Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften (Dr. rer. pol.)
von der Fakultät für Wirtschaftswissenschaften
der Universität Fridericiana zu Karlsruhe
genehmigte Dissertation.

Query-Based Multicontexts for Knowledge Base Browsing

Julien Tane
D.E.A

Tag der mündlichen Prüfung: 05. Februar 2007
Referent:
Prof. Dr. Rudi Studer, Universität Karlsruhe (TH)
Koreferent:
Prof. Dr. Gerd Stumme, Universität Kassel

2007 Karlsruhe

Abstract

The amount of accessible information is rising at a great pace. At the same time, the need to grasp and understand information becomes even more crucial. To achieve this, it is important to improve the interaction between users and data collections.

Querying and browsing are the two main kinds of interaction with complex data. It is crucial to be able to combine them efficiently and effectively. On the one hand, the purpose of querying is to return the precise content of some part of a large data collection, but requires some knowledge of the data model to query it. On the other hand, browsing usually combines relational exploration with some sort of topic or conceptual hierarchy, allowing the user to interact with the data to refine or adapt the representation to his specific needs. However, current techniques cannot mix both approaches successfully to offer an exploration paradigm capable of displaying complex relationships in the data.

In this thesis, we propose a new approach to create and navigate contextual views *by integrating browsing and querying to explore complex data collections such as knowledge bases. Inspired by existing techniques from the field of Formal Concept Analysis, we introduce a new structure called* Query-based multicontext. *This structure can be seen as the space of views over the data collection. Each view is generated from a data source using a query-based intensional representation of their content. We further define different operators as well as a generic template mechanism which simplify the view definition process. As a main application of the approach, we designed and implemented a knowledge base browser relying on different strategies to define pertinent views.*

Finally, we evaluated our work in two manners. First, we discuss the benefits of our knowledge browsing approach with respect to other approaches found in the literature. Then we compared the performance of users on diverse visualisation tasks when using one of three visualisation.

To Philipp and Jorge, for their patience...

0.1 Acknowledgements

After spending so much time at a weary task, fighting so many different negative feelings such as frustration, disappointment, which sometimes lead you to the road of depression, you really know then how important friends and colleagues have been in the final success. The least I can do is to show my gratitude with these few lines.

First of all I want to thank Philipp Cimiano and Jorge Gonzales for their support at a turning point of my life as well as the many discussions and frustrations which they had to endure. In particular, I want to tell my utmost respect and gratitude to Philipp for the time, energy he needed in trying to keep me focused.

I would like to express my extreme gratitude to Prof. Studer and Stumme for offering me the possibility to begin and finish this thesis in the most comfortable positions. Thanks to their wise guidance, the work atmosphere of their research groups have been ideal for the work I had to do. Their calm, understanding and professionalism have been exemplary during the whole time.

Thanks to Dr. Philipp Cimiano again and Dr. Pascal Hitzler for many crucial comments and corrections which improved the quality of this thesis greatly.

I owe also many thanks to all the proof and comment readers of my thesis or of parts of it: Philipp Cimiano, Prof. Studer, Prof. Stumme, Pascal Hitzler, Daniel Oberle, Raphael Volz, as well as to my brother Pierre and my dear friend Etienne Ailloud who read many pages of my dissertation and gave me a more in dependant feedback.

My work would have not been such a pleasure, if my colleagues had not been so fun to work with, not to mention their excellent work. I want to mention in particular Christoph Schmitz, Siegfried Handschuh, Philipp Cimiano, Andreas Hotho, Duc Thanh Tran and Prof. Stumme for their patience. A special thank you goes to Christoph, with whom I cooperated closely for the PADLR project and who made useful comments on the early idea behind this thesis.

A great thank to Boris Motik and Serhiy Yevtushenko, whose works have served as a solid basis for much of my own work.

Once someone said: 'Go to a woman...' when you need to pump up your confidence. This has proved to be true for me to a point that I want to thank many different women for their cares and words but also for their just being there. So thanks to Saartje Brockmann, Sventje Dieter, Sofia Pinto, Julia Rothmann, Julia Dernbach, Juliane Päsch, Lotta Koch, Frederieke Layer, Jenny Ross, Sylvie, Friede und die Swerge Königin.

A special 'thank you Saarjte' goes to Saartje Brockmann for the right words at a very crucial moment.

I must also thank my flat mates: Hartmut Aichert, Jens Pliester and Yimin Wang for all their useful comments, their friendship and their tolerance.

My gratitude goes to Klaus Täger whose friendship and presence has meant much to me during this time. For coming often to Karlsruhe and many great discussions, I also thank Etienne. For somewhat useful comments on life, much fun and many interesting discussions, thanks to Jochen and Toby.

Last but not least, since they were always there for me, I want to thank my family, especially my mother and father, but also my brothers as well as Karin Grankvist Tane who all know how to remind but also forgive me my many shortcomings. Thanks to my uncle, aunts, cousins and cousines, who I should have seen much more often than I did. A particular thanks to Florence and Elise Tane for their wonderful little film.

Contents

0.1 Acknowledgements . . . 5

1 Introduction **17**
1.1 Motivation . . . 18
1.1.1 A Scenario . . . 18
1.1.2 Formal Concept Analysis . . . 21
1.2 Research Questions . . . 24
1.2.1 Contextualised Views . . . 24
1.2.2 Exploration Process . . . 24
1.3 Contribution . . . 26
1.4 Outline of the Thesis . . . 27

2 Preliminaries **29**
2.1 Knowledge Representation and Applications . . . 29
2.1.1 Aspects of Knowledge Representation . . . 29
2.1.2 Application: Visualising the Semantic Web . . . 33
2.2 Logical and Rules Paradigms . . . 37
2.2.1 Description Logics . . . 38
2.2.2 Datalog . . . 46
2.3 Formal Concept Analysis . . . 48
2.3.1 Basic FCA . . . 48
2.3.2 Context Operators . . . 51
2.3.3 Many-valued Contexts and Conceptual Scaling . . . 53
2.3.4 Multicontexts . . . 57
2.4 Summary . . . 58

3 Query-Based Multicontexts **59**
3.1 Intensional Context Representations . . . 59
3.2 Query-based Multicontexts . . . 62
3.2.1 Definition: Query-based Multicontext . . . 62
3.2.2 A Simple Example . . . 64
3.2.3 An SQL Example . . . 66
3.3 Operators . . . 68
3.3.1 Generic Query Operators . . . 71
3.3.2 Typed Query Language . . . 75
3.3.3 Context Index Operators . . . 77
3.4 Semantic Query-Based Multicontext . . . 86
3.4.1 Generic Ontology Model . . . 86
3.4.2 Semantic Query Infrastructures . . . 88

3.5 Constructors . . . 91
3.5.1 Relation Constructors . . . 94
3.5.2 Hierarchy Constructor . . . 95
3.5.3 Instantiation Hierarchy Constructors . . . 96
3.5.4 Subsumption/Instantiation Hierarchies Constructors . . . 97
3.5.5 The Join Relation Context Constructor . . . 99
3.5.6 The CoRelation Constructor . . . 101
3.5.7 Application-dependent Context Constructors . . . 101
3.6 Summary . . . 105

4 Knowledge Base Exploration **107**
4.1 Exploring Using Views . . . 107
4.1.1 Visualising the Information Space . . . 107
4.1.2 Views for Knowledge Base Exploration . . . 111
4.1.3 Scenario: Getting to Know AIFB . . . 114
4.2 View Interaction . . . 121
4.2.1 Visualisation . . . 124
4.2.2 Selection . . . 128
4.2.3 Paradigm Comparison . . . 132
4.3 View Definition . . . 133
4.3.1 View Definition Strategies . . . 133
4.3.2 Default Views . . . 136
4.3.3 View Parametrisation . . . 138
4.3.4 View Adaptation . . . 143
4.4 Summary . . . 147

5 Implementation **149**
5.1 Architecture Overview . . . 149
5.1.1 The Main Components . . . 149
5.1.2 One Round Example . . . 152
5.2 QBMC Infrastructure . . . 153
5.2.1 Query Management Module . . . 153
5.2.2 Context Index Management Module . . . 156
5.2.3 Context Construction Engine . . . 157
5.3 Context Infrastructure . . . 159
5.3.1 Query Evaluation and Query Results Storage . . . 162
5.3.2 Context Storage . . . 164
5.3.3 Context Creation . . . 164
5.3.4 Lattice Generation . . . 168
5.4 Semantic Query Infrastructure Implementation . . . 168
5.4.1 Methodology for Adapting Data Sources . . . 168
5.4.2 Adapting KAON . . . 170
5.4.3 Adapting OWL Using KAON2 . . . 171
5.5 The Graphical User Interface . . . 173
5.5.1 Main Components . . . 174
5.5.2 Parametrisation using Direct Manipulation . . . 179
5.5.3 The Lattice Panel . . . 183
5.5.4 Lattice Partitioning . . . 184
5.6 Underlying FCA Framework . . . 186
5.6.1 Alternatives . . . 189

5.6.2 Concept Explorer 190
5.7 Summary 191

6 Evaluation **193**
6.1 Comparison with other Approaches 193
6.1.1 ClusterMap 193
6.1.2 Toscana Framework 195
6.1.3 Docco Framework 197
6.1.4 The Conceptual Email Manager 201
6.2 Visualisation Evaluation 203
6.2.1 Description of the View Paradigms 204
6.2.2 Question Types 206
6.2.3 Context of the Evaluation 208
6.2.4 Results 210
6.3 Summary 212

7 Discussion **213**
7.1 Contributions 213
7.2 Future Work 214
7.2.1 Open Questions 214
7.2.2 Other Applications 217
7.3 Final Words 218

A Ontology **229**

List of Figures

1.1 Diagram showing a browsing path for our researcher Mikko Malli. 19
1.2 The line diagram showing the concept lattice for the relation between researchers and the research topics *text-mining*, *knowledge discovery* and *data mining*. 23
1.3 The knowledge exploration process. 25

2.1 The Semantic Web layer cake (source [Horrocks et al., 2005]). . . 34
2.2 An example of RDF graph together with a RDF schema stating that Tim Berners-Lee is the author of a document with the title Information Management: A Proposal. 35
2.3 A small example of a description logic knowledge base. 42
2.4 The concept lattice of the food and people context of Example 2. 51
2.5 Example of the constructors defined in [Ganter and Wille, 1999] 53
2.6 The formal concept lattice for the processor scale. 55
2.7 The formal concept lattice for the rating scale. 56
2.8 The nested line diagram for the computers embedding the rating scale in the processor scale. 57

3.1 The model work flow 61
3.2 The concept lattice corresponding to the context generated from the context index $p :=$ (project,topic,isOnTopic). 65
3.3 The lattice for the SQL example using the Tables 3.3, 3.4 and 3.5. 68
3.4 Intuitive diagram for the union context index operator. 81
3.5 A schematic representation of the subsumption hierarchy, instantiation hierarchy, subsumption/instantiation hierarchy constructor. 96
3.6 The lattice of the concept hierarchy. 96
3.7 The lattice of the instance hierarchy for the subconcepts of: Person, Document, Research Group and Conference. 98
3.8 The lattice of the instance hierarchy for the concepts: Person, Professor, PhDStudent, Assistant. 99
3.9 The lattice of the concept hierarchy for the instances: **Rudi Studer, York Sure, Julien Tane**. 100
3.10 The lattice of the join between research topics and research groups through the project. 102
3.11 The main idea behind the CoRelation constructor. X is the start author and all the light-coloured nodes are in the resulting context. 103
3.12 The concept lattice displaying the relation between the publication written by Julien Tane and his coauthors. 103

3.13 The lattice for news filtering. 104

4.1 Diagram schematising the concept of Database Content Space (source: [Boyle et al., 1993]). 108
4.2 Diverse abstract representations of some views. a) represents an abstract representation of a view, b) is an example of simple relation view, c) is a view resulting from a join constructor and d) shows a view with a complex definition for the attribute set. . 113
4.3 The interaction process for a query-based multicontext browser. . 115
4.4 The start view presenting the main concepts of the knowledge base.115
4.5 The distribution of the research topics of the AIFB text-miners. 118
4.6 The distribution of the publications written by the text-mining researchers. 119
4.7 The concept lattice of Andreas Hotho's publications visualised with authors as attributes. 119
4.8 The concept lattice of Andreas Hotho's research topics and the colleagues sharing them. 120
4.9 The tree displaying part of the subsumption-instance hierarchy of the ontology. 124
4.10 The tree displaying the distribution of the projects carried out by the research groups. 126
4.11 The graph displaying the distribution of the projects carried out by the research groups. 127
4.12 The lattice displaying the distribution of projects carried out by the research groups. 128
4.13 The selection settings panel. 132
4.14 An example of web formular to create a simple context of author publications. 140
4.15 The wizard allowing the creation of the join. 142
4.16 The join relation indirectly indicating the research topics of AIFB research groups by looking at which topic its members work on. . 144
4.17 The adapted view displaying the publications assigned to the Efficient Algorithm Group or to the Knowledge Management Group and who were written by a Professor or an Assistant. 146

5.1 An overview of the architecture of the QBMC Browser. 150
5.2 The lattice of the join which shows which research topics are dealt by some AIFB professors through one of the projects they coordinate. The topics at the top are not coordinated through some projects. 154
5.3 The class diagram for the implementation of the context infrastructure. 161
5.4 The layout of the main components of the query-based multicontext browser. 175
5.5 The main panels while exploring the different projects of the AIFB research groups. 176
5.6 The context index panel for the context showing the research topics for the projects of the Knowledge Management Chair. . . 177

5.7 The constructor parametrisation panel with the parameters for the view displaying the distribution of the research topics of the projects carried out by the Knowledge Management Group. . . . 177
5.8 The constructor parametrisation panel with the parameters for the view displaying the distribution of the research topics of the projects carried out by the Knowledge Management Group. . . . 178
5.9 The projects of the Knowledge Management Group dealing with text-mining displayed in the entity panel. 179
5.10 A panel to refine a query by dropping Person over the query parameter inst({Publication}). 180
5.11 The wizard obtained when dropping the relations and on the join context construction panel. 182
5.12 A panel to save parts of a formal concept of the lattice for the context index (inst({Topic}),inst({Person}),$<$ isWorkedOnBy $>$). . 184
5.13 A lattice with a partition of the top elements in two classes: {Att1,Att3} and {Att4,Att5,Att6}). 185
5.14 The concept lattices of the the two classes {Att1,Att3} and {Att4,Att5,Att6} of the partition of the lattice of Figure 5.13. 186
5.15 The concept lattice of a class of a partitioning from the concept lattice of $<$ workedAt $>$. 187

6.1 Job vacancies organised by economic sector. 194
6.2 The nested diagram of the lattice for comparing PC price with type of case. 196
6.3 The zoomed diagram lattice of desktop computer. 198
6.4 The lattice for the query: Java "Formal Concept Analysis" "Conceptual Graphs" in Docco. 199
6.5 The Mail-Sleuth user interface (source [Cole et al., 2003a]). . . . 202
6.6 The nested diagrams showing the project subject for the members of the KVO Institute (source: [Cole et al., 2003a]). 202
6.7 The lattice showing the mail project topic relation of the KVO work group. 203
6.8 The Graph view 205
6.9 The Tree View 206
6.10 The Lattice View 206
6.11 Average Time in seconds needed to answer the three questions in the three para–digms. 210

List of Tables

1.1 Table representing the works–on–topic relation between text-mining authors and their research topics. 22
1.2 The relation between the research topics *text-mining*, *knowledge discovery* and *data mining* and researchers working on one of these topics. 23
1.3 An intensional representation of a table. 24

2.1 Concept constructors 40
2.2 Role constructors 40
2.3 The semantics of the description logic A-box axioms. 41
2.4 The semantics of the description logic T-box axioms. 43
2.5 A simple example of a formal context 49
2.6 An example of many-valued context. 54

3.1 The context $\mathbb{K}_p$ for the context index $p :=$ (project,topic,isOnTopic). 65
3.2 The context $\mathbb{K}_p$ of the car sharing context index. 67
3.3 The vehicle table for the car sharing example. 69
3.4 The vehicle-location table for the car sharing example. 69
3.5 The location-position table for the car sharing example. 69
3.6 The table defining the operators symbols for the generic query operators (q_1, q_2, q_3, $q_4 \in L_1$ and q_3 and $q_6 \in_2$). 72
3.7 The datalog implementation of the query operators of table 3.6. . 75
3.8 Table stating the main use of the context index operators. 86
3.9 Definition of the query primitives of the query languages L_C, L_R and L_I for query infrastructure $\mathbb{Q}_{Kb}$. 92
3.10 Definition of the query primitives of the query languages L_{Ri},L_{CI}, L_{CR}, L_{CC} and L_{RR} of query infrastructure $\mathbb{Q}_{Kb}$ 93

4.1 The main phases of an interaction cycle of the knowledge base exploration framework. 114
4.2 Techniques helping the visualisation. 123
4.3 Table summarising the advantages and disadvantages of the three interaction paradigms for views. 133
4.4 Table recalling the diverse query definition paradigms (part A, cited from [Shneiderman and Plaisant, 2005]). 134
4.5 Table recalling the diverse query definition paradigms (part B, cited from [Shneiderman and Plaisant, 2005]). 135

4.6 The table recalling the diverse query definition paradigms (part C, cited from [Shneiderman and Plaisant, 2005])). 136

5.1 Typical steps for the evaluation of queries in a database system (adapted from [Vossen, 1994], Chapter 17, page 428). 155
5.2 The naive algorithm for the evaluation of a context index. 159
5.3 Features of the context operators implementations. 165
5.4 Features of the constructors implementations. 167
5.5 List of the methods of the main entities of KAON API 172
5.6 The algorithm used to compute queries with nominals. 174
5.7 A summary of the main possibilities for the creation of queries using a drag-and-drop. 181
5.8 Calculate the partition of the top elements of the lattice. 188

6.1 Illustration of different types of queries which can be formulated when using the Docco framework. 200
6.2 Confidence and t-values level of lattice-tree and lattice-graph time comparisons. 211

Chapter 1

Introduction

In recent years, the amount of information available electronically has risen tremendously. Coping with this information growth remains one of the great research challenges for the computer science community. Search engines on the Internet are quite successful at providing access to much of the available knowledge, but their techniques remain limited to advanced indexing mechanisms in which background knowledge only plays a minor part. Yet, semantic technologies have matured in recent years with new advances concerning the acquisition, organisation and storage of knowledge (see [Studer and Staab, 2003]). Many application domains have recognised their benefits. For instance, the crucial task of integrating vocabularies and taxonomies from diverse branches of medicine, has led to tremendous efforts of standardisations in the field of biomedical informatics (see [Smith and Ceusters, 2006]). One of the key elements of these efforts is the use of *ontologies*, which have been defined in [van Harmelen and Horrock, 2000] in the following terms:

An ontology is a consensual, shared and formal description of the concepts that are important in a given domain.

In other words, ontologies describe knowledge of a domain such as bioinformatics (see [Rector, 2003]) using a formal structure capable of specifying relations between the concepts of a domain. For example, the Galen[1] ontology captures clinical terminology and has been used in a number of applications such as language generation, disambiguation, user interfaces or quality assurance (see [Rector, 2003, Rector and Rogers, 2006]). The benefit of ontologies in comparison to traditional schemas or vocabularies lies in their formal semantics. The formal semantics ensure a stable interpretation of the meaning given to an ontology as well as allow the deployment of supplementary services such as inferencing, consistency checking, etc.

While reasoning about the knowledge is important, humans are also key contributors and consumers of knowledge. It is therefore crucial to provide them with means of interacting with this knowledge. Diverse approaches have been proposed in the literature to achieve this goal (see [Fluit et al., 2002, Hearst, 1999, Ahlberg et al., 1992, Furnas and Zacks, 1994]). Most approaches either focus on interacting with the basic objects of the knowledge base (instances and their relations) or on summarising the large amount of information

[1]See `http://www.opengalen.org`.

available in the knowledge base. A third approach consists in focusing on specific parts of the knowledge base. Yet, defining these parts is a non trivial process and is the main topic of this thesis. Our purpose is to investigate the definition and navigation of *contextualised semantic views* over knowledge bases. These views may be seen as snapshots of specific parts of a knowledge base. Using our formalisation of the concept of contextualised views, we present a new way of exploring and interacting with knowledge bases. The novelty of our work lies in the combination of knowledge base queries and *Formal Concept Analysis*, a conceptual clustering theory.

Formal Concept Analysis is an expanding field of research dealing with the relation between tabular data, called *formal contexts* (i.e. binary tables or attribute-value tables) and specific order structures, called *concept lattices*. Concept lattices possess interesting features for knowledge browsing and information management in general.

While previous approaches using Formal Concept Analysis can also be seen as contextualised views, our approach differs in that it introduces a way of defining and manipulating precisely the content of formal contexts. To achieve this, we introduce a new structure: the *Query-Based Multicontext*. A Query-Based Multicontext is a formal space consisting of all the formal contexts which can be generated by querying a given data source. We show that it is possible to manipulate the elements of this space by manipulating intensional representations called *context indices*. Each of these context indices represents a contextual view. We describe how to create and manipulate the views in order to explore the knowledge base. Moreover, we describe the implementation of this framework on top of a specific knowledge base framework.

Finally, before concluding this thesis, we present the evaluation we performed in order to assess the benefits of our approach. We then use this evaluation as a basis for discussing supplementary related work as well as some of the possible extensions we consider relevant for our approach.

1.1 Motivation

To motivate the goals behind our work, we introduce a scenario about a researcher who is interested in the research done at the AIFB institute.[2] Then we describe briefly Formal Concept Analysis in order to motivate its use in this thesis.

1.1.1 A Scenario

Mikko is a researcher in the field of Formal Concept Analysis interested amongst other things in *text-mining* and *ontology-learning*. He noticed that some persons from the AIFB Institute work on topics relevant to his own interests and would like to get an overview of their profiles and the projects they work on.

Mikko browses the AIFB portal. The AIFB portal offers a simple means of learning more about the institute. His browsing pattern is illustrated by the diagram in Figure 1.1. Starting from the AIFB main page, he follows the links found in each page (the order of the visits is indicated by a number beside each arrow). The dashed lines show that other choices would have been possible.

[2]See `http://www.aifb.uni-karlsruhe.de`.

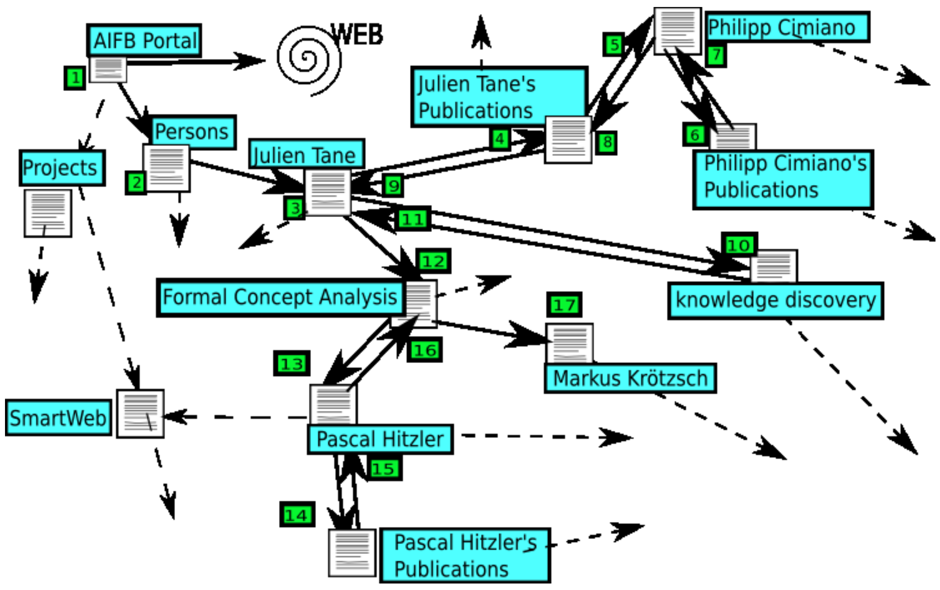


Figure 1.1: Diagram showing a browsing path for our researcher Mikko Malli.

This kind of browsing is *entity-centred*. It displays one entity at a time (one per page actually), displaying all its properties, for example relations to other entities. Unfortunately, this makes it difficult for Mikko Malli to get an overview of the relations between researchers, topics and projects. To get an overall picture of the institute, Mikko needs to browse many pages to find out what are the topics and projects AIFB researchers work on. At each page, he must decide between diverse links or may backtrack to some of the pages he has already read. For instance in Figure 1.1, when looking at Philipp Cimiano's page, Mikko may go on to Philipp Cimiano's publications or go back to Julien Tane's page to continue to the Formal Concept Analysis page. He also needs to remember some information he gathered at each page. While this browsing approach allows him to get an idea of the relation between persons, projects and research topics, it is time consuming and does not give a good overview of the relationships between persons and topics for example. Fortunately, the portal provides a downloadable knowledge base[3] describing the research done at the AIFB institute. Using this knowledge base, he hopes to get an overview more quickly.

Exploring the Knowledge Base

The AIFB knowledge base contains information about many aspects of the research at AIFB. The conceptual schema of this knowledge base is a research oriented ontology used in other applications. Since this ontology is used in most of the examples of this thesis, we describe it in more detail in Appendix A.[4] The following list gives an idea of the kind of information present in this knowledge

[3]See http://www.aifb.uni-karlsruhe.de/about.html.

[4]We refer the interested reader to the recent article: [Sure et al., 2005] on the design decisions that underlie this ontology.

base:

- research groups: name, members, projects
- persons: name, telephone, fax, homepage, publications, research group
- publications: title, authors, publication, location, topics
- projects: name, financing institution, research groups carrying them out
- research topics: projects dealing with these topics, persons working on these topics

It specifies, for example, that the *PhD Student* Julien Tane is a member of the *research group* Knowledge Management. It also contains metadata over his publications and states that he is interested in the *research topic* Formal Concept Analysis.

The amount of information available from the knowledge base is large. It contains metadata on more than 900 persons and even more publications. Most of these persons are authors of several publications, some authors having more than 100 publications, while others are only present because they coauthored one publication with a person working at the institute.

Moreover, Mikko Malli's prime interest is not the AIFB in general. He would rather focus on the part of the knowledge base corresponding to his research topic. Here are some questions which Mikko might have in mind:

1. Who worked or is working on research topics related to Formal Concept Analysis at the institute?
2. What are the other topics these researchers work on?
3. Who published articles on a research topic relevant to my research?
4. What are or were the projects relevant to my interests?
5. Which researchers work together on which project for my field of research?
6. Which of these researchers published together, on which field of research?

These questions show that Mikko Malli wishes to focus on a part of the knowledge base centred around Formal Concept Analysis. To achieve this, three solutions come to his mind. The first would be to use a typical ontology browser like Protege,[5] but like the AIFB portal, these browsers use an entity centred approach. Mikko would have to take a look at many entities individually and follow their relationships to others in order to gather knowledge, while he is mostly interested in relations between groups of entities, for example the text-mining researchers and their projects or publications, etc.

The second alternative would be to use a graph-based ontology browsing tool like the KAON OIModeller.[6] Contrary to the previous approaches, using a graph visualisation paradigm enables him to see the relations between diverse individuals at the same time, but the graph is filled in a more or less ad hoc

[5]See http://protege.stanford.edu.
[6]See http://kaon.semanticweb.org.

manner, depending on the paths he follows and the particular elements he selects. This means that he would have to concentrate on the next step instead of the information presented. Moreover, this approach does not scale when the size of the graph increases.

The third alternative is to query the knowledge base, but this solution has major limitations. The first limitation is that he needs to know how to formulate the corresponding queries. If he manages to formulate the queries, he needs a way to visualise their results. Finally, he must be able to use part of the results to formulate new queries.

From the drawbacks of these three alternatives, it is possible to extract the basic requirements of a knowledge base browsing tool.

Contextualised Views of AIFB

The above six questions give an idea of the interests of Mikko. Each of these questions can be seen as a conceptual representation of what should be displayed. Each of these questions focuses on a certain context of interpretation; therefore we call them *contextualised views*.[7]

While the answers of the first question build a set, which can be easily displayed as a list, the answers of the second question form a relation between two sets: the text-mining researchers and their research topics. This relation can be presented to the user using some table equivalent to Table 1.1. Though the table is still readable, it does not show the intrinsic structure of the relationship between these two groups. This leads to the relevant research problem of finding a pertinent way of displaying this binary relation. While for many purposes a table like Table 1.1 may be suitable, this representation is not suitable to give more insight on which of these researchers share certain research topics. Research in the field of Formal Concept Analysis has shown that the diagram of concept lattices can be used for this purpose (see [Eklund et al., 2004]). This is what we show in the following paragraph.

1.1.2 Formal Concept Analysis

Formal Concept Analysis is a field of applied mathematics and computer science introduced by Rudolf Wille in 1982 (see [Wille, 1982]). Its goal is to investigate the relations between data tables called *formal contexts* and ordered structures, called *concept lattices*. While tables are frequently used to represent and store data, concept lattices have been used in fields as diverse as software engineering, data mining, ontology learning, conceptual exploration and psychology (see [Huchard et al., 2002, Hotho and Stumme, 2002, Cimiano et al., 2003, Wille, 1997, Borg, 1992]).

In Formal Concept Analysis, binary relations are usually called *formal contexts* and correspond to some cross table as shown in Table 1.2. The rows of the table are usually called objects and the columns attributes. For instance, Table 1.2 represents the works-on-topic relation between researchers as objects and three attributes: *knowledge discovery*, *text-mining* and *data mining*.

[7]The term *view* has been chosen because of the similarity of the notion with the well-known notion of views in database terminology, where views are table which are generated using queries over the database.

Table 1.1: Table representing the works–on–topic relation between text-mining authors and their research topics.

Researcher	Research Topics
Andreas Hotho	knowledge discovery, text-mining, knowledge portal, data mining, knowledge management, ontology engineering, semantic web, Scalable Data Mining, business engineering
Julien Tane	knowledge discovery, text-mining, ontology-based KM systems, knowledge systems, Formal Concept Analysis
Stefan Bloehdorn	knowledge discovery, text-mining, knowledge representation and reasoning, machine learning, multimedia systems
Steffen Staab	artificial intelligence, knowledge discovery, text-mining, knowledge portal, ontology-based KM systems, knowledge systems, data mining, knowledge management, ontology engineering, semantic web, development of KM systems, information extraction, ontology learning, semantic annotation, knowledge representation and reasoning, agent systems, constraint programming, E-learning, e-business, hypermedia systems, KM methodology, modelling, semantic web infrastructure
Alexander Mädche	knowledge discovery, text-mining, knowledge portal, artificial intelligence, data mining, Ontology Engineering, knowledge representation languages, development of KM systems, information extraction, ontology learning, semantic annotation, knowledge representation and reasoning
Stefan Klink	text-mining, knowledge portal, Digital libraries, Semantic web Services, office information systems, virtual university

Interestingly, the theory of Formal Concept Analysis states that for any given formal context, a new structure called a *concept lattice* can be generated which contains exactly the same information. The concept lattice of the formal context found in Table 1.2 is displayed using the diagram on the left[8] of Figure 1.2. Using this diagram, it is possible to visualise the relationship between these researchers and the three research topics selected. Each node stands for a set of researchers and a set of research topics. For example, the node labelled with data mining represents the pair of sets:

- attributes: {knowledge discovery, data mining}
- objects: {Gerd Stumme, Jorge Gonzalez, Jens Hartmann, Ketut Ngurah Sudharma, Olivier Sandel, Alexander Mädche, Steffen Staab Andreas Hotho.}

We recalled at the top of Figure 1.2 the two crucial conventions necessary to read the diagram. These conventions state that for any given node the elements

[8]The right part of the Figure displays meta information to help the reader in understanding the diagram.

Table 1.2: The relation between the research topics *text-mining*, *knowledge discovery* and *data mining* and researchers working on one of these topics.

person/topics	knowledge discovery	data mining	text-mining
Alexander Mädche	×	×	×
Andreas Hotho	×	×	×
Steffen Staab	×	×	×
Julien Tane	×		×
Stefan Bloehdorn	×		×
Gerd Stumme	×	×	
Jorge Gonzalez	×	×	
Jens Hartmann	×	×	
Ketut Ngurah Sudharma	×	×	
Philipp Cimiano	×		
Stefan Klink			×
Rudi Studer	×		

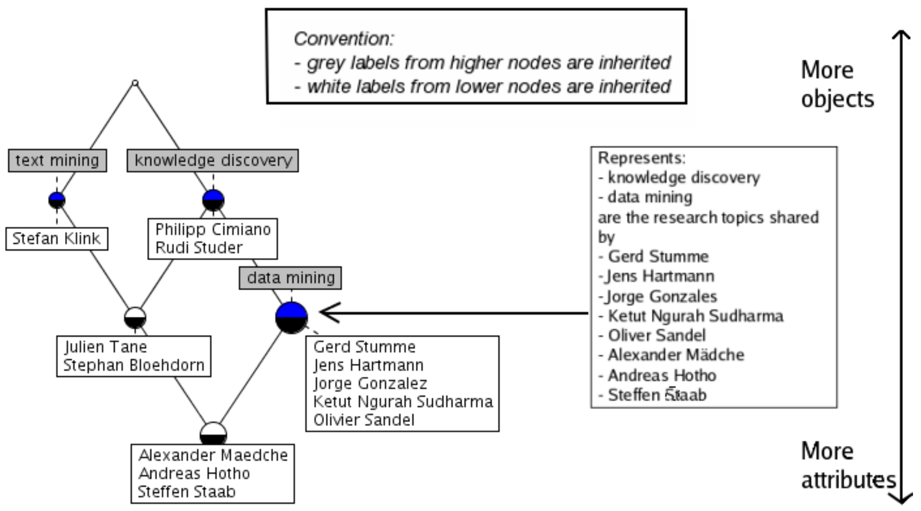


Figure 1.2: The line diagram showing the concept lattice for the relation between researchers and the research topics *text-mining*, *knowledge discovery* and *data mining*.

found in the grey labels of higher nodes are also in the attribute set of a node, while the elements found in the white labels of lower nodes are also in the object set of a node.

1.2 Research Questions

1.2.1 Contextualised Views

The scenario we presented shows that for some purposes it is useful to focus on only some parts of the data available. In some way, this task can be seen as contextualisation, i.e. choosing a context[9] of interpretation of the data and displaying the data in this context. The purpose of our work is to provide methods and tools to perform such a task.

Consider the *author* relation between publications and their authors. In the knowledge base, the amount of persons and publications present is too large to be browsed in a sensible manner (the AIFB knowledge base contains more than 900 authors and even more publications). Typically a knowledge base offers a number of primitives to express specific parts of the knowledge base. For example, it is possible to define a query[10] retrieving *PhD Students* of the Knowledge Management *research group*:

$$\exists x,\ \mathsf{phdStudent}(x) \sqcap \mathsf{memberOf}(x, \mathsf{Knowledge\ Management})$$

In the same way, it is possible to restrict the search on publications of a certain kind: conference paper, journal article which have been published in or before a given year.

In general, it is useful to consider a view as a table of the form given in Table 1.3. This kind of table occurs very frequently and diverse paradigms have been developed to display the information stored in them (see [Ziegler et al., 2002]). However, in this thesis we mainly concentrate on the use of *concept lattices* which we deem appropriate as visualisation paradigm in certain situations. The next section discusses briefly the process necessary to explore the knowledge base.

Table 1.3: An intensional representation of a table.

	attributes: criteria used in the structuring
objects	relation between objects and attributes

1.2.2 Exploration Process

In order to motivate our approach, we first consider a generic exploration process consisting of four phases. This model is depicted in Figure 1.3. The user starts with a goal or a task and some hypotheses in phase I. In phase II, he interacts with the tool in order to communicate what he wants to see. In analogy to the process of query definition, we call this phase *view definition*. The result of

[9]In this thesis, the word *context* can refer to different things such as a given formal context or the context of use. To avoid confusions between the two, we prefer using *formal context* when talking about the mathematical structure. At the places where the word context refers to the situation or the environment, we use the expressions: *situation's context* or *context of interpretation*.

[10]This query uses a syntax adapted from first order logic. Capitalised words correspond to concepts or relations, $\mathsf{memberOf}(x, \mathsf{Knowledge\ Management})$ denotes the elements memberOf the group Knowledge Management and $\sqcap$ is an operator denoting the intersection of the result of the two sides.

Phase II is a request transformed into queries to the knowledge base in Phase III, the *view creation* phase in order to create the corresponding view. Once the answer to the queries has been computed, it is presented to the user to be visualised. In the *view interaction phase*, Phase IV, the user analyses the content of the view. This analysis may lead to new goals and hypotheses and a new cycle of the browsing process may be entered.

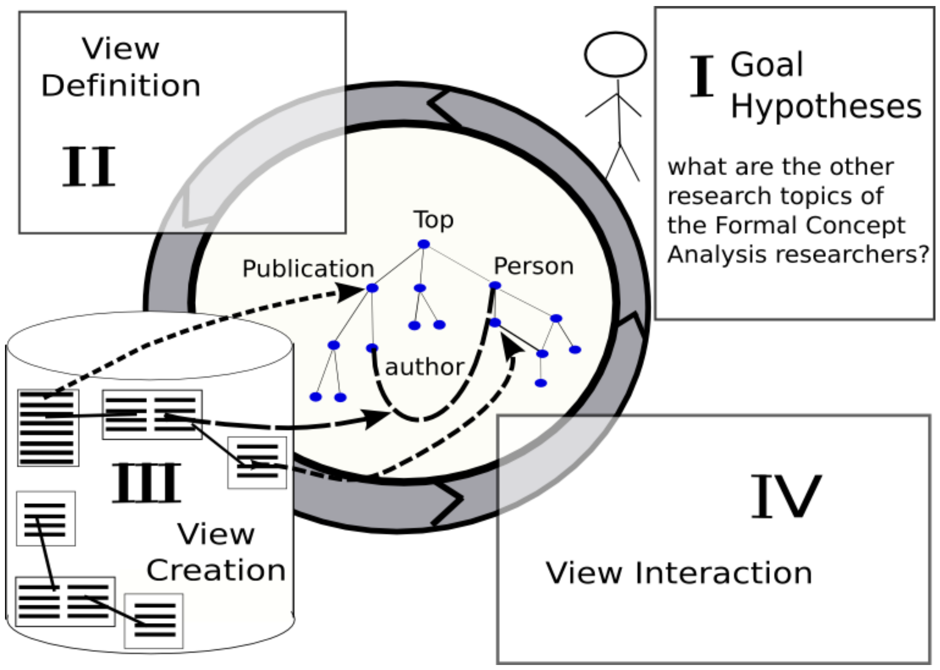


Figure 1.3: The knowledge exploration process.

If we investigate in further details this model, a few questions arise:

- What kind of model is suitable for a contextualised view?
- What kind of knowledge representation and query languages can or should be used?
- How does the user formulate his requests?
- How can or should the results of these requests be visualised?

The approach we propose in this thesis investigates possible answers to these questions using a combination of ontologies and Formal Concept Analysis. We focus particularly on Phases II, III and IV. Though we believe that Phase I is crucial for specific applications,[11] we discuss these aspects only briefly when this has an impact on the design of the actual implementation.

[11]Some interesting ideas might be found for example in [Kuhlthau, 2005].

View Definition

In exploration approaches centred on entities, the choice of the next view is determined by choosing the next entity to visualise among diverse alternatives. The contextualised approach relies on the definition process of the next relevant view to display. It is critical to study the diverse approaches which can be used to create views. Moreover, since the view definition process is not the primary goal, this process should not be too cumbersome for the user. This means that the design of an exploration framework should find an appropriate trade-off between the expressivity of the knowledge representation language and the overall complexity of the definition process.

Another important requirement is that the exploration method must be capable of reusing the elements selected in previous phases or cycles of the browsing. To do this, we consider diverse alternatives for the selection process during the exploration.

View Interaction

Once a view has been created, it must be displayed to the user. The choice of the most appropriate view visualisation paradigm is not a trivial one, since diverse aspects may play a role in the exploration process. Therefore, it is important to investigate criteria which help decide which view paradigm is the most suitable to support the user in visualising the content of the view presented to him.

1.3 Contribution

The requirements of the browsing approach presented in the previous section lead us to develop a new framework combining ontology queries and Formal Concept Analysis to browse knowledge bases. The contributions of this thesis to achieve this goal can be split in two main parts: a contextualised view framework for knowledge bases and a novel knowledge base browsing approach.

Contextualised Views On Knowledge Bases

We develop a new model allowing to construct complex contextualised views by combining Formal Concept Analysis with queries on knowledge bases. We define for that purpose a new structure called *query-based multicontext*. This structure consists of formal contexts which can be used as input to traditional Formal Concept Analysis algorithms. Each of these contexts is generated from a surrogate representation[12] which we call *context index*. Each context index consists of a triple (q_1, q_2, q_3) of three queries which must respect some basic properties in order to be evaluated as a formal context of a query-based multicontext. Moreover, we define operators on context indices which correspond to operators on formal contexts. In addition to the simplification they bring to operate and combine diverse formal contexts, the context index operators play a central role in the definition of *constructors*, which are kinds of high-level templates to ease the construction of formal contexts. Finally, as a proof of concept

[12]A surrogate is an object representing a more complex object and often used in their place for manipulation purposes.

of the possibility of manipulating context indices intensionally, we implemented the query-based multicontext theory.

A Browsing Framework

The query-based multicontext theory is used to implement a knowledge browsing framework for knowledge bases. This browsing paradigm uses formal contexts as semantic views of parts of a knowledge base. Each view can be displayed either as a graph, as a tree or as a lattice. Our browsing approach relies mainly on three aspects: the query-based multicontext infrastructure, a methodology for creating complex views based on constructors, and diverse paradigms to present the information to the users. Part of the evaluation of our approach was to compare the performance of users as well as their reactions when interacting with the three different visualisation paradigms.

Finally, we also compare our approach to other Formal Concept Analysis based approaches and study how their approach could be integrated in the query-based multicontext browser.

1.4 Outline of the Thesis

This thesis is divided in seven chapters which we now outline.

Chapter 2 deals with the state-of-the-art and the necessary preliminaries to understand the work presented in this thesis. We first discuss aspects of knowledge representation as well as an application.

Chapter 3 presents the basic theory of the query-based multicontext. After a general overview of the principle behind our approach, we introduce the basic definition of query-based multicontexts, followed by some illustrative examples. The basic structure is then extended using more expressive query infrastructures as well as operators which allow for complex operations on formal contexts. Finally, a high level template mechanism is presented which simplifies the role of the user in user interaction.

An application of this theory to knowledge browsing is presented in Chapter 4. Notably, we first discuss the principle of browsing with user defined views. We also address the important issues of view interaction and view definition.

Chapter 5 describes our implementation of the query-based multicontext.

Chapter 6 addresses the evaluation of our approach using two methods. While the first one is a qualitative comparison with other approaches, the second one presents a user evaluation we performed to compare three visualisation paradigms for our views.

Finally, Chapter 7 concludes the thesis with a discussion of some open problems and future applications of the query-based multicontext theory.

Chapter 2

Preliminaries

Before introducing the results of our research in the next few chapters, we need to introduce basic preliminaries which underlie either the theory we develop or our design decisions. This chapter consists of three sections. The first section discusses the main definitions and issues of knowledge representation and introduces the Semantic Web as one possible domain of application of our results (see Section 2.1.1). Then, we introduce the main logical and rule paradigm necessary for the understanding of our approach (see Section 2.2). We finally present the mathematical theory of Formal Concept Analysis which plays a central role in the view mechanism we developed (see Section 2.3).

2.1 Knowledge Representation and Applications

In this thesis we introduce a new notion of contextualised semantic views capable of representing well-defined parts of a knowledge base. These views build on existing knowledge representation approaches. Before introducing a given knowledge representation paradigm in the next section, we first recall some of the basic definitions as well as some important issues related to the field. The second part of this section presents the vision of the Semantic Web which provides an interesting use case for our work.

2.1.1 Aspects of Knowledge Representation

Many definitions of knowledge, information and data have been proposed in the literature and no real consensus has been reached over this issue due to the heterogeneous requirements of possible applications. John Sowa discusses many of the relevant issues in his book: *Knowledge Representation: Logical, Philosophical and Computational Foundations* (see [Sowa, 2000], pages 51–123). In this thesis, we mainly focus on knowledge bases. In particular, we investigate knowledge bases using standardised ontology languages which are likely to gain importance in the coming years, but a generic definition of the notion of knowledge representation helps highlight the main features of these languages.

Knowledge Representation

The representation of knowledge has played a great role in the development of mankind. Since the hunt drawings on caves' walls, not only the means of storing and displaying knowledge have evolved, but the current computer technology also enables the automation of many tasks requiring human knowledge. Since its inception, the field of Artificial Intelligence has offered many approaches to further enhance this capacity, some focusing on symbolic encodings of knowledge. The rational behind the use of symbols is that symbols can be used to easily manipulate surrogates of entities in an abstract manner. While different approaches have been proposed such as semantic networks, frames, etc, [Davis et al., 1993][1] gives a generic definition to the notion of *knowledge representation* paradigm.

A knowledge representation (hence KR) is:

1. most fundamentally a *surrogate*,
2. encoding a *set of ontological commitments*,
3. to serve as *a medium of human expression*,
4. while being a *fragmentary theory of intelligent reasoning*,
5. and a *medium for pragmatically efficient computation.*

To illustrate this definition, we use a scenario of a travel agency broker. Similar scenarios can be found in the literature (see for example [Tempich et al., 2004]).

Scenario 1 (Travel Agency Web Services) *The company ACME is a tourist agency reseller on the Internet which offers some special travel packages. From the company's web site, a client can select particular standard offers or customise the offers to his wishes. To be able to provide up-to-date services, the company ACME must be able to integrate the information from a number of other companies which provide specific services (typical examples of these service providers may be hotels, car renting companies). Moreover, once the information has been integrated in the knowledge base of the company, offers to specific client may be required to satisfy a number of criteria (for example, the client may require to know the number of beds in a hotel room).*

As already mentioned, a knowledge representation is a surrogate, in other words, it is designed to serve as a substitute for real world entities. In the above scenario, the company manipulates information about the possible services it provides to the users. The diverse manipulation tasks are performed on a representation of knowledge.

When designing a knowledge representation, a number of assumptions about the structure of the world must be made. A model of the domain must be created which encodes these assumptions, and represents the commitments as to how the world is modelled. Depending on the complexity of a domain, the modelling primitives may be required to model the knowledge. For instance, some domain may require only propositional languages, whereas others require more expressive languages such as first-order logic or fuzzy logic, etc.

[1] John Sowa mentions this definition in the book cited above (see pages 134–141 of [Sowa, 2000]).

A knowledge representation should also support human communication, that is, humans should be able to understand and manipulate it. In order to integrate the products from diverse providers, the name and nature of the products should be easily understandable so as to ease their use by non-experts.

One of the purposes of using a knowledge representation is to enable the inference of new facts from previous knowledge or to verify the consistency of the knowledge. A knowledge representation should be adapted to the practical situations where it is to be used.

This last criterion makes explicit that an appropriate trade-off between efficiency and expressivity has to be found. While some applications might require low response time, others might not require that queries terminate. This issue is relevant for our thesis because the choice of the language used influences our approach on two levels. From the querying point of view, we consider the complexity of answering specific questions given a query infrastructure. From the knowledge representation point of view, ontology languages like OWL focus on the modelling of knowledge while ensuring certain computability properties. As we recall in Chapter 2, Section 2.1.2, description logics restrict the modelling primitives of the language to ensure decidability.

Ontologies and Knowledge Bases

The purpose of a knowledge representation as defined in the previous section is to provide a language to describe a domain. From the introduction we recall the definition of an *ontology* given in [van Harmelen and Horrock, 2000].

An ontology is a consensual, shared and formal description of the concepts that are important in a given domain.

An ontology does not consists only of the concepts occurring in a domain, it also includes relations between these concepts. For example, ontologies to represent the knowledge about a field of research usually includes persons, publications and events, specifying that person are authors of publications and that publications are presented at events.

The notion of domain is particularly relevant when considering a knowledge exploration approach because the purpose of some users' tasks stems from specific aspects of the domain or of the application to consider.

Some authors do not make any distinction between an ontology and a *knowledge base*. In this thesis, we make that distinction. We consider that an ontology constitutes a general theory of a domain, whereas a knowledge base describes particular circumstances pertaining to such a theory.

The distinction is however not strict. As noted in [de Bruijn and Fensel, 2005], the literature presents different common layers of knowledge: *top level ontologies*, *domain ontologies* and *application ontologies*. Each level corresponds to a degree of generality of the ontologies. Top level ontologies capture aspects of knowledge which can be seen as independent of the domain. Domain ontologies model valid knowledge for a certain domain, say medicine or the automobile industry. Application ontologies capture specific knowledge for a specific application. The AIFB ontology used throughout this thesis is an example of an application ontology. The ontologies of the first two layers are usually meant as more generic knowledge to be reused in application ontologies, but there is no strict separation between these layers. For example, the AIFB knowledge base has been defined by extending an ontology designed to describe research activities.

In other words, we use knowledge base as a synonym for application ontology. This distinction is related to the distinction between *Closed and Open World Assumptions* which have practical consequences in the algorithms used.

Traditional Knowledge Representation Assumptions

Databases have traditionally been the main mean of storing structured information in computers. In particular, the relational theory developed by Codd in [Codd, 1970] relies on certain assumptions under which most industrial databases (as well as traditional deductive databases approaches) are designed. These assumptions are:

- *Unique Name Assumption* (UNA): there is a unique name for individuals, so two different names refer to two different individuals

- *Closed World Assumption* (CWA): everything that is not explicit in the current knowledge or cannot be inferred from it is considered false.

The *Unique Name Assumption* is particularly used in database theory because values can then be used as keys in the database. For example, two products of a given company should receive different serial numbers. Using this assumption, an application can reasonably infer that the products are different. In a more distributed setting, the uniqueness of a name for an entity is difficult if not impossible to ensure. This topic has been extensively treated in the database literature and can be traced back to Raymond Reiter's article [Reiter, 1980]).

The second assumption stems from the fact that it is sometimes realistic to assume complete knowledge about the world. For example, the list of items of a store can be listed completely. If the item of a given type is not returned by a query, it can be reliably assumed that this item is not available. According to the *Closed World Assumption*, if a fact is not contained in the knowledge base, the negation of this information can be inferred. This means that true and false assertions are not on an equal footing as the truth value "false" is preferred. On the other hand, under the *Open World Assumption*, the truth value of a statement can be either true, false or unknown. The literature abounds on the topic of the possible definitions of the Closed World Assumption (see for example [Grimm and Motik, 2005]).

Taxonomy of Knowledge Entities and Relations

For the purpose of our methodology, we were interested in the kind of entities occurring in a knowledge base. From the diverse paradigms we investigated we could draw a taxonomy of entity types . The purpose of this taxonomy is to guide our design since it allows to abstract from the particular ontology and have a more generic picture of our approach.

As starting point, we first determine the possible knowledge entity types occurring in a knowledge base. From our study of different knowledge representation paradigms, we retained the following type of entities: *individuals*, *categories*, *relation instances*, *relations* and *contexts*. The main criteria we retained from these entities is that it is relevant in some application to study, compare or manipulate the properties of these entities.

We now describe these types of entities and illustrate them using an example. These entities are surrogates and therefore represent some entities, group of entities or some situations. They can be seen as names or identifiers representing intensionally an element or a group of elements in the real world.

- *Individuals* (also called *instances*) represent atomic entities of the real world. In the web service example introduced above, a product item is an individual.

- *Categories* (also called *classes, types, or concepts*) represent groups of individuals assigned to this category according to some criteria. For example, a product series or even the set of all product items is a category.

- *Relation instances*: represent a certain relation between instances. For example, a computer is sold with a certain motherboard. A relation is usually labelled by a predicate describing the specific relation. For example, the predicate used in the previous example is *isSoldWith*.

- *Relations* represent sets of relation instances. For example, the relation hasPart between computers and their building parts.

- *Contexts* represent situations. Typically a knowledge base in itself represents a context containing a number of statements all considered valid. For example, a knowledge base may represent the status of the storage room of a store at different moment. Each of these states represent a number of statements valid in at the given moment. Some applications may then need to refer to these states as the context of interpretation.

All these kinds of entities play a role in this thesis because they are the elements to be visualised and manipulated and can be found in the knowledge representation languages presented in the following section.

2.1.2 Application: Visualising the Semantic Web

One of the goals of this thesis is to present a new approach to knowledge exploration of knowledge bases. In particular, we focus on the visualisation of knowledge bases used in the context of the Semantic Web which we describe in detail in this section.

The Semantic Web Vision

The *Semantic Web* is an initiative pushed by the W3C aiming at creating a new generation of the web where content is described semantically using some standard representation language that both machines and humans may understand and manipulate. We introduce here the Semantic Web by first describing the general architecture, then we present in more details crucial building blocks such as RDF(S), OWL and rule languages. A more complete overview of the Semantic Web can be found in the literature (see [Antoniou and van Harmelen, 2004,

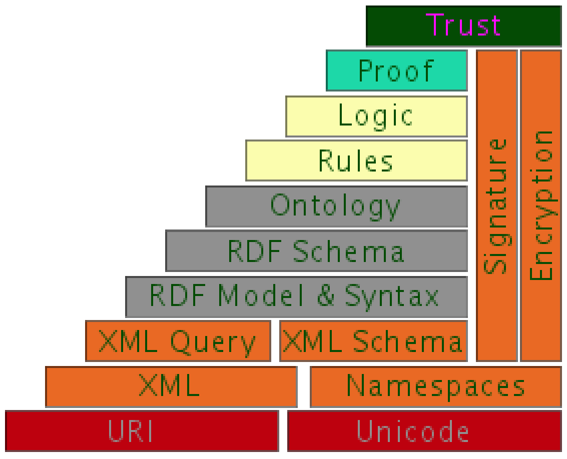


Figure 2.1: The Semantic Web layer cake (source [Horrocks et al., 2005]).

Studer and Staab, 2003, Fensel et al., 2003] for good entry points on the subject).

In the seminal article [Berners Lee et al., 2001], a basic idea of the interdependence of the different layers of the Semantic Web have been introduced as a diagram known as the *Semantic Web layer cake*. but with the evolution of the research field this idea evolved. A diagram illustrating the stratification of the layers is displayed in Figure 2.1. This particular version stems from [Horrocks et al., 2005]. The lower levels of the architecture proposed rely on diverse XML related standards like Unicode, URIs, XQuery or XML Schema for data interchange. On top of these layers, the RDF Model provides a basic graph model to describe data. This model serves as underlying model for the RDF(S) and OWL ontology languages. These languages correspond to the RDF Schema and Ontology layer of the diagram and are used to describe available knowledge.

On top of the ontology layer, the rules layer allows to perform more complex or less standard inferences using the knowledge available from an ontology. In the same way, the proof and logic layer enhance the diverse capacities with extended proof and inference capabilities. Finally, the trust layer offers mechanisms to ensure the validity and origin of the data.

The Resource Description Framework

Designed in order to be compatible with XML technologies, the *Resource Description Framework* (see [Klyne and Carroll (eds), 2004, Hayes (editor), 2004]), henceforth RDF, is an assertional modelling language to express properties of web resources using precise formal vocabularies. Contrary to XML documents which require a tree structure, the model behind RDF is a graph and was designed to simplify the integration of diverse sources of metadata. A RDF graph can be seen as a set of statements also called triples due to their general form

(Subject, Predicate, Object). For example, a statement[2]

(**ontoworldwiki:Julien_Tane**, **foaf:member**, **aifb:WBS**)

indicates that Julien Tane is a member of the group Knowledge Management. **ontoworldwiki:Julien_Tane** is an identifier for the person **Julien Tane**, the URI **foaf:member** is a relation defined in the Friend of a Friend RDF Schema and **aifb:WBS** is an identifier for the research group Knowledge Management.

RDF is designed to provide a basic foundation for more advanced declarative languages. The RDF Schema standard, henceforth RDF(S) provides for example both syntax and semantics for the definition of conceptual schemas and can be used as a simple ontology language (see [Brickley and Guha (eds), 2004]). Using RDF(S), named concepts and properties can be defined as well as the means to express subsumption relations between concepts or between relations.

A simple example RDF graph together with a corresponding RDF schema is displayed in Figure 2.2.

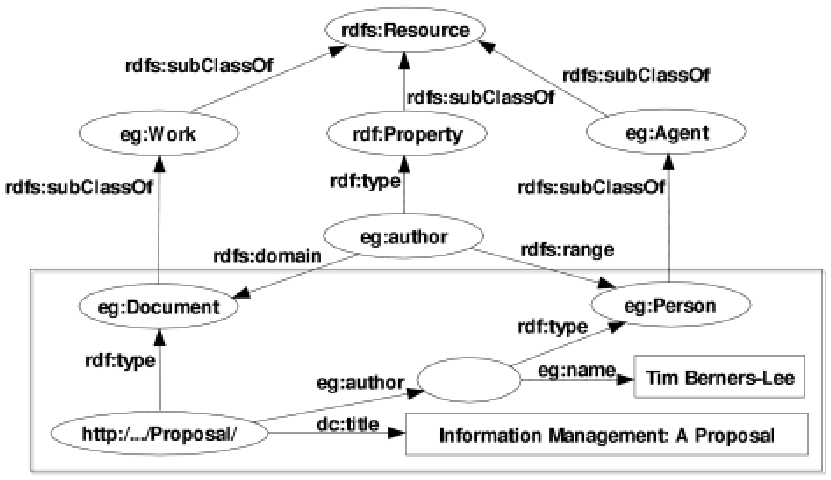


Figure 2.2: An example of RDF graph together with a RDF schema stating that Tim Berners-Lee is the author of a document with the title Information Management: A Proposal.

This simple graph in the box states that a document identified by a URI: http://.../Proposal/ is a document. It specifies that its title is Information Management: A Proposal and that its author is a person named Tim Berners-Lee. At the top of the figure, it is possible to see that eg:author is a property with domain eg:Document and range eg:Person. According to the semantics of RDF(S), using this schema, it is possible to infer automatically that the proposal is also an instance of the Class work and that Tim Berners-Lee is also an agent. If the type of the author of the proposal had not been stated explicitly, an inference engine for RDF(S) could have also inferred that Tim Berners-Lee is a person and therefore an agent.

[2]In this statement, the strings **ontoworldwiki**, **foaf** and **aifb** are abbreviations for URLs.

Supplementary properties of relations can also be specified. For instance, a relation can be specified as being the inverse of another one or that it is functional (i.e. if an indivual has a value for this relation, it is the only one).

Finally, RDF(S) also offers other powerful modelling primitives such as reification and metamodelling. The former allows to make assertions on the statements of an RDF graph. The latter allows to add supplementary properties to redefine some aspects of the schema. In this thesis, we do not address these issues.

Note that RDF and RDF(S) are relevant for our work since the first working implementation of our framework uses KAON ontologies which correspond to RDF(S) ontologies extended with modelling primitives adapted to lexical information as well as some metamodelling primitives (see [E. Bozsak et al., 2002]). We omit further description of these primitives since they are not central for our purpose.

RDF(S) provides a simple ontology language which has already been used in many applications. Yet, some domains and applications like medicine require more complex primitives which are not covered by the RDF(S) standard (see [Rector, 2003, Rector and Rogers, 2006]). To remedy this issue more expressive ontology languages are required and the more expressive language OWL has been developed.

OWL - The Web Ontology Language

The Web Ontology Language (OWL) provides both a syntax and corresponding formal semantics to specify ontologies. It has been released as W3C recommendation in April 2004 (see [MacGuiness and van Harmelen (eds), 2004]) and is designed as an extension of RDF(S). Using these primitives, it is possible to define classes in a constructive manner from other classes. We now give three examples of such primitives, but others are given in Section 2.2.1. Our first example is the possibility to use the set connectives such as intersection, union and negation. The class carnivorous plant can be defined as the intersection of the two classes carnivore and plant. Using the negation primitive, it is also possible to define the class non-carnivorous plants. Unlike in RDF(S), it is also possible to define classes using relations. For example, the class Father can be defined as the individual having a relation hasChild with some other (possibly unknown) individual. Finally, in OWL it is possible to state that a given class consists of only a restricted list of individuals. For example, the 8 G8Members can be listed exhaustively.

As mentioned in Section 2.1.1, different applications and domains may influence the choice of the knowledge representation primitives needed. Depending on the application, there may be a trade-off between efficiency and decidability of the reasoning. This requirement has been considered in the design of the OWL ontology language. The OWL recommendation provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users. These sublanguages are respectively called *OWL Lite*, *OWL DL* and *OWL Full*.

- **OWL Lite**: OWL Lite restricts some of the more complex primitives in order to simplify the ease of implementation.
- **OWL DL**: The features of the fragment were chosen to correspond closely

to a highly expressive but decidable logical language called $\mathcal{SHOIN}$(D). This language belongs to a class of knowledge representation called *description logics* for which the ontology primitives are carefully selected in order to ensure decidability of all possible inferences.

- **OWL Full**: OWL Full allows the unrestricted use of many modelling primitives. However, this expressive power leads to undecidability of the inference tasks in the general case.

Note also that OWL Full is seen as an extension of RDF, but OWL Lite and OWL DL extend only a restricted subset of RDF. However, we abstract from these issues here because we focus in this thesis on OWL DL since it is the language which corresponds the most to our purpose.

Rule Languages

An ontology describes a state of knowledge shared by a community, but many applications require the possibility to define complex inferences on this knowledge which are not specified in the ontology. In [Battle et al., 2005], a number of application scenarios are presented for a Semantic Web Service Language based on a rule language. Among these applications, the authors mention service discovery and policy rules for e-commerce. In the Semantic Web layer cake, these issues are addressed using the rule layer, yet, the actual form of this layer remains unclear because different kinds of rules are required depending on the kind of inference needed.

The preferable place of the rule layer is still controversial. Some authors (for example [Horrocks et al., 2005]) suggest that the rule layer should be defined on top of the OWL language. Others (see for example [de Bruijn et al., 2005] advocate that some of the assumptions used in OWL are not suitable for some rule frameworks. In particular, they argue that the Open World Assumption (see Section 2.1.1) used by OWL is not compatible with the traditional Closed World Assumption found in most rule-based paradigms. Another approach for the combination of rules and ontologies can be found in [Hitzler et al., 2005a].[3] This approach corresponds to determining the intersection between the OWL and the logic programming approaches to modelling. In 2005, the *Rules Interchange Format*[4] working group has been created by W3C in order to increase communication on this topic and develop standards for the exchange of rules over the Internet.

Rules are relevant for our approach because they offer the possibility to implement query languages on top of an ontology framework.

2.2 Logical and Rules Paradigms

In this section, we present two of the main theoretical frameworks necessary for the development of our approach. We first introduce description logics by defining the general syntax and semantics of languages of this family (see Section 2.2.1). Then, we present the syntax and semantics of the traditional deductive

[3]Information can also be found in [Hitzler et al., 2005b].

[4]See `http://www.w3.org/2005/rules`.

language *Datalog* which is closely related to the rule layer of the Semantic Web architecture (see Section 2.2.2).

2.2.1 Description Logics

The extensive literature on knowledge representation proposes many approaches to model knowledge. In this section, we introduce description logics as underlying knowledge representation paradigm. Description logics offer a logical language appropriate for the approach because description logics focus on determining the trade-off between expressivity, efficiency and decidability.

Originally developed to create terminologies, *description logics* (henceforth DLs) are logical formalisms for representing and reasoning about conceptual and terminological knowledge (see [Nardi and Brachman, 2003]). In recent years, their importance has been recognized as central for the research field of knowledge representation and they have been used in a wide range of applications. The motivations for using description logics can be diverse, but traditionally description logics are used for applications requiring a trade-off between using an expressive modelling language or using a language which has nice logical and algorithmic properties. Using a sound and complete language garantees that all inferences using this language are correct and that all correct inferences are be found. From the algorithmic point of view, the inference engines should be able to perform inferences efficiently.

In practice, this means that most description logics correspond to decidable fragments of first order logic (i.e. they restrict the expressivity of the language constructs used so that the truth value[5] of statements of the language can be determined in finite time).

A description logic knowledge base can be divided into two sets of assertions (also called axioms) respectively called *A-box* and *T-box*. The first contains facts about individuals, while the second provides a set of axioms considered true in the given knowledge base. Both build upon a common vocabulary of concept and role expressions. Using a combination from A-box and T-box axioms together with the basic axioms of logic, it is possible to infer new knowledge from the already specified knowledge.

Description logics are given model-theoretic semantics. A *interpretation* is a mapping I which maps the elements of the language to sets and relations of a set called *domain of discourse*. The axioms of a knowledge base put constraints on the acceptable interpretation of a knowledge base. An interpretation which satisfies the axioms of the knowledge base is then called a *model* of the knowledge base.

Concept and Role Expressions

The vocabulary of a description logic language consists of *concept expressions*, which denote sets of individuals and *role expressions*, which traditionally denote binary relations between individuals. Some description logic languages accept n-ary relations (see [Calvanese et al., 1997]) but for the purpose of our exposition we restrict ourselves to the binary case. In order to allow a highly expressive language, description logics can leverage the use of constructors to

[5]Description logics usually assume the open world assumption, the possible truth values are true, false and unknown.

build more complex concept and role expressions from simpler ones. The most simple expressions consist of the set $\mathcal{N}_C$ of *atomic concept names* and the set $\mathcal{N}_R$ of *atomic role names*[6] and provide the basic vocabulary. For instance, *PhDStudent* is an atomic concept denoting the set of Ph.D students. Using the constructors given in Tables 2.1 and 2.2, it is possible to build more complex concepts and roles. For example, $\mathit{PhDStudent} \sqcap \mathit{Father}$ is a complex concept expression representing the concept of PhD Students who are also fathers. These notions are formalised in the following definition.

Definition 1 (Concept and Role Expressions) *Let $\mathcal{N}_I$ be the set of individual names and let $\mathcal{N}_C$ and $\mathcal{N}_R$ be the sets of atomic concept names and atomic role names, respectively.*

A role expression is either

- *an atomic role name, i.e. an element of $\mathcal{N}_R$, or*
- *an expression recursively constructed from simpler role and concept expressions using the role constructors found in Table 2.2.*

A concept expression is either

- $\top$*,*
- $\bot$*,*
- *an atomic concept name, i.e. an element of $\mathcal{N}_C$, or*
- *an expression recursively constructed from simpler role and concept expressions using the concept constructors found in Table 2.1.*

An interpretation $I = (\Delta^I, .^I)$ consists of a non empty domain Δ^I and an interpretation function $.^I$ called valuation *which maps individual names to elements of Δ^I, atomic concept names to sets $A^I \subseteq \Delta^I$, and atomic roles names to relations $P^I \subseteq \Delta^I \times \Delta^I$. Tables 2.2 and 2.1 specify the semantics of concepts and roles.*

The concept and role constructor correspond to typical modelling operations. The operators $\sqcup$, $\sqcap$ and $\neg$ correspond to the typical set operations.

Using these concept and role expressions in an unrestricted manner leads to intractability or undecidability. Therefore, many description logics only allow certain combinations of constructors. For instance, $\mathcal{ALC}$ concept expressions may not contain qualified number restrictions and nominals, thus ensuring the tractability of most of the relevant tasks.

The A-box

The *A-box* (for assertional box) contains statements which either specify the membership of an individual a to a concept C, or specify the existence of a relationship between individuals a and b. The syntax of these axioms are specified in the following definition.

[6]Per convention, atomic concept names are capitalised as opposed to atomic role names, which are not.

Table 2.1: Concept constructors

Name	Syntax	Semantics
Top	$\top$	Δ^I
Bottom	$\bot$	$\emptyset$
Negation	$\neg C$	$\Delta^I \setminus C^I$
Union	$C \sqcup D$	$C^I \cup D^I$
Intersection	$C \sqcap D$	$C^I \cap D^I$
Value restriction	$\forall R.C$	$\{x : \forall y : (x, y) \in R^I \rightarrow y \in C^I\}$
Existential restriction	$\exists R.C$	$\{x : \exists y : (x, y) \in R^I \wedge y \in C^I\}$
Qualified at-most restriction	$\leq n\ R.C$	$\{x : \sharp\{y \vert (x, y) \in R^I \wedge y \in C^I\} \leq n\}$
Qualified at-least restriction	$\geq n\ R.C$	$\{x : \sharp\{y \vert (x, y) \in R^I \wedge y \in C^I\} \geq n\}$
Nominals	$\{a, b, c...\}$	$\{a^I, b^I, c^I...\}$

Note: $\sharp S$ is the number of elements in the set S.
a, b, ... are elements of $\mathcal{N}_I$
C and D are concept expressions
R is a role expression

Table 2.2: Role constructors

Name	Syntax	Semantics
Intersection	$R \sqcap S$	$R^I \cap S^I$
Union	$R \sqcup S$	$R^I \cup S^I$
Complement	$\neg R$	$(\Delta^I \times \Delta^I) \setminus R^I$
Inverse	R^-	$\{(b, a) \in \Delta^I \times \Delta^I \vert (a, b) \in R^I\}$
Composition	$R \circ S$	$R^I \circ S^I$
Transitive Closure	R^+	$\bigcup_{n \geq 1} (R^I)^n$
Reflexive-transitive closure	R^*	$\bigcup_{n \geq 0} (R^I)^n$
Role restriction	$R_{\vert C}$	$R^I \cap (\Delta^I \times C^I)$
Identity	id(C)	$\{(d, d) \vert d \in C^I\}$

Note: a, b, ... are elements of $\mathcal{N}_I$
C is a concept expression
R and S are role expressions

Definition 2 (A-box Axioms) *An A-box axiom is an expression of one of the following form:*

$$\begin{array}{rl} C(a) & R(a, b) \\ \neg S(a, b) & \\ a \approx b & a \not\approx b \end{array}$$

where C is a concept expression, R and S are role expressions and a and b are individual names.

The semantics of these axioms are given by the mapping given in the Table 2.3.

A *concept assertion* is a statement $C(a)$ for a concept expression C and an individual a, while a *role assertion* is a statement of the form: $R(a, b)$ for R a role expression and a and b individuals. The axioms $\neg S(a, b)$ specifies that the a does not have any relation S with b. The axioms $a \approx b$ and $a \not\approx b$ state that the individual names a and b represent the same entity, respectively do not represent the same entity.

Table 2.3: The semantics of the description logic A-box axioms.

Name	Axiom	Semantics of Axioms
concept instantiation	$C(a)$	$a \in C^I$
role instantiation	$R(a, b)$	$(a, b) \in R^I$
role refutation	$\neg R(a, b)$	$(a, b) \notin R^I$
same individual	$a \approx b$	$a^I = b^I$
different individual	$a \not\approx b$	$a^I \neq b^I$

To introduce description logics in a more intuitive manner, we illustrate these definitions using an example knowledge base presented in Figure 2.3. The A-box contains the following statements:

$$\begin{aligned}
\{&\mathsf{Professor(Rudi\ Studer),\ PhDStudent(Julien\ Tane)},\\
&\mathsf{publication(Rudi\ Studer,\ Ontologies...IEEE\ 2003)},\\
&\mathsf{publication(Julien\ Tane,\ Query\text{-}Based\ Multicontext...)},\\
&\mathsf{InProceedings(Query\text{-}Based\ Multicontext...)},\\
&\mathsf{InProceedings(Ontologies...IEEE\ 2003)},\\
&\mathsf{isWorkedOnBy(Formal\ Concept\ Analysis,\ Julien\ Tane)},\\
&\mathsf{isSubTopicOf(Machine\ Learning,\ Computer\ Science)},\\
&\mathsf{isSubTopicOf(ILP,\ Machine\ Learning)}\}
\end{aligned} \tag{2.1}$$

This A-box contains assertions of the sort: PhDStudent(Julien Tane) to denote that Julien Tane is a PhD student or isWorkedOnBy(Formal Concept Analysis, Julien Tane) to denote that the topic Formal Concept Analysis is worked on by Julien Tane.

The T-box

The *T-box* (for terminological box) contains axioms which enable an inference engine to draw new facts from the stated knowledge. There are mainly two kinds of axioms.

The first kind establishes relationships between concepts or between roles. For concept expressions C and D and roles R and S, axioms of the form: $C \sqsubseteq D$, $R \sqsubseteq S$ or $C \equiv D$ state relationships between two concepts or two roles. For example, the axiom PhDStudent $\sqsubseteq$ Person, states that all PhD students are also persons. The second type of axiom states some important property of roles, for example, Trans(isSubTopicOf) states that the role isSubTopic is transitive. Using this axiom with the A-box axioms: isSubTopicOf(Machine Learning, Computer Science) and isSubTopicOf(ILP, Machine Learning) an inference engine supporting transitivity axioms would infer: isSubTopicOf(ILP, Computer Science).

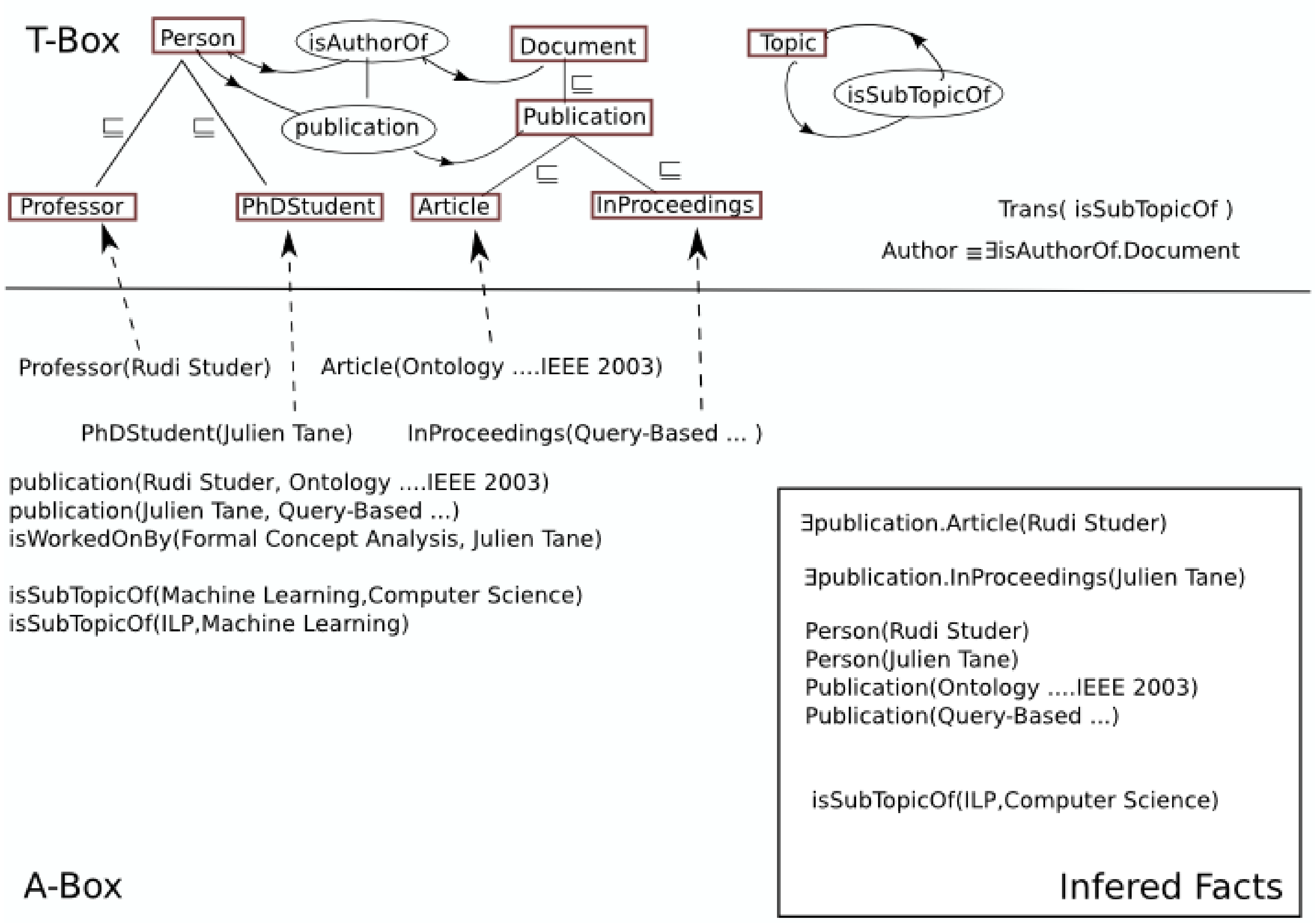

T-Box
Person
isAuthorOf
Document
Topic
isSubTopicOf
publication
Publication
⊑
Professor
PhDStudent
Article
InProceedings
Trans(isSubTopicOf)
Author ≡∃isAuthorOf.Document
Professor(Rudi Studer)
Article(Ontology ....IEEE 2003)
PhDStudent(Julien Tane)
InProceedings(Query-Based ...)
publication(Rudi Studer, Ontology ....IEEE 2003)
publication(Julien Tane, Query-Based ...)
isWorkedOnBy(Formal Concept Analysis, Julien Tane)
isSubTopicOf(Machine Learning,Computer Science)
isSubTopicOf(ILP,Machine Learning)
A-Box
∃publication.Article(Rudi Studer)
∃publication.InProceedings(Julien Tane)
Person(Rudi Studer)
Person(Julien Tane)
Publication(Ontology ....IEEE 2003)
Publication(Query-Based ...)
isSubTopicOf(ILP,Computer Science)
Infered Facts

Figure 2.3: A small example of a description logic knowledge base.

Definition 3 (T-box Axioms) *A T-box axiom is an expression of one of the following forms:*

$$\begin{array}{rl} C \sqsubseteq D & C \equiv D \\ R \sqsubseteq S & R \equiv S \\ \mathsf{Trans}(S) & \mathsf{Ref}(R) \\ \mathsf{Sym(R)} & \mathsf{Irr}(R) \end{array}$$

where C and D are concept expressions and R and S role expressions.

The interpretation of these axioms is given in Table 2.4.

In Figure 2.3, a few named concept are given in the T-box at the top. They are represented in boxes, wheras role names have been encircled. A line between two named concepts with the $\sqsubseteq$ symbol can be interpreted as a T-box axiom: $C \sqsubseteq D$, for example in the figure: PhDStudent $\sqsubseteq$ Person. The same thing can be said of publication $\sqsubseteq$ isAuthorOf.

Using T-box axioms, it is also possible to specify the domain and range of a role. For example, the role isAuthorOf has Person as domain. This means that an A-box statement isAuthorOf(Peter,Letter) implies that Peter is a Person. The T-box axiom Author $\equiv$ $\exists$isAuthorOf.Document means that all individuals of Author have written at least a document, but also that all person having written a document are authors, so Peter is not only a Person but also an Author.

Table 2.4: The semantics of the description logic T-box axioms.

Name	Axiom	Semantics of Axioms
concept subsumption	$C \sqsubseteq D$	$C^I \subseteq D^I$
subrole subsumption	$R \sqsubseteq S$	$R^I \subseteq S^I$
concept equivalence	$C \equiv D$	$C^I = D^I$
role equivalence	$R \equiv S$	$R^I = S^I$
transitivity	Trans(R)	$(R^I)^+ \subseteq R^I$
reflexivity	Ref(R)	$Diag^I \subseteq R^I$
irreflexivity	Irr(R)	$Diag^I \cap R^I = \emptyset$
symmetry	Sym(R)	$(R^-)^I = R^I$

It is well known that an unrestricted use of concept and role expressions in T-box axioms leads to undecidability. To address these issues, some restrictions can be set either on the production rules for concept expressions and role expressions or on the use of certain kinds of roles in concept expressions. For example, the use of transitive roles with qualified number expressions leads to undecidability (see [Horrocks and Sattler, 2004]). For this reason, the ontology language OWL, which we present below disallows the use of roles subsuming transitive roles.

Using the previous definition, we can now give a definition of knowledge base:

Definition 4 (Knowledge Base) *A knowledge base KB is defined as a tuple KB=($KB_{\mathcal{T}}$,$KB_{\mathcal{A}}$), where $KB_{\mathcal{T}}$ is a set of T-box axioms and $KB_{\mathcal{A}}$ is a set of A-box axioms.*

An interpretation $I = (\Delta^I, .^I)$ consists of a non empty domain Δ^I and an interpretation function $.^I$ which maps individuals to elements of Δ^I, atomic concepts A to sets $A^I \subseteq \Delta^I$, and atomic roles P to relations $P^I \subseteq \Delta^I \times \Delta^I$.

An interpretation I is a model of KB if it satisfies the axioms from Tables 2.3 and 2.4 when concept and role expressions are interpreted using the semantics of Tables 2.1 and 2.2.

One of the purposes of a knowledge base is to store facts which can be used to infer new knowledge. In description logics systems, the means of acquiring this new knowledge is often characterised by the type of *inference task* necessary.

Definition 5 (Inference tasks) *For a knowledge base KB, we define the following inference tasks.*

- satisfiability*: The knowledge base KB is* satisfiable *if and only if there is a model I for KB.*
- concept satisfiability*: For a concept expression C, a concept is* satisfiable *if and only if there is a model I for KB where C^I is not empty.*
- subsumption*: For two concept expressions C and D, a concept C is subsumed by a concept D, written $KB \models C \sqsubseteq D$ if and only if $C^I \subseteq D^I$ is true in all models of KB.*
- concept equivalence*: Two concepts C and D are* equivalent*, if $KB \models C \sqsubseteq D$ and $KB \models D \sqsubseteq C$.*
- instance checking*: An individual a is an* instance *of a concept C if and only if $a^I \in C^I$.*
- role checking*: A role R relates individuals a and b with respect to KB if and only if $(a^I, b^I) \in R^I$.*

It is worthwhile to note that all these tasks can be reduced to the satisfiability task. Using such a reduction enables to obtain worst-case complexity results for the other inference tasks.

Description Logics Examples

The choice of a description logic flavor is generally guided by the necessities of the domain to be modelled. For example, the AIFB portal ontology uses a description logic which lack constructs such as nominals or qualified number expressions, whereas the medicine domain generally requires more expressive constructs such as role composition.

In the next Chapter of this thesis we introduce a number of operators which are closely related to some of the description logic constructors presented in this section. In order to appreciate the relation between these operators and the one presented here, we consider first the $\mathcal{ALC}$ description logic. This description logic can be described by presenting the production rules allowing the construction of concepts.

$\mathcal{ALC}$Concept Expression $\rightarrow A|C \sqcup D|C \sqcap D|\neg C|\exists R.C|\forall R.C$

This means that concept can be constructed in $\mathcal{ALC}$ using boolean operator on concepts (i.e. $\sqcup$, $\sqcap$, $\neg$) and the existential restriction ensuring that the members of a concept $\exists R.C$ have a relation R with an individual instance of C, whereas the concept $\forall R.C$ is the concept consisting of all the instances which have the property that if they possess a relation R then the elements they link to through R are of type C. While the boolean operators and the existential restriction are useful and very frequent for writing queries. The value restriction is more useful to draw supplementary inferences than for querying purposes.

A number of constructs can be added to $\mathcal{ALC}$ to obtain an appropriate language. The next important primitive are nominals and are usually denoted using a $\mathcal{O}$ in the logic name. Nominals allow the definition of extensional sets of elements. For example, the professors of the AIFB institute are all known. Using nominals it is possible to define the concept:

$$\begin{aligned}\mathsf{AIFBProfessors} \equiv \{&\mathsf{Prof.\ Oberweis, Prof.\ Schmeck, Prof.\ Seese,} \\ &\mathsf{Prof.\ Studer, Prof.\ Stucky}\}\end{aligned}$$

For example, a hierarchy on roles can be added. This is usually denoted by the use of $\mathcal{H}$ in the name. Just as for concept expressions, boolean operators can be used on role constructors.

The definition given above corresponds to descriptions logics with binary predicates, but a pertinent aspect to consider in our approach is the use of more expressive logics, in order to capture the content of databases. Certain description logics have been specially designed to adapt easily to relational databases. Notably, Calvanese et al. (see [Calvanese et al., 1997]) introduced such a logic, the $\mathcal{DLR}$ description logic, which allows relations of arity greater than two as well as other crucial operators relevant for the database context. Our original approach in [Tane, 2005] introduced a query language very similar to this description logic. Other approaches have also been suggested, for example [Grädel, 1999] introduced a description logic based on the Guarded Fragment of first order logic. However, we postpone further discussions to the future work section (see Chapter 7) of this thesis.

The literature on description logics is extensive, covering a wide range of issues from theoretical results to applications. We refer the interested reader to the *Description Logic Handbook*, (see [Baader et al., 2003]) which covers most of the state-of-the-art in the domain of description logics, in particular the first chapter (see [Nardi and Brachman, 2003]) gives a good introduction to the topic.

With the advances of research in the domain of description logics, a cooperative effort from the research community is paving the way for the next version of OWL DL. The purpose of this initiative is to extend the actual OWL DL (which corresponds to the $\mathcal{SHOIN}(D)$ description logic) with features which can be easily implemented in state-of-the-art inference engines. This new description logic would correspond to the $\mathcal{SROIQ}$(D) description logic. It corresponds to the description logic $\mathcal{ALC}$ together with the transivive and reflexive axioms, and enhanced with the inverse role constructor $.^-$ (the $\mathcal{I}$ in the language name indicates its presence), as well as qualified number expressions (indicated by $\mathcal{Q}$),[7] “nominals” concept constructor and concrete domains (indicated using the D in between the parentheses). We omited concrete domains here. The decidability and NExp-Time-completeness of this logic has been proved in [Horrocks et al., 2006].

[7]There is a technical limitation on the use of complex roles in qualified number expression in order to prevent loosing the decidability of the logic.

2.2.2 Datalog

The interest in using logic in databases gave rise to the field of deductive databases. For an overview of the issues in deductive databases, we greatly recommend the so-called *Alice*-book (see [Abiteboul et al., 1995]) as well as [Eiter et al., 1997].

In this thesis, we study the consequences of using different flavors of query languages to see how well they could be integrated in our approach. For this, we first consider their syntax.

Syntax

We now introduce the syntax of the diverse datalog flavors. We give a general definition of the datalog syntax but other definitions can be found in the literature (see in particular [Eiter et al., 1997] though we were mainly inspired from [Motik, 2006], as well as [Abiteboul et al., 1995]).

Definition 6 (Syntax) *A* first order signature Σ *is a 4-tuple* $\Sigma := (\mathcal{P}, \sigma, \mathcal{F}, \mathcal{V})$*, where* $\mathcal{P}$ *is a set of* predicate names *together with a mapping* σ *from* $\mathcal{P}$ *into the set of non-negative integers denoting the arity of a predicate.* $\mathcal{F}$ *is a finite set of* constants *and finally* $\mathcal{V}$ *is a countable set of* variables.

We call a term[8] *over* $\mathcal{V} \cup \mathcal{F}$ *a tuple over* $\mathcal{V} \cup \mathcal{F}$*. The mapping* length *maps a term* t *to a non-negative integer* n*, so that* n *is the arity of* t*. We call* ground term *a term where only constants appear.*

An atom *is an expression of the form* $p(\mathbf{t})$ *where* p *is a predicate name (i.e.* $p \in \mathcal{P}$*) and* $\mathbf{t}$ *is a term. A* ground atom $p(\mathbf{t})$ *is an atom where* $\mathbf{t}$ *is a ground term. The* negation of an atom *is an expression of the form* $\sim A$ *where* A *is an atom. A* literal *is an atom or a negated atom.*

A datalog rule *is an expression of the form*

$$A_1 \vee \cdots \vee A_n \leftarrow l_1, \ldots, l_m,$$

where $n \geq 0$ *and* $m \geq 0$ *and* $A_1, \ldots, A_n$ *is a sequence of atoms called the* head of the rule *and* $l_1, \ldots l_m$ *is a sequence of positive or negative literals (i.e. an atom or the negation of an atom) called the* body *of the rule. The set of atoms of the head is denoted* $head(r)$*, whereas the set of positive atoms (respectively negative atoms) of the body is written* $body^+(r)$ *(respectively* $body^+(r)$*).*

Moreover a rule is safe *if each variable occurring in the head also occurs in the body.*

A datalog program *is a finite set of safe datalog rules.*

A datalog program P *is called* stratified *if there is a decomposition of the set of predicates of* P *into disjoint sets* P_0*,* P_1*, ...,* P_r *called* strata *so that for every rule* r*, there is a stratum* P_j*, so that:*

- *all predicates of the head belong to* P_j
- *for all positive predicates of the body of* r*, there is a stratum* P_i*, such that* $i \leq j$*.*

[8]This corresponds to terms as used in first order logic but in datalog only 0-ary function symbols are allowed.

- *for all negative predicates of the body of r, there is a stratum P_i, such that $i < j$.*

In addition, it is useful to denominate the type of rules. We call a rule positive *if all $l_1, \ldots l_m$ are atoms, we call it* disjunctive *if the head contains more than one atom. Finally, a non-disjunctive positive rule is called a* Horn-rule*. A* positive *(respectively,* Horn-*) datalog program is a program consisting only of positive (respectively, Horn-rules). A disjunctive program is a program containing at least one disjunctive rule.*

Semantics

Depending on the flavor of datalog used, different semantics can be given to a datalog program: *minimal*, *stable* and *perfect* semantics (see [Eiter et al., 1997]). The properties of these semantics are well known. In particular, the perfect and stable model semantics are equivalent for the disjunction-free stratified datalog. We focus here on the latter semantics for which we give the corresponding definition.

Definition 7 (Semantics) *The* Herbrand universe *HU_P of a datalog program P is the set of constants occurring in P. The* ground instance *of P over the Herbrand Universe HU_P, written $ground(P, HU_P)$ is the set of ground rules obtained by replacing all variables of each rules of P with constants from HU_P in all possible ways while respecting variable bindings.*

For a program P, the Herbrand base HB_P of P is the set of all ground atoms defined over predicates from $\mathcal{P}$ using only constants from HU_P. An interpretation *M of P is a subset of HB_P. An interpretation M of P* satisfies *a ground rule r: $A_1 \vee \cdots \vee A_n \leftarrow l_1, \ldots, l_m$, if Ml_i for every literal l_i implies MA_j for at least one A_j. A model M is a model for a program P if M satisfies all the ground rules of the ground instance $ground(P, HU_P)$ of P. Finally, a model is* minimal *if no subset of M is a model of P. The semantics of P is defined as the sets of all minimal models of P, denoted $\mathcal{MM}(P)$.*

The Gelfond-Lifschitz transform *of P^M with respect to an interpretation M is a set of ground rules obtained by removing from $ground(P, HU_P)$ all rules containing a negative literal in the body and removing from the remaining rules all negative literals.*

A model M is said to be stable if $M \in \mathcal{MM}(P^M)$. The semantics of a disjunction-free stratified datalog program P are given by the unique stable model of P.

Three supplementary theoretical results play also an important role when the chosen datalog is restricted to stratified datalog. First of all the semantics of stratified datalog programs are independent of the stratification chosen. The second interesting result is that the traditional stable and perfect semantics match in the stratified case. Finally, the model of a non-disjunctive datalog program is unique (see [Eiter et al., 1997]).

Example

We illustrate the above definitions using an example.

Example 1 (Datalog Examples) *Consider a directed graph with labelled arcs modelled as a database predicate graph.*

The following datalog program returns the sets of pairs for which there is a path in the graph from the first to second argument.

$\mathsf{path}(x, y) \leftarrow \mathsf{graph}(x, y, a).$
$\mathsf{path}(x, y) \leftarrow \mathsf{graph}(x, y, a), \mathsf{path}(y, z).$

The Herbrand Universe in this case are the nodes and labels of the graph, that is the constants occurring in the database graph. The minimal model of this program corresponds to the transitive closure of the graph.

For some practical application, it is sometimes desirable to avoid some labels. For example, suppose the graph models the network of a city. Some streets may be under constructions and therefore should be avoided when looking for a path. We introduce a new predicate street_under_construction. The following program lists all the paths between to endpoints which avoid streets under construction.

$\mathsf{path}(x, y) \leftarrow \mathsf{graph}(x, y, a), \sim \mathsf{street\_under\_construction}(a).$
$\mathsf{path}(x, z) \leftarrow \mathsf{path}(x, y), \mathsf{graph}(y, z, a), \sim \mathsf{street\_under\_construction}(a).$

A stratification of this program could be for example
$P_0 = \{street\_under\_construction\}$, $P_1 = \{path, graph\}$.

Note that queries written in the stratified flavor of datalog can be evaluated in polynomial time. Diverse implementations of these flavors exist; in particular, the Horn fragment. Disjunctive datalog has been implemented in several systems such as dlv[9] (see [Leone et al., 2002]).

Note that the unique name assumption and the closed world assumption underlie the datalog paradigm.

2.3 Formal Concept Analysis

In Chapter 1, Section 1.1.2 we gave an informal introduction to the field of Formal Concept Analysis. We now present the necessary background to understand the work in this thesis.

2.3.1 Basic FCA

We start by recalling the basic definitions of Formal Concept Analysis and illustrate them using simple examples. Unless stated otherwise, all the definitions and theorems presented here are taken from [Ganter and Wille, 1999]. We already explained the intuituin behind *formal contexts* and the following definition formalises this notion.

Definition 8 (Formal Context) *A formal context $\mathbb{K}$ is a triple (G, M, I), where G and M are sets and I is a binary relation between G and M, i.e. $I \subseteq G \times M$. Per convention, we call the set G the object set, while M is called the attribute set. The relation I is called the incidence relation.*

Two aspects should be noted about this definition. First, the definition is symmetrical. The two sets G and M can be interchanged, as long as the

[9]See http://www.dbai.tuwien.ac.at/proj/dlv/.

inverse relation I^{-1} is used instead of I (in other words, the arguments of I are swapped). The second important aspect is that a formal context can be seen as a binary relation between two sets. Observe that not every element of G or M needs to appear in the set of pairs of the relation. This means that the relation I is not enough to describe a formal context. This fact is illustrated in the following example:

Table 2.5: A simple example of a formal context

$\mathbb{K}_p$	Italian	Spanish	French	Indian	Thailand	Chinese	Japanese	Vegetarian	Fast food
Paul	×				×	×	×	×	
Maria	×	×			×		×		
Greg	×		×				×		
Stefanie	×			×	×		×		
Julien		×	×		×	×	×		
Karin	×	×		×			×	×	
Mario									

Example 2 (Person–Food Context) *A couple wants to organise a dinner with a group of friends. In order to satisfy their guests, they draw a table of the food appreciated by these friends. Table 2.5 shows this table as a simple formal context. The objects are persons who they plan to invite, while attributes are the type of food they like to eat.*

If the intersection of G and M is empty (i.e. $G \cap M = \emptyset$), then a formal context could be seen as a bipartite graph (a graph with two distinct sets of nodes[10] with a relation between the two) and could be displayed as a graph.

We need to introduce two derivation operators.

Definition 9 (Derivation) *For $A \subseteq G$ and $B \subseteq M$, we introduce a new notation: "$'$" on subsets of G and M:*
$A' := \{m \in M \; : \; \forall g \in A, (g, m) \in I\}$, *and*
$B' := \{g \in G \; : \; \forall m \in B, (g, m) \in I\}$

The meaning of these notations can be interpreted as finding all the elements of the other set which are shared by a given subset of elements. In the example given above, {Maria, Paul, Karin}$'$ = {Italian, Japanese} and {Italian, Thai}$'$ = {Maria, Paul, Stefanie}. Note that these operators can be composed. For example, let A be {Maria, Paul, Karin}, A'' = {Maria, Paul, Karin, Greg, Stefanie, Paul}. If neither Julien nor Mario are invited every one should be satisfied.

[10]Of course in this case, the two sets of nodes are the object and attribute sets and the set of edges consists of the pairs in I.

Definition 10 (Formal concept) *A formal concept of a formal context (G, M, I) is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. A is called the* extension *of the formal concept, while B is called its* intension.

Moreover, we denote by $\underline{\mathfrak{B}}(G, M, I)$ the set of all the formal concepts of (G, M, I).

We define on the set $\underline{\mathfrak{B}}(G, M, I)$ a partial order[11] $\leq$*, such that for two formal concepts (A_1, B_1) and (A_2, B_2) of the context (G, M, I):*

$(A_1, B_1) \leq (A_2, B_2) :\Leftrightarrow A_1 \subseteq A_2$ *(or equivalent $B_2 \subseteq B_1$).*

Actually, the partial order we have just defined on $\underline{\mathfrak{B}}(G, M, I)$ has particular mathematical properties which makes its structure particularly suitable for information visualisation.

Definition 11 (Complete Lattice) *In a partially ordered set $(L, \leq)$, a* lower bound *for a subset A of L is an element $x \in L$, such that $\forall a \in A$, $x \leq a$, whereas an* upper bound *for a subset A of L is an element $x \in L$, such that $\forall a \in A$, $x \geq a$. A* complete lattice *$(L, \leq)$ is a partially ordered set with the property that for any subset A of elements of the lattice , there exists two elements of L, respectively denoted by $\wedge A$ and $\vee A$, such that*

- $\bigwedge A$ *is the largest lower bound.*
- $\bigvee A$ *is the smallest upper bound.*

  A concept lattice is called supremum dense *if every element a of L there is a subset A of L, so that a is the upper-bound of the elements of A, in other words $a = \bigvee A$. A concept lattice is called* infinimum dense *if every element a of L there is a subset A of L, so that so that a is the lower-bound of the elements of A, $a = \bigwedge A$.*

The element of $\vee A$ is also called the[12] *least upper-bound of A, while the element of $\wedge A$ is called the greatest lower-bound of A.*

From [Ganter and Wille, 1999], we now state the *Fundamental Theorem of Formal Concept Analysis*:

Theorem 1 (Fundamental Theorem of Formal Concept Analysis) *The partially ordered set $\underline{\mathfrak{B}}(G, M, I)$ is a complete lattice, henceforth called the concept lattice of (G, M, I). For a family $((A_t, B_t))_{t \in T}$ of formal concepts of the concept lattice $\underline{\mathfrak{B}}(G, M, I)$, the smallest upper bound and greatest lower bound are of the following form:*

$\bigwedge_{t \in T}(A_t, B_t) = (\bigcap_{t \in T} A_t, (\bigcup_{t \in T} B_t)'')$

$\bigvee_{t \in T}(A_t, B_t) = ((\bigcup_{t \in T} A_t)'', \bigcap_{t \in T} B_t)$

Conversely, a complete lattice $\boldsymbol{V}$ is isomorphic to $\underline{\mathfrak{B}}(G, M, I)$, if and only if there exists two mappings $\gamma \colon G \to V$ and $\mu \colon M \to V$ such that $\gamma(G)$ is ***supremum dense*** *in* $\boldsymbol{V}$*, and $\mu(M)$ is* ***infinimum dense*** *in* $\boldsymbol{V}$ *and $(g, m) \in I$ is equivalent to $\gamma(g) \leq \mu(m)$ for all $g \in G$ and all $m \in M$. In particular, $\boldsymbol{V} \cong \underline{\mathfrak{B}}(V, V, \leq)$.*

[11] A *partial order relation* is a reflexive, antisymmetric and transitive binary relation on a set.

[12] The antisymmetry of the $\leq$ relation implies that if a least upper bound exists, it is unique.

The Fundamental Theorem first states that $\underline{\mathfrak{B}}(G, M, I)$ is a complete lattice. But, in order to discuss the properties of concept lattices, we need a way to visualise them. In Order Theory, partial orders are usually displayed using *line diagrams*. A line diagram is a diagram consisting of nodes representing the elements of the ordered sets and edges representing the neighbouring[13] $\preceq$ relation of two elements for the order relation $\leq$.

In Figure 2.4, the line diagram of the concept lattice of the formal context given in Example 2 is shown. This diagram visualises the persons who share the same taste for food. For instance, it is possible to see that Japanese food is eaten by everyone but Mario and that Italian, Thailandese and Spanish food cover most of the people always leaving two persons out. One of these two persons is Mario, who is either not satisfied by any of the food proposed or whose taste is not known.

Lattices have the interesting property that for any set of formal concepts, it is possible to find the greatest lower-bound and the lowest upper-bound.

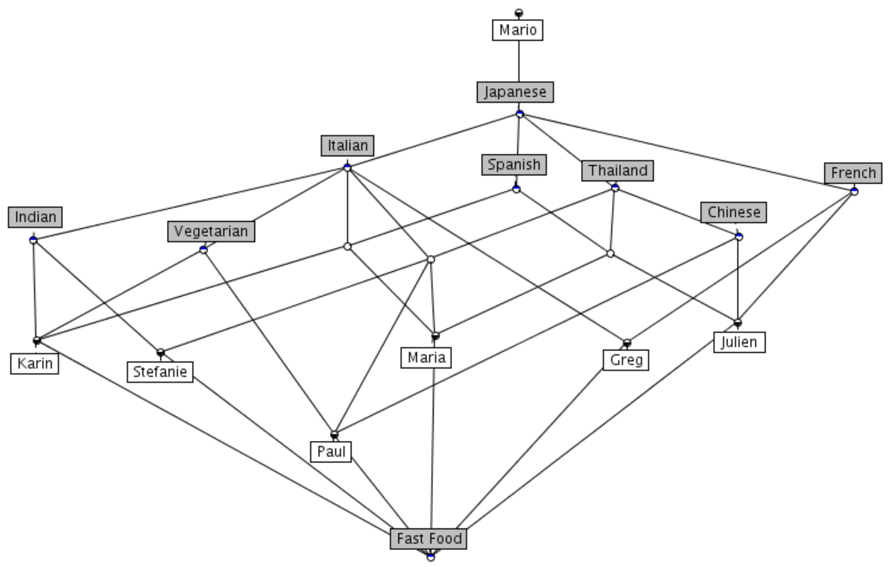


Figure 2.4: The concept lattice of the food and people context of Example 2.

2.3.2 Context Operators

In this section, we introduce basic operators on formal contexts. Most of the definitions we use here come from [Ganter and Wille, 1999]. The purpose of context operators is to be able to combine or transform the formal contexts to obtain new formal contexts. Some of these operators can be understood as conceptual operations on the information contained in these contexts as we explain after having introduced them.

In order to introduce certain operators, we need a supplementary notation to ensure that unions are disjoint:

[13]For a partial order relation on S, the neighbouring (if it exists) consists of all the pairs (a,b), such that $\forall z \in S$, if $a < z$ and b comparable with z, then $b \leq z$.

Definition 12 (Disjoint Union Identifiers) *Let T and T^- be two disjoint sets. Let t be an element of $T \cup T^-$ and let A be a set. We define the* t-marked set $\overset{t}{A}$ *in the following way:*

$$\overset{t}{A} := \{(t, a) : a \in A\}$$

Let $t \in T \cup T^-$ and let I be a binary relation over sets A and B, we define the following sets:

$$\overset{t.}{I} := \{(a, (t, b)) : (a, b) \in I\}$$

$$\overset{.t}{I} := \{((t, a), b) : (a, b) \in I\}$$

We call them the right t-marked relation I *and* left t-marked relation I *respectively.*

Note that these notations used in [Ganter and Wille, 1999] differ a little from the one we introduced. Instead of using a ".", we use an element of $T \cup T^-$ to distinguish sets of elements. Moreover, the notation simplifies part of the proof given in Chapter 3, Section 3.3.3.

The following context operator definitions correspond to the one given in [Ganter and Wille, 1999].

Definition 13 (Context Operators) *Let $\mathbb{K} : (G, M, I)$ be a formal context, then the following unary operators are defined:*

- *dual:* $\mathbb{K}^d := (M, G, I^{-1})$.
- *complement:* $\mathbb{K}^c := (G, M, (G \times M) \setminus I)$

Let $\mathbb{K}_1 : (G_1, M_1, I_1)$ and $\mathbb{K}_2 := (G_2, M_2, I_2)$ be two formal contexts, then the following operators are defined:

- *if $G_1 = G_2$, the apposition:* $\mathbb{K}_1|\mathbb{K}_2 := (G_1, \overset{t}{M_1} \cup \overset{t^-}{M_2}, \overset{t.}{I_1} \cup \overset{t^-.}{I_2})$
- *if $M_1 = M_2$, the subposition:* $\frac{\mathbb{K}_1}{\mathbb{K}_2} := (\overset{t}{G_1} \cup \overset{t^-}{G_2}, M_1, \overset{.t}{I_1} \cup \overset{.t^-}{I_2})$
- *union:* $\mathbb{K}_1 \cup \mathbb{K}_2 := (G_1 \cup G_2, M_1 \cup M_2, I_1 \cup I_2)$
- *intersection:* $\mathbb{K}_1 \cap \mathbb{K}_2 := (G_1 \cap G_2, M_1 \cap M_2, I_1 \cap I_2)$

The operators of the above definition are illustrated in Figure 2.5. The dual operator inverses binary relations of the contexts and sets the objects as attributes and attributes as objects. The concept lattice of the dual context of a context $\mathbb{K}$ is the inversed order of the lattice of formal context $\mathbb{K}$. The complement operator complements the binary relation. The union operator merges the information from both contexts, while the intersection operator only keeps the information which is present in both context. Finally, the subposition and apposition operators can be seen as disjoint merge operators.

Figure 2.5: Example of the constructors defined in [Ganter and Wille, 1999]

$\mathbb{K}_p$	a	b
1	×	
2	×	×

$\mathbb{K}_p^d$	1	2
a	×	×
b		×

$\mathbb{K}_p^c$	a	b
1		×
2		

$\mathbb{K}_{p_1}$	a	b
1		×
2	×	
3	×	×

$\mathbb{K}_{p_2}$	a	b
1	×	×
2		×

$\frac{\mathbb{K}_{p_1}}{\mathbb{K}_{p_2}}$	a	b
(1,a)		×
(2,a)	×	
(3,a)	×	×
(1,b)	×	×
(2,b)		×

$\mathbb{K}_{p_3}$	a	b	c
1	×		×
2	×	×	

$\mathbb{K}_{p_4}$	b	c	d
1	×	×	×
2		×	

$\mathbb{K}_{p_3} \mid \mathbb{K}_{p_4}$	(a,a)	(b,a)	(c,a)	(b,b)	(c,b)	(d,b)
1	×		×	×	×	×
2	×	×			×	

$\mathbb{K}_{p_3} \cup \mathbb{K}_{p_4}$	a	b	c	d
1	×	×	×	×
2	×	×	×	

$\mathbb{K}_{p_3} \cup \mathbb{K}_{p_4}$	b	c
1		×
2		

2.3.3 Many-valued Contexts and Conceptual Scaling

While the basic Formal Concept Analysis theory focuses on the case of binary contexts, the approach has also been extended to a more generic class of context: the *many-valued contexts*. This kind of context is much closer to the model used in relational databases and their relation has been studied in a number of articles (see [Correia, 2002, Priss, 2005]). We first present their definition then we introduce the notion of *scaling*. Using apriori models of the data, called *scales*, many-valued contexts may be transformed as a one-valued context, and can thus be visualised using a line diagram.

Many-valued Contexts

We give here again the definition of a many-valued context which can also be found in [Ganter and Wille, 1999].

Definition 14 (Many-valued Context) *Let G be a set of objects, M be a set of attributes and W a set of possible values for these attributes, then a* many-valued context *is a tuple $\mathbb{K} = (G, M, W, I)$ where I is a ternary relation $I \subseteq G \times M \times W$, with*

$$(g, m, w) \in I, (g, m, v) \in I \quad \Longrightarrow \quad w = v.$$

$(g, m, w) \in I$ means that the object g has attribute m with value w. We sometimes use the usual notation m(g)=w, where the attribute is seen as a partial function from G to W.

We illustrate this definition using a simple example.

Example 3 (Many-Valued Context Example) *In a computer magazine, it is common to use a table like Table 2.6 to help the reader in comparing the features of different computers. The rows of the table are the computers to be compared. The columns of the table inform the reader about the company producing it, the price and the processor clock rate.*

Table 2.6: An example of many-valued context.

Model	Company	Price	Processor	User Rating
THX-39	THX	399	3 GHz	$++++$
ASC-2	LLA	599	3,5 GHz	$++$
MLD-56	MLD	349	2,7 GHz	$+++$
XTL-22	KLT	350	2,5 GHz	$++$
XTL-25	KLT	379	3 GHz	$++++$
...	...	...	...	...

Note that a many-valued context can be seen as a one-valued context directly, by using a restructuration of the definition. Indeed, we can define the context $\mathbb{K}_N := (G, \pi_{2,3}(I), I_2)$ where $\pi_{2,3}$ is the projection of I on the cross product of $M \times W$ and I_2 is defined by $I_2 := \{(g, (m, w)) \in G \times M \times W \ : \ (g, m, w) \in I\}$.

Conceptual Scaling

While we saw that it is possible to transform a many-valued context into a one-valued one, the traditional Formal Concept Analysis theory introduces the notion of *scales* to allow the use of partial orders on the values of the many-valued context. The following definition stems from [Ganter and Wille, 1999] and formalises the notion of scale.

Definition 15 (Scale) *A* scale *for the attribute m of a many-valued context is a one-valued context $\mathbb{S}_m$ with $m(G) \subseteq G_m$, where G_m is the set of possible values for the attribute m and $m(G)$ is the set of actual values for the attribute m.*[14] *The objects of the scale are called* scale values, *while the attributes are called* scale attributes.

Using scales, a many-valued context can easily be transformed into a one-valued one by replacing each attribute m of a many-valued context by the scale attributes of the corresponding scale and each value w of the attribute m for an object g by the corresponding line of the scale.

Figure 2.6 shows the scale for the attribute processor of the many-valued context.

Figure 2.6 shows the scale for the attribute rating of the many-valued context.

[14]Some of the values of an attribute m may not occur as value for any element of G.

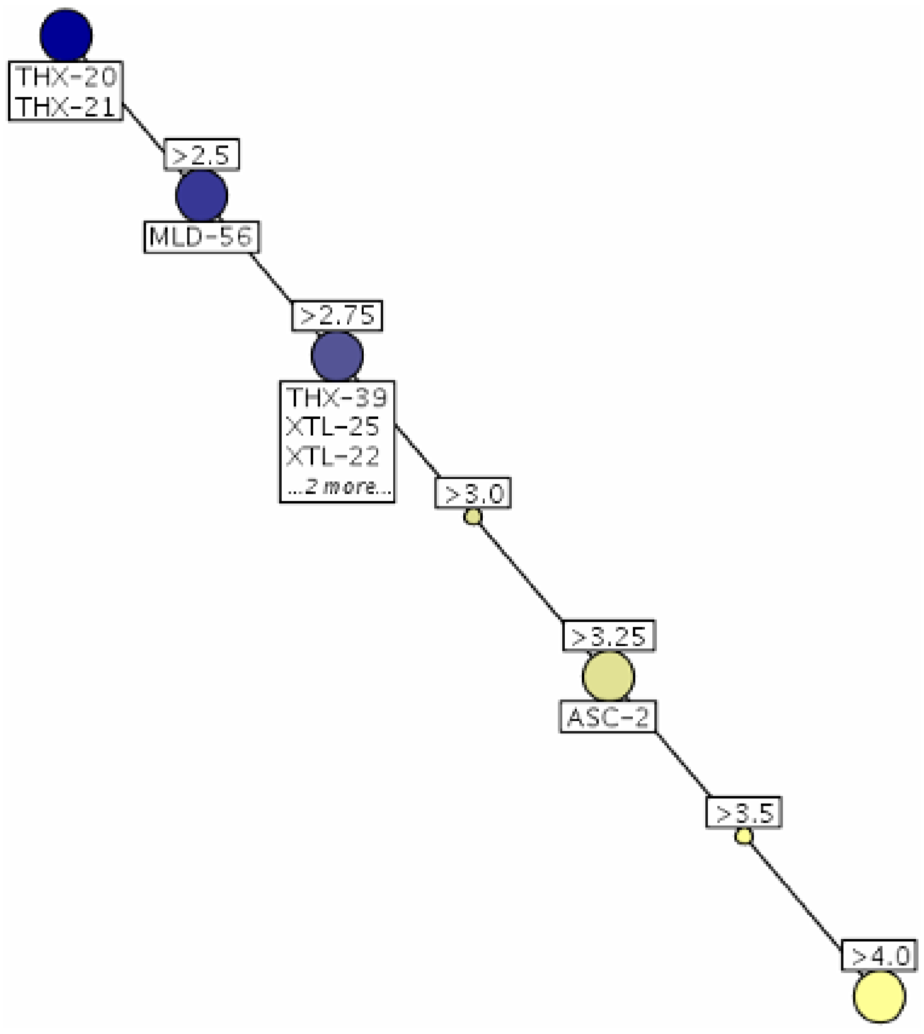


Figure 2.6: The formal concept lattice for the processor scale.

Nested Line Diagrams

In addition to allowing the transformation of the many-valued contexts into one-valued formal contexts, the scaling process is particularly adapted to a new kind of line diagrams called *nested line diagrams*. The idea behind nested diagrams is to simplify the visualisation of the line diagrams of contexts which can easily be split in a set of formal contexts with disjoint attribute sets. Each of these formal context can then be seen as a scale.

To understand this combination, it is best to use an example.

Example 4 (Scaled Many-Valued Context Example) *In Figure 2.8, the concept lattice of a scaled many-valued context of Example 3 is shown.*

This lattice is read in the following way. The nested line diagram is composed of two different lattice. The outer lattice which is in this case the processor *scale. and the inner lattice, here the* rating *scale. Each node of the outer lattice is*

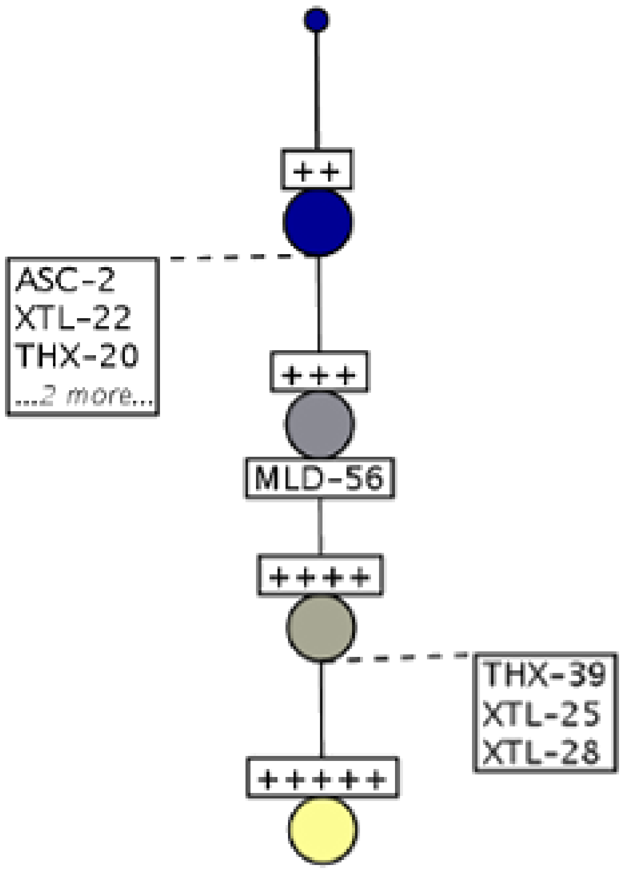


Figure 2.7: The formal concept lattice for the rating scale.

replaced by a version of the inner lattice indicating the objects at their position in the inner lattice. Both scales are read in the same manner as normal concept lattices. This means that if an object occurs at some node of the inner lattice, this object is found at the same position in the scale of Figure 2.7. And this object is also at the same node of the outer lattice as it was in the scale on Figure 2.6.

For example, the computer ASC-2 has the fastest processor but only a rating of ++.

As the example shows, the resulting lattice combines information from both scales, in that way it is possible to create different views on the data.

Nested line diagrams are extremely useful and powerful. In Chapter 6, Section 6.1.2, an application which uses these techniques is compared to the approach we present in the next chapter.

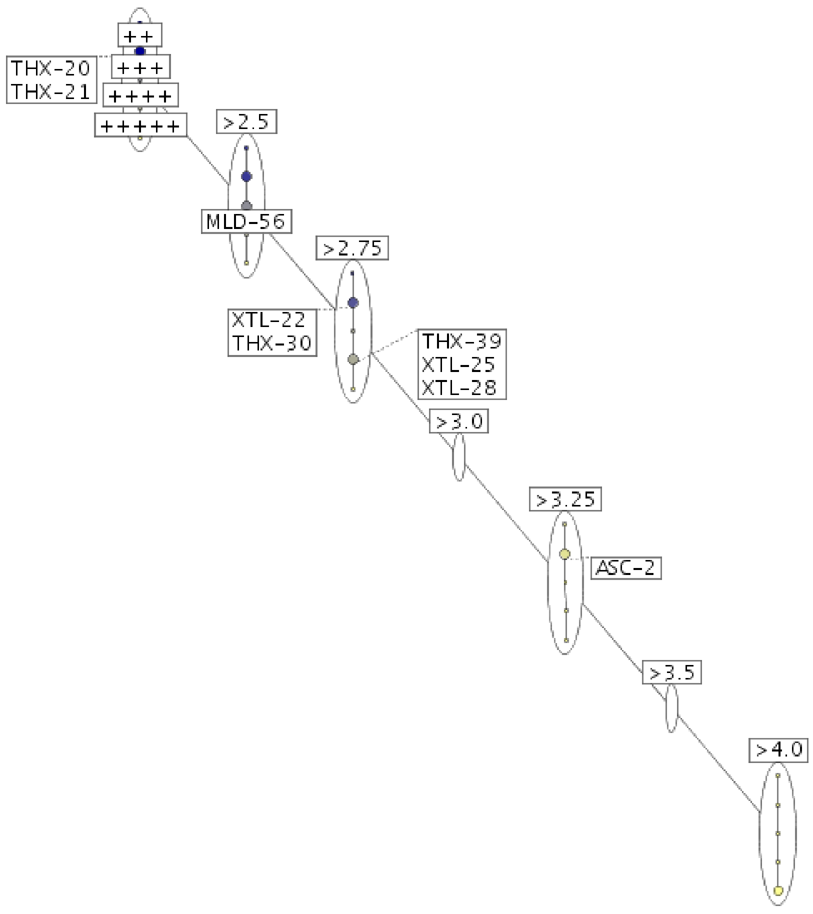


Figure 2.8: The nested line diagram for the computers embedding the rating scale in the processor scale.

2.3.4 Multicontexts

For the sake of completeness, we present here the multicontexts as presented in [Wille, 1996, Gast, 1996]. A multicontext can be seen as a network of formal contexts, which sometimes share object and attribute sets. We introduce them here because this model was one of the starting points of our investigations. The limitations and the added value of this simple model guided us in our research.

The following definition formally describes multicontexts.

Definition 16 (Multicontext) *A* multicontext *of signature $\sigma : P \to I^2$, where I and P are non-empty sets, is defined as a pair (S_I, R_P) consisting of a family $S_I := (S_i)_{i \in I}$ of sets and a family $\mathcal{R}_P := (R_p)_{p \in P}$ of binary relations with $R_P \subseteq S_i \times S_j$ if $\sigma p = (i, j)$.*

As suggested in [Wille, 1996, Gast, 1996], such a network of formal contexts can be used to investigate the coherence of the data. Since we do not use this notion in this thesis, we do not introduce the supplementary notations necessary to use multicontexts for this purpose. Little work has been published on the topic of multicontexts, but it is worthwhile to note the master thesis of Petra Gast (see [Gast, 1996]) where different aspects of multicontexts are investigated and illustrated. Notably, she extended the notion of multicontexts to that of many-valued multicontexts.

2.4 Summary

In this chapter, we introduced the necessary preliminaries to understand the approach proposed in this thesis. Notably we discussed the definition of knowledge representation, as well as introduced the main kinds of knowledge representation for knowledge bases. Then we recalled the basic methods and techniques used in database theory which play a role in this thesis. Finally, we presented the basic notions of Formal Concept Analysis which we use throughout this thesis. The next chapter introduces the query-based multicontext theory, which deals with the generation of contexts from a given data source.

Chapter 3

Query-Based Multicontexts

In this chapter we present a novel approach to model multicontexts. This approach is based on a new structure called *Query-Based Multicontext*. A query-based multicontext can be seen as a formal space of all the formal contexts which can be generated by querying a given data source. Each formal context in this space is represented by an intensional representation based on queries. In Chapter 4 we show that this intensional representation can be used for knowledge browsing, and we argue in Chapter 7 that other application domains can benefit from this approach.

This chapter addresses different topics concerning the query-based multicontext theory. In Section 3.1, we give an intuition of the ideas behind Query-Based Multicontexts. Following this informal description, we give a formal definition of the query-based multicontext in Section 3.2.1, which we further illustrate in Sections 3.2.2 and 3.2.3 using two examples. In the following sections 3.3 and 3.4, we introduce various specialisations of query-based multicontexts by considering several kinds of query infrastructures. In particular, we consider query infrastructures with the usual set operations, typed query infrastructures and semantic query infrastructures. We also show in Section 3.3.3 that by using certain set operations it is possible to define useful operations on our intensional representations of contexts. We also demonstrate how these operations correspond to the commonly used context operations we introduced in Chapter 2, Section 2.3.2.

The semantic query infrastructure which we introduce in Section 3.4 serves as a basis for the later chapters of this thesis to provide a novel way of browsing knowledge bases. To ease the browsing, two further issues are addressed in the two last sections of this chapter. In Section 3.5, we introduce constructors which are parametrisable templates to ease the definition of the intensional representations of formal contexts. They play a fundamental role in our browsing approach.

3.1 Intensional Context Representations

In Chapter 1, Section 1.1, we argued for the importance of being able to create views. We defined views as a mechanism to focus on a specific aspect of a knowledge base. We illustrated our point considering the publication-author relation, which is too large to be visualised as a whole. In the next section, we propose a model to create this kind of views using queries on the knowledge base. Throughout this chapter, we introduce different extensions of this model, which we illustrate and motivate through corresponding examples.

The novelty of our approach lies in the capability of carefully tailoring the content of the concept lattice. In order to achieve this precise selection, we introduce an intensional representation for contexts called a *context index.* Given a data source and a context index, it is possible to generate a context containing only the data relevant for the user. For knowledge visualisation purposes, a Formal Concept Analysis browsing tool can generate the concept lattice of this context and visualise this lattice using a line diagram. The purpose is then to help the user in defining context indices which either give an overview over the data or which focus on well specified parts of the data. We show in Chapter 4 how to define and manipulate context indices for this purpose. Now that we have set the goal, the first question to be answered is: what is an appropriate intensional representation? Looking at the formal definition of a formal context should help us in answering this question.

Formal contexts consist of three sets: an object set, an attribute set and a relation[1] between them. Each of these sets carries information potentially useful to the user. In order to allow the user to manipulate the contexts using context indices, we define a context index as a triple $p := (q_1, q_2, q_3)$ of queries which are used to generate the formal context. For a given data source, these queries can be evaluated using a function κ to create a context $\mathbb{K}_p$.

$$p := (q_1, q_2, q_3) \longrightarrow \mathbb{K}_p := \kappa(p)$$

The function κ is a function which stands for the evaluation of these queries and the combination of their results.

Context indices are useful because of their basic properties. We list here these properties.

- symbolic: context indices are intensional representations of the result of their evaluation,
- concise: a context index is usually a short string describing the content of the context,
- operable: operations can be defined on queries and therefore on context indices. These operations correspond to operations on the represented objects or on the context, and
- generic: their meaning is intensionally defined by the queries, but their evaluation depends on the data source used.

These properties make context indices useful at different levels for the design of a Formal Concept Analysis application. We illustrate this by presenting the life cycle of such an application.

The goal of our approach is to browse knowledge bases. The idea relies on displaying the line diagram of lattices pertinent to the user. In order to achieve this, the application sets on a life cycle with three phases: context index definition, context generation and lattice generation. Figure 3.1 shows

[1]A relation is nothing else than a set of pairs.

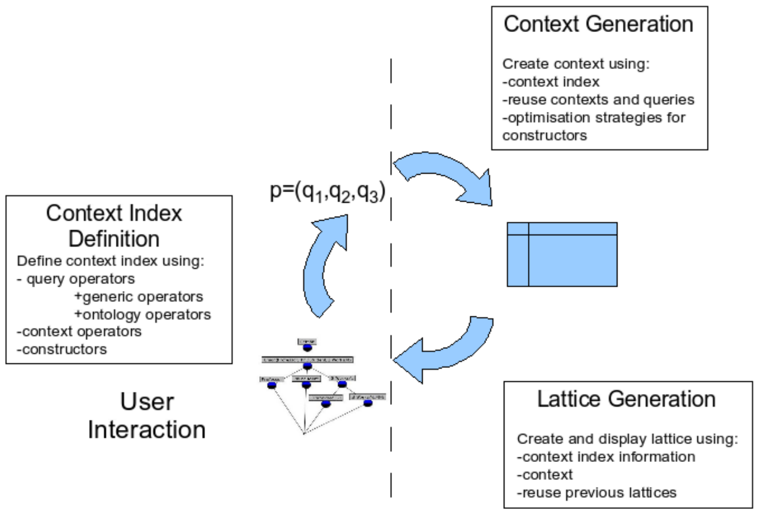


Figure 3.1: The model work flow

this life cycle. At the beginning of any given cycle, a context index is either given or created. It is then evaluated to create a formal context which can then be presented to user using some visualisation paradigm. For example, the generated formal context can be displayed using the line diagram of its concept lattice, thus preserving the complete information. The user can interact with it to create a new context index, thus entering a new round of the life cycle where the new context index is evaluated and another context is created, and displayed in the form of a concept lattice.

Context indices play diverse roles in the phases of this life cycle as outlined in Figure 3.1 and explained in further details in the following paragraphs.

In the *context index definition* phase, context indices are used as intensional representations of contexts. The user can define a context index using query operations, context operations or finally context index constructors. The latter can be seen as high-level templates to be parametrised using queries in order to create some complex context index.

During the *context generation* phase, context indices can be used to make the evaluation of contexts more efficient and economical in resources.

Finally, in the *lattice generation* phase, context indices allow for the reuse of previous lattices, and provide supplementary information which can be used when generating and displaying lattices.

The overall goal of this chapter is to expose the theoretical background to the query-based multicontexts. We start with a formal definition, illustrated by two examples. Next, we introduce the generic query and context index operators which allow to define, create and manipulate formal contexts in query-based multicontexts. Then, we propose a further extension of query-based multicontexts, where the query languages used are based on ontologies. This extension is used in the later chapters to create our browsing application. In addition to this extension and the use of operators, we present a method to create context indices based on the ideas of templates, which we call constructors. These constructors also serve in the next chapter to guide the user in the definition of the concept lattices to be used for visualisation.

Now that we have given an outline of the ideas behind the query-based multicontext, we can proceed by giving its formal definition.

3.2 Query-based Multicontexts

In this section, we first give a formal definition of the query-based multicontext. Then we illustrate this definition with two examples. The first example introduces the most simple kind of query-based multicontext, whereas the second one presents a query-based multicontext on top of an SQL database. The definition of query-based multicontexts used for knowledge bases is given in Section 3.4 of this chapter.

3.2.1 Definition: Query-based Multicontext

A query-based multicontext is a formal space containing all the contexts which can be generated by querying a given data source. In order to define this space formally, we first define query infrastructures (see Definition 17), then we introduce the structure used to represent formal contexts intensionally: *context indices* (see Definition 18). A context index represents an intensional view of the data contained in a context. These two definitions allow us to introduce the query-based multicontexts as mathematical structure.

Query Infrastructures

We first introduce some preliminary definitions: given a set Σ, called *alphabet*, a sequence of elements of Σ is called a *word* over Σ. Let Σ^* be the set of finite words over Σ. A subset L of Σ^* is called a *language L over the alphabet* Σ, in other words, a language is a set of finite words over Σ. In a similar manner, we define Σ^+ as the set of non empty sequence of elements of Σ. In this thesis, we call certain languages[2] *query languages* and we call their words *queries*. These languages are defined in the following definition.

Definition 17 (Query Infrastructures) *Let Ω be a set called* universe. *We call a* query infrastructure $\mathbb{Q}$ *a pair (L_1, L_2) of query languages over some possibly disjoint alphabet. An* instance Q of a query infrastructure (L_1, L_2)*, is a 5-tuple $(L_1, L_2, \Omega, eval_1, eval_2)$ where $eval_1$ is a map from L_1 into $\mathcal{P}(\Omega)$ and $eval_2$ a map into $\mathcal{P}(\Omega \times \Omega)$. The elements of L_1 are called* set queries*, while elements of L_2 are called* relation queries*.*

An instance of a query infrastructure is a structure to represent the necessary elements to query a given data source (i.e. the data source schema plus the data). For example, the AIFB knowledge base is a data source. The abstraction *query infrastructure* corresponds to the data source schema shared by all the instances of this query infrastructure, that is, the data sources themselves.

Since these data sources might have their own query language or means of providing the data, it is important to prevent confusions in terms. Thus, we

[2]We use the term query language, when the purpose of its words is to represent or retrieve data.

call *data source* the actual data providing mechanism, and *data source language* the language used to communicate with the data source.

The elements of L_1 can be seen as monadic predicates (that is, unary predicates), the elements of L_2 can be seen binary predicates, whereas the evaluation functions play a role similar to an interpretation in logic. We prefer the more generic term *query* for two reasons.

First of all, though many query languages can be formulated in a logical language, doing so would have greatly reduced the genericity of the query-based multicontext definition we present next. Many query languages cannot be easily formulated as a logical language. For example, one could think of using such a language as CQL.[3] We discuss briefly a similar topic in Chapter 6, Section 6.1.3.

Finally, the term *query* denotes perfectly their role in our approach. While the purpose of formulas is usually to describe concisely some set of entities and their relation independently of a specific interpretation, the goal of a query is to find a narrow set of items in a large collection that satisfy a well-understood information need ([Marchionini, 1995]). A query is a request for information which is interpreted with respect to a given data source. The answer of the query depends on this data source, but the intended meaning lies in the query and some assumptions regarding its interpretation. In Section 3.3, we introduce query operators with given semantic interpretations. It is assumed that all the query answering mechanisms of the data sources respect these interpretations.

Query-Based Multicontexts

Using the query infrastructure we just introduced, we can formalise the idea of a query-based multicontext:

Definition 18 (Query-Based Multicontext) *Let $\mathbb{Q} = (L_1, L2)$ be a query infrastructure. We define $\mathbb{P} := L_1 \times L_1 \times L_2$ and call an element p of $\mathbb{P}$ a* context index. *Let $Q_\Omega := (L_1, L_2, \Omega, eval_1, eval_2)$ be an instance of the query infrastructure $\mathbb{Q}$.*

For a context index $p = (q_1, q_2, q_3) \in \mathbb{P}$, we define its induced query-based context *as: $\mathbb{K}_p := (eval_1(q_1), eval_1(q_2), eval_2(q_3) \cap (eval_1(q_1) \times eval_1(q_2)))$. A* query-based multicontext *for Q_Ω, denoted by* QBMC(Q_Ω)*, is defined as the set $\{\mathbb{K}_p | p \in \mathbb{P}\}$. We call the mapping from $\mathbb{P}$ in QBMC(Q_Ω) the context realisation and we denote it by: κ_{Q_Ω} (or simply κ if it is clear from the context). So, for all $p \in \mathbb{P}$, $\kappa(p) = \mathbb{K}_p$.*

Before commenting on this definition, observe that we use certain conventions throughout this thesis to denote queries from L_1 and L_2; queries q_3, q_6, etc. denote elements of L_2, while all the others denote queries of L_1. Moreover, to increase the readability, we sometimes use the vector notation

$$\begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$$

instead of (q_1, q_2, q_3) when denoting a context index.

[3]CQL is a text-based language used to query the data collections of libraries. It has been standardised by the Library of Congress (see `http://www.loc.gov/standards/sru/cql/`).

For a given instance of a query infrastructure, a context index fully specifies a formal context. The object set of the formal context, respectively the attribute set, contains the result of the evaluation of the first query, respectively the second query. The incidence relation is defined as the pairs common to the relation returned by the third query and to the cross product of the object set with the attribute set. The idea behind this construction is that the relation desired is only between the objects and attributes of the context. The relation between other objects and attributes is not relevant at that point. In other words, the only pairs of the relation which are kept are the pairs with a first argument belonging to the object set and with a second argument belonging to attribute set, thus building a formal context. For instance, if the goal of the formal context is to display the relation between professors and their research topics, then the pairs of the relation between PhD Students and research topics are not relevant. The underlying database, however, might not make any difference and return all the pairs.

From the definition, it is clear that a query based multicontext is fully defined once an instance of a query infrastructure has been chosen. The actual content of each context of the query-based multicontext is dependent on the actual evaluation of the queries used to create it. query-based multicontexts sharing the same query infrastructure (i.e. the same query languages) might not return the same contexts. However, their answers might coincide for a part of the languages. This distinction is therefore particularly interesting for the integration of diverse data sources.

Now that we have provided the formal definition of the query-based multicontext, the next couple of sections provide two examples: one showing a very basic query-based multicontext, the other demonstrating a query-based multicontext on top of an SQL database.

3.2.2 A Simple Example

To illustrate the ideas behind Query-Based Multicontexts, we start by presenting a type of query-based multicontext which is extremely restricted in expressive power. This lack of expressivity is due to the absence of operators in the language. Such a kind of query-based multicontext is called *operator free.*

Example 5 (Operator Free Query-Based Multicontext) *As a base example, we consider a simple database, containing projects and topics, and a relation* isOnTopic *between them specifying to which research topic the project mainly contributes.*

Ω *consists of a number of projects and research topics:* Ω := {*PADLR, SEKT, SWAP, SWWS, Semantic Web, P2P, Web Services*}

We can specify the languages L_1 *and* L_2 *in the following way:*

- L_1 *corresponds to class names:*
  $L_1 := \{project,\ topic\}$
- L_2 *corresponds to relation names:*
  $L_2 := \{isOnTopic\}$

L_1 *and* L_2 *can be seen as languages. Using* $q_1 :=$ project, $q_2 :=$ topic *and* $q_3 :=$ isOnTopic, *it is possible to define the context index p:= (project,topic,isOnTopic). Now, we consider the following evaluation functions:*

- *$eval_1$*
  - $eval_1(q_1) = eval_1(project) = \{PADLR, SEKT, SWAP, SWWS\} \subseteq \Omega$
  - $eval_1(q_2) = eval_1(topic) = \{Semantic\ Web, P2P, Web\ Services\} \subseteq \Omega$
- *$eval_2$*
  - $eval_2(q_3) = eval_2(isOnTopic) =$
    $\{(PADLR, Semantic\ Web), (SEKT, Semantic\ Web), (SWAP, Semantic\ Web), (SWWS, Semantic\ Web), (PADLR, P2P), (SWAP, P2P), (SWWS, Web\ Services)\} \subseteq \Omega \times \Omega$

Table 3.2.2 presents the context $\mathbb{K}_p := \kappa(p)$, whereas Figure 3.2 shows the lattice for this context.

Table 3.1: The context $\mathbb{K}_p$ for the context index $p :=$ (project,topic,isOnTopic).

$\mathbb{K}_p$	Semantic Web	P2P	Web Services
PADLR	×	×	
SWAP	×	×	
SWWS	×		×
SEKT	×		

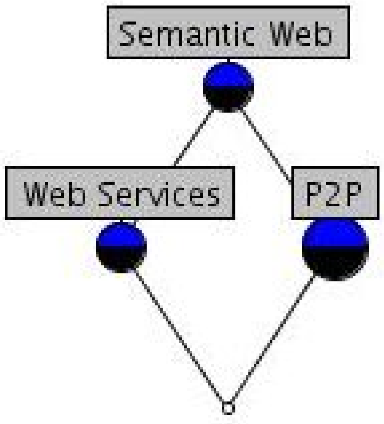


Figure 3.2: The concept lattice corresponding to the context generated from the context index $p :=$ (project,topic,isOnTopic).

The preceding example is very simple. No operators were defined on the languages L_1 and L_2. query-based multicontexts of these types are the most generic instances of multicontexts in the sense of [Wille, 1996]. However, they do not give any means of manipulating and refining the content of the contexts. Often it can be useful to characterise more precisely the components of a formal context. For example, it does not make sense to display the whole concept lattice of authors/publications of the AIFB knowledge base as the resulting lattice contains 1,021 formal concepts. A means of focusing on specific parts of the data would hence be useful. In order to achieve this, we extend the definition of query-based multicontexts by allowing the use of query operators to specify smaller formal contexts. These query operators can then be used in the definition of more meaningful context indices. To illustrate the role of operators in the expressivity of formal contexts, we present an example of a query-based multicontext on top of a database in the following section.

3.2.3 An SQL Example

This section introduces a more complex example of a query-based multicontext based on relational databases. Relational databases are omnipresent in today's business world, and the visualisation of the data found in a database is a crucial task of today's manager. It is therefore useful to apply the query-based multicontext approach to relational databases.

The specification of an instance of a query infrastructures fully defines a query-based multicontext. We define Ω_{DB} as the set of possible tuples of the database.[4] This means that for any given instance of a query infrastructure on Ω_{DB}, the evaluation of a query q_1 belonging to L_1 by $eval_1$ returns a set of tuples of the database, and the evaluation of a query q_3 belonging to L_2 by $eval_2$ returns a set of pairs of tuples. However, since traditional SQL database management systems return relations, i.e. sets of tuples and do not return pairs of tuples, this means that to implement a query infrastructure on a relational database each of the database rows of a result of a relation query (i.e. belonging to L_2) must be split in a corresponding pair of tuples. To achieve this, the assignment is performed using namespaces A and B which specify whether a given argument belongs to the first component or the second.

Example 6 (Car Sharing) *Suppose that database $\mathcal{DB}$ belongs to a car sharing facility to offer the possibility for the user to choose their cars according to the proximity. The database $\mathcal{DB}$ consists of three tables:* vehicle, location-vehicle *and* location-position. *These tables have the following database schemas:*

vehicle:

vehicle	*category*	*price*

location-vehicle:

location	*vehicle*

location-position:

location	*geographicalPosition*

The first database table contains the available vehicles, their price and their type, while the second contains the name of the location and the vehicle available at that location. Finally, the last table gives the geographical position (as GPS coordinates) of the locations.

From the web site of the car sharing agency, a member of the agency can consult the cars suitable for his purpose by filling two parameters: the maximal hourly price of the car $MAXIMAL\_PRICE$, and the user's start position $USER\_POSITION$.[5]

[4]Two tuples are considered identical, if their respective components are equal.

[5]We abstract here from the method used to specify the value of this parameter.

For example, the value of these parameters for a specific user could be
MAXIMAL_PRICE=3.00 and
USER_POSITION= (51.3219444444440,9.5086111111111126)

Using the three following queries q_1, q_2 and q_3, a context index (q_1, q_2, q_3) can be defined.

q_1='(SELECT vehicle AS A_vehicle FROM Vehicle
WHERE category IN (catA, catB, catC)
AND price < MAXIMAL_PRICE)'

q_2='(SELECT location AS B_location,
distance(geographicalPosition, USER_POSITION) AS B_distance
FROM location-position
WHERE distance(geographicalPosition, USER_POSITION) < 500)'

q_3='SELECT A.vehicle AS A_vehicle , B.location AS B_location,
distance(GeographicalPosition, USER_POSITION) AS B_distance
FROM vehicle A , location-vehicle B, location-position C
WHERE vehicle.vehicle= location-vehicle.vehicle
AND location-position.location= location-vehicle.location'

Table 3.2: The context $\mathbb{K}_p$ of the car sharing context index.

$\mathbb{K}_p$	Hanaustr 5, 60m	Hardystr 16, 70m	Esmachstr 24, 90m	Bermannstr 145, 105m	Lilienstr 13, 450m
Renault Twingo	×				
Volkswagen Golf		×	×		
Volkswagen Combi	×	×	×		
Ford Focus	×	×		×	×

The function called distance *takes two GPS coordinates as input and returns the distance between the two points.*

The realisation of this context index is a formal context with the following content:

- *objects: the vehicles of one of the three types A, B or C, whose price is less than 3.00 euros.*
- *attributes: are pairs indicating the location and distance pairs situated at less than 500 meters*
- *relation: the vehicles and locations together with their distances where the* vehicle *column are equal and where the vehicle is present in the object set and the pair (location,distance) in the attribute set.*

Table 6 presents the context generated from the context index presented above for the data found in Tables 3.3, 3.4 and 3.5. We show in Figure 3.3 the concept lattice for this context.

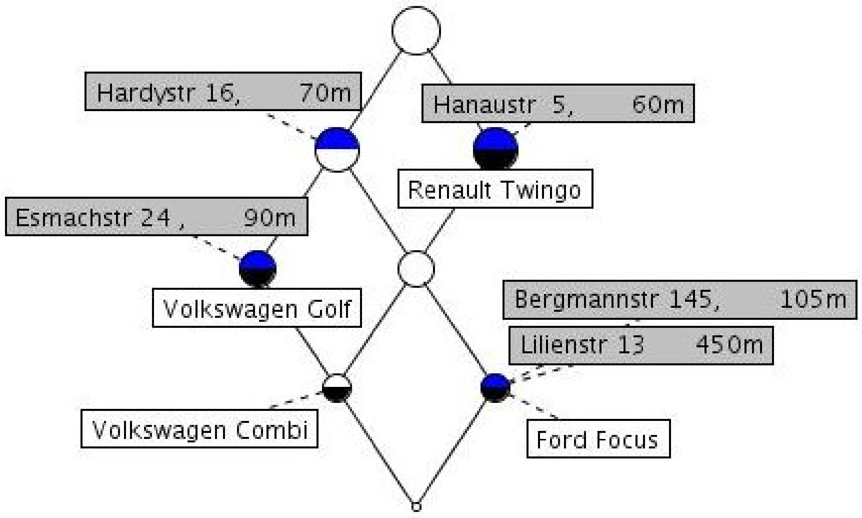


Figure 3.3: The lattice for the SQL example using the Tables 3.3, 3.4 and 3.5.

This example shows that the use of more complex operators allows for the definition of more interesting contexts. These formal contexts are more interesting because they filter out many objects and attributes which are not needed in a given context. The filtering is performed at two places. First of all in q_1 the cars are selected according to their categories and for their price. The second filtering occurs in query q_2 where only locations which are closer than 500m are considered. Note also that the attributes are pairs (i.e. 2-tuples) whereas objects are singletons (i.e. 1-tuples).

In the following section, we introduce diverse generic operators for any query infrastructure. Using these operators, it is not only possible to define more expressive formal contexts, but also typical and useful operators on context indices. These context operators are typical in the sense that they correspond to traditional context operators as defined in [Ganter and Wille, 1999].

The use of these query operators also allows for the definition of context index operators. The goal of the following section is to introduce such operators on the languages L_1 and L_2 in order to define meaningful context index operations, i.e. operations on context indices which correspond to pertinent operations on contexts.

3.3 Operators

In order to manipulate more effectively the formal contexts generated from context indices, it is useful to define new operators. In this section, we deal with two kinds of operators *query operators* and *context index operators*.

Table 3.3: The vehicle table for the car sharing example.

Vehicle	Category	Price
Ford Focus	A	1.35
Mercedes Transporter	C	4.24
Renault Espace	B	3.70
Renault Twingo	A	1.20
Volkswagen Golf	A	1.50
Volkswagen Combi	B	2.25

Table 3.4: The vehicle-location table for the car sharing example.

Location	Vehicle
Bergmannstr 145	Ford Focus
Esmachstr 24	Renault Twingo
Esmachstr 24	Renault Espace
Esmachstr 24	Volkswagen Combi
Esmachstr 24	Volkswagen Golf
Hanaustr 5	Renault Espace
Hanaustr 5	Ford Focus
Hanaustr 5	Renault Twingo
Hanaustr 5	Volkswagen Combi
Hanaustr 5	Volkswagen Combi
Hardystr 16	Mercedes Transporter
Hardystr 16	Volkswagen Golf
Hardystr 16	Volkswagen Combi
Hardystr 16	Ford Focus
Lilienstr 13	Renault Espace
Lilienstr 13	Ford Focus
Lilienstr 13	Mercedes Transporter

Table 3.5: The location-position table for the car sharing example.

Location	GeographicalPosition
Bergmannstr 145	(51.32194444444455,9.508611111111129)
Esmachstr 24	(51.32194444444455,9.508611111111129)
Hanaustr 5	(51.32194444444445,9.508611111111125)
Hardystr 16	(51.32194444444435,9.508611111111123)
Lilienstr 13	(51.32194444444466,9.508611111111118)

We first present a specialisation of the query infrastructures defined in Section 3.2.1 by considering generic query operators. For practical purposes, it is also useful to introduce the new notion of typed query infrastructures.

At the end of this section, we also define operators on context indices which allow the definition of new formal contexts from existing ones. We also show that these operators on context indices correspond to the traditional context operators as defined by Ganter and Wille in [Ganter and Wille, 1999].

Operator Definition

For the rest of this section, we need a formal definition of what we mean by an operator for queries, respectively context indices. In the case of query languages, we need an extended version in order to combine different query languages.

Definition 19 (Operators) *Let A be a set, and n a natural number. An n-ary operator ω on A is a mapping from A^n to A. For a query infrastructure $\mathbb{Q} = (L_1, L_2)$ and two natural numbers k and j (possibly equal to zero) and for i=1,2, a L_i-query operator is a mapping from $L_1^k \times L_2^j$ into L_i.*

The domain $L_1^k \times L_2^j$ of the mapping is a way of formulating that the arguments of a specific operator may be from L_1 or L_2. For example, we define below a union operator on $L_1 \times L_1$ returning queries of L_1.

Purpose of Operators

The use of operators in query-based multicontext has several advantages. First, using generic operators, it is possible to create more expressive contexts than in an operator-free query-based multicontext. This is illustrated by the following example.

Example 7 (Union and Intersection Operators) *Let L_{n1} and L_{n2} be two such operator-free languages so that $L_{n1} := \{$Journal article, Conference article, Workshop article, Professor, Assistant, PhDStudent, Member of Group 1, Member of Group 2 $\}$, and $L_{n2}:=\{$isauthorOf$\}$.*

To add union and intersection operators to these languages, we only need to define the semantics of these operators. The union $\cup$ of two queries is evaluated as the union of their evaluations, and the intersection $\cap$ of queries is evaluated as the elements common to the evaluations of these queries.

For example, the following context indices can be constructed:

1. *p_1:=($\cup$(PhDStudent, Assistant), $\cup$(Journal article, Conference article), isAuthorOf)*
2. *p_2:= ($\cap$($\cup$(PhDStudent, Assistant), Member of Group 1), $\cup$(Workshop article, Conference article), isAuthorOf)*

The context corresponding to the first context index contains in its object set PhD Students and Assistants, in its attribute set only articles published in a journal or in the proceedings of a conference. Note that professors have not been selected for the objects of the first context. Writing the necessary operations in an operator-free language, would have required at least[6] the definition of four supplementary concepts corresponding to professor or assistant, professor Or PhDStudent, assistant or PhD student and professor or PhD Student or assistant, that is one for each of the possible combinations. The use of operators simplifies the basic language because only the atomic concept and role names need to be defined.

The second example shows how the two operators can be combined to express even more complex element sets. The set of objects of the induced context $\mathbb{K}_{p_2}$

[6]The more elements L_{n1} has, the more combinations exist. The number of combinations grows exponentially with the size of L_{n1}.

contains the PhD Students and the assistants who are also members of Group 1. Interestingly, the concept member of Group 1 *corresponds to the individuals which are member of Group 1, that is this concept is defined by the role the instances play in relation to the Group 1.*[7]

Constructing these two formal contexts using an operator-free query language would have required finding more names for the supplementary queries, whereas introducing operators simplifies the definition of the basic language.

The second advantage of the introduction of operators on queries or context indices lies in the reuse of existing structures and techniques. On the one hand, database literature offers a large amount of interesting optimisations, some of which may be used to optimise the evaluation of the operators we define. On the other hand, the evaluation of these operators may be transmitted to a database or a knowledge base to be executed remotely. Data sources, such as database management systems or knowledge bases usually have highly optimised algorithms to evaluate certain type of queries more efficiently. It is therefore preferable to let them perform the tasks for which they have been designed. Chapter 5, Section 5.3.1 discuss the implementation details regarding the delegation of operators.

3.3.1 Generic Query Operators

We already presented informally two query operators, namely the union and intersection, but for our purpose, other operators are needed. In this section, we introduce operators that can be implemented on top of any query infrastructure. For this reason we call them *generic operators*. All of these operators can be easily defined using operators from the relational algebra (see for example [Vossen, 1994] for an explanation of traditional relational algebra operators). Some of these, like the union, intersection, cross product and difference (i.e. the \-operator) are very frequently used. [Abiteboul et al., 1995] presents the main complexity results regarding the different database language flavors.

Table 3.6 lists these operators and defines the sets and relations they return. In the rest of this thesis, we consider that the languages L_1 and L_2 contain these generic operators.

Let us now define the semantics of these operators. Table 3.6 defines some useful generic query primitives, which we consider from now on contained in the query languages L_1 and L_2. In order to simplify the table, we use the following conventions: $q_1, q_2 \in L_1$ and $q_3, q_6 \in L_2$ and R is a binary relation over Ω, i.e. $R \subseteq \Omega \times \Omega$.

Note also that we disallow self reference of queries.

Note that except the disjoint union operators and indexed result operator, all these operators can be implemented in a relational database. Note, however, that the operator q_3^+ and $q_3^+(q_1)$ cannot be implemented in a SQL92 database since these databases cannot answer recursive queries.

The choice of these particular operators has been guided by different reasons. Except the recursive operators q_3^+ and $q_3^+(q_1)$, all operators are necessary for our purpose because they play an important role in many context creation process. The recursive operators are useful in the case of an RDF implementation, because it can then be used to simulate the subumption hierarchy.

[7]It is a role since memberOf is a relation and Group 1 is an instance.

Table 3.6: The table defining the operators symbols for the generic query operators (q_1, q_2, q_3, $q_4 \in L_1$ and q_3 and $q_6 \in_2$).

Notation	evaluation	Meaning
L_{1gen}	Result in Ω	
$q_1 \cap q_2$	$eval_1(q_1) \cap eval_1(q_2)$	intersection of the results
$q_1 \cup q_2$	$eval_1(q_1) \cup eval_1(q_2)$	union of the results
$\overset{t}{q_1}$	$\overset{t}{eval_1}(q_1)$	indexed result
$q_1 \overset{t}{\cup} q_2$	$\overset{t}{eval_1}(q_1) \cup \overset{t^-}{eval_1}(q_2)$	disjoint results
$q_1 \backslash q_2$	$\{a \in \Omega : a \in eval_1(q_1) \backslash eval_1(q_2)\}$	elements in q_1 but not in q_2
$dom(q_3)$	$\{a \in \Omega : (a,b) \in eval_2(q_3)\}$	domain of the relation
$range(q_3)$	$\{b \in \Omega : (a,b) \in eval_1(q_3)\}$	range of the relation
$q_3(q_1)$	$\{b \in \Omega : \exists a \in eval_1(q_1)$ and $(a,b) \in eval_2(q_3)\}$	elements related through q_3 to elements of q_1
$q_3^{-1}(q_1)$	$\{a \in \Omega : \exists b \in eval_1(q_1)$ and $(a,b) \in eval_2(q_3)\}$	elements related through the inverse of q_3 to elements of q_1
$q_3^+(q_1)$	$\{y \in \Omega : \exists x \in eval_1(q_1)$ and $(x,y) \in eval_2(q_3^+)\}$	reachable through q_3 from q_1
L_{2gen}	Result in $\Omega \times \Omega$	
$id(q_1)$	$\{(a,a) \in \Omega \times \Omega : a \in eval_1(q_1)\}$	identity
$q_1 \times q_2$	$\{(a,b) \in \Omega \times \Omega : a \in eval_1(q_1) \wedge b \in eval_1(q_2)\}$	cross product
$q_3 \backslash q_6$	$\{(a,b) \in \Omega \times \Omega : (a,b) \in eval_2(q_3) \backslash eval_2(q_6)\}$	pairs present in q_3 but not q_6
q_3^c	$\{(a,b) \in \Omega \times \Omega : (a,b) \in eval_2(dom(q_3) \times range(q_3)) \backslash eval_2(q_3)\}$	complement of the relation q_3
$q_3 \cap q_6$	$eval_2(q_3) \cap eval_2(q_6)$	intersection of the relations (i.e. of the returned pairs)
$q_3 \cup q_6$	$eval_2(q_3) \cup eval_2(q_6)$	union of the relations (i.e. of the returned pairs)
$q_3 \overset{t.}{\cup} q_6$	$\overset{t.}{eval_2}(q_3) \cup \overset{t^-.}{eval_(}q_6)$	make range disjoint results of the pairs
$q_3 \overset{.t}{\cup} q_6$	$\overset{.t}{eval_2}(q_3) \cup \overset{.t^-}{eval_(}q_6)$	make domain disjoint results of the pairs
q_3^{-1}	$\{(b,a) \in \Omega \times \Omega : (a,b) \in eval(q_3)\}$	dual (or inverse) relation
$q_3 \bowtie q_6$	$\{(a,c) \in \Omega \times \Omega : \exists b \in eval(range(q_3) \cap dom(q_6))$ and $((a,b) \in eval(q_3)$ and $(b,c) \in eval(q_6))\}$	simple join
$q_3 \bowtie_{q_1} q_6$	$\{(a,c) \in \Omega \times \Omega : \exists b \in eval(q_1) and((a,b) \in eval(q_3)$ and $(b,c) \in eval(q_6))\}$	join with selection
q_3^+	$\{(x,y) \in \Omega \times \Omega : \exists x_1, \ldots, x_n \in \Omega,\ xRx_1R \ldots Rx_nRy$ with $R = eval(q_3)\}$	transitive closure of the relation resulting from q_3

To give an idea of the semantics of these operators, consider them as set and graph operations. For the sake of understandability, we write that an element belongs to a query if it belongs to the result of its evaluation. For example, we say that x belongs to $q_1 \cap q_2$ instead of writing x belongs to the result of the evaluation of the query $q_1 \cap q_2$.

An idea of the union and intersection operators has already been given in the Example 7 of the previous section. The evaluation of "\" returns the elements from the results of the first query which are not contained in the results of the second one. The "dom" and "range" operators make a projection on the first, respectively second component of the relation given as parameter. The operator q_3^{-1} swaps the components of the pairs of the result of q_3, such that if a pair (a, b) belongs to the results of the evaluation of q_3, the pair (b, a) belongs to the result of the evaluation of q_3^{-1}. The operators $q_3(q_1)$ return the elements b that there is an element a belonging to q_1 and a pair (a, b) belonging to q_3. The operator $q_3^{-1}(q_1)$ returns all the elements b such that there exists an element a belonging to q_1 (as above) and a pair (b, a) belonging to q_3. The $\times$-operator creates the cross product of the results of its parameters. The complementation operators q_3^c returns the pairs which belong to the cross product of the set of elements of the domain of q_3 with the set of elements of the range of q_3 and which do not belong to the results of q_3. The q_3^+-operator returns the pairs of elements for which there is a path from one to the other, that is a sequence of pairs of the results of q_3, where two neighbouring elements are equal. The operator $q_3^+(q_1)$ restricts itself to the elements of the second component of q_3 reachable from an element of the result of q_1 using a path in q_3^+.

Finally, the join operators: $q_3 \bowtie q_6$ and $q_3 \bowtie_{q_1} q_6$ are similar to the join operators found in databases. They return pairs of elements (a, c), so that a belongs to the domain of q_3 and c to the range of q_6, and that there is a common element b belonging both to the range q_3 and to the domain of q_6.

The following example demonstrates the usage of some of these operators.

Example 8 (Operator Example) *A researcher is interested in the topics of the publications of the PhD Students of group 1. The context index to represent this can then be written as* $p = (q_1,\ q_2,\ q_3)$*,*

where these three queries are defined as:

1. $q_1 =$ PhDStudent $\cap$ Member of Group 1*,*
2. $q_2 = q_3(q_1)$ *and*
3. $q_3 =$ isAuthorOf $\bowtie_{\textsf{Conference article}}$ hasTopic

The results of the evaluation of q_1 *contain only the elements shared by the results of the queries* PhDStudent *and* Member of Group 1*. The results of* q_3 *are pairs* (a, b) *such that:*

- a *belongs to the domain of* isAuthorOf*,*
- b *belongs to the range of* hasTopic
- *there exists an element* c *of the results of* Conference article *so that:*
  - (a, c) *is a pair in the result of* isAuthorOf*.*

– *(c, b) is a pair in the result of hasTopic*

In other words, the result of q_3 contains all the pairs (p, t) consisting of a person p who published a conference article on a given topic t.

From the definition, we can see that q_2 returns the elements of the range of the result of q_3, which have a first component which belongs to the results of q_1. In other words, the results of q_2 are only the topics of the publications of at least one of the PhD Student members of group 1.

As the example shows, is it possible to express precise and complex context indices using these operators.

There are two further reasons to consider them as central to the query-based multicontext theory. On the one hand, they allow the definition of context index operators and constructors. The details for these aspects are presented in this chapter, Sections 3.3.3 and 3.5. On the other hand, some of these operators are very suitable for the interaction with a lattice.

Queries and Datalog

The query operators presented above have been given a semantics by giving a mapping of the query terms to Ω and $\Omega \times \Omega$ directly. In this paragraph, we introduce the datalog programs corresponding to these query operators. Since the expressivity of the different flavors of datalog have been well studied in the literature (we refer again to the Alice-book [Abiteboul et al., 1995]), the purpose of this correspondence is to be able to determine the type of flavors needed to implement a query-based multicontext infrastructure on top of a datalog engine.

To illustrate our purpose, we first consider the context index: $p := (q_1, q_2, q_3)$ where the three queries are datalog programs: $P_1(x)$, $P_2(y)$ and $P_3(x, y)$ with the respective rules' heads: $q_1(x)$, $q_2(x)$, $q_3(x)$.

The evaluation of the context index can be rewritten:

$$(eval_1(q_1(x)), eval_1(q_2(x)), eval_2(q_r(x, y)))$$

where $q_r(x, y)$ is the head predicate of a new rule defined by
$q_r(x, y) \leftarrow q_1(x), q_2(y), q_3(x, y)$.

The evaluation of datalog queries and rules in general is a topic well studied in the database literature. To map our generic query operators to datalog program, for each query q of L_1 or L_2 we associate a datalog program P_q with a single and unique head predicate pred-q_1 of corresponding arity. The predicate $\sim$ applies to such a program and is interpreted as a negation as failure of the head predicate of this program.

Following the convention used in Table 3.6, $q_1, q_2 \in L_1$ and $q_3, q_6 \in L_2$, Table 3.7 presents the implementation of the generic queries using the datalog syntax. In the same way, to each query q_3 in L_2, we associate a datalog program P_{q_3} with a single goal predicate pred-q_3 of arity 2. For the complement query, a supplementary rule is needed.

It is important to note that the datalog programs resulting from this process are stratified and non-disjunctive (see Definition 6 from Chapter 2, Section 2.2.2). The absence of disjunction is easily seen when noting that none of the heads of these datalog rules contains more than one predicate. Moreover, the programs are stratified since self-reference is not allowed in their construction.

Table 3.7: The datalog implementation of the query operators of table 3.6.

Operator	Datalog program
$q_1 \cap q_2$	intersection-q_1-$q_2(x) \leftarrow$ pred-$q_1(x)$, pred-$q_2(x)$.
$q_1 \cup q_2$	union-q_1-$q_2(x) \leftarrow$ pred-$q_1(x)$. union-q_1-$q_2(x) \leftarrow$ pred-$q_2(x)$.
$q_1 \setminus q_2$	q_1-sub-$q_2(x) \leftarrow$ pred-$q_1(x)$, $\sim$pred-$q_2(x)$.
dom(q_3)	dom-$q_3(x) \leftarrow$ pred-$q_3(x,y)$.
range(q_3)	range-$q_3(y) \leftarrow$ pred-$q_3(x,y)$.
$q_3(q_1)$	related-q_3-$q_1(y) \leftarrow$ pred-$q_3(x,y)$, pred-$q_1(x)$.
$q_3^{-1}(q_1)$	invrelated-q_3-$q_1(x) \leftarrow$ pred-$q_3(x,y)$, pred-$q_1(y)$.
$q_3^{+}(q_1)$	trans-$q_3(x,y) \leftarrow$ pred-$q_3(x,y)$. trans-$q_3(x,z) \leftarrow$ trans-$q_3(x,y)$, pred-$q_3(y,z)$. trans-q_3-$q_1(y) \leftarrow$ pred-$q_1(x)$, trans-$q_3(x,y)$.
$id(q_1)$	id-$q_1(x,x)\leftarrow$ pred-$q_1(x)$.
$q_1 \times q_2$	cross-q_1-$q_2(x,y)\leftarrow$ pred-$q_1(x)$, pred-$q_2(y)$.
$q_3 \setminus q_6$	q_3-sub-$q_6(x,y) \leftarrow$ pred-$q_3(x,y)$, $\sim$pred-$q_6(x,y)$.
q_3^c	cross$q_3(x,y) \leftarrow$ dom-$q_3(x)$, range-$q_3(y)$. complement-$q_3(x,y) \leftarrow$ cross$q_3(x,y)$, $\sim$pred-$q_3(x,y)$.
$q_3 \cap q_6$	intersection-q_3-$q_6(x,y) \leftarrow$ pred-$q_3(x,y)$, pred-$q_6(x,y)$.
$q_3 \cup q_6$	union-q_3-$q_6(x,y) \leftarrow$ pred-$q_3(x,y)$. union-q_3-$q_6(x,y) \leftarrow$ pred-$q_6(x,y)$.
q_3^{-1}	inv-$q_3(x,y) \leftarrow$ pred-$q_3(y,x)$.
$q_3 \bowtie q_6$	join-q_3-$q_6(x,y)\leftarrow$ pred-$q_3(x,z)$, pred-$q_6(z,y)$.
$q_3 \bowtie_{q_1} q_6$	join-q_3-q_1-$q_6(x,y)\leftarrow$ pred-$q_1(z)$, pred-$q_3(x,z)$, pred-$q_6(z,y)$.
q_3^{+}	trans-$q_3(x,y) \leftarrow$ pred-$q_3(x,y)$. trans-$q_3(x,y) \leftarrow$ trans-$q_3(x,z)$, pred-$q_3(z,y)$.

The complexity results described in [Abiteboul et al., 1995] show that non disjunctive stratified logic programs remain tractable.

We now consider a typed version of these operators.

3.3.2 Typed Query Language

In Section 3.4 of this chapter, we define an alternative kind of query infrastructure to query knowledge bases. In order to define it, we need to be able to type the results of queries. If queries are typed, it is then possible to ensure what the content of the evaluated context should be. For example, knowing whether the elements of the contexts are instances or concepts is important when displaying, interacting or defining views. The colors and forms of some of the display elements can be adapted depending on types. Finally, type information can simplify the implementation of certain context creation algorithms. While we focus in this thesis on the ontology based query infrastructure which we present in Section 3.4, the definition given here provides a generic and useful way to define typed query-based multicontexts.

Depending on the application, a different set of types may be used in a typed query-based multicontext. For instance, in ontology-based applications, this set might[8] contain the types *concept*, *relation*, *instance* and *relation instance*. In some cases, it is useful to consider the union of queries of different types, to address this requirement, a special type λ is also introduced. For example, a query returning both concepts and instances would have a return type λ.

We introduce the notion of *typed query infrastructure* in Definition 20. The operators of this query infrastructure refine the generic operators presented in Section 3.3.1, Table 3.6. These new operators are distinguished from the generic ones using a subscript to denote the type of the result. For example, instead of the query $\cup$, the new operators $\cup_t$ and $\cup_{t_1}$ return respectively sets of elements of types t and t_1.

First, let us define formally what we mean by a typed query infrastructure:

Definition 20 (Typed Query Infrastructure) *Let E_λ be a set of identifiers and let T be a non-empty set of elements called* types. *Let $(E_t)_{t\in T}$ be a partition of E_λ (i.e. $E_\lambda = \bigcup_{t\in T} E_t$ and $\forall t_1, t_2 \in T$, with $t_1 \neq t_2$, $E_{t_1} \cap E_{t_2} = \emptyset$). Moreover, let λ be the generic type. The set $T_\lambda := T \cup \{\lambda\}$ is called the set of types.*

The following enumeration defines recursively families of typed query languages $(L_t)_{t\in T_\lambda}$ and $(L_{t_1 t_2})_{t_1,t_2\in T_\lambda}$:

1. ***set construction operator**: for $t \in T_\lambda$, $n \geq 1$, and $i_1, \dots, i_n \in E_t$, the expression "$\{i_1, \dots, i_n\}$" is a query belonging to L_t*

2. ***typed set operators**: for $t \in T_\lambda$, and q_1, $q_2 \in L_t$ and for all operators $\circ_t \in \{\cup_t, \cap_t, \dot{\cup}_t, \setminus_t\}$, the expression "$q_1 \circ_t q_2$" is a query of L_t*

3. ***cross-product operator**: for $t_1, t_2 \in T_\lambda$, $q_1 \in L_{t_1}$ and $q_2 \in L_{t_2}$,*
   - $q_1 \times_t q_2 \in L_{t_1 t_2}$

4. ***relation operators**: for $t_1, t_2 \in T_\lambda$, and $q_3, q_6 \in L_{t_1 t_2}$,*
   - $dom_{t_1}(q_3) \in L_{t_1}$
   - $range_{t_2}(q_3) \in L_{t_2}$
   - $q_3 \setminus_{t_1 t_2} q_6 \in L_{t_1 t_2}$
   - $q_3^c \in L_{t_1 t_2}$
   - $q_3^{-1} \in L_{t_2 t_1}$
   - *and if* $t_1 = t_2$, *then* $q_3^+ \in L_{t_1 t_1}$

5. ***role operators**: for $t_1, t_2 \in T_\lambda$, $q_1 \in L_{t_1}$, $q_3 \in L_{t_1 t_2}$, $q_6 \in L_{t_2 t_1}$ as well as $q_9 \in L_{t_1 t_1}$*
   - $q_3(q_1) \in L_{t_2}$
   - $q_6^{-1}(q_1) \in L_{t_1}$
   - $q_9^+(q_1) \in L_{t_2}$

6. ***join operators**: for $t_1, t_2 \in T_\lambda$, $q_1 \in L_{t_2}$, $q_3 \in L_{t_1 t_2}$ and $q_6 \in L_{t_2 t_3}$,*

[8]As noted in Chapter 2, Section 2.1.1 contexts could also be used as elements useful for manipulation. Unfortunately, considering the context as entities create some technical difficulties we could not address satisfactorily...

- $q_3 \bowtie_{q_1} q_6 \in L_{t_1 t_3}$
- $q_3 \bowtie q_6 \in L_{t_1 t_3}$

We define the two query languages $L_T := \bigcup_{t \in T_\lambda} L_t$ and $L_{TT} := \bigcup_{t_1,t_2 \in T_\lambda} L_{t_1 t_2}$. A typed query infrastructure *for the type set T is a query infrastructure of the form (L_T, L_{TT}).*

Example 9 *We present here a small example which illustrates the use of typed query-based multicontexts.*

We want to browse documents. For this, we create a file system query-based multicontext*. Let the type set be defined as $T := \{$F, D, L$\}$. F stands for file, D for directory and L for labels. The basic query language L_D consists of three operators:*

- *files $\in L_{D,F}$ which returns the binary relation between directories and the files they contain,*[9]
- *subdirectories $\in L_{D,D}$ which lists the direct (subdirectory,superdirectory) pairs, and*
- *labels $\in L_{FL}$ which lists the (file, label) pairs. Such a pair means that file is tagged using the label label.*

We now give a number of queries as examples.

1. *$\{$Formal Concept Analysis, Logic$\}$ $\in L_L$*
2. *labels^{-1}($\{$Formal Concept Analysis, Logic$\}$) $\in L_F$*
3. *files^{-1}(labels^{-1}(Formal Concept Analysis, Logic)) $\in L_D$*
4. *subdirectories^{-1}(files^{-1}(labels^{-1}($\{$Formal Concept Analysis, Logic$\}$)))$^+ \in L_D$*

Query 1 is an example of the set construction operator and returns a set of labels. Query 2 uses Query 1 which returns labels and returns the sets of files tagged with one of these labels and Query 3 and Query 4 return a set of directories. Note that there is no operators to obtain directly the labels of the directories.

Note that the set construction operator is defined using newly created predicate pred-set-x together with a set of ground atoms pred-set-x(i_1), ... pred-set-x(i_n).

We now see that these operators allow the definition of some interesting context index operators.

3.3.3 Context Index Operators

Different context operators have been proposed throughout the Formal Concept Analysis literature. In particular, the properties of some of these operators have been studied in detail in [Ganter and Wille, 1999]. In Chapter 2, Section 2.3.2, we recalled some of these operators: complementation, dual, apposition, subposition and direct sum. Some of these operators, for instance apposition, play

[9]A directory contains a file, but is not considered to contain the files of its subdirectories.

an important role in the conceptual models of Formal Concept Analysis applications [Stumme and Mädche, 2001, Hereth-Correira and Kaiser, 2004]. In this section, we introduce context index operators so that the realisation function κ of a given QBMC(Q) can be seen as a homomorphism from $\mathbb{P}$ to $QBMC(Q)$. Applications based on the query-based multicontext theory will be able to manipulate intensional representations of these context operators, profiting in this way from the advantages mentioned earlier: greater expressivity, flexible (remote, optimised or delegated) implementations.

Context Index Equivalence

Queries and context indices can be seen as strings representing their respective evaluation intensionally. In many cases, different intensional represention may be equivalent in the sense that they are known to represent the same result. To study the properties of context index operators, we formalise the notions of context index equality and context index equivalence in the following definition.

Definition 21 (Query and Context Index Equivalence) *We define two queries q_1 and q_2 are* equivalent with respect to an instance Q of a query infrastructure*, if $eval_1(q_1) = eval_1(q_2)$. Query equivalence is denoted by $q_1 \sim_Q q_2$.*

For an instance Q of a query infrastructure $\mathbb{Q}$, for $p_1 := (q_1, q_2, q_3)$ and $p_2 := (q_4, q_5, q_6)$ in $\mathbb{P}$, we say that context indices p_1 and p_2 are equivalent with respect to Q*, denoted by $p_1 \equiv_Q p_2$, if $\kappa(p_1) = \kappa(p_2)$ is true. In other words, the following equalities have to be valid:*

1. $q_1 \sim_Q q_4$
2. $q_2 \sim_Q q_5$ *and*
3. $(q_1 \times q_2) \cap q_3 \sim_Q (q_4 \times q_5) \cap q_6$.

We say that p_1 equals *p_2, denoted by $p_1 = p_2$ if $q_1 = q_4$, $q_2 = q_5$ and $q_3 = q_6$.*

These two relations are important because they are equivalence relations on queries, respectively on context indices. Note that when the instance of query infrastructure is clear, we may write the relations $\sim_Q$ and $\equiv_Q$ as $\sim$ and $\equiv$ respectively.

In order to simplify calculations, we introduce the functions ρ and $\overline{\rho}_Q$:

ρ: $\mathbb{P} \longrightarrow L_2$:
$p = (q_1, q_2, q_3) \mapsto \rho(p) = q_3 \cap (q_1 \times q_2)$ and
$\overline{\rho}_Q$: $\mathbb{P} \longrightarrow \Omega \times \Omega$
$p = (q_1, q_2, q_3) \mapsto \overline{\rho}_Q(p) = eval_2(q_3) \cap eval_2(q_1 \times q_2)$

ρ returns a query which returns exactly the incidence relation of the resulting formal context, while $\overline{\rho}$ returns the incidence relation of the induced context for a given instance of a query infrastructure:
This notation is also useful in the rest of this paragraph.

Lemma 1 *The relation $\sim_Q$ is an equivalence relation on queries and $\equiv_Q$ is an equivalence relation on context indices.*

Proof

Since set equality is an equivalence relation, we can immediately conclude that $\sim_Q$ is an equivalence relation on queries.

We now need to prove that $\equiv_Q$ is an equivalence relation on context indices. The definition of $\equiv_Q$ implies that it is an equivalence relation if and only if the components satisfy the properties of an equivalence relation. While it is trivial to see that these properties are valid for the two first components, the third one is more complex.

So if $p_1 \equiv p_1$, $\overline{\rho}_Q(p_1) = \overline{\rho}_Q(p_1)$. So $\equiv_Q$ is reflexive.

For p_1,$p_2 \in \mathbb{P}$, $p_1 \equiv p_2$ implies $\rho(p_1) \sim \rho(p_2)$. So $\overline{\rho}_Q(p_1)$=$\overline{\rho}_Q(p_2)$. Therefore $\equiv_Q$ is symmetric.

For p_1, p_2, $p_3 \in \mathbb{P}$, $p_1 \equiv p_2$ and $p_2 \equiv p_3$ imply $\rho(p_1) \sim \rho(p_2)$ and $\rho(p_2) \sim \rho(p_3)$. Since $\sim$ is transitive, then, $\equiv_Q$ is transitive and thus an equivalence relation. $\square$

The equivalence relation $\equiv_Q$ partitions the context index set $\mathbb{P}$ into a set of equivalence classes $\mathbb{P}_{/Q}$. Two context indices belong to the same equivalence class $[p]_Q$ if and only if their realisation maps to the same context. In the following paragraph, we define context index operators which preserve the equivalence relation $\equiv_Q$.

Definition 22 (Equivalence Preservation) *Let Q be an instance of a query infrastructure. A unary operator f on* $\mathbb{P}$ preserves the equivalence relation $\equiv_Q$ *if and only if* $\forall p_1, p_2 \in \mathbb{P}$, $p_1 \equiv p_2$ *implies* $\kappa(f(p_1)) = \kappa(f(p_2))$.

A binary operator g on $\mathbb{P}$ preserves the equivalence relation $\equiv_Q$, *if and only if* $\forall p_1, p_2, p_3, p_4 \in \mathbb{P}$, $p_1 \equiv p_2$, $p_3 \equiv p_4$ *implies* $\kappa(g(p_1, p_3)) = \kappa(g(p_2, p_4))$.

An operator which preserves the equivalence relation $\equiv_Q$ *can be also called* equivalence preserving for $\equiv_Q$.

Context Index Operators

We define the following context index operators:

Definition 23 *For* $p_1 := (q_1, q_2, q_3)$ *and* $p_2 := (q_4, q_5, q_6)$ *in* $\mathbb{P}$, *we define the following operators:*

- dual *(or sometimes called transpose):* $p_1^d := (q_2, q_1, q_3^{-1})$
- complement*:* $p_1^c := (q_1, q_2, (q_1 \times q_2) \backslash q_3)$
- apposition*: if* $q_1 \sim_Q q_4$ *then the apposition of* p_1 *and* p_2 *is defined by* $p_1 | p_2 = (q_1, q_2 \dot{\cup} q_5, q_3 \overset{t.}{\cup} q_6)$.
- subposition*: if* $q_2 \sim_Q q_5$ *then the subposition of* p_1 *and* p_2 *is defined by* $\frac{p_1}{p_2} = (q_1 \dot{\cup} q_4, q_2, q_3 \overset{.t}{\cup} q_6)$.
- union*:* $p_1 \cup p_2 := (q_1 \cup q_4, q_2 \cup q_5, ((q_3 \cap (q_1 \times q_2)) \cup (q_6 \cap (q_4 \times q_5)))$
- intersection*:* $p_1 \cap p_2 := (q_1 \cap q_4, q_2 \cap q_5, q_3 \cap q_6)$.

Most of these operators are simple since they each correspond to some of the formal context operators presented in Chapter 2, Section 2.3.2. We show in a theorem in this section, that these operators can be used to manipulate formal contexts by applying the context index operators instead of the formal context operators.

To help the reader understand the proof of the theorem, we now describe the goal operations performed by these operators. The dual operator first swaps the object set and attribute set, then transposes the relation. The dual context index of (Person, Publication, isAuthorOf) is (Publication, Person, isAuthorOf^{-1}). Using the complement operator on the same context instead creates (Person, Publication, isAuthorOf^{c}), that is, the object and attribute sets remain the same, but a person is linked to a publication if and only if it is not linked to that publication through isAuthorOf.

If the object sets are the same, then the apposition operator is best understood as putting the two contexts side by side. If the attributes are seen as criteria of classification, then apposition corresponds to adding criteria. Similarly, the subposition operator can be seen as putting two contexts on top of each other in such a way that it corresponds to the merging of two separate views on the data. In Section 3.5, we show how the subposition is used to create hierarchies combining instances with their classification hierarchy. The union operator merges two contexts (i.e. the realised context corresponds to the merging of the corresponding components of the formal contexts), while the intersection operator only keeps the elements present in both contexts.

We illustrate this situation using Figure 3.4. Intuitively, the union of the relations of the two contexts $\mathbb{K}_{p_1}$ and $\mathbb{K}_{p_2}$ corresponds to the plain colored space. However, the union of the results of the relation q_3 and q_6 with the cross product of $(q_1 \cup q_4)$ and $(q_2 \cup q_5)$ may contain pairs which do not belong to the relation of the union of the incidence relations of the two contexts $\mathbb{K}_{p_1}$ and $\mathbb{K}_{p_2}$. The squares marked with an $\emptyset$ should be empty in the merged context. But query q_3 for example could return a pair (a, b) so that $a \in eval_1(q_4) \backslash eval_1(q_1)$ and $b \in eval_1(q_2)$. To avoid this, the union operator requires the supplementary constraints: $\cap(q_1 \times q_4)$ and $\cap(q_2 \times q_5)$ ensuring that the acceptable pairs must occur in the plain colored space of Figure 3.4.

We now introduce a theorem showing that these operators correspond to the operators of [Ganter and Wille, 1999].

Theorem 2 (Basic Context Index Operator Properties) *The context index operators introduced in Definition 23 are equivalence preserving for the equivalence relation $\equiv_Q$. Moreover, these context index operators return the formal context resulting from the context operator of the same name presented in Chapter 2, Section 2.3.2 and which originate from [Ganter and Wille, 1999].*

Moreover, the following properties can be proved:

1. $p_1^{dd} \equiv p_1$
2. $p_1^{cc} \equiv p_1$
3. $p_1^{cd} \equiv p_1^{dc}$

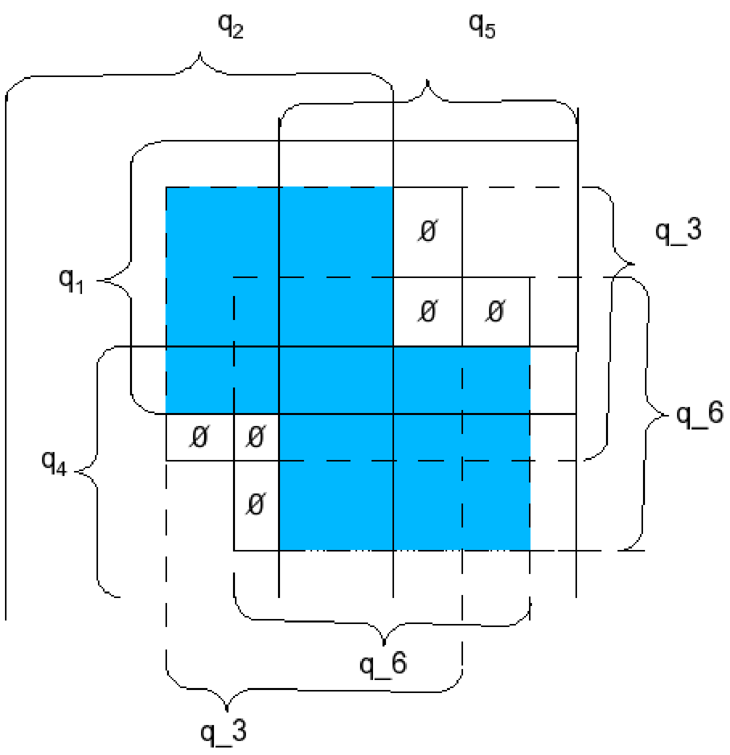


Figure 3.4: Intuitive diagram for the union context index operator.

Proof

In the following, we first prove that the results of each of these operator returns a context index which when evaluated return the same result as the corresponding operation on the results of the evaluation of the operands. The operators on formal contexts were defined in Chapter 2, Section 2.3.2. Note that in most proofs when we need to prove a statement of the form: $\kappa(p_1) = \kappa(p_2)$, we consider each argument separately. We first prove that the first two arguments of these contexts are equal, then we prove the equality of the relations. The first two proofs are exceptions, since the proofs are very simple.

- **dual:**

  We need to prove: $\kappa(p_1^d) = \kappa(p_1)^d$.

  Recall that $p_1^d = (q_2, q_1, q_3^{-1})$. It is straightforward to see that the first components get swapped.

$$\begin{aligned}
\kappa(p_1^d) &= (eval_1(q_2), eval_1(q_1), eval_2(q_3^{-1}) \cap eval_2(q_2 \times q_1)) \\
&= (eval_1(q_2), eval_1(q_1), eval_2(q_3)^{-1} \cap eval_2(q_2 \times q_1)) \\
&= (eval_1(q_2), eval_1(q_1), eval_2(q_3)^{-1} \cap (eval_1(q_2) \times eval_1(q_1))) \\
&= \kappa(p_1)^d
\end{aligned}$$

  The incidence relations are also equal since for any sets A, B and any relation I, $((A \times B) \cap I)^{-1} = ((B \times A) \cap I^{-1})$.

Let p_1 and p_2 be two context indices so that $p_1 \equiv p_2$. We know that $\kappa(p_1) = \kappa(p_2)$, and $\kappa(p_1)^d{=}\kappa(p_1^d)$, but also $\kappa(p_1)^d = \kappa(p_2)^d$ and finally $\kappa(p_2)^d = \kappa(p_2)^d$. We can conclude that the dual operator is equivalence preserving.

- **complement:**

  We need to prove that $\mathbb{K}^c_{p_1}$ and $\mathbb{K}_{p_1^c}$ are equal.

  $$\begin{array}{rcl} \kappa(p_1^c) & = & (eval_1(q_1), eval_1(q_2), eval_2(q_1 \times q_2 \backslash q_3) \cap eval_2(q_1 \times q_2)) \\ & = & (eval_1(q_1), eval_1(q_2), (eval_2(q_1 \times q_2) \backslash eval_2(q_3)) \cap eval_2(q_1 \times q_2)) \end{array}$$

  Since for all set A and B, $(A\backslash B)\backslash A = A\backslash B$, the two contexts $\mathbb{K}^c_{p_1}$ and $\mathbb{K}_{p_1^c}$ are equal.

  To prove the equivalence preserving property of the complement operator, we use the same technique as for the dual.

  Let p_1 and p_2 be two context indices so that $p_1 \equiv p_2$. We know that $\kappa(p_1) = \kappa(p_2)$, and $\kappa(p_1)^c{=}\kappa(p_1^c)$, but also $\kappa(p_1)^c = \kappa(p_2)^c$ and finally $\kappa(p_2)^c = \kappa(p_2)^c$. We can conclude that the complement operator is equivalence preserving.

- **apposition:** We need to prove: $\kappa(p_1|p_2) = \kappa(p_1)|\kappa(p_2)$. We prove it component wise. According to the definition, the apposition is defined only if $q_1 \sim_Q q_4$, so the first components of p_1 and p_2 are equivalent. Moreover, all occurrences of q_4 can be replaced by q_1. Due to the definition of the set disjunction $\overset{t}{\cup}$, the second components of both contexts are also equal, i.e. $eval_1(q_2 \overset{t}{\cup} q_5) = \overset{t}{eval_1}(q_2) \cup \overset{t^-}{eval_1}(q_5)$.

The third component is more complex. We have to prove:

$$\begin{array}{rcl} \overline{\rho}_Q(p1|p2) & \overset{?}{=} & \overset{t.}{\overline{\rho}_Q}(p_1) \cup \overset{t^-.}{\overline{\rho}_Q}(p_2), \text{ i.e.} \\ \overline{\rho}_Q(p1|p2) & \overset{?}{=} & (\overset{t.}{eval_2}(q_3) \cap (eval_1(q_1) \times \overset{t}{eval_1}(q_2))) \cup \\ & & ((\overset{t^-.}{eval_2}(q_6) \cap (eval_1(q_1)) \times \overset{t^-.}{eval_1}(q_5))) \\ \overline{\rho}_Q(p1|p2) & \overset{?}{=} & (\overset{t.}{eval_2}(q_3 \cap (eval_2(q_1 \times q_2))) \cup (\overset{t^-.}{eval_2}(q_6 \ \cap (q_1 \times q_5))) \end{array}$$

We start with $\overline{\rho}_Q(p1|p2)$:

$$\begin{array}{rcl} \overline{\rho}_Q(p1|p2) & = & (eval_2(q_3 \overset{t.}{\cup} q_6)) \cap eval_2(q_1 \times (q_2 \overset{t}{\cup} q_5)) \\ & = & ((\overset{t.}{eval_2}(q_3) \cup \overset{t^-.}{eval_2}(q_6)) \cap eval_2(q_1 \times (q_2 \overset{t}{\cup} q_5)) \\ & = & ((\overset{t.}{eval_2}(q_3) \cup \overset{t^-.}{eval_2}(q_6)) \cap eval_1(q_1) \times eval_1(\overset{t}{q_2} \cup \overset{t^-}{q_5})) \end{array}$$

Since $A \times (\overset{t}{B} \cup \overset{t^-}{C}) = (A \times \overset{t}{B}) \cup (A \times \overset{t^-}{C})$ for any sets A, B and C, we obtain:

$$\begin{aligned}
\overline{\rho}_Q(p_1|p_2) &= (\overset{t.}{eval_2}(q_3) \cup \overset{t^-.}{eval_2}(q_6)) \cap \\
&\quad ((eval_1(q_1) \times \overset{t}{eval_1}(q_2)) \cup (eval_1(q_1) \times \overset{t^-}{eval_1}(q_5))) \\
\overline{\rho}_Q(p_1|p_2) &= (\overset{t.}{eval_2}(q_3) \cup \overset{t^-.}{eval_2}(q_6)) \cap (\overset{t.}{eval_2}(q_1 \times q_2) \cup \overset{t^-.}{eval_2}(q_1 \times q_5)) \\
\overline{\rho}_Q(p_1|p_2) &= (\overset{t.}{eval_2}(q_3) \cap (\overset{t.}{eval_2}(q_1 \times q_2))) \cup (\overset{t^-.}{eval_2}(q_6) \ \cap (\overset{t^-.}{eval_2}(q_1 \times q_5))) \\
\overline{\rho}_Q(p_1|p_2) &= (\overset{t.}{eval_2}(q_3 \cap (eval_2(q_1 \times q_2))) \cup (\overset{t^-.}{eval_2}(q_6 \ \cap (q_1 \times q_5))) \\
\overline{\rho}_Q(p_1|p_2) &= \overline{\rho}_Q(p_1) \overset{t.}{\cup} \overline{\rho}_Q(p_2)
\end{aligned}$$

So for the third component, the evaluation of the apposed context relation is equal to the disjoint union of the two contexts.

- **subposition:** The subposition operation is the dual operation of the apposition, the proof differs only in taking the first component instead of the second.

- **union:** We must now demonstrate that each of the components of the two contexts $\kappa(p_1 \cup p_2)$ and $\kappa(p_1) \cup \kappa(p_2)$ are equal.

  It is clear that the first two components of both contexts are the union of the results of the corresponding components from the two contexts. We discussed using Figure 3.4 that the union operator must ensure the abscence of all the pairs of elements which are absent of the two formal contexts. In order to check this, we consider the different possible cases. We consider pairs $(a, b) \in \Omega \times \Omega$.

  1. Let $(a, b) \in eval_2(q_3)$, $a \in eval_1(q_1)$ and $b \in eval_1(q_2)$ then $(a, b) \in \mathbb{K}_{p_1}$ therefore $(a, b) \in eval_2(q_3 \cap (q_1 \times q_2))$ and also $(a, b) \in \overline{\rho}_Q(p_1 \cup p_2)$.
  2. Similarly, for (a, b) such that $(a, b) \in eval_2(q_6)$, $a \in eval_1(q_4)$ and $b \in eval_1(q_5)$, the pair (a, b) belongs to both $\overline{\rho}_Q(p_1 \cup p_2)$ and $\overline{\rho}_Q(p_1) \cup \overline{\rho}_Q(p_2)$.
  3. Let $(a, b) \in eval_2(q_3)$, $a \in eval_1(q_4) \backslash eval_1(q_1)$ then $(a, b) \notin eval_2(q_3 \cap (q_1 \times q_2))$ and from the definition of the union, we can also conclude that $(a, b) \notin \overline{\rho}_Q(p_1 \cup p_2)$ and that $(a, b) \notin \overline{\rho}_Q(p_1) \cup \overline{\rho}_Q(p_2)$. This implies that (a, b) is neither in the incidence relation from $\mathbb{K}_{p_1} \cup \mathbb{K}_{p_2}$ nor the one from $\mathbb{K}_{p_1 \cup p_2}$.
  4. Similarly, for (a, b) such that $a \in eval_1(q_1) \backslash eval_1(q_4)$ and $(a, b) \in eval_2(q_6)$, we can conclude that (a, b) neither in the incidence relation from $\mathbb{K}_{p_1} \cup \mathbb{K}_{p_2}$ nor the one from $\mathbb{K}_{p_1 \cup p_2}$.
  5. If $(a, b) \notin eval_2(q_1 \times q_2) \cup eval_2(q_4 \times q_5)$, then $(a, b) \notin \overline{\rho}_Q(p_1) \cup \overline{\rho}_Q(p_2)$ and $(a, b) \notin \overline{\rho}_Q(p_1 \cup p_2)$.

  Considered as a whole, these diverse cases imply that $(a, b) \in \overline{\rho}_Q(p_1) \cup \overline{\rho}_Q(p_2)$ if and only if $(a, b) \in \overline{\rho}_Q(p_1 \cup p_2)$. This enables us to conclude that $\kappa(p_1 \cup p_2) = \kappa(p_1) \cup \kappa(p_2)$.

- **intersection:** Since per definition $eval_1(q_1 \cap q_4) = eval_1(q_1) \cap eval_1(q_4)$, the first two components are equivalent.

We now need to prove $\overline{\rho}_Q(p_1 \cap p_2) = \overline{\rho}_Q(p_1) \cap \overline{\rho}_Q(p_2)$. Let (a, b) be a pair of $\overline{\rho}_Q(p_1 \cap p_2)$. We must prove that it belongs to $\overline{\rho}_Q(p_1) \cap \overline{\rho}_Q(p_2)$. Then: $a \in eval_1(q_1)$ and $a \in eval_1(q_4)$, $b \in eval_1(q_2)$ and $b \in eval_1(q_5)$. So $(a, b) \in eval_1(q_1) \times eval_1(q_2)$ and $(a, b) \in eval_1(q_4) \times eval_1(q_5)$. And of course, $(a, b) \in eval_2(q_3)$ and $(a, b) \in eval_2(q_6)$. Therefore both $(a, b) \in eval_2((q_1 \times q_2) \cap q_3)$ and $(a, b) \in eval_2((q_4 \times q_5) \cap q_6)$ are valid. This means that $\overline{\rho}_Q(p_1 \cap p_2) \subseteq \overline{\rho}_Q(p_1) \cap \overline{\rho}_Q(p_2)$ is also true.

Let (a, b) be a pair of $\overline{\rho}_Q(p_1) \cap \overline{\rho}_Q(p_2)$. Then $(a, b) \in eval_2(q_3 \cap q_6)$, $a \in eval_1(q_1 \cap q_4)$ and $b \in eval_1(q_2 \cap q_5)$ are valid. The last two statements imply that $(a, b) \in eval_2((q_1 \cap q_4) \times (q_2 \cap q_5))$. Finally, we can infer that $(a, b) \in eval_2((q_3 \cap q_6) \cap ((q_1 \cap q_4) \times (q_2 \cap q_5)))$.

In other words, $(a, b) \in \overline{\rho}_Q(p_1 \cap p_2)$. Finally we can conclude that $\mathbb{K}_{p_1} \cap \mathbb{K}_{p_2} = \mathbb{K}_{p_1 \cap p_2}$.

We have proved the first part of the theorem, and we turn to the second part:

- To see that $p_1^{dd} \equiv p_1$, we note that $p_1^d = (q_2, q_1, q_3^{-1})$, and obtain in the same way $p_1^{dd} = (q_1, q_2, (q_3^{-1})^{-1})$. The result of the operator $.^{-1}$ on queries of L_2 swaps the components of the results. If the same operator is used twice, then the components go back to their original positions. In other words, $p_1^{dd} \equiv (q_2, q_1, q_3) = p_1$.

- The proof of the property: $p_1^{cc} \equiv p_1$ is similar but more complex.

  We have: $p_1^c = (q_1, q_2, (q_1 \times q_2) \setminus q_3)$. Then
  $p_1^{cc} = (q_1, q_2, (q_1 \times q_2) \setminus ((q_1 \times q_2) \setminus q_3))$.

  We need to prove $p_1^{cc} \equiv_Q p_1$, in other words, we need to demonstrate: $\kappa(p_1^{cc}) = \kappa(p_1)$ (and not $p_1^{cc} = p_1$ which is not true in general).

  The first two components are equal. Moreover, we know that
  $eval_2(q_1 \times q_2) = (eval_2(q_3) \cap eval_2(q_1 \times q_2)) \cup (eval_2((q_1 \times q_2) \setminus q_3) \cap eval_2(q_1 \times q_2))$
  $eval_2(q_1 \times q_2) = (eval_2(q_3) \cup eval_2((q_1 \times q_2) \setminus q_3)) \cap eval_2(q_1 \times q_2)$

  Therefore, when evaluating the third component of p_1^{cc}, we obtain:
  $eval_2((q_1 \times q_2) \setminus ((q_1 \times q_2) \setminus q_3)) \cap eval_2(q_1 \times q_2)=$
  $(eval_2(q_1 \times q_2) \cap eval_2(q_1 \times q_2)) \setminus (eval_2((q_1 \times q_2) \setminus q_3) \cap eval_2(q_1 \times q_2))$

  Since for a set A, $A \cap A = A$, the first part is simplified, and for the second part, we know that: $eval_2((q_1 \times q_2) \setminus q_3)= eval_2(q_1 \times q_2) \setminus eval_2(q_3)$

  Therefore: $eval_2((q_1 \times q_2) \setminus ((q_1 \times q_2) \setminus q_3)) \cap eval_2(q_1 \times q_2)=$
  $eval_2(q_1 \times q_2) \setminus ((eval_2(q_1 \times q_2) \cap eval_2(q_1 \times q_2)) \setminus eval_2(q_3) \cap eval_2(q_1 \times q_2))$
  Again we use $A \cap A = A$:
  $eval_2((q_1 \times q_2) \setminus ((q_1 \times q_2) \setminus q_3)) \cap eval_2(q_1 \times q_2)=$
  $eval_2(q_1 \times q_2) \setminus ((eval_2(q_1 \times q_2)) \setminus (eval_2(q_3) \cap eval_2(q_1 \times q_2)))$
  Since for any set A and B, the following equality is valid:[10]
  $A \setminus (A \setminus B)= B \cap A$. $eval_2((q_1 \times q_2) \setminus ((q_1 \times q_2) \setminus q_3)) \cap eval_2(q_1 \times q_2)=$
  $eval_2(q_1 \times q_2) \cap eval_2(q_3) \cap eval_2(q_1 \times q_2)))$

[10]It is easy to see using a Venn diagram for the two cases $A \cap B = \emptyset$ and $A \cap B\neg = \emptyset$.

$$eval_2((q_1 \times q_2) \setminus ((q_1 \times q_2) \setminus q_3)) \cap eval_2(q_1 \times q_2) = eval_2(q_1 \times q_2) \cap eval_2(q_3)$$

This last quality shows that $p_1^{cc} \equiv (q_1, q_2, q_3) = p_1$.

- Finally we need to prove: $p_1^{cd} \equiv p_1^{dc}$.

  This means that we have to prove: $\kappa(p_1^{cd}) = \kappa(p_1^{dc})$.

  $\kappa(p_1^{cd}) = \kappa(p_1^{c})^{d} = \kappa(p_1)^{cd}$

  Since for every context $\mathbb{K}$, we know that $\mathbb{K}^{cd} = \mathbb{K}^{dc}$, we obtain:
  $\kappa(p_1^{cd}) = \kappa(p_1)^{dc} = \kappa(p_1^{d})^{c} = \kappa(p_1^{dc})$

  So we can conclude $p_1^{cd} \equiv_Q p_1^{dc}$.

□

Advantages

The introduction of the context index operators has practical advantages. These advantages are of three kinds:

- the language used to define the contexts is more expressive and corresponds to usual tasks, for instance merging data from two views,
- the evaluation of these operators can be optimised, and
- new data structures can be coupled with the intensional description of these operators.

During the context definition process, the use of operators can greatly ease the work of the user. A typical example of this is given by the dual operator (sometimes also called the *transpose* operator). The transposition of a context is equivalent to taking the dual of the context. By applying the transpose operator on a given context index, the user creates the transposition of a context he had already defined. Since the basic definitions of Formal Concept Analysis are dual (i.e. the role of the objects and attributes can be swapped), the lattice obtained from the transposed context is the dual concept lattice.

The intensional description can also be used to support a better context management system. For instance, the dual and complement contexts can be implemented by using a facade mechanism (see [Gamma et al., 1995]) instead of performing a new access to the data. For the dual context, attribute and object sets swap and the relation is transposed, the complement operator is implemented using a logical operator inverting the result of the relation of the inverted context. In this case, only one of the two contexts needs to be kept in memory.[11] These approaches are described in Chapter 5, Section 5.3.3.

Even though it is possible to implement naively these operators, it might be preferable in some cases to use databases and their related theory. From database theory we know a great number of optimisation techniques. These optimisation techniques can be used to implement the context index operators.

[11]Depending on the actual implementation of the context, the transposition might have an effect on the performance of the lattice layout algorithm. We comment on this in the Section on the operator implementation in Chapter 5, Section 5.3.3.

The intensional description can then be used to trigger an alternative optimisation process or to delegate the evaluation to remote implementations. Chapter 5, Section 5.3.3 describes how these operators can be evaluated using these strategies.

Table 3.8: Table stating the main use of the context index operators.

Operator	Use
dual	transposition, inverse visualisation, lattice creation optimisation
complement	complement context visualisation
apposition	supplementary criteria, scaling
subposition	partitioned subgroups
union	merging
intersection	focusing

3.4 Semantic Query-Based Multicontext

Until now the query-based multicontext model has been kept generic to allow several extensions. One of the main goals of this thesis is to show that it is possible to develop a novel knowledge base exploration paradigm using a query-based multicontext architecture. To achieve this goal, we introduce a new query infrastructure, the *semantic query infrastructure*, which provides the necessary query operators to interact with a knowledge base. Before we describe our semantic query language, we present the generic model for ontologies and knowledge bases which is used to abstract from the actual knowledge base language of the data sources.

3.4.1 Generic Ontology Model

In Chapter 2, Section 2.1.1, we introduced different knowledge representation languages used in the context of the Semantic Web. In this section, we discuss the model we follow to allow the visualisation of the content of a knowledge base.

This model is designed as a generic way of integrating ontologies and as been presented in a similar form in [E. Bozsak et al., 2002] though the notations have been adapted to our purpose. It is based on a partition of the elements which play a role in the modelling of the knowledge base. The partition we propose corresponds to the distinction we presented in Chapter 2, Section 2.1.1 except that we do not consider here the contextual relations. The use of contextual relations is out of the scope of this thesis because their introduction generates some issues which remain open problems. We discuss briefly this topic in the future work section at the end of this thesis (see Chapter 7, Section 7.2.1). In the following presentation, we only concentrate on the first four aspects we described.

Just as the description logic ontologies are usually divided between a T-box and an A-box, the model we propose here is divided into two layers: an *ontological layer* and a *fact layer*. We begin our presentation by giving a definition of the ontology layer.

Ontology Layer

Definition 24 (Ontology Layer (see [E. Bozsak et al., 2002])) *We define an* ontology *as a tuple $Onto = (\mathcal{C}, \leq_C, \mathcal{R}, \leq_R, \sigma)$ where $(\mathcal{C}, \leq_C)$ and $(\mathcal{R}, \leq_R)$ are partially ordered sets and where σ, called the* signature function*, is a mapping from $\mathcal{R}$ to tuples over $\mathcal{C}$, in other words $\sigma : \mathcal{R} \longrightarrow \mathcal{C}^+$: $r \mapsto \sigma(r) = (c_1, \ldots, c_i)$ for some $i \in \mathbb{N}, i > 0$.*

The elements of $\mathcal{C}$ are called concepts*, while the elements of $\mathcal{R}$ are called* relations*.*

Depending on the ontology language chosen, different kinds of elements are allowed in $\mathcal{C}$ and $\mathcal{R}$. For example, if the ontology language used is RDF(S), the elements of $\mathcal{C}$ are the concept names occurring in the vocabulary of the ontology. In an OWL ontology, the situation is more complex due to the use of non atomic concepts and relations. For example, the more expressive description logic $\mathcal{SHOIN}$ (see Chapter 2, Section 2.2.1) allows building of concept expressions using negation, union and intersection of concepts. Moreover, some concepts may also be defined extensionally using nominals (i.e extensional list of instances of this concept).

In order to ensure compatibility with different frameworks, we chose to keep only named concepts, since query infrastructure provides to the user the possibility to create more complex queries using the operators of the query infrastructure of the query-based multicontext.

In Section 3.3, we introduced a number of operators on queries which are very similar to the description logics operators. There are mainly two purpose to this introduction. First, it allows us to abstract from the operators available in the description logic. This means that we could implement our framework on top of RDF(S) while enabling supplementary modelling primitives for querying such as union, intersection or role composition and nominals. Note that the semantics of some query operators are not identical with the semantics of the description logics language. For instance, the transitive axiom used in description logic and the transitive query operator do not correspond completely. Combining the transitive axiom with existential axioms implies an infinite model, while the computation of the transitive closure of a relation can be determined completely.

On the other hand, the use of the generic ontology model also implies that to use the ontology framework of the data source efficiently, queries should be mapped to expressive concepts of the knowledge base. In order to ensure this, we consider that the concept set $\mathcal{C}$ and the relation set $\mathcal{R}$ correspond to the named concept and named roles of the $T - box$. Note that qualified number expressions, negation and value restriction are not expressible using the generic query operators. These could be easily emulated by using new names for the corresponding concepts and axioms of the form

$$\leq 45 \mathsf{publication.Publication} \sqsubseteq \mathsf{ExperiencedAuthor}$$

Note that the difference operator $\setminus$ corresponds to a negation by failure. We believe that for the purpose of exploration the negation as failure is more appropriate because it allows to focus on individuals which are not known to have some property though they may potentially have one. Another alternative would be to consider the classical negation in logic.

Fact Layer

The fact layer of our semantic infrastructure is straightforward in comparison to the ontology layer. The choice of the elements is much more restricted since we only allow two different types: *instances* and *relation instances* which naturally correspond to the individuals and the pairs of individuals (a,b) of the domain Δ of the interpretation (see Chapter 2, Section 2.2.1). However, it should be noted that using the typed query infrastructure presented in Section 3.3.2 makes it possible to further divide the sets of instances into individuals and concrete objects. Since we did not introduce special query operators for concrete objects, both the individuals and the concrete objects are seen as instances.

Definition 25 (Fact Base (see [E. Bozsak et al., 2002])) *A fact base FB for an ontology $Onto := (\mathcal{C}, \leq_C, \mathcal{R}, \leq_R, \sigma)$ is a tuple $(I_\mathcal{C}, \iota_C, I_\mathcal{R}, \iota_R)$ so that:*

- $I_\mathcal{C}$ *is a set of elements called* instances*,*
- $I_\mathcal{R}$ *is a set of elements called* relation instances*,*
- ι_C *is a mapping from $\mathcal{C}$ to $\mathcal{P}(I_\mathcal{C})$,*
- ι_R *is a mapping from $\mathcal{R}$ into $\mathcal{P}(I_\mathcal{R}^+)$ such that:*
  $\forall r\ \iota_R(r) \subseteq \prod_{i=1}^{i=|\sigma(r)|} \iota_C(\pi(i, \sigma(r))$ *where*
  - $|\sigma(r)|$ *is the length of the tuple $\sigma(r)$ and*
  - *for all tuples, $\pi(i,t)$ returns the i-th component of t*

Moreover the following conditions must be valid:
$\forall c, c_2 \in \mathcal{C},\ c \leq_C c_2 \Rightarrow \iota_C(c) \subseteq \iota_C(c_2)$*, and*
$\forall r, r_2 \in \mathcal{R},\ r \leq_R r_2 \Rightarrow \iota_R(r) \subseteq \iota_R(r_2)$.

We can now define what we mean by a knowledge base:

Definition 26 (Knowledge Base) *A* knowledge base *is a pair $KB = (O, FB_O)$, where O is an ontology and where FB is a fact base for the ontology O.*

Using this generic ontology model, we define query infrastructures for knowledge bases in the following section.

3.4.2 Semantic Query Infrastructures

One of the goals of this thesis is to show that the combination of knowledge bases and a query-based multicontext based browser creates an interesting framework to manipulate data. The details about the browsing mechanism are given in Chapter 4. In this section, we deal with the language needed to combine knowledge bases with query-based multicontexts.

In order to use knowledge bases with query-based multicontexts, we need to define an ontology-based query infrastructure. This query infrastructure may be implemented and deployed for different knowledge bases, each deployed implementation is an instance of this query infrastructure as stated in Definition 17.

In Chapter 2, Section 2.1.2, we presented an overview of some of the proposed languages. The semantic query infrastructures we introduce in this chapter can be seen as a new kind of semantic query language. There are two reasons behind our choice of using our own language. First of all, each query language restricts itself to certain primitives. Some primitives might not be useful in the query-based multicontext, while others might be needed. Using our own query language allows us to use the query-based multicontext with different query languages and ontology frameworks. The second reason for creating a semantic query language special for the query-based multicontext lies in the type of queries which can be defined using the graphical user interface of our tool. Using our own language allows us to reduce the size of the queries needed. In Chapter 5, Section 5.4.2 we describe a way to adapt existing ontology infrastructures to the semantic query infrastructures defined in this section.

Definition 27 (Ontology Query Infrastructure) *An* ontology query infrastructure $\mathbb{Q}_{onto}$*:=(*L_{Onto1}*,*L_{Onto2}*)*[12] *is a typed query infrastructure with two type elements:*[13] $\{C, R\}$ *(which stand respectively for concepts and relations). In addition to the typed generic operators, a number of supplementary operators are defined.*

The following query constructions can be used to create queries in L_C *(the set of queries returning concepts):*

- *subconcepts*(q_C)*, which returns the subconcepts*[14] *of* q_C
- *superconcepts*(q_C)*, which returns the superconcepts of* q_C
- *related*(q_C) *returns the concepts which are related through some relation to a concept of* q_C
- *related*(i, q_R, q_C) *returns the concepts which are subsumed by the domain of the signature of a relation of* q_R *and for which the range of the relation subsumes a concept in* q_C *at the position* i

The following query constructions can be used to create queries in L_R *(the set of queries returning relations).*

- *subrelations*(q_R)*, which returns the subrelations of* q_R
- *superrelations*(q_R)*, which returns the superrelations of* q_R
- *relations*(q_C) *the relations which have some concept of* q_C *or some subconcept of a concept of* q_C *in their signature*
- *relations*(q_{I1}, q_{I2}) *the relations which have instances of* q_{I1} *in the domain of their extensions and instance of* q_{I2} *in the range of thei evaluation*

The following query constructions can be used to create queries in L_{CC} *(the sets of queries returning pairs of concepts).*

- *subconcepts which returns the subconcepts/superconcept pairs of the concept subsumption hierarchy*

[12]L_{Onto1} is the L_1 and L_{Onto2} is the L_2 of the query-based multicontext definition.
[13]This means that $L_{Onto1} = L_C \cup L_R$ and $L_{Onto2} = L_{CC} \cup L_{RR} \cup L_{CR} \cup L_{RC}$
[14]This return all the subconcepts, not only the direct subconcepts.

- concepts(q_R, j, k) *return the pairs of concepts at position j and k of the signature of some relation of q_R (sometimes abbreviated* concepts(q_R) *for $j = 1$ and $k = 2$)*

The following query construction can be used to create queries in L_{RR} (the sets of queries returning pairs of relations).

- subrelation *which the relation subsumption hierarchy*

This query infrastructure only allows to browse the ontology model. There is absolutely no possibility to access the instances. In order to query the fact base, we finally introduce the *knowledge base query infrastructure* $\mathbb{Q}_{kb}$:=(L_{Kb},L_{KbKb}):

Definition 28 (Knowledge Base Query Infrastructure) *A knowledge base query infrastructure $\mathbb{Q}_{Kb}$:=(L_{Kb},L_{KbKb}) is an ontology query infrastructure, with two supplementary type elements:*[15] *$\{I, Ri\}$ (which stand respectively for instances and relation instances), and with supplementary operators:*

The query language L_{CI} has been extended with the following query construction: (the queries returning pairs (c, i) consisting of a concept c and an instance i).

- instantiation *which returns the pairs of the instantiation relation*

The query language L_C have been extended with the following query construction:

- concepts(q_I) *which returns the concepts which have at least one instance of q_I.*

The query language L_I has been extended with the following query constructions:

- inst(q_C)*(sometimes abbreviated I(q_C)) which returns all the instances of the concepts q_C.*
- role(j, q_R, k, q_C) *(sometimes abbreviated role(q_R,q_C) for $j = 1$ and $k = 2$) which returns the instances at position j of relation instances of q_R where at position k is an instance of concept q_C.*

L_{Ri} has been extended with the following query construction:

- relinstances(q_R) *which returns all the instances of the relations of q_R.*

L_{II} has been extended with the following query construction:

- $< q_R >_{jk}$ *return the pairs of instances of some relation instance of q_R at positions j and k respectively.*[16]
- $< q_R >$ *return the pairs of instances of some relation instance of q_R at positions 1 and 2 respectively.*[17]

[15]This means that $L_{Kb} = L_{Onto1} \cup L_I \cup L_{Ri}$ and $L_{KbKb} = L_{Onto2} \cup L_{CI} \cup L_{CRi} \cup L_{RI} \cup L_{RRi} \cup L_{II} \cup L_{RiRi} \cup L_{IRi} \cup L_{RiI} \cup L_{IC} \cup L_{IR} \cup L_{RiC} \cup L_{RiR}$.

[16]This notation is the generic method to obtain pairs of instances at position j and k of a relation instance tuple.

[17]This construction is a specific case of the preceeding one, but it simplifies the notation for relations having only two components.

We can now define *ontology query-based multicontexts* and *knowledge base query-based multicontexts* :

Definition 29 (Semantic Query Structures) *We call* semantic query infrastructure *a query infrastructure which is either an ontology query infrastructure or a knowledge base query infrastructure.*

We call a semantic query-based multicontext *a query-based multicontext where the underlying query infrastructure is an instance of a semantic query infrastructure.*

Moreover, the use of formulas in a first order logic language would have required the introduction of variables. Some formalisms like description logics as presented in Chapter 2, Section 2.2.1 also avoid variables. The syntax of the query languages is directly inspired from these logics. The absence of variables corresponds also to the unnamed notation paradigm of relational databases.

3.5 Constructors

While designing the theory of the query-based multicontext, we noticed that defining useful context indices would be too cumbersome and unintuitive for most users. In order to remedy this problem, we introduce the new notion of *constructors*. Constructors ease tremendously the task of defining context indices because they work as templates. Instead of defining all the three queries required in a context index, a user only needs to fill in the parameters of a constructor and a new context index will be generated using this input. Moreover, constructors also simplify the browsing because they usually represent common conceptual constructions. We chose the term constructor by analogy to constructors in object-oriented programming, but it is often useful to think of them in terms of *macros* or *templates*.

We define constructors formally in the following generic way:

Definition 30 (Constructor) *For a given query-based multicontext QBMC(Q), a* constructor *is a function whose values belong to the context index set* $\mathbb{P}$.

Each constructor is denoted using a constructor name. The important aspect of Definition 30 is that a constructor is any function whose range lies in $\mathbb{P}$, but which may depend on some parameters (i.e. the domain). Note that the type of set used as domain of a constructor is left unspecified, though the constructors presented in this section are all defined on query languages, other kind of arguments could be useful. In many cases, the parameters of the functions are queries from Q.

Once you have set these parameters, the constructor can be evaluated by the query-based multicontext infrastructure in order to create a complex context index. This new context index can be used in exactly the same manner as any other context index. The evaluation of a constructor is constructor specific.

In Chapter 4, Section 4.3 , we discuss diverse strategies to ease the *constructor parametrisation.*

To illustrate this, consider a $q_r \in L_2$, then the constructor $q_r \mapsto R(q_r)$ where $R(q_r) := (dom(q_r), range(q_r), q_r)$ is the context index resulting from this constructor. While this constructor is very simple and limited, it ensures the

Table 3.9: Definition of the query primitives of the query languages L_C, L_R and L_I for query infrastructure $\mathbb{Q}_{Kb}$.

Notation	Meaning	Query Result
Notations for L_C	returns (concept names)	Results: Concept Sets
subconcepts(q_C)	all subconcepts of elements of q_C	$\{d \in \mathcal{C} : \exists c \in eval_1(q_C), d \leq_C c\}$
subconcepts(q_C)	all superconcepts of elements of q_C	$\{d \in \mathcal{C} : \exists c \in eval_1(q_C), d \geq_C c\}$
concepts(q_I)	the set concepts of which the instances in q_I are instances	$\{c \in \mathcal{C} : \exists i \in eval_1(q_I), i \in \iota(c)\}$
related(q_C)	concepts attached to some concept of q_C through some relation	$\{e \in \mathcal{C} : \exists m, n \in \mathbb{N}, m \neq n, \exists c \in q_C, \exists r \in \mathcal{R}, e \in \pi(n, \sigma(r)) \wedge c \in \pi(m, \sigma(r))\}$
related(j, q_R, k, q_C)	concepts at the position j of a relation of q_R, and where a concept of q_C is in the position k of this relation	$\{d \in \mathcal{C} : \exists c \in eval_1(q_C), \exists r \in eval_2(q_R), c = \pi(k, \sigma(r)) \wedge d = \pi(j, \sigma(r))\}$
Notations for L_R	returns (relation names):	Results: Relation Sets
subrelations(q_R)	all subrelations of elements of q_R	$\{s \in \mathcal{R} : \exists r \in eval_2(q_R), s \leq_R r\}$
superrelations(q_R)	all superrelations of elements of q_R	$\{s \in \mathcal{R} : \exists r \in eval_2(q_R), s \geq_R r\}$
relations(q_C)	relations attached to a concept of q_C	$\{r \in \mathcal{R} : \exists c \in eval_1(q_C), \exists n \in \mathbb{N}, c \leq_C \pi(n, \sigma(r))\}$
relations(q_R, j, q_C)	the set of relations belonging to q_R for which there is a concept c compatible with its signature	$\{r \in \mathcal{R} : \exists c \in eval_1(q_C), r \in eval_2(q_R), c \leq_C \pi(2, \sigma(r))\}$
relations(i, q_{I_1}, j, q_{I_2})	relations which have in their extension some instance pair (i_1, i_2) where q_{I_1} returns i_1 and q_{I_2} returns i_2	$\{r \in \mathcal{R} : \exists i_1 \in eval_1(q_{I_1}), \exists i_2 \in eval_1(q_{I_2}), r \in eval_2(q_R), \exists w \in \iota_R(r), i_1 = \pi(1, w) \wedge i_1 = \pi(2, w)\}$
Notations for L_I	returns (instance names):	Results: Instance Sets
inst(q_C)	the set of instances of concepts belonging to q_C	$\{i \in \mathcal{I}_C : i \in \bar{\iota}(eval_2(q_C))\}$
instances(j, q_R, k, q_C)	set of instances at position j of a relation instance of q_R for which at position k some instance of one concept of q_C is found	$\{i \in \mathcal{I}_C : \exists c \in eval_1(q_C), \exists r \in eval_2(q_R), c = \pi(k, \sigma(r)) \wedge \exists w \in \iota_R(r), i = \pi(j, w)\}$
instances(j, q_R, k, q_I)	set of instances at position j of a relation instance of q_I and where at position k an instance of q_i is set q_R	$\{i \in \mathcal{I}_C : \exists i_1 \in eval_1(q_I), \exists r \in eval_2(q_R), \exists w \in \bar{\iota}_R(r), \pi(k, w) = i_1 \wedge \pi(j, w) = i\}$

Table 3.10: Definition of the query primitives of the query languages L_{Ri},L_{CI}, L_{CR}, L_{CC} and L_{RR} of query infrastructure $\mathbb{Q}_{Kb}$

Notation	Meaning	Query Result
Notations for L_{Ri}	returns (relation instances names):	Results: relation instance sets
relinstances(q_R)	set of relation instances of the relations resulting from q_R	$\{i \in \mathcal{I}_R : \exists r \in eval_1(q_R), \iota_{\mathcal{R}}(i)\}$
Notations for L_{II}	returns pairs of instances names:	Results: sets of pairs of instances
$< q_R >_{jk}$	the sets of pair of instances obtained from evaluating each relation one relation of q_R at position j and k	$\{(i_1, i_2) \in \mathcal{I}_C \times \mathcal{I}_C : \exists r \in eval(q_R), \exists w \in \iota_R(r)\ (i_1, i_2) = (\pi(j, w), \pi(k, w))\}$
$< q_R >$	the sets of pair of instances obtained from evaluating each relation one relation of q_R at positions 1 and 2	$\{(i_1, i_2) \in \mathcal{I}_C \times \mathcal{I}_C : \exists r \in eval(q_R), \exists w \in \iota_R(r)\ (i_1, i_2) = (\pi(1, w), \pi(2, w))\}$
Notations for L_{IC}	returns pairs of instance and concept names:	Results: sets of pairs of the form (instance,concept)
INSTANTIATION	the instantiation function for concepts	$graph(\iota_C) := \{(c, i_1) \in \mathcal{C} \times \mathcal{I}_{\mathcal{C}} : i_1 \in \iota_C(c)\}$
Notations for L_{CR}	returns pairs of:	Results: sets of pairs ( )
CONCEPT-RELATIONS	the relation between concepts and relations	$\{(c, r) \in \mathcal{C} \times \mathcal{R} : \exists i \in \mathbb{N}, c \leq_C \pi(i, \sigma(r))\}$
Notations for L_{CC}	returns pairs of concept names:	Results: sets of pairs of concepts
subconcepts	the graph of $\leq_C$	graph($\leq_C$)
Notations for L_{RR}	returns pairs of relation names:	Results: sets of pairs of relations
subrelations	the graph of $\leq_R$	graph($\leq_R$)

construction of a correct context index. However, most interesting constructors in practice generate more complex structures.

An important aspect to consider is that the generic definition of a context index can also be seen as a constructor with three parameters: the object query, attribute query and the relation query. This allows us to deal principally with constructors instead of context indices.

In order to define some constructors, it is sometimes important to constrain the types of the parameters which can be used. Therefore, the following definition specifies the notion of queries over a given subset of Ω.

Definition 31 (Query over a Set) *Let A be a subset of Ω, i.e. $A \subseteq \Omega$; we call a query $q \in L_1$, a* query over the set A *if $eval(q_1) \subseteq A$.*

We start the presentation of the main constructors by considering the relations constructors.

3.5.1 Relation Constructors

We already gave an informal idea of a *relation constructor*. But there are actually four relevant relation constructors depending on which arguments are set.

Definition 32 (Relation Constructors) *We call* default relation constructor *the mapping R from L_2 into $\mathbb{P}$ which maps q_3 to the context index*

$$\begin{aligned} R : L_2 &\longrightarrow \mathbb{P} \\ (q_3) &\mapsto (domain(q_3), range(q_3), q_3) \end{aligned}$$

We call relation constructor with domain *the mapping R_d from $L_1 \times L_2$ into $\mathbb{P}$ mapping (q_1, q_3) to the context index*

$$\begin{aligned} R_d : L_1 \times L_2 &\longrightarrow \mathbb{P} \\ (q_1, q_3) &\mapsto (q_1, range(q_3), q_3) \end{aligned}$$

In the same manner, the relation constructor with range[18] *is the mapping R_r from $L_2 \times L_1$ into $\mathbb{P}$ mapping (q_2, q_3) to the context index*

$$\begin{aligned} R_r : L_2 \times L_1 &\longrightarrow \mathbb{P} \\ (q_3, q_2) &\mapsto (domain(q_3), q_2, q_3) \end{aligned}$$

Finally, the fully specified relation constructor[19] *is the mapping R_{dr} from $L_1 \times L_1 \times L_2$ into $\mathbb{P}$ mapping (q_2, q_3) to the context index*

$$\begin{aligned} R_{dr} : L_1 \times L_1 \times L_2 &\longrightarrow \mathbb{P} \\ (q_1, q_2, q_3) &\mapsto (q_1, q_2, q_3) \end{aligned}$$

To illustrate these constructors, we use the following example.

[18]The difference in order of the parameters allows to determine the constructor type from the query types. The indices are obsolete in this case.

[19]The fully specified relation constructor corresponds actually to the generic definition of a context index. We consider it as a constructor because it is also clear that every context index can be created by parametrising the fully specified relation constructor.

Example 10 (Relation Constructors) *We start by using the default relation constructor on the publication relation,*[20] *to build the following context index:*

$$R(<\mathsf{publication}>) := \begin{pmatrix} dom(<\mathsf{publication}>) \\ range(<\mathsf{publication}>) \\ <\mathsf{publication}> \end{pmatrix}$$

Since the concept lattice obtained from this context index is very large, it is sensible to concentrate on a smaller number of publications. For example, the publications of the members of the PADLR *project can be obtained by using the query* $q_1 := \mathsf{role}(<\mathsf{member}>, \{PADLR\})$.

$$R_d(q_1, <\mathsf{publication}>) := \begin{pmatrix} \mathsf{role}(<\mathsf{member}>, \{PADLR\}) \\ range(<\mathsf{publication}>) \\ <\mathsf{publication}>) \end{pmatrix}$$

This can be further refined by requiring the publications to deal with text-mining *using the query* $q_2 := \mathsf{related}(<\mathsf{isOnTopic}>, \{\mathsf{text\text{-}mining}\})$.

$$R_{dr}(q_1, q_2, <\mathsf{publication}>) := \begin{pmatrix} \mathsf{role}(<\mathsf{member}>, \{PADLR\}) \\ \mathsf{role}(<\mathsf{isOnTopic}>, \{\mathsf{text\text{-}mining}\}) \\ <\mathsf{publication}>) \end{pmatrix}$$

3.5.2 Hierarchy Constructor

Since hierarchical relations are very common in practice, it is useful to have a constructor which builds a context capable of representing the order relation between elements of a hierarchy. Since every partially ordered set can be embedded in a minimal lattice,[21] hierarchies can be quite naturally coded in a formal context (see [Davey and Priestley, 1994, Ganter and Wille, 1999] for more details on the Dedekind-MacNeille completion).

Definition 33 (Subsumption Hierarchy Constructor) *Let* $q_{\mathcal{H}}$ *be a query over a partially ordered set* $(\mathcal{H}, \leq)$. *We define the hierarchy-based context index* $S(q_{\mathcal{H}}, \leq)$ *for* $(q_{\mathcal{H}}, \leq)$ *as*

$$S(q_{\mathcal{H}}, \leq) := (q_{\mathcal{H}}, q_{\mathcal{H}}, \leq)$$

Figure 3.5 shows in a) a schematic representation of this first hierarchy constructor. The other one are described later in this Section.

To illustrate the use of this constructor, we imagine a scenario on the web.

Example 11 *A typical use of this kind of constructor is the display of a concept hierarchy. For example, the concept lattice corresponding to the context index* $S(\mathsf{subconcepts}(\{\mathsf{Person}, \mathsf{ResearchGroup}, \mathsf{document}, \mathsf{Conference}\}), \leq_C)$ *is displayed in Figure 3.6. Here the query* $q_{\mathcal{H}}$ *is* subconcepts({Person, ResearchGroup, Document, Conference}).

[20] We are interested in the concept lattice of the publication relation.

[21] Minimal in the sense that this lattice can be embedded in all the lattices into which the partially ordered set can be embedded.

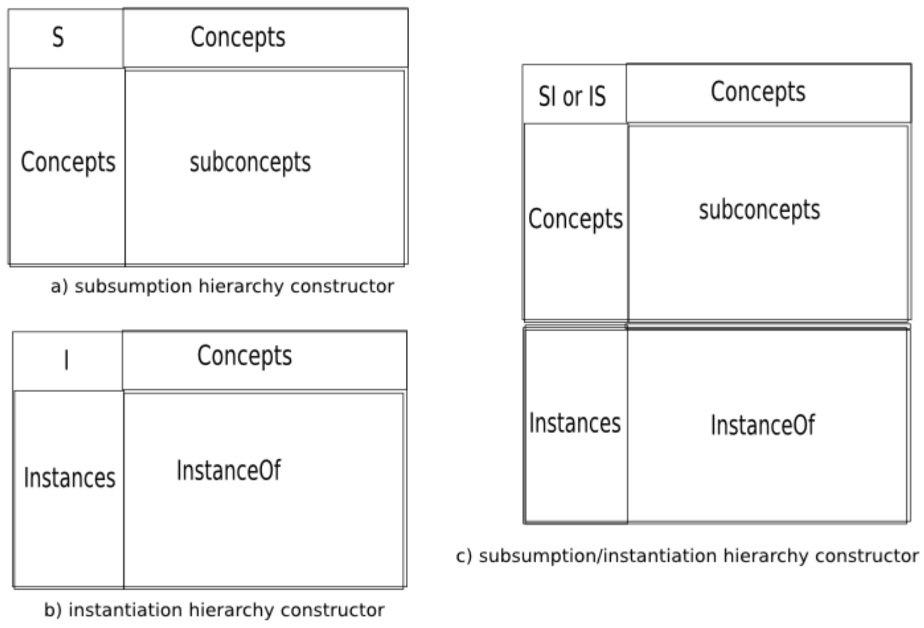

S
Concepts
Concepts
subconcepts
a) subsumption hierarchy constructor
I
Concepts
Instances
InstanceOf
b) instantiation hierarchy constructor
SI or IS
Concepts
Concepts
subconcepts
Instances
InstanceOf
c) subsumption/instantiation hierarchy constructor

Figure 3.5: A schematic representation of the subsumption hierarchy, instantiation hierarchy, subsumption/instantiation hierarchy constructor.

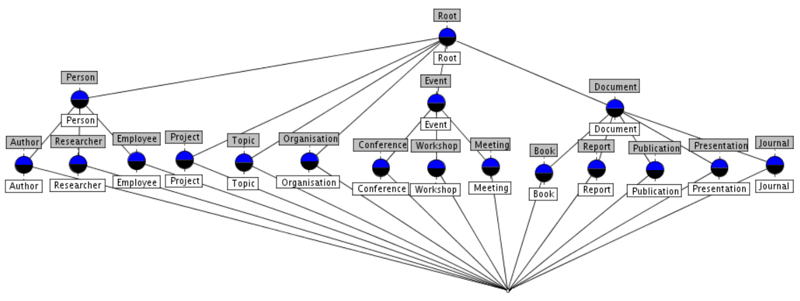


Figure 3.6: The lattice of the concept hierarchy.

3.5.3 Instantiation Hierarchy Constructors

Instantiation relations occur frequently in data under different forms. The most typical example is the instance/concept relation present in the ontology. However, any tagging or labelling can be seen as a kind of instantiation relation. For example, a book dealing with the Semantic Web can be seen as a document instance of the topic Semantic Web. In order to capture notions similar to instantiation, we give a generic definition of the notion of *instantiation function*.

Definition 34 (Instantiation Structure) *For two subsets C and I of Ω a function ι defined on C and with values in $\mathcal{P}(I)$ is called an* instantiation function. *$(C,\ I,\ \iota)$ is then called an* instantiation structure. *For a subset $A \subset C$, $\overline{\iota}(A) := \bigcup_{a\in A} \iota(a)$, and for a subset $B \subset I$, $\underline{\iota}(B) := \bigcup\{c \in C | \iota(c) \cap B \neq \emptyset\}$. For* *an instantiation structure, we call C the* concept set*, and I the* instance set*.*

Though a particular instantiation function may apply to other elements than concepts and instances, we use the term *concept* in this paragraph, (respectively the term *instance*) to denote the elements of the range of an instantiation function (respectively the domain).[22]

We can now define two kinds of instantiation hierarchies depending on whether the query is defined on the concept set or on the instance set:

Definition 35 (Instantiation Hierarchy Constructors) *For an instantiation structure (C, I, ι)) and for a query q_C over the set C, we define the* instance hierarchy context index for concepts from q_C *as*

$$\begin{aligned} C: \quad & L_1 \times L_2 \longrightarrow \mathbb{P} \\ & (q_C, \iota) \mapsto (\bar{\iota}(q_C), q_C, \iota^{-1}) \end{aligned}$$

For an instantiation structure (C, I,ι) and for q_I a query over the set I, we define the concept hierarchy context index for the instances from q_I *as*

$$\begin{aligned} I \quad & L_1 \times L_2 \longrightarrow \mathbb{P} \\ & (q_I, \iota) \mapsto (q_I, \underline{\iota}(q_I), \iota^{-1}) \end{aligned}$$

In b) from Figure 3.5, a schematic representation of the instantiation hierarchies is displayed. Note that all hierarchy constructors have concepts as attributes.

Example 12 *Figure 3.7 shows the lattice of the instantiation hierarchy for the subconcepts of* Person*,* Document*,* Research Group *and* Conference*. Note that the concepts* Document *and* Publication *are in the attribute contingent of the same formal concept. This happens because there is no instance in the knowledge base which is at the same time a* Document *but not a* Publication*. Due to the number of objects in some of the nodes the object labels are not displayed. However, in a user interface, the extensions of the nodes can be visualised through other means.*

3.5.4 Subsumption/Instantiation Hierarchies Constructors

The instantiation relation is usually coupled with a subsumption relation on the concept set. But since not all instantiation structures are compatible with the hierarchy on the concept set, some property formalising the compatibility between subsumption hierarchy and instantiation hierarchy is needed. This is the goal of the following definition.

Definition 36 (Instantiation Compatibility) *Let (C, I, ι) be an instantiation structure. An instantiation structure is* compatible with the partial order $(C, \leq)$ *if and only if the following property is valid*

$$\forall c, c_1 \in C, c \leq c_1 \Longrightarrow \iota(c) \subseteq \iota(c_1)$$

[22]For example, a topic hierarchy and the assignment of topics to documents is not a strict instantiation relation, since documents are not topics.

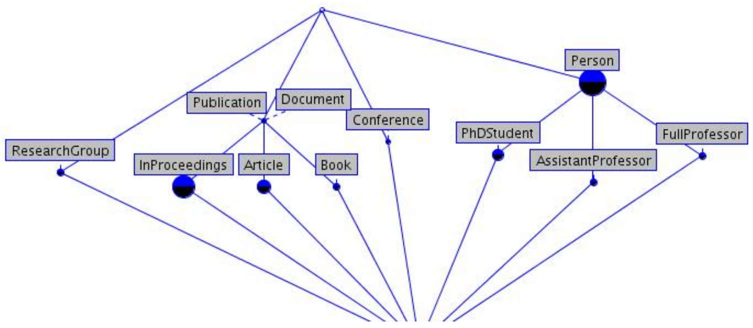


Figure 3.7: The lattice of the instance hierarchy for the subconcepts of: Person, Document, Research Group and Conference.

Using this notion of compatibility, we now define a constructor which combines concept hierarchies with instantiation hierarchies. Again two kinds of context indices are considered, depending on the type of query which is used as a parameter.

Definition 37 (Subsumption/Instantiation Constructors) *Let (C, I, ι) be an instantiation structure compatible with the partially ordered set $(C, \leq)$. Let $q_C \in L_1$ be a query on C, and $q_I \in L_1$ be a query on I. The* subsumption/instantiation context index for the concepts of q_C *is defined as*

$$SI(q_C, \leq, \iota) := (q_C \overset{.t}{\cup}_q \bar{\iota}(q_C), q_C, \leq \overset{.t}{\cup}_q \iota^{-1}) = \frac{S(q_C, \leq)}{C(q_C, \iota)}$$

and the instantiation/subsumption context index for the instances of q_I *is defined as*

$$IS(q_I, \leq, \iota) := (\iota^{-1}(q_I) \overset{.t}{\cup}_q q_I, \underline{\iota}(q_I), \leq \overset{.t}{\cup}_q \iota^{-1}) = \frac{S(\underline{\iota}(q_I), \leq)}{I(q_I, \iota)}$$

As shown in the section c) from Figure 3.5, the subsumption/instantiation hierarchies can be built using the results from the other hierarchy constructors. Note that the set of concepts of both the subsumption and instantiation hierarchies must be the same. The subsumption/instantiation constructor can then be built by using the subposition operator on the subsumption constructor for the concepts in q_C and the instance hierarchy constructor for the concepts of q_C. The instantiation/subsumption is also built using the instantiation/subsumption on on the subsumption constructor for the parent concepts of q_I and the concept hierarchy constructor for the instance of q_I.

The following example illustrates both instantiation/subsumption hierarchies.

Example 13 *Figure 3.8 shows the lattice of the subsumption/instantiation hierarchy for the concepts* Person, Professor, PhDStudent, Assistant. *This hierarchy has the context index*

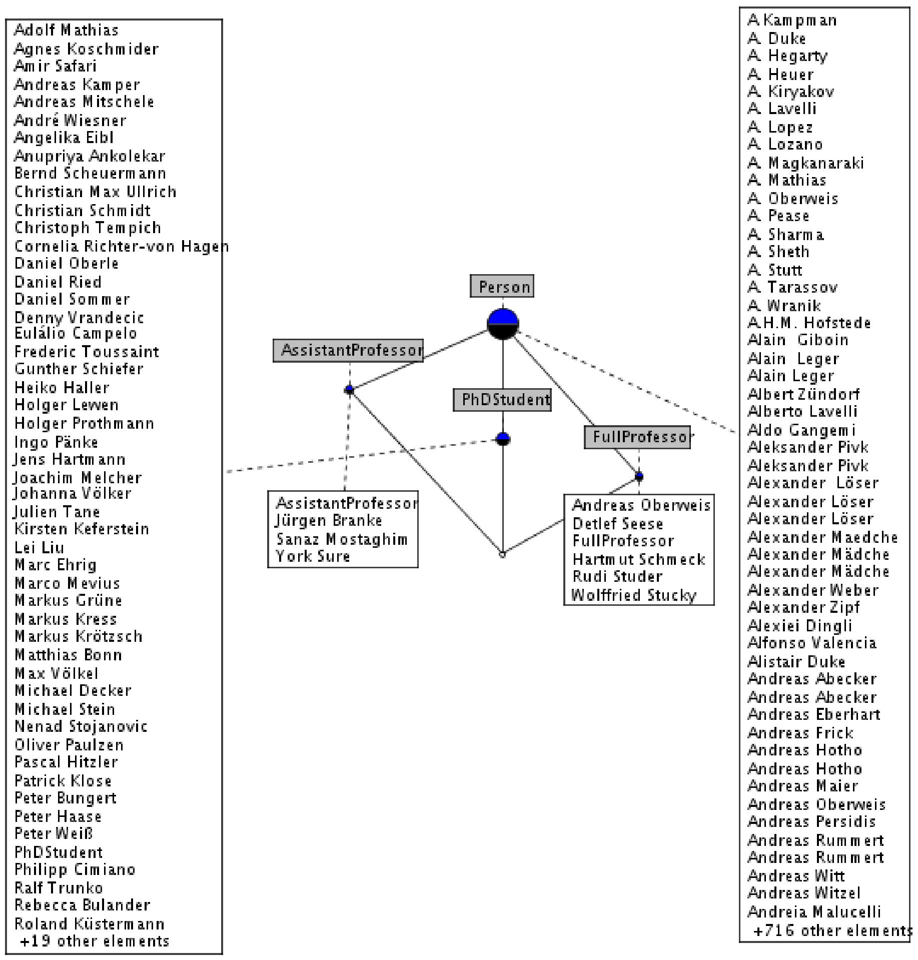


Figure 3.8: The lattice of the instance hierarchy for the concepts: Person, Professor, PhDStudent, Assistant.

$$SI(\{\textsf{\textit{Person, Professor, PhDStudent, Assistant}}\}, \leq_C, \iota_C)$$

Figure 3.9 shows the lattice of the instantiation/subsumption hierarchy for the instances: ***Rudi Studer, York Sure, Julien Tane*** *The context index of this hierarchy is*

$$IS(\{\textbf{\textit{Rudi Studer, York Sure, Julien Tane}}\}, \leq_C, \iota_C)$$

3.5.5 The Join Relation Context Constructor

Another important constructor is the *join*. Informally a join is a relation which is defined as the pairs of elements which are linked by the composition of two relations. For example, the join of the two relations: isAuthorOf and isOnTopic links authors of publications to the topics addressed in one of their publications.

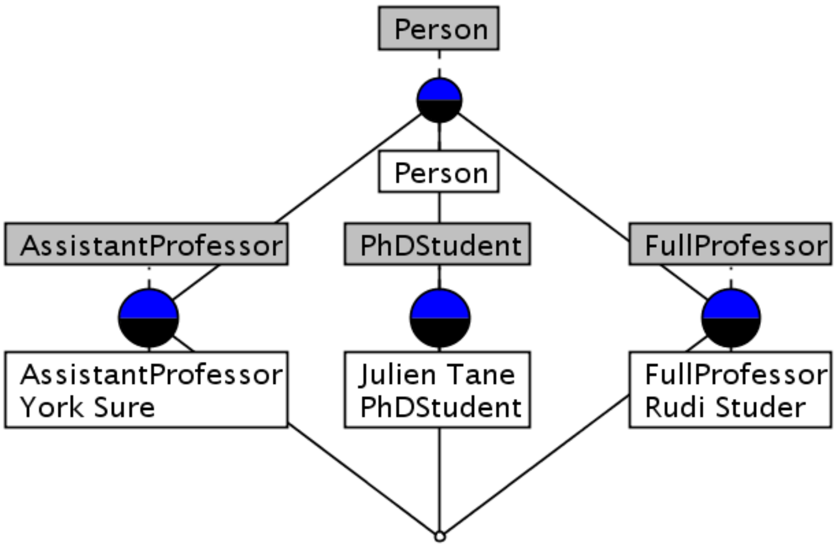


Figure 3.9: The lattice of the concept hierarchy for the instances: **Rudi Studer, York Sure, Julien Tane**.

Among the operators presented in Section 3.3.1, the two operators $\bowtie$ and $\bowtie_{q_1}$ enable the representation of the composition of relations. We see that the former operator can be easily implemented from the latter one. Joins are central operators for databases. A great amount of literature has been written on them. Introducing the join operators could allow more efficient implementations of the context construction if the data source is a relational database, because relational databases are traditionally optimised for this type of operation.

Definition 38 (Join Relation Context Constructor) *Let q_A, q_B, q_C be three queries in L_1 and let q_{R_1} and q_{R_2} be queries in L_2. The* Join *context index $Join(q_A, q_{R_1}, q_B, q_{R_2}, q_C)$ between q_A and q_C through q_{R_1} and q_{R_2} over q_B is defined as*

$$\begin{aligned} Join : L_1 \times L_2 \times L_1 \times L_2 \times L_1 &\longrightarrow \mathbb{P} \\ (q_A, q_{R_1}, q_B, q_{R_2}, q_C) &\mapsto (q_A, q_C, q_{R_1} \bowtie_{q_B} q_{R_2}) \end{aligned}$$

Of course, it is possible to generalise the notion of join context index, by using more than two relation queries.

Example 14 (Join Relation Constructor Example) *Figure 3.10 shows the lattice of the join*[23]

$$Join \begin{pmatrix} inst(\{\mathit{ResearchTopic}\}) \\ < \mathit{dealsWith} >^{-1} \\ \mathit{inst}(\{\mathit{Project}\}) \\ < \mathit{carriedOutBy} > \\ inst(\{\mathit{ResearchGroup}\}) \end{pmatrix}$$

[23]We use the vector notation to enhance the readability of the parameters of the constructor.

This join displays a relation between research topics and research groups built from the join of the two relations dealsWith and carriedOutBy. Note that the SWRC ontology (see Appendix A) does not contain any direct relation between research topics and research groups.

3.5.6 The CoRelation Constructor

Our preliminary experience with a query-based multicontext for knowledge browsing has quickly shown that a certain kind of context index had to be frequently constructed. The purpose of the corresponding formal context is to visualise the relation between a set of elements and the elements sharing some property with one of these elements. This is best explained using an example. Consider a set of selected authors, in particular consider the special case where this set contains only one author: Julien Tane. Figure 3.11 is used to illustrate this situation. It is possible to look at the list of publications of this author. But, a researcher might be more interested in his coauthors. The CoRelation constructor is a view designed to explore the shared relations of a group of individuals.

This constructor takes two parameters: a set query and a relation query. In the figure, the set query returns the author marked with an X while the relation query returns all the pairs of the authorOf relation, that is all the pairs of nodes linked to each other. The purpose of the CoRelation constructor is to consider only the coauthors of the start authors.

The following definition formalises the intuitive idea of the CoRelation constructor.

Definition 39 (CoRelation Context Constructor) *For a query q_1 in L_1 and a query $q_3 \in L_2$, the* CoRelation constructor *is defined by*

$$\begin{aligned} CoR : L_1 \times L_2 &\longrightarrow \mathbb{P} \\ (q_1, q_3) &\mapsto (q_3^{-1}(q_3(q_1)), q_3(q_1), q_3) \end{aligned}$$

Example 15 (CoRelation Example) *In Figure 3.12, the lattice representing the relation between Julien Tane together with his coauthors and his publications is shown.*

3.5.7 Application-dependent Context Constructors

In contrast to the well defined constructors, there is a class of constructors which are extremely useful but have a purpose only in a well defined context. This happens when some kind of context indices occurs frequently or plays an important role in the visualisation process. It is then useful to define an *application-dependent constructor*. For example, we introduce below a constructor capable of structuring news items according to some personalisation criteria. A new context index can then be generated every day by setting the set of daily news items as objects.

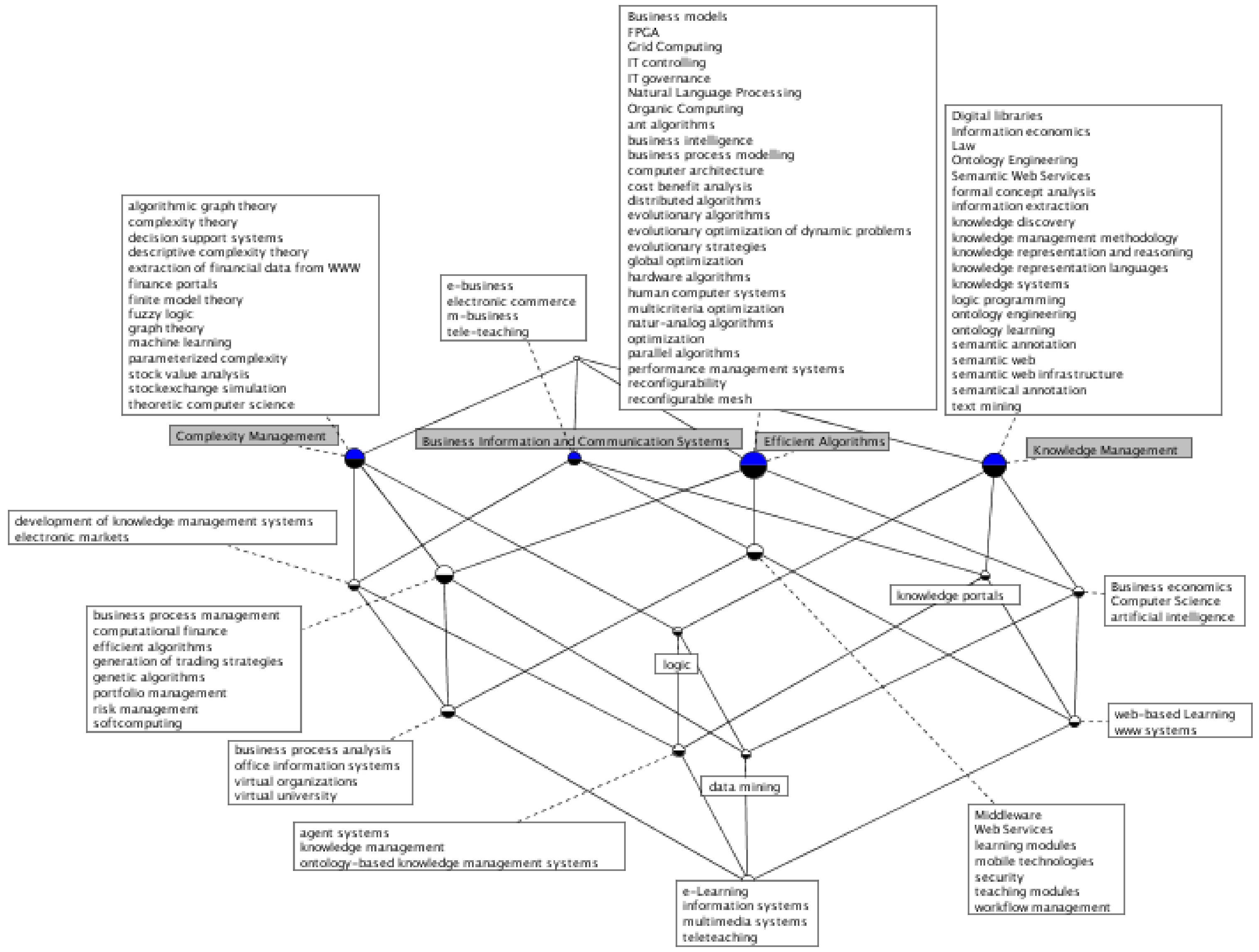


Figure 3.10: The lattice of the join between research topics and research groups through the project.

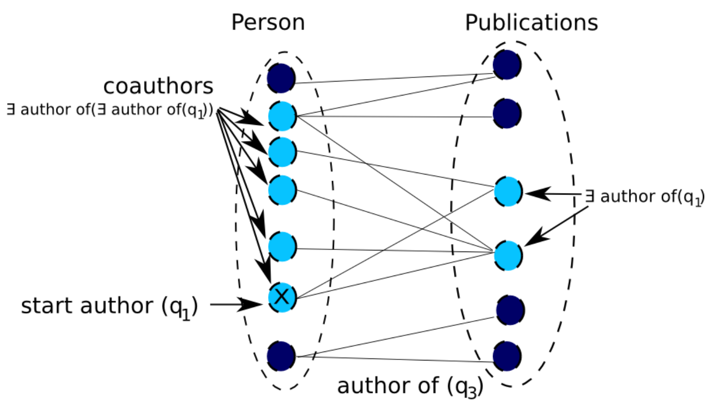


Figure 3.11: The main idea behind the CoRelation constructor. X is the start author and all the light-coloured nodes are in the resulting context.

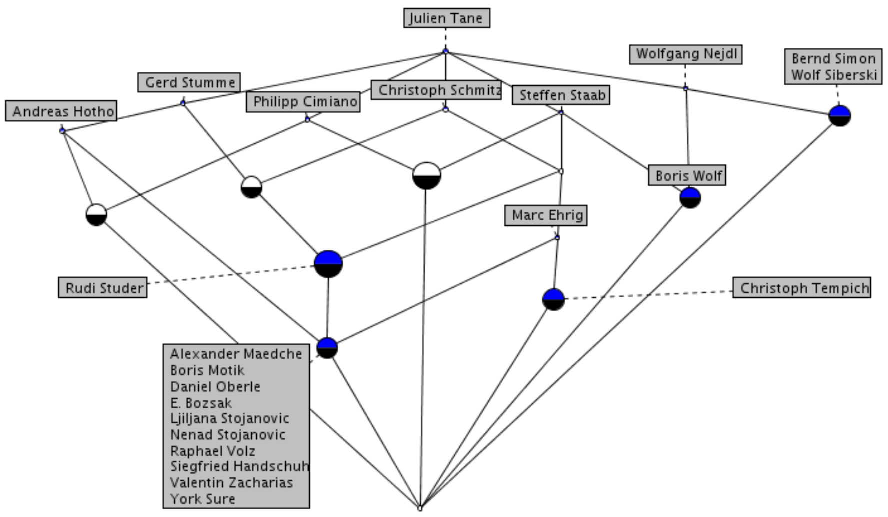


Figure 3.12: The concept lattice displaying the relation between the publication written by Julien Tane and his coauthors.

Example 16 (Application-Dependent Constructor Example) *Suppose the query-based multicontext browser is used to classify news items.*[24] *Each news item is parsed to return two relations:* topic *and* occurrence*, where*

- *topic:*[25] *a relation returning the pairs (n, t) where n is a news item and t one of the topics attached to it*

[24]An RSS feed could be for example used for this purpose (see http://www.webressource.org/rss/1.0/spec for more information on RSS feeds) Note that RSS Feed are usually in RDF which makes it easier to integrate into a knowledge base.

[25]Some RSS feeds annotate their items with topic informations.

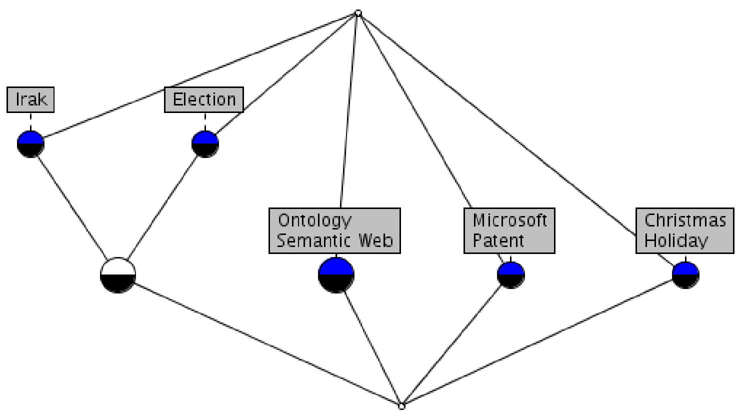


Figure 3.13: The lattice for news filtering.

- *occurrence:*[26] *a relation returning the pairs* (n, w) *where* n *is a news text and* t *one of the relevant*[27] *words contained in the text (for instance: Semantic Web, Ontology, Irak...).*

This news reader needs some parametrisation. The first parameter needed is the list of news item as input. The topics of the news items are retrieved from the news item automatically. Finally, the user can choose a list of terms of particular interest.
Using these input elements, a context index (q_1,q_2,q_3) *can be defined using the following queries.*

- q_1 *is the query returning the news items of the day.*
- q_2 *is a query returning the union of the topics of the day and the preselected terms.*
- q_3 *is a query returning the pairs between these attributes and the news of the day.*

This application-dependent constructor can be written as

$$News = (q_n,\ topic(q_n) \cup \{\textsf{Semantic Web},\ \textsf{Ontology},\ \textsf{Irak}\},\ topic \cup occurrence)$$

where q_n *is the query returning the set of news items taken as input,* topic *a query representing the relation between news and their topics. Finally, occurrence is a query returning the relation between the news items and the terms occurring in their text. Figure 3.13 shows the lattice created for this news filtering mechanism.*

To implement such a constructor, it is also often useful to implement some application dependent operators. For example, the operators occurrence and

[26]Most RSS feeds contain some abstract about the news item.
[27]These words are picked out using an algorithm computing the most relevant words of a text.

topics which we introduced in the example, are implemented independently. Another example of an application dependent constructor can be found in Chapter 6, Section 6.2. The evaluation we present in that section was implemented by presenting diverse concept lattices of predefined views. These views were created using an application dependent constructor returning a certain context index depending on the step of the evaluation.

3.6 Summary

In this chapter, we introduced a novel generic approach for dealing with multicontexts: the query-based multicontext theory. This theory relies on the introduction of *context indices* which serve as intensional representations for formal contexts and can be generated from a data source.

We showed that considering certain operators on queries allows to define operations on context indices which correspond to commonly used operations on contexts. Moreover, we presented several extensions of basic query-based multicontexts, where the underlying query infrastructure can be typed or serve to access knowledge bases.

In order to facilitate the interaction with context indices, we introduced constructors, a template mechanism, to simplify the definition of interesting context indices, thus easing the creation of relevant contexts.

Arrived at this point of our explanation, it is natural to ask how this theory can be used. The following chapter introduces a new approach to the exploration of knowledge bases.

Chapter 4

Knowledge Base Exploration

We propose a novel approach for the exploration of a knowledge base relying on a process allowing the user to define and visualise contextualised semantic views. Using the theory developed in the previous chapter, we show how the notions of constructors and context indices can be used in the definition process. Moreover, we investigate the properties of three different paradigms for the visualisation of the generated views, insisting in particular on the means of selecting relevant individuals for the selection.

In the first section of this chapter, Section 4.1, we motivate the approach by stating its purpose, principle and main issues. In Section 4.2, we cover the issue of interacting with a view, in particular we discuss alternatives to selecting elements. Finally, we discuss in Section 4.3 the main means of creating a view using query definition and constructor parametrisation.

4.1 Exploring Using Views

In the last decade, the literature on information visualisation has increased tremendously. An overview of these different paradigms can be found in diverse publications related to the domain such as [Herman et al., 2000], [Chen, 2006], [Frasincar et al., 2003] or in the Information Retrieval literature (see for example [Hearst, 1999]). These paradigms also benefit from the more general user interface design literature (see [Shneiderman and Plaisant, 2005] for an overview of the research in the domain).

In this section, we first present the main goals of information visualisation introducing a generic model from the literature. This leads us to a discussion of the issues we had to address. Finally, we use the scenario presented in Chapter 1, Section 1.1.1 to illustrate the general principle of our approach.

4.1.1 Visualising the Information Space

The visualisation of information generally involves complex interaction between users and the machine providing the visualisation. In order to describe our approach, we refer to a generic model for information visualisation which has been first presented in [Boyle et al., 1993]. This model is built upon the notion of an abstract space and offers many similarities with our approach since a query-based multicontext can be seen as a kind of specialisation of the abstract space model proposed in this model.

Information Space

To build a conceptual model of visualisation in databases, the notion of an *information space* has been introduced in [Boyle et al., 1993]. It provides a generic view of any exploratory process. Such an information space can be seen as an all-encompassing space consisting of cognitive, presentation and database components. The *cognitive space* consists of a mental model the user has of the components of the system, the objective of the interaction, and the visual perception. The *presentation space* contains data, constructs and methods that allow to visualise the information. The purpose of the presentation space is to reveal important or relevant features for the fulfilment of the user's goals.

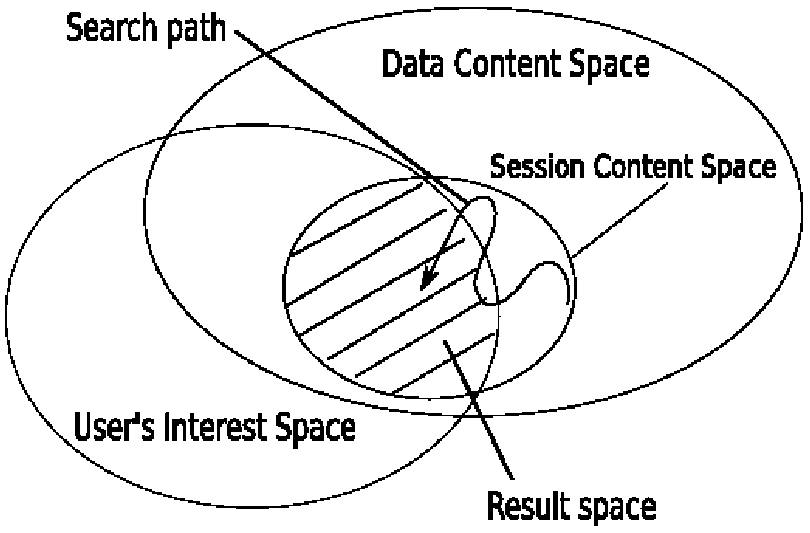


Figure 4.1: Diagram schematising the concept of Database Content Space (source: [Boyle et al., 1993]).

Finally, the *database space* (see Figure 4.1 for a schematic view of this space) consists of the *content space* consisting of all the information which can be interpreted from the database together with the *user's interest space*, that is, all the structures that should be found or uncovered during an exploration session. During a given session, only a part of the content space, the space visited during the session is the *session content space*. Finally, the *result space* consists of the areas of the content space relevant to the user, the uncovered parts of the interest space.

Interaction Purpose

From the model presented above, it is difficult to address the cognitive aspects in a satisfactory manner, yet it is possible to discuss the purpose of interaction on the design of an exploration framework. The purpose of interaction sets constraints on the nature of the user's interest space.

A user may have different motivations to interact with a visualisation framework. Considering the use and context of the interaction helps defining the basic requirements for the implementation. The possible tasks a user might want to perform have been studied in the literature. We refer here to the work found in [Shneiderman, 1997a], where four main tasks have been proposed: open-ended browsing, exploration of availability, specific fact-finding (known item search) and extended fact-finding. We considered these four purposes, but we chose other terms which suit better the purpose of exploring a knowledge base as opposed to the original terms which were chosen for discussing interactions with the Web. Moreover, these four purposes are categorised in two more generic classes.

- Exploration
  - familiarising, i.e. trying to understand better the structure of the knowledge base
  - visualising, i.e. trying to discover interesting properties or configurations
- Problem solving
  - searching, i.e. trying to locate some elements satisfying certain criteria
  - exhaustive search, i.e. trying to locate all the elements satisfying certain criteria

The two exploratory approaches are related to each other but their focus is somewhat different. In the first case, the tool should provide an overview of the content of the knowledge base. In the second case, the tool should emphasise particular properties of the displayed entities.

The difference between searching and exhaustive search lies in the extent of the recall or precision needed. *Recall* is a measure corresponding to the ratio of the number of elements actually found or visualised over the number of elements which should have been found or visualised. *Precision* is a measure corresponding to the ratio of the number of elements which should have been visualised over the total number of elements visualised. When searching, only one relevant element is needed. For example, it is sufficient to find the phone number of a person in one document and not all occurrences, that is, precision plays a greater role. Whereas when selecting, the purpose is to find all or at least most of the relevant elements. For instance, it is important for a travel agency broker to find all or most of the holiday packages matching a given set of criteria, in other words, in that case a high recall is important.

To illustrate these purposes, we recall the scenario from the introductory chapter. Our researcher Mikko Malli wants to visualise the relationships between researchers, their topics and their projects. This is typically an exploration task.

The focus of this thesis is not to investigate in detail the best strategies for these different approaches. However, considering these different goals provides a clearer picture of the elements of the knowledge base which are relevant to the user and sets a number of constraints on the approach to be used. Note also that the extensive research on Information Retrieval has already provided a number of effective solutions for some search problems. In the field of *information visualisation*, a number of techniques have been developed which help users in familiarising with a knowledge base. And some search tasks necessitate the use of exploratory methods due to the difficulty of formulating the relevant queries. Using a more exploratory approach, the user can gradually focus on the relevant part of the knowledge base.

To achieve his goal a user must find its way through the content space. An exploration framework should support the user by providing appropriate strategies.

Exploration Strategies

An exploration process consists of a number of steps and is modelled as an *exploration path* across the content space. A schematic representation of such a path is presented in Figure 4.1, showing an hypothetical navigation in the session's content space. An effective exploration framework can then be seen as having two main optimisation tasks. On the one hand, the steps in the path should be chosen so that they maximise the visualisation of the user's interest space. On the other hand, at each step of the path, the framework should also emphasise the most relevant aspects of the currently visualised space.

Depending on the user's interest space, different strategies can be used. Ben Shneiderman proposed in [Shneiderman, 1996] the following mantra

Overview first, zoom and filter, then details-on-demand.

and baptised it the *Information Seeking Mantra*. The goal of this mantra is to help the designers of information seeking interfaces by stating the main stages of the search process.

It means that a knowledge exploration tool should start with an overview and offer means of zooming, that is, specifying more precisely the current focus of the exploration. Parallel to the process of zooming, irrelevant pieces of information can be filtered out. Finally, once the correct focus is found, more information about the elements can be requested. If the search space would be metaphorically represented as a space where generic information is found at the top, and the details are at the bottom, then the mantra suggests a gradual vertical descent in the data space. Starting from the top where the most generic information is found, the user progressively reduces the search space using zooming and filtering until he can select the elements to be displayed in details.

The different alternatives for the exploration strategies are extremely dependent on the visualisation paradigms.

Alternative Visualisation Paradigms

An Entity-centred approach as presented in Chapter 1, Section 1.1 can only display on a very fine granular level and does not display relationships between groups of entities. The success of the information search is extremely dependent on the path chosen. This has different consequences. The number of steps required to cover the user's interest space is likely to be greater since every entity visited implies one search step. Then, the chances of following useless paths are greater, since there may not be any cues for the user that he is choosing the right path. Finally, since entity centred approaches are usually based on a neighbourhood principle, it might not be possible to relate to entities which do not share a neighbourhood. For example, the publications of a researcher working on some project may not belong to the set of entities in the neighbourhood of this project. Note that the context indices presented in Chapter 3 allow the definition of the neighbourhood of entities.

A number of other visualisation paradigms present views of the data using visualisation paradigms like hierarchies and graphs. These paradigms rely on zooming and filtering as proposed by Shneiderman, but zooming is usually performed in the sense of zooming a map or an image representing the data (see for example [Shneiderman, 1992]). Filtering on the other hand is usually limited to a number of fields (a traditional technique has been proposed by Shneiderman in [Ahlberg et al., 1992]). Though our approach is compatible with these paradigms, we use the contextualised view mechanism to specify the content of the views.

In addition to the filtering and zooming mechanism available in these views, our framework allows for another kind of zooming. Our approach can be seen as a conceptual zooming approach but is actually better defined as a view definition paradigm, since the interface supports the user in the definition of the data to be displayed. We present in the next section the model for this exploration framework.

4.1.2 Views for Knowledge Base Exploration

The approach we propose in this thesis relies on the use of views to visualise the information accessible from a knowledge base. In order to understand their utility, we discuss some important aspects.

Views: A Definition

We understand the term *view* in a sense close to the database terminology and define it in the following terms.

Definition 40 (View) *A* view *is a specific focus on a part of the content of the data source defined using an intensional representation. A* visualisation paradigm *provides a set of means for the user to visualise and interact with the content of the view.*

A query-based multicontext view *is a formal context generated from a context index defined using appropriate queries on an instance of a query infrastructure.*

Note that the intensional representation may take different form: structural, positional or semantic, etc. For example, in a geographical application, the view may be defined as the entities which are located in a given region. Since context indices are intensional representations of the generated, the generated formal contexts from these context indices form views on the data.

At first, we implemented a simple knowledge exploration mechanism using Formal Concept Analysis as one of the components of an E-Learning framework. The limitations of this first approach led us to design the query-based multicontext theory as underlying model for knowledge exploration.

The Courseware Watchdog

The *Courseware Watchdog*[1] is an ontology-based knowledge management application designed for maintaining E-Learning resources. The goal of the application was to discover, organise and explore collections of E Learning resources obtained from the web or a peer-to-peer application. The retrieved resources could then be organised by a semantic clustering mechanism. Finally, we developed a simple exploration framework combining Formal Concept Analysis and ontologies which allowed the user to visualise the existing data. More details on the Courseware Watchdog can be found in diverse publications (see [Tane et al., 2004, Tane et al., 2003, Schmitz et al., 2002]).

While other knowledge browsing approaches used graphs or trees to display the knowledge base, the Courseware Watchdog used concept lattices as a display paradigm. The concept lattices it displayed can be seen as simple views with little expressivity. It could for example create lattices displaying hierarchies of concepts, instance subsumption hierarchies for a set of instances and finally concept lattices of ontology relations. By interacting with the interface, the user could activate simple constructors to build the corresponding concept lattices. For example, a user could browse a number of levels of the concept hierarchy or display one of the relations defined on a given concept. While some of the views created by the system could be useful, the absence of means to further focus on parts of relations as well as elements of the domain were important limitations to overcome.

These limitations were rooted in the lack of expressivity of the lattice definition process, itself limited by the lack of expressivity of the context infrastructure which had strong similarities with the multicontexts introduced by Rudolf Wille (see [Wille, 1996] or refer to Chapter 2, Section 2.3.4). In order to overcome the lack of expressivity of the views, we looked at means of creating meaningful contextualised semantic views.

More Expressive Views

In Figure 4.2, we show a schematic representation of different formal contexts. a) represents the general case for formal contexts which represents a formal context as consisting in a set of objects, a set of attributes and a relation between them. The goal of defining expressive views implies finding a trade-off between expressivity and complexity. On the one hand, an expressive view allows the user to focus on his interest space. On the other hand, a complex definition process has different limitations. First of all, if the exploration framework has a steep learning curve, user would not use it. The second limitation lies in the complexity of the specification process. Studies have shown (see [Hearst, 1999]) that users tend to prefer simple interfaces with immediate results. Finally, a complex view definition process is likely to be error prone and increase the session path with views not containing any relevant information.

Dealing with the trade-off between the expressivity of the views and the complexity of their definition is one of the main issues in the design of a view-based exploration framework. However, the lack of expressivity has other limitations.

b) shows a particular context as it may be found in a multicontext in Wille's sense or as used in the Courseware Watchdog. It only uses names for concepts

[1]See http://cwatchdog.sf.net.

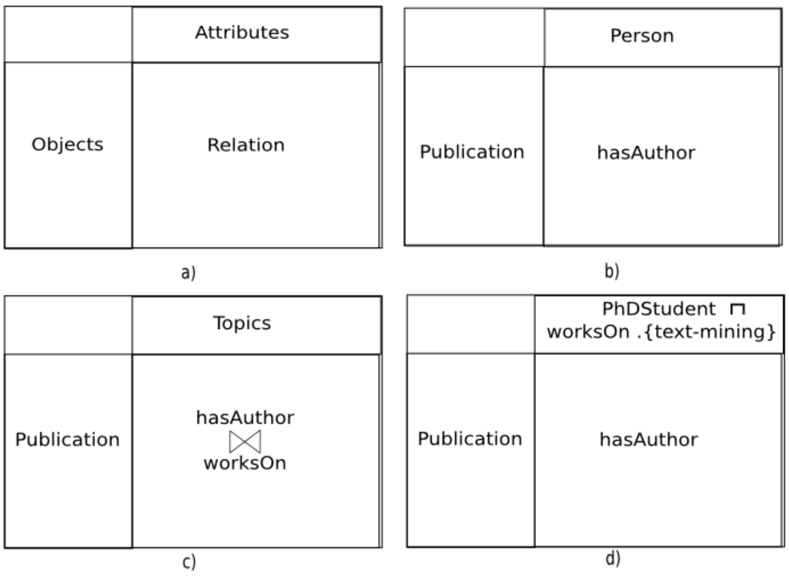


Figure 4.2: Diverse abstract representations of some views. a) represents an abstract representation of a view, b) is an example of simple relation view, c) is a view resulting from a join constructor and d) shows a view with a complex definition for the attribute set.

and relations but no operator. c) on the contrary defines the attributes using operators. Finally d) is defined using a join constructor. The ontology does not contain any relation between Topics and author. The context c) and d) could not be constructed without query operators. In particular, it is important to note the use of the nominal {text-mining}. Nominals play an important role since they allow the representation of a selection of instances for which there is no intensional definition. The last view d) is also interesting because it is based on a constructor.

Requirements for View Exploration

The user's final goal for the exploration is to obtain the relevant information from the exploration, the closer the presented information is to the user's interest space the most likely this will happen. However, to achieve this goal three issues must be addressed. First of all, the view displaying the data should suit the purpose. The relevant elements should be emphasised appropriately. The second issue is closely related to this issue since selection can be used to communicate which entities should be more closely studied. The graphical user interface should therefore provide the means of selecting relevant information. Moreover, selection is a crucial step for the definition of new views. This means that in order to improve the exploration of different views, the selection paradigm should provide appropriate methods of selection. Finally, while for many purposes default methods of using selections can be used to define new views, a powerful exploration framework should also provide more refined methods of providing information.

The Browsing Model

Before illustrating the exploration principle of our framework, we recall the main steps involved from a user point of view. A cycle of interaction with the framework can then be summarised using the following steps: view visualisation, selection, view definition.

Table 4.1: The main phases of an interaction cycle of the knowledge base exploration framework.

Phase	Description	Example
Visualisation	user interacts with the view to find relevant information	the user must note the text-mining researchers who work on topic similar to his
Selection	the user selects the relevant information from the view	the user must select text-mining or text-mining researchers
View Definition	the user performs specifies the parameters of the view	the user must be able to define the view displaying the relations between text-mining researchers and the research topic they work on

In the first phase, the user first familiarises himself with the content of the present view and may note which elements are relevant for further exploration. In the second phase, the user selects the relevant elements. This selection may consist in a single element, or in a set of relevant elements. In the third phase, the user makes use of the elements previously selected in order to define the next view. Once these steps have been performed, the framework constructs the next view to be displayed. These phases are summarised and illustrated in Table 4.1.

The phases introduced above emphasise the main difficulties and research questions of our framework. They differ from the cycles presented earlier in this thesis. In Chapter 1, Section 1.2.2, the cycle highlighted the general phases of the whole framework whereas the three phases presented here emphasise the steps the user must perform to visualise the next view. In Chapter 3, Section 3.1, the focus of the cycle was on the construction phase, which we described all along Chapter 3.

Our next step is to introduce a scenario describing the main principle behind our approach.

4.1.3 Scenario: Getting to Know AIFB

To illustrate the purpose of our approach, we return to the scenario presented in the introductory chapter (see Chapter 1, Section 1.1.1), where a researcher is interested in the AIFB publications by authors working on the topic of text-mining. Such a user may have different purposes to use the tool. For example, he might want to retrieve one or more specific publications or to retrieve all the publications by certain authors. Another goal might be to visualise the publications of the authors interested in text-mining. In the latter case, a grouping

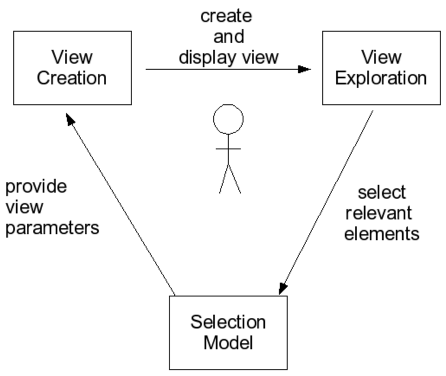


Figure 4.3: The interaction process for a query-based multicontext browser.

of the authors according to their common publications is useful.

To prevent a cumbersome description of the exact interaction at this stage, we focus on diverse information the user may want to know and the way it is displayed by our browsing framework. We assume here that our researcher is provided with the relevant views through some actions he performs. The actual description of the means to create a view is presented in Section 4.3. Moreover, this scenario could be implemented in a web browser. A user would click on a link to obtain a new predefined view. Yet, a static line diagram (i.e. the user cannot improve manually the layout of the diagram) is unsuitable for this task.[2]

Starting Point

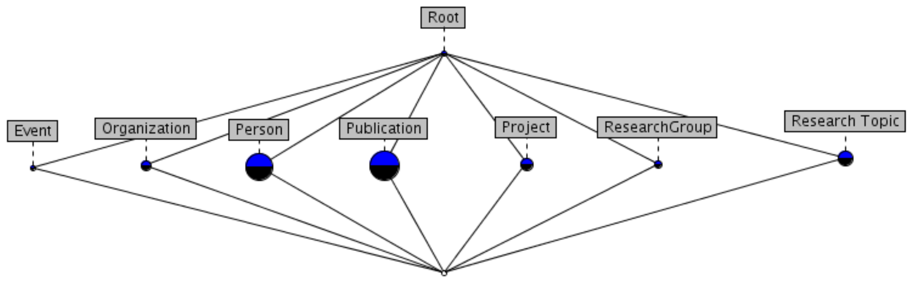


Figure 4.4: The start view presenting the main concepts of the knowledge base.

As a starting point, the view displayed consists of the top concepts of the ontology: Event, Organization, Person, Publication, Project, ResearchGroup, ResearchTopic. This view is shown in Figure 4.4. The user can then select the centre of interest of the data he actually wishes to visualise. For example, a person looking for certain publications would select the concept Publication and

[2]Since the current web technology are mature enough for this, it is to be expected that a suitable interaction framework will eventually be developed. We performed some experiments in this direction, but no working prototype has yet been implemented.

start defining views using this concept. If the concept hierarchy is small, displaying it as a whole may be feasible and would ease the selection of the most relevant concepts. This is illustrated by the actions of Mikko Malli.

Since Mikko Malli is primarily interested in persons, he selects Person as a concept of interest. He may obtain more details about this concept and display the subsumption hierarchy view of the subconcepts of Person. He may wish to study the distribution of instances among the subconcepts of Person and note that there is just a few Professors and Assistants but that there is a large number of PhDStudents as well as a large number of persons who are not instances of any of the subconcepts of Person.

Text-Mining Researchers and their Research Topics

Since our researcher is mainly interested in the researchers working in the field of text-mining, he first wants to find out the researchers who work on text-mining. The query corresponding to this simple request is:

$$q_1{:=}role(<\mathsf{isWorkedOnBy}>^{-1},\{\mathsf{text\text{-}mining}\})$$

The definition of such a query by the user can be performed in different ways by the user. When selecting the instance text-mining, some menus points enable him to create role queries with the relations which this instance has. Other approaches are also possible, for example, the user may fill in the entries of a form.

This query consists of two specific information which are both necessary to answer the question. In an entity-centred approach, our researcher can look for text-mining and then look at the values of the relation isWorkedOnBy. The result of this query can be easily presented as a list:

- Alexander Maedche
- Andreas Hotho
- Julien Tane
- Stefan Bloehdorn
- Stefan Klink
- Steffen Staab

This list informs the user that these persons share an interest in text-mining, but the user would like to have a better insight into their relationships. For instance, Mikko could be interested in the research topics shared by the persons working on text-mining. To visualise this kind of relations, our exploration framework provides a useful type of view: the views based on the CoRelation constructor (see Chapter 3, Section 3.5.6 for its formal definition). Using this constructor, Mikko can generate the formal context consisting of the set of all the researchers working on text-mining as attribute set, the set of all their research topics as object set and the relation indicates which topics are worked on by which researchers. Since the main focus are the research topics, this context is transposed[3] (i.e the object and attribute sets are swapped and the

[3]To simplify the explanation, the CoRelation context considered the start element to be objects, but it is often useful to consider the dual of this constructor.

components of the pairs of the resulting relation) as denoted by the small d as exponent. The final context index of this view is:

$$(\mathrm{CoR}(\{\textsf{text-mining}\},<\mathrm{isworkedOnBy}>))^d$$

One of the questions to be answered is which view paradigm should be used to visualise this context: a lattice, a graph, a tree, or something else? The tree paradigm does not allow to visualise easily which persons share research topics. The graph approach allows to visualise how projects are shared, but unless the research topics shared by persons are grouped together, it does not bring forward that these topics are shared by certain researchers. Concept lattices can be seen as a clustering structure which address this issue appropriately.

Figure 4.5 shows the concept lattice where research topics are grouped according to the persons working on them (the edges at the bottom element have been hidden, since the bottom element of the lattice does not contain any object[4]).

In this example, the label of the lowest concept lattice node which contains Steffen Staab and Alexander Mädche as object consists in seven elements. The node contains actually 13 research topic. This means that the two researchers share many research topics, yet it does not show the cooperation between these persons. To show their cooperation, two other views can be interesting: the shared projects and the shared publications. The latter is shown in Figure 4.6.

A few comments on this lattice are in order. First, no object label[5] are displayed in this diagram. While the number of elements per label in the first view is limited[6] (one line per box), in the second view the number of publications for at least some of the nodes is larger than 100 items.[7] If the view is displayed using the graphical user interface (as opposed to the Figure 4.6 which is static), several features allow the user to visualise dynamically the content of the nodes. For example, a context menu can be used to list elements of the extent of a given node.

A Researcher's Publications and Topics

Our researcher is now interested in the publications of one of these text-mining researchers: Andreas Hotho. He recognised the name and knows that the author writes on topics relevant for his research. This researcher is the author of 46 publications. A clustering of the publications according to their authors is useful here, since researchers cooperate often on more than one publication and shared publications tend to cover different aspects of one research question. Such a clustering is shown in Figure 4.7. The bottom element is not displayed, since no publication was written by all the coauthors of Andreas Hotho.

Furthermore, our researcher could be interested in researchers working at AIFB on the same topics as Andreas Hotho. Our researcher could be interested in seeing a lattice like the one displayed in Figure 4.8. The attribute set of the context of this lattice contains the research topics of Andreas Hotho, whereas

[4]In other words, there is no researcher working on all the topics.
[5]We recall that per convention the object labels are shown in white boxes whereas the attribute labels are in grey boxes.
[6]Research do work on many research topics.
[7]Some researchers write a large number of publications.

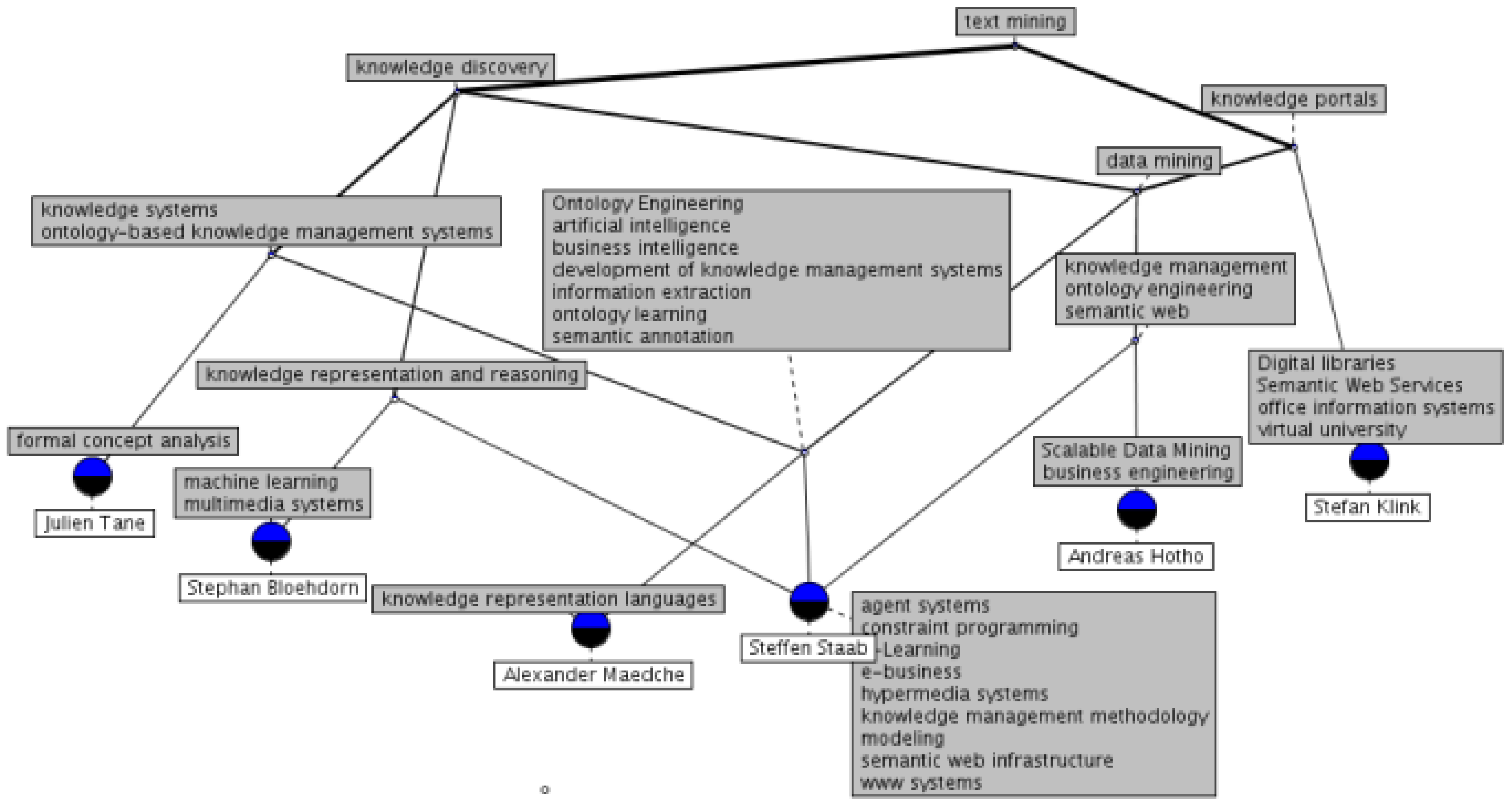


Figure 4.5: The distribution of the research topics of the AIFB text-miners.

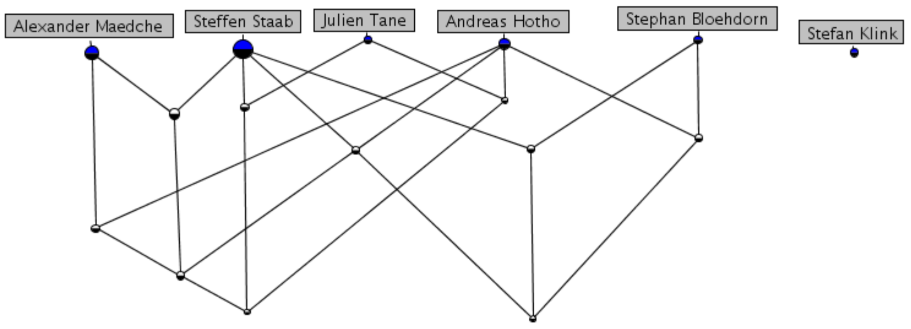


Figure 4.6: The distribution of the publications written by the text-mining researchers.

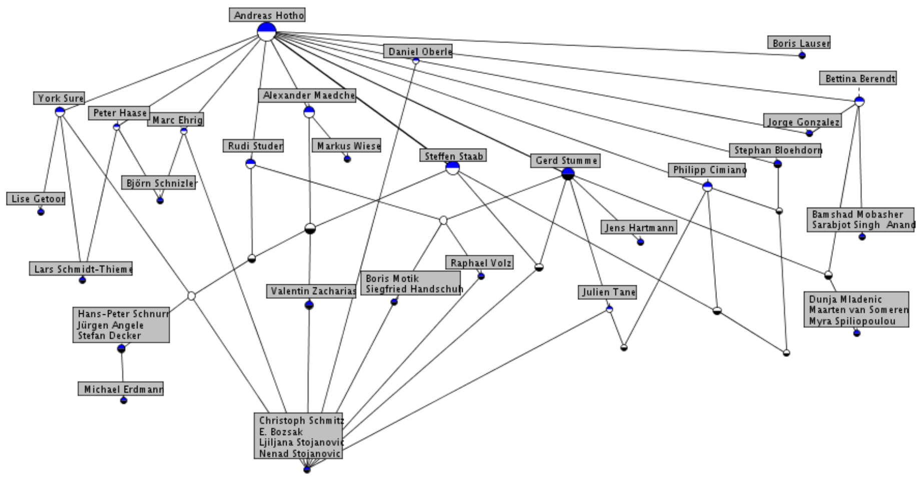


Figure 4.7: The concept lattice of Andreas Hotho's publications visualised with authors as attributes.

the object set contains the persons who share at least one research topic with Andreas Hotho.

We selected a few question which might interest our user. For each of these questions, we give the corresponding context index.

1. What are the subelements of a given set of elements?

   In particular, in the ontology model presented in Chapter 3, Section 3.4.1, two kinds of hierarchies are present: the concept subsumption hierarchy and the role subsumption hierarchy. For example, what are the subconcepts of the concept *Person*? This information is available from the realised context of the following context index:

$$\mathrm{S(subconcepts(\{Person\}),subconcepts)}$$

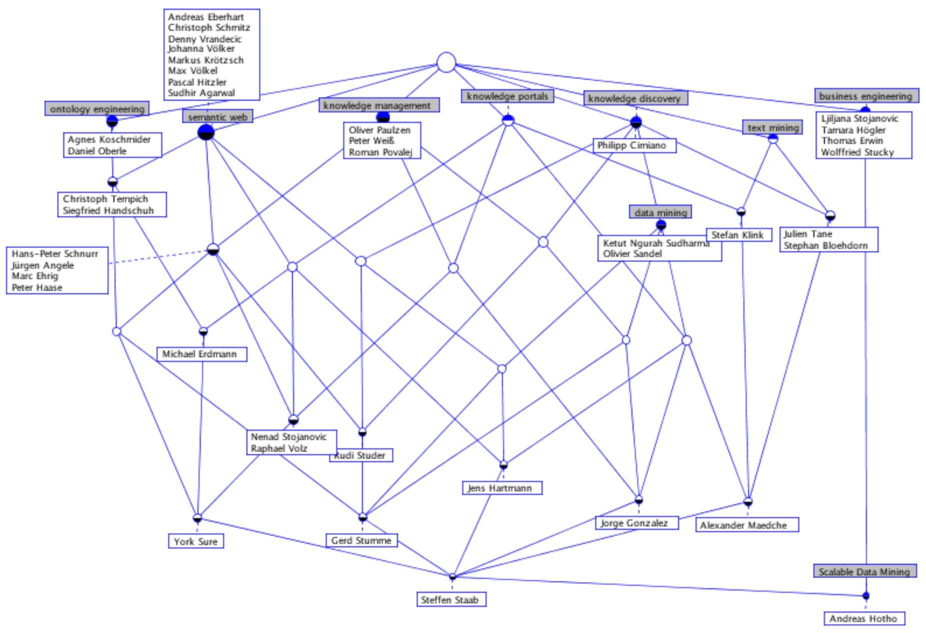


Figure 4.8: The concept lattice of Andreas Hotho's research topics and the colleagues sharing them.

2. How are the instances of a group of concepts distributed among their subconcepts? The subsumption instance context index is an intensional representation of the view required to answer this question. For example, the context index

   $$\text{SI}(\text{subconcepts}(\{\mathsf{Person}\}),\text{subconcepts},\text{instantiation})$$

   can be evaluated, and the concept lattice used to visualise the distribution of the instances of the concept Person among its subconcepts?

3. How is the distribution of a group of instances with respect to the concepts they instantiate? For example, who are the persons affiliated to the Efficient Algorithm research group and what is their position (i.e which subconcept of Person do they instantiate) can be displayed using the line diagram of the lattice of the following instance subsumption context index

   $$\text{IS}(\text{role}(<\mathsf{affiliation}>,\{\mathsf{Efficient\ Algorithm}\}),\ \text{subconcepts},\text{instantiation})$$

4. How is the configuration of a given relation? It is often useful to see how elements relate to each others. For example it is relevant to find the elements which share a greater number of elements or which share elements with a greater amount of elements. For example, using the concept lattice of the worksAtProject relation, it is possible to note which person participate in more projects. For this, it is interesting to study the distribution

of researchers with respect to the projects they work on. The following context index enables this formulation.

$$\mathrm{R}(<\mathsf{worksAtProject}>,\mathsf{dom}(<\mathsf{worksAtProject}>),\mathsf{range}(<\mathsf{worksAtProject}>))$$

5. Given a set of individuals, which are the individuals which share the most with these? It is often relevant to see the other authors which published with a given author and to note whether some groups can be recognised. For example, it may be useful for a researcher to see who co-authored with Andreas Hotho on which of his publications?

$$\mathrm{CoR}(\{\mathsf{Andreas\ Hotho}\},\ <\mathsf{publication}>)$$

6. How is the distribution of elements across two composed relations? In some situations, it is relevant to study the distribution of elements with respect to composed relations. For example, the SWRC ontology does not contain any direct relation between the organisations financing projects and the research groups carrying them out. This can be visualised using the concept lattice generated using the following parametrisation of a join constructor.

$$\mathrm{J}\begin{pmatrix} \mathsf{inst}(\{\mathsf{Organization}\}) \\ < \mathsf{financedBy} >^{-1} \\ < \mathsf{carriedOutBy} > \\ inst(\{\mathsf{Research\ Group}\}) \end{pmatrix}$$

4.2 View Interaction

Once a view has been generated from the data source, different visualisation alternatives are possible. Among the diverse paradigms, we focus on three visualisation paradigms which are particularly adapted to visualise the content of a formal context. Each of these paradigms, the *tree view*, *the graph view* and the *lattice view*, have advantages and drawbacks depending on the configuration of the data and the underlying purpose of the interaction. We discuss in this section two aspects which are particularly relevant for our approach and which correspond to different ways of looking at the human-computer communication. On the one hand, the purpose of a visualisation paradigm is to communicate the relevant information to the user. On the other hand, the user must be able to communicate his needs and intentions to the visualisation paradigm.

We start this section by discussing some preliminary aspects relevant for the interaction with views. Then we describe the diverse view paradigms to visualise a context index.

Visualisation Tasks

Ben Shneiderman investigates in [Shneiderman, 1996] different aspects of information visualisation such as the different types of data to be visualised and the main tasks occurring during information visualisation. These visualisation tasks are relevant in our approach since they also occur in our approach.

Each of these tasks plays a role in the visualisation of views, therefore we recall here the seven tasks[8] proposed in [Shneiderman, 1996].

1. **Overview**: Gain an overview of the entire collection.
2. **Relate**: View relationships among items.
3. **History**: Keep a history of actions to support undo, replay and progressive refinement.
4. **Focus**: Focus on items of interest.
5. **Filter**: Filter out uninteresting items.
6. **Details-on-demand**: Select an item or group and get the details when needed.
7. **Extract**: Allow extraction of sub-collections and the specification of query parameters.

Note that some of these tasks may require some interaction from the user. In particular, Tasks 3 to Tasks 7 can only be applied if the focus of the task is supplied. Task 1 and Task 2 may also require interaction but may also be more static. Finally, Task 7 is not an actual visualisation task but rather a task necessary for the adaptation of the visualisation.

In [Shneiderman, 1996], the term zoom was used instead of focus for Task 4. Different techniques can be used to focus and zooming is only one of them. Zooming can be seen as a method of focusing on relevant elements, but also as a means of filtering irrelevant elements.

Task 7 plays a crucial role in our approach since an appropriate selection mechanism is necessary for the query and view definition process.

Though a knowledge exploration framework should support all these tasks, we do not deal with each of them in detail, especially since their study is extremely dependent on the implementation and usability of the tools used. We focus particularly on the visualisation of relations as well as the extraction of sub-collections and query parameters.

Visualisation and Interaction Assistant

While we discuss the features of three different visualisation paradigms, it is important to recall the crucial roles played by a number of generic techniques which assists the user in visualising the view. Typically the main purpose of these techniques is to prevent the overloading of screen space and to minimise the cognitive effort of the user, thus allowing him to concentrate on the most relevant information. We summed this techniques in Table 4.2.

The first manner of enhancing the visualisation of a view is the use of a *preview* mechanism. A preview mechanism consists in a mean of giving more details about entities by displaying more information on the object when the mouse moves over its representation or some selection is performed. There are

[8]We changed the order of the tasks to simplify their presentation.

Table 4.2: Techniques helping the visualisation.

Technique	Description
preview	during interaction with the visualisation, supplementary information is automatically provided by the interface.
zooming	elements nearer to the selection are displayed more predominantly while other less.
spanning	the visualisation is seen through a movable window to display certain zones of the data.
focusing	the visualisation changes to show predominately some elements.
highlighting	some elements are highlighted.
context menu	a popup menu or window opens providing diverse action to visualise information.

diverse means of implementing such a preview mechanism. A first way is to use tool tips which appear when the mouse lingers over an area of interest. An other way is to use an auxiliary panel coupled with some partial selection by the user. For example, a user clicking on the concept Person may see the list of its instances in the auxiliary panel.

In order to reduce the screen space needed for visualisation, *zooming* and *spanning* are traditional techniques. They usually allow the user to keep a mental picture of the view. We understand here zooming only as the process of magnifying the size of some part of the view. Spanning is performed by displaying the visualisation through a window which can be moved to access other part of the data. This is typically performed using scroll bars.

Another important technique is the implementation of means of *focusing* and *highlighting*. Depending on the visualisation paradigm, different means of focusing and highlighting may be used. The purpose of highlighting is to emphasise some aspect of the data, while focusing often results in some emphasis, it may also result in a reorganisation of the layout of the view.

The techniques *filtering* and *hiding* consist in hiding or filtering out a number of elements so as to emphasise the remaining elements. These techniques have the same purpose as focusing and highlighting and are usually used together.

These techniques related to the well-known notion of *Focus+Context* (see [Lamping et al., 1995]). This approach is based on the principle that one of the entities to be visualised is currently focused upon. A number of other elements corresponding to the current context of use are emphasised as long as they are related to the focused entity. The user may then modify the visualisation by either changing the focus, i.e. choosing another entity as central or switching to another kind of context. A typical example of such a view is the bird's eye view which magnifies central elements while the elements which are less related to central elements are depicted smaller, further from the centre of the picture and may be greyed out (see [Lamping et al., 1995] for a typical example of Focus+Context approach). The main purpose of the Focus+Context technique is to be able to switch quickly both the focus and the context of exploration.

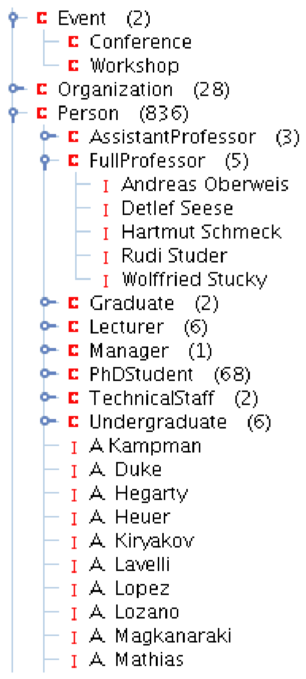
We also used context menus as a means of visualising the content of the lattice. A *context menu* is a small window appearing at the request of the user when selecting some entities. It provides a number of actions for the user to select from. Typically, context menus are used for tasks such as filtering, details-on-demand or the extraction of sub-collections or query parameters. For our purpose, we show in Section 4.3.2 that they provide a simple way of defining views. The context menu can also be used to display supplementary information. It can be displayed directly in the context menu,[9] or trigger the display of further information in an auxiliary panel. This is a simple mean of providing a support for the details-on-demand task.

4.2.1 Visualisation

Different kinds of visualisation paradigms exist to display views. In the context of this thesis, we investigate three paradigms based on three different kinds of structures: trees, graphs and lattices. While the first two paradigms are usual ways of displaying information to the user, the last one has shown to be practical for certain purposes (see [Eklund et al., 2004, Cole et al., 2003b, Hearst, 1999]). Each of these structures and corresponding paradigms have their own advantages and limitations.

To study the visualisation paradigms, it is important to note that the views of our framework can be grouped in two categories: hierarchies and relations. While all views are relational to some extent (hierarchies are special kinds of relations), the hierarchical views display in addition some kind of order between its attributes (the order is coded in the context, see in Chapter 3, Section 3.5).

The Tree Paradigm



```
C Event (2)
    C Conference
    C Workshop
C Organization (28)
C Person (836)
    C AssistantProfessor (3)
    C FullProfessor (5)
        I Andreas Oberweis
        I Detlef Seese
        I Hartmut Schmeck
        I Rudi Studer
        I Wolffried Stucky
    C Graduate (2)
    C Lecturer (6)
    C Manager (1)
    C PhDStudent (68)
    C TechnicalStaff (2)
    C Undergraduate (6)
    I A Kampman
    I A. Duke
    I A. Hegarty
    I A. Heuer
    I A. Kiryakov
    I A. Lavelli
    I A. Lopez
    I A. Lozano
    I A. Magkanaraki
    I A. Mathias
```

Figure 4.9: The tree displaying part of the subsumption-instance hierarchy of the ontology.

[9]Using the context menu to display information is usually considered bad design.

The *tree visualisation paradigm* is the simplest of the three paradigms used in our browsing tool. It can be used to display hierarchies in an intuitive manner. For example, Figure 4.9 shows the subsumption instance hierarchy for the AIFB knowledge base. It can also be used to display binary relations. An example of this is given in Figure 4.10. The view in this figure corresponds to the *carriedOutBy* relation between projects and research groups. However, the tree paradigm has the drawback that it does not illustrate the real structure of the data because a branch of the tree only indicates the relation between one upper element and the lower elements. In Figure 4.10, the projects of the diverse research groups are shown. Yet, it is difficult to extract from the tree view which projects are shared between research groups as well as which research groups work on common projects. In some way, the complement question of the sharing of properties can be seen as the elements not sharing properties. For that purpose, the tree view is not suitable either.

The drawbacks of the tree structure for visualisation of the sharing of elements should be contrasted with the structured possibilities of the tree views. A tree is able to display a large number of elements at the same time by putting the emphasis on the relation of an element with a group of elements. This emphasis enables user to optimise the search for the properties of the properties of a given element. This optimisation is further enhanced by the use of a sorting mechanism for siblings[10] in the tree. A user can pick an element easily by browsing a sorted list of labels.

Though the tree view is not suitable for visualising shared properties, the evaluation we present in Chapter 6, Section 6.2 shows that even for tasks where the tree paradigm does not seem suitable, some users perform amazingly well to solve the questions asked.

Another interesting property of trees is that they are also suitable to display multiple relations. For example, the properties of a set of instances can be displayed using a four-leveled tree with the instances as the children of the root element (the root node is hidden), the relations as children of the instances and the values as leaf.

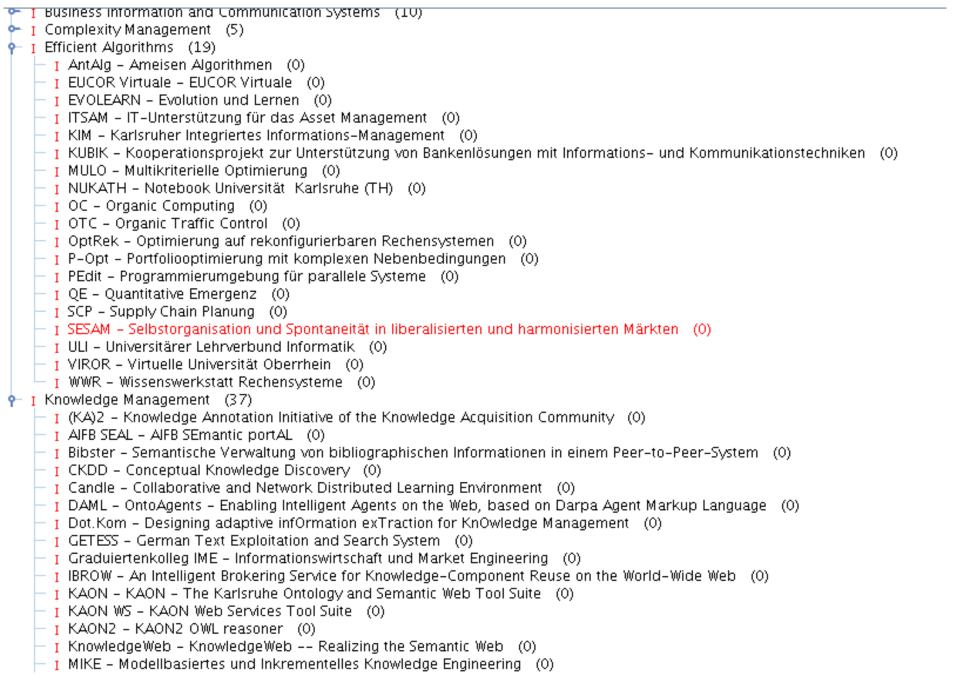
A few simple heuristics can be used in order to check whether the tree view paradigm is suitable. A very simple heuristic consists in checking whether the data is a one-to-one mapping. In that case, the number of objects and attributes is equal and the number of concepts in the lattice is equal to the number of objects plus 2 (bottom and top and as many formal concepts as objects or attributes). The other heuristic is to use an algorithm to partition the lattice. Such an approach is shown in Chapter 5, Section 5.5.4.

If the structure resulting from the removal of the bottom element of the lattice is a tree, the use of a line diagram to visualise the tree is often inappropriate. A tree panel may be more suitable, especially if the attribute contingent are small[11].

The literature on the visualisation of tree structure is extensive. While some of the approaches use visualisation similar to the one presented in Figure 4.9, many other means of displaying trees have been proposed for example some use graphs, maps or circles. [Kosba, 2004] compares different tree visualisation paradigms. For its comparisons, the paper uses the InfoVis Contest 2003

[10]Siblings are elements sharing a father in the tree.

[11]If the attribute contingent are large, the line diagram of a tree may still be more appropriate.



```
Business Information and Communication Systems   (10)
Complexity Management   (5)
Efficient Algorithms   (19)
    AntAlg – Ameisen Algorithmen   (0)
    EUCOR Virtuale – EUCOR Virtuale   (0)
    EVOLEARN – Evolution und Lernen   (0)
    ITSAM – IT-Unterstützung für das Asset Management   (0)
    KIM – Karlsruher Integriertes Informations-Management   (0)
    KUBIK – Kooperationsprojekt zur Unterstützung von Bankenlösungen mit Informations- und Kommunikationstechniken   (0)
    MULO – Multikriterielle Optimierung   (0)
    NUKATH – Notebook Universität  Karlsruhe (TH)   (0)
    OC – Organic Computing   (0)
    OTC – Organic Traffic Control   (0)
    OptRek – Optimierung auf rekonfigurierbaren Rechensystemen   (0)
    P-Opt – Portfoliooptimierung mit komplexen Nebenbedingungen   (0)
    PEdit – Programmierumgebung für parallele Systeme   (0)
    QE – Quantitative Emergenz   (0)
    SCP – Supply Chain Planung   (0)
    SESAM – Selbstorganisation und Spontaneität in liberalisierten und harmonisierten Märkten   (0)
    ULI – Universitärer Lehrverbund Informatik   (0)
    VIROR – Virtuelle Universität Oberrhein   (0)
    WWR – Wissenswerkstatt Rechensysteme   (0)
Knowledge Management   (37)
    (KA)2 – Knowledge Annotation Initiative of the Knowledge Acquisition Community   (0)
    AIFB SEAL – AIFB SEmantic portAL   (0)
    Bibster – Semantische Verwaltung von bibliographischen Informationen in einem Peer-to-Peer-System   (0)
    CKDD – Conceptual Knowledge Discovery   (0)
    Candle – Collaborative and Network Distributed Learning Environment   (0)
    DAML – OntoAgents – Enabling Intelligent Agents on the Web, based on Darpa Agent Markup Language   (0)
    Dot.Kom – Designing adaptive infOrmation exTraction for KnOwledge Management   (0)
    GETESS – German Text Exploitation and Search System   (0)
    Graduiertenkolleg IME – Informationswirtschaft und Market Engineering   (0)
    IBROW – An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web   (0)
    KAON – KAON – The Karlsruhe Ontology and Semantic Web Tool Suite   (0)
    KAON WS – KAON Web Services Tool Suite   (0)
    KAON2 – KAON2 OWL reasoner   (0)
    KnowledgeWeb – KnowledgeWeb -- Realizing the Semantic Web   (0)
    MIKE – Modellbasiertes und Inkrementelles Knowledge Engineering   (0)
```

Figure 4.10: The tree displaying the distribution of the projects carried out by the research groups.

dataset[12] which deals with the visualisation and the comparison of trees.

The Graph Paradigm

The *graph view paradigm* displays the view by representing each object and attribute as a node and each relation as an edge in the graph. For example, the graph representing the carriedOutBy relation presented in the previous section (i.e. Section 4.2.1) is shown in Figure 4.11. Graphs structures are frequently used to represent information (see for example, [Sowa, 1984, Sowa, 2000, Munzner, 2000]). Graph visualisation is a field in itself and the literature on the subject is quite extensive (many algorithms are described in [Battista et al., 1999, Munzner, 2000]).

The graph view has some advantages:

- multiple relation displayed at the same time
- intuitive representation
- display of relations
- diverse layout strategies

Contrary to the tree paradigm, which is not suitable for complex structures, the graph paradigm may visualise relations between sets of elements.

[12]The datasets and the results can be downloaded from http://www.cs.umd.edu/hcil/iv03contest/datasets.html.

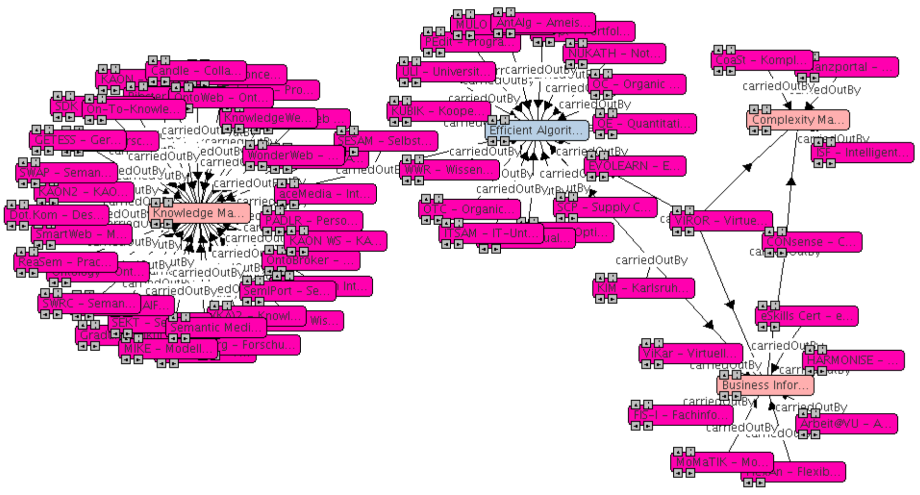


Figure 4.11: The graph displaying the distribution of the projects carried out by the research groups.

The Lattice Paradigm

The *lattice view paradigm* used in this thesis should actually be called concept lattice view paradigm since the lattices displayed are the concept lattices of the realised formal context of some context index. The traditional technique to display partially ordered sets is to use a line diagram though other techniques have also been proposed in the literature (see for example [Carpineto and Romano, 2005]).

While a line diagram can be seen as a specialisation of a graph, the underlying assumptions and interaction features differ greatly from the more generic graph approach.

The main common aspect with the graph is that both paradigms require some appropriate layout algorithms to draw their structures. As in the case of the graph, different approaches have been considered for the layout of the line diagram (see [Cole, 2001a, Cole, 2001b]).

A closer approach is to consider the visualisation of partially ordered sets. Line diagrams are useful for all partially ordered sets, yet we use concept lattices to display information.

First of all, every partially ordered set can be embedded in a concept lattice which is the smallest complete lattice for this set. This is called the Dedekind MacNeille completion. For a parially ordered set $(P, \leq)$, the formal context $(P, P, \leq)$ can be built and the formal concept for these contexts are all the pairs (X, Y), where X is the set of all lower bounds of Y and Y is the set of all the upper bounds of X. The size of this lattice can much larger than the size of the original set.

Concept lattices offer a number of interesting properties which make them more interesting to display information. An important feature of concept lattices is that each node forms a cluster of objects and a cluster of attributes. This means that it is easy to access a great number of elements by clicking on only one element.

In addition to representing clusters, the line diagram of a concept lattice also provides supplementary information. First, the cluster indicate in a compact manner which elements share properties (attributes can be seen as properties). The graph also displays this information, but if many different kinds of relations exist, the number of elements displayed explodes, and it cannot be visualised properly.

For example, the lattice displayed in Figure 4.12 shows how the project distribution of the different research groups. The labels for objects are not displayed but they are easily accessed using a preview mechanism.

Using a concept lattice it is possible to visualise which objects share more attribute than others (or the dual property, i.e. which attributes share more objects). This information can be displayed using the size of the nodes, but the place in the partial order gives also informations, since the highest objects share all the object of the lower objects (as long as their are comparable) and will have suplementary ones. In Figure 4.12, the size of the node gives an idea of the extent of each node, i.e it indicates the number of projects of a research group or the one it shares. The label gives the procentual distribution of research projects for each node.

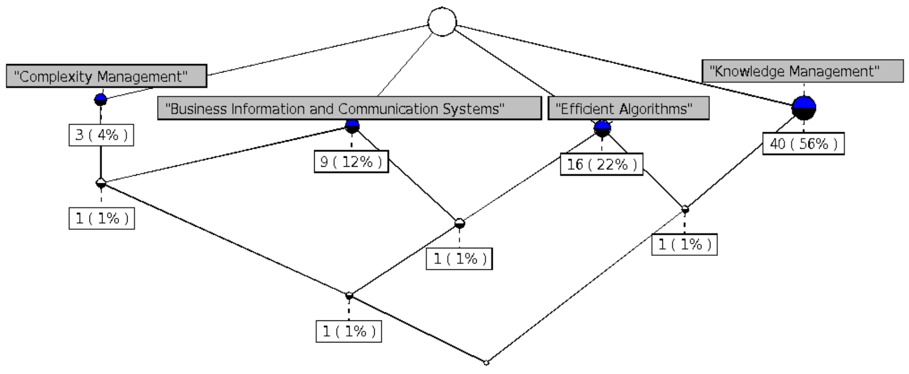


Figure 4.12: The lattice displaying the distribution of projects carried out by the research groups.

4.2.2 Selection

As noted in Section 4.1.2, the exploratory approach is extremely dependent on the means of specifying the next view to be visualised, in particular, the user must be able to specify the elements which are used in the definition of this new view. We discuss in the following paragraphs diverse means of selection which can be used in the views.

It is important to note that the selections we consider in this section consist each of a set of entity. These sets of entities may be typed in order to be able to offer appropriate context menu operations. Moreover, in order to support the user in the selection process, it is crucial to offer him the possibility to visualise the current selection. The panel used to visualise this selection must also allow direct manipulation.

Selection Types

Before discussing the main selection types of the different visualisation paradigms, we briefly introduce the main selection mechanisms. In both approaches, mainly two methods can be used to selected entities. Using the first method, the user can incrementally add entities to the selection. The second approach relies on the topological properties of the corresponding view together with a feedback mechanism based on the highlighting mechanism.

In all paradigms a number of entities are displayed on the screen. In the tree paradigm, these entities are the are the nodes of the tree. In the graph, edges can additionally be selected. The entities of the tree and graphs are entities present in the knowledge base. On the contrary, the artifacts which can be directly selected in the lattice paradigm are mainly formal concepts. Since formal concepts cannot be directly used for the definition of queries, it is crucial to provide a mean of selecting entities from them.

In all three approaches, these artifacts may be selected incrementally by clicking on each artifact, or a more topological approach may be used.

Lattice-based Selection

One of the main reasons for using the line diagram of a concept lattice is that a single click allows the selection of a great number of elements, since each node represents at the same time a set of attributes and a set of objects. Moreover, it is important to recall that the concept lattice structure has two interesting properties:

1. The extent of the greatest lower bound of any set of formal concepts coincides with the intersection of the extents of the elements of the set.

2. The intent of the smallest upper bound of any set of formal concepts coincides with the intersection of the intents of the elements of the set (dual property of 1).

These properties are direct interpretations of the fundamental theorem of Formal Concept Analysis (see Chapter 2, Section 2.3.1, or the book by Ganter and Wille [Ganter and Wille, 1999]). However, though these properties can be useful, they are also the source of the complexity of the line diagram of concept lattices. We assume that users are able to interact with this visualisation mechanism. Moreover, we claim that power users, who understand the nature of the lattice, can answer certain questions quickly and easily. A reason for this is that different sets of elements can be selected from the concept lattice with two simple clicks: the first one corresponding to mode selection and the second one to node selection as explained in the following.

In order to give the user a more powerful selection mechanism, an advanced user can choose from a set of selection modes. Depending on the current lattice selection mode, the user selects a part of the currently displayed lattice which suits his purpose. The interface offers eight lattice selection modes, which can be divided into two groups: the *entity selection modes* and the *concept selection modes*.

1. The four entity selection modes are:

   (a) *Attributes*: selects the attributes of the chosen formal concept,
   (b) *Objects*: selects the objects of the chosen formal concept,
   (c) *Attribute contingent*: selects the context attributes, which are in the attribute contingent of the chosen formal concept (i.e. the attributes belonging to this formal concept, but no formal concept above it.), and
   (d) *Object contingent*: selects the objects in the object contingent of the chosen formal concept (i.e. objects belonging to the extent of this formal concept but to none of the formal concepts below it).

2. The four concept selection modes are the following ones:
   (a) *Concepts*: selects the chosen formal concept,
   (b) *Filter Concepts*: selects the formal concepts belonging to the filter of the chosen formal concept (the formal concepts that are higher than the selected element in the lattice),
   (c) *Ideal Concepts*: selects the formal concepts belonging to the ideal of the chosen formal concept (the formal concepts that are lower than the selected element in the lattice), and
   (d) *Neighbours*:[13] selects the neighbouring formal concepts to the chosen formal concept.

While the elements selected in the first four selection modes are entities, i.e. elements of the universe Ω, in the last four, the selected elements are formal concepts, i.e. nodes of the lattice.

As their names indicate, the first two entity selection modes select objects and attribute sets of a node. The meaning of the other selection modes might be more difficult to grasp. The attribute contingent of a formal concept consists of all the attributes shared by all the objects of the extent of the formal concept but not shared by any smaller subset. The object contingent is the dual property of the attribute contingent, in other words it contains all the objects shared by all attributes of the formal concept but not shared by any subset of the intent.

The four formal concept selection nodes (i.e. the selection nodes in 2.), select some subsets of the set of formal concepts and are based on structural aspects of the lattice.

In this thesis, we focus on the use of the four entity selection modes, but we mention the four formal concept selection modes because we refer to them when dealing with future query-based multicontext browsing investigations in Chapter 7 Section 7.2.1.

For example, if the user wants projects shared by the three research groups Efficient Algorithm, Complexity Management and Business Information and Communication Systems, he could select them by choosing the mode (usually the default one) *Objects* and clicking on the node located at the crossing of the descending paths from the nodes labelled Efficient Algorithm, Complexity Management and Business Information and Communication Systems (see Figure 4.12, page 128).

Performing the same task using the tree view and graph view is much more cumbersome.

[13]In our application we defined neighbours as the formal concepts which are the direct neighbours in the lattice of this formal concepts.

Tree-based Selection

The tree selection offers a precise and simple mean for selecting elements. This selection paradigm also comes in two flavors depending on whether one or more elements may be selected. In both cases, the user browses the elements of the tree and may select one (or more in the multi-element mode) parent or child appearing in the tree. This mean of selection may also be enhanced by using sorting mechanisms for the display of the parents or of the children. For example, the elements may be sorted alphabetically allowing a faster search for the most relevant elements.

Graph-based Selection

The graph selection modes are mainly of two types. Both types correspond to some notion of closeness. The first types may use neighbouring information on the graph to highlight and select elements sharing properties. The second main approach to selection in the graph is the use of the proximity of the elements on the plane. Using the mouse to draw geometrical figures such as rectangles and circles, the user may select all the elements located in the perimeter of the figure. In some layout, the zones defined by these figures may contain elements which should not have been picked and may have to be then removed from the result of the selection. Observe that this second approach can be also easily adapted to the tree and lattice view.

The Type Selection Mode

Whereas the lattice selection mode only considered elements of the lattice or the context entities, the type selection mode lets the user restrict the type of the selected elements to certain types. This mode can be useful in all the three paradigm and is one of the benefits of using a typed infrastructure.

When browsing knowledge bases, the possible types of selections are:

- *Concepts*
- *Instances*
- *Properties*
- *Property Instances*
- *or any combination of these basic types*

The different mode settings can be set by the user before selecting elements. For our example, a user interested in actual publications may select only instances and set the filter accordingly. Figure 4.13 shows a small panel where the user can set the modes he needs at the moment. The type selection mode corresponds to the checked boxes, while the lattice selection mode is set using the pull down menu in the middle. The button “Clear Selection” on the right clears the selected elements from the current selection.

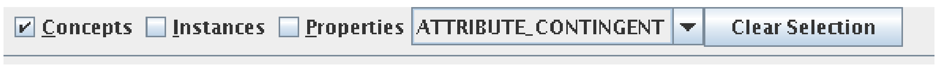


Figure 4.13: The selection settings panel.

4.2.3 Paradigm Comparison

Now that we have introduced the three visualisation paradigms, we discuss their respective differences, limitations and advantages. Table 4.3 summarises these advantages and limitations which we discuss in the following paragraphs.

Intuitive Views

The notion of intuitivity is difficult to define. People often have to learn to use tools. For example, the mouse is a tool most computer users use widely. But, who ever taught new computer users notices quickly that some users require some time to get acquainted to. This is also true when comparing visualisation paradigms. We performed a preliminary[14] evaluation which showed the need for a basic training for the lattice approach, while the use of the graph approach required to get acquainted with the possibilities of the paradigm. Without preliminary training, the tree view tended to outperform the other paradigms. We suppose that the reason for this is the omnipresence of the tree view as interaction paradigm. Indeed, the tree view is well known due to its use in the file managers of computers.

However, the final evaluation of the interaction presented in Chapter 6, Section 6.2, showed that if users were required to undergo a brief training phase, the lattice view could outperform the tree view on some tasks.

Screen Space

As observed in [Shneiderman and Plaisant, 2005], screen space is often a costly resource, either because the computer screen is small in itself, or because some other task must be performed in parallel with the use of the browsing paradigm. In these cases, the tree paradigm is the most suitable since it requires little space. In particular cases, such as the views presented in Figures 4.10, 4.11 and 4.12, the lattice view is as compact as the tree view. Some strategies to minimize the size of the structure to be drawn has been presented in [Aeschlimann and Schmid, 1992].

Structural Hints

In most applications, a tree view is well suited since it requires little space and users are usually accustomed to it. However, the purpose of a visualisation paradigm is often to help users notice some particular aspects of the data. Some of the paradigms are more suitable for certain tasks than others. For example, on the one hand, the lattice is more suitable for displaying shared properties than the tree view or the graph view. This does of course not come as a surprise, since the lattice structure reflects exactly this structural aspect.

[14]The evaluation was preliminary, because we noticed quickly that users using the lattice and graph view required some basic training in order to perform as well as with the tree view.

Table 4.3: Table summarising the advantages and disadvantages of the three interaction paradigms for views.

Paradigm	Tree	Graph	Lattice
User acquaintance	high	medium	low
Intuitive	yes	yes	no
Screen space	restricted	important	important
Layout	simple	complex	complex
Predictable	yes	not always	not always
Multiple inheritance	no	yes	yes
Multiple relations	yes	yes	no
Outlier detection	no	yes	yes
Shared elements	no	yes	yes

On the other hand, the graph visualisation approach is more suitable to display multiple relationships. The Conceptual Graphs of John Sowa (see [Sowa, 1984, Sowa, 2000]) or UML diagrams (see [Knöpfel et al., 2006]) are good examples of this.

Table 4.3 shows that the tree view has many advantages over the other paradigms. But, it is not suitable to display some structural aspects of a view: shared elements, multiple inheritance and multiple instantiation as well as outliers.

4.3 View Definition

We motivated in Section 4.1 the importance of the means of defining views for our exploration approach. The purpose of this section is to investigate several strategies for simplifying the definition of views. We first give an overview of the possible strategies before developing each of them.

4.3.1 View Definition Strategies

The views which may be defined using the query-based multicontext theory may vary from very simple ones (for example using a default relation constructor as shown by Example 10 in Chapter 3, Section 3.5) to more complex ones involving combinations of intersection, unions and role operators.

According to [Shneiderman and Plaisant, 2005],[15] query definition methods belong to one of the following approaches: *command language*, *form fillin*,[16] *menu selection*, *direct manipulation* and *natural language*. The command language approach consists in defining a textual language for defining queries. In the form fillin approach, a user fills the parameters of a query template. The direct manipulation approach is based on the idea of manipulating objects to perform diverse actions, for example drag and drop actions. Finally, the natural language approach to query definition consists in letting user express their query in natural language. Tables 4.4, 4.5 and 4.6 (originating from [Shneiderman and Plaisant, 2005]) summarise the advantages and drawbacks of these methods. The above distinctions is especially important for our approach since these methods can be used not only for query definition but also for view

[15]This taxonomy of query definition approach was first presented in [Shneiderman, 1997b].

[16]Note that the term used by Shneiderman is form fillin as opposed to *form filling*.

Table 4.4: Table recalling the diverse query definition paradigms (part A, cited from [Shneiderman and Plaisant, 2005]).

Method	Advantages	Disadvantages
Command language	• Flexible. • Appeals to expert users. • Supports the definition of user-defined "scripts" or macros. • Is suitable for interacting with networked computers even with low bandwidth.	• Retention of commands is generally very poor. • Learnability of commands is very poor. • Error rates are high. • Error messages and assistance are hard to provide because of the diversity of possibilities plus the complexity of mapping from tasks to interface concepts and syntax. • Not suitable for non-expert users.

definition. Each of these methods is suitable for a given situation. Therefore, we implemented all but the natural language approach which could be integrated with a query answering framework such as ORAKEL (see [Cimiano, 2004]).

In [Tane, 2005], [Tane, 2004] and [Tane et al., 2006], we used the form fillin and menu selection to ease the knowledge exploration. These approaches are described in Section 4.3.3 for the form fillin and in Chapter 5, Section 5.5.1. The direct manipulation method has not been described yet and its way of working is sketched in Section 4.3.3. The command language cannot really be seen as a browsing approach, but remains a useful method of creating views and this approach was used in the visualisation evaluation to create the necessary views. We describe it in the next paragraph.

Command Language

The command language approach has been implemented on top of a SQL database as a commandline tool for an early prototype of the query-based multicontext architecture. A user of the system could write in a file[17] the three queries of a context index and the corresponding context index was created. This approach for view definition is more suitable for prototyping purposes, batch applications. The logging of the context indices used in a navigation session can also be used to keep track of the last view visualised.

We now show how the command language approach can be used to defined complex views for applications.

Example 17 (Command Language Example) *One of the early prototypes of the query-based multicontext approach used files to create complex formal context. In order to generate the formal context indicating the publications of professors, the context index*

[17] A text file has the advantage that the content can be easily edited, generated or even exchanged.

Table 4.5: Table recalling the diverse query definition paradigms (part B, cited from [Shneiderman and Plaisant, 2005]).

Method	Advantages	Disadvantages
Form Fillin	• Simplifies data entry. • Shortens learning in that the fields are predefined and need only be 'recognised'. • Guides the user via the predefined rules.	• Consumes screen space. • Usually sets the scene for rigid formalisation of the business processes.
Menu Selection	• Ideal for novice or intermittent users. • Can appeal to expert users if display and selection mechanisms are rapid and if appropriate "shortcuts" are implemented. • Affords exploration (users can "look around" in the menus for the appropriate command, unlike having to remember the name of a command and its spelling when using command language.) • Structures decision making. • Allows easy support of error handling as the user's input does not have to be parsed (as with command language).	• Too many menus may lead to information overload or complexity of discouraging proportions. • May be slow for frequent users. • May not be suited for small graphic displays.

$$R(<\mathsf{author}>, \mathsf{role}(<\mathsf{author}>, \{\mathsf{Publication}\}),\ inst(\{\mathsf{Professor}\}))$$

can be created using a file with one query per line as:

$\mathsf{role}(<\mathsf{author}>, \{\mathsf{Publication}\})$
$inst(\{Professor\})$
$<author>$

An application can generate the corresponding context as well as the realised lattice.

The command language approach is particularly adapted for distributed applications as well as for data mining purposes, where other tools can request views from a server using the defined command language.

While the user may be able to simply define some views by selecting an action from a menu, more expressive views require a more complex view definition

Table 4.6: The table recalling the diverse query definition paradigms (part C, cited from [Shneiderman and Plaisant, 2005])).

Method	Advantages	Disadvantages
Direct Manipulation	• Visually presents task concepts. • Easy to learn. • Errors can be avoided more easily. • Encourages exploration. • High subjective satisfaction.	• May be more difficult to program. • Not suitable for small graphic displays. • Spatial and visual representation is not always preferable. • Metaphors can be misleading since the “the essence of metaphor is understanding and experiencing one kind of thing in terms of another” (Lakoff and Johnson 1983: p. 5), which, by definition, makes a metaphor different from what it represents or points to. • Compact notations may better suit expert users.
Natural Language	• intuitive • no formal language to learn • high expressivity • no knowledge about schema required	• low correctness • ambiguity • initial learning effort (effort at guessing the limits of the system)

process. The goal of this section is to expose the important issues for the view definition process. In the first part of this section, we present *default views* which are views easily defined using some simple query as parameter. In the second part of this section, we deal with more complex view definition strategies. We also highlight some open issues which have not been implemented but would greatly enhance the quality of the views created.

4.3.2 Default Views

Default views are very simple views which require very little or no parametrisation. Due to this minimal requirement, they are especially suitable for the menu selection approach to view definition. In that case, the user can activate the view by activating a button, a menu item or a link. If parametrisation is required, it can be either hard coded (in the link or menu item's action) or directly inferred from the situation.

Default views usually take only one parameter or no parameters. This makes them suitable for creating a browsing framework in a web browser. The idea of using default views was our starting point for our Formal Concept Analysis browsing framework. It was first implemented in the Courseware Watchdog,[18] which we briefly described in Section 4.1.2 (see also [Tane et al., 2004, Tane et al., 2003, Schmitz et al., 2002]). Though the approach allowed to browse the ontology to some extent, using exclusively the default views approach restrict the navigation process to very simple views. One of the reasons for this is that some query operators or constructors require at least two parameters. This is particularly true of the role query operator, though it is sometimes possible to use information on the context to give default values to the query parameters

In our approach the main method to parametrised default views is the use of a context menu on the elements of the visualisation. Depending on the kind and number of elements selected, the actions proposed differ. As seen in Section 4.2.2, a number of entities may be selected in an entity or topological manner in the tree and graph paradigms. In the lattice paradigm, the concept part selection mechanism can be used to create selection based on criteria.

Most of the relevant default views in the context of a knowledge base exploration framework are dependent on the type of element used to parametrise. We first consider the case where the elements selected are elements of the knowledge base, i.e. concepts, relations or instances. Then we discuss the use of context operators.

Concept Views

From Chapter 3, Section 3.5, the constructors of Sections 3.5.2, 3.5.3 and 3.5.4 play an important role in browsing knowledge bases. The subsumption hierarchy is usually a powerful means of grouping related elements. Most knowledge representation frameworks contain at least one hierarchy primitive for classes or concepts. Whereas in RDF(S) (see [Hayes (editor), 2004]) the subsumption relations properties or classes can be given, more complex subsumption relations can be specified in more expressive ontology languages. For example, the OWL language allows to create concept definition using roles.

Relation Views

For a given set of relations, it is possible to display three different views: the relation concept lattice, its inverse or the lattice of subrelation of this relation. In other words, for a selection q_r consisting of relations the three kinds of concept lattices which can be built correspond to the following context indices:

- $\mathrm{R}(q_r)$
- $\mathrm{R}(q_r^{-1})$
- $\mathrm{S}(q_r,\mathsf{subrelation})$

The first of the three relation default views builds the context index of the default relation constructor (see Chapter 3, Section 3.5.1, page 94). The second constructor applies the same relation constructor but the inverse operator $.^{-1}$ is applied on the query q_r. The third operator however builds the subsumption hierarchy for the relations resulting from q_r.

[18]See `http://cwatchdog.sf.net`.

Instance Views

Though the other default views are often useful, default views are particularly interesting when the selection is a set of instances. There are mainly two reasons for this. The first reason for this is that one of the main purposes of visualisation is to obtain some kind of model or categorisation of sets of instances. Most importantly, two kinds of views are particularly relevant: concept categorisation and role categorisation. The second reason is that once the user selected a number of instances from the extent of a concept, he can use these two default views to categorise the elements of this extent further.

Constructing Views Using Context Operators

Another possibility to easily construct views is to use context index operators as defined in Chapter 3, Section 3.3.3. Since two of these operators: the dual operator and the complement operators, require only one parameter, both can be directly applied on the current context index. However, context index operators can be used appropriately on two or more context indices.

For example, imagine two views defined using the following context indices p_1 and p_2.

$$p_1 := \begin{pmatrix} \mathsf{role}(<\mathsf{isWorkedOnBy}>^{-1}, \{\mathsf{text\text{-}mining}\}) \\ \mathsf{role}(<\mathsf{isWorkedOnBy}>, \mathsf{role}(<\mathsf{isWorkedOnBy}>^{-1}, \{\mathsf{text\text{-}mining}\})) \\ <\mathsf{isWorkedOnBy}> \end{pmatrix}$$

$$p_2 := \begin{pmatrix} \mathsf{role}(<\mathsf{isWorkedOnBy}>^{-1}, \{\mathsf{text\text{-}mining}\}) \\ \mathsf{role}(<\mathsf{carriedOutBy}>, \{\mathsf{Knowledge\ Management\ Group}\}) \\ <\mathsf{worksAtProject}> \end{pmatrix}$$

If these two views are presented in a list (for example, a list showing the views which have been previously defined), the user interface can offer the possibility for the user to appose the two resulting contexts.[19] This method is particularly suitable to generate scales or the many-valued contexts presented in Chapter 2, Section 2.3.3. The traditional technique of using nested line diagrams can then be used for visualising of the resulting lattice.

4.3.3 View Parametrisation

Since default views lack expressive power due to their requirement for simplicity, a methodology for defining more complex views is necessary. Our view definition methodology is based on the use of the constructors which were introduced and defined in Chapter 3, Section 3.5. In particular, we saw that the generic definition of a context index can be seen as a constructor with three parameters: the object query and attribute query and the relation query. This means that the general view definition case can be seen as a special case of constructor definition. The user of the framework can set the parameters of a given constructor and trigger its evaluation. The result of this evaluation is a new view.

[19]The apposition operator can be used since the requirement that the object queries of these context indices are equals, is satisfied.

The process of view parametrisation can be implemented in two orthogonal manners: *view guided parametrisation* or *parameter oriented parametrisation*. Each of these parametrisation methods is useful in some situation. On the one hand, view guided parametrisation corresponds to a process where the type of view to be created guides the way the user can parametrise the view. We present shortly an example where the user is guided in parametrising a join constructor. On the other hand, parameter oriented parametrisation consists in setting values to the parameters of a constructor independently of its nature. This may be in some cases necessary because the guided process may restrict the kind of query which can be used as parameter.

To give an example of the parametrisation process, the relation constructor has three parameters. On the graphical interface, the user selects the relation constructor. For each of the constructor parameters a corresponding button is displayed on the user interface. The state of the buttons indicates the currently selected parameter of this constructor (per default the first). The user may then either select an action from the context menu of the selection, he may drag and drop a selection to one of the parameters. This example illustrates that either the menu selection or the direct manipulation approach may be used in the view definition process.

Form Fillin

The purpose of form fillin is to help the user in setting up the parameters of the context indices. It can be seen as a basic mean of parametrisation, though it may be used in combination with default views approach by fixing a number of parameters to be evaluated. The form fillin approach can also be used together with other query definition approaches. In particular, our constructor parametrisation can be seen as a process mixing direct manipulation and form fillin.

The principle of form fillin is to present to the user a number of predefined fields with clear meanings. Form fillin approaches are typical of library user interfaces which allow users to set up the parameters of the search.

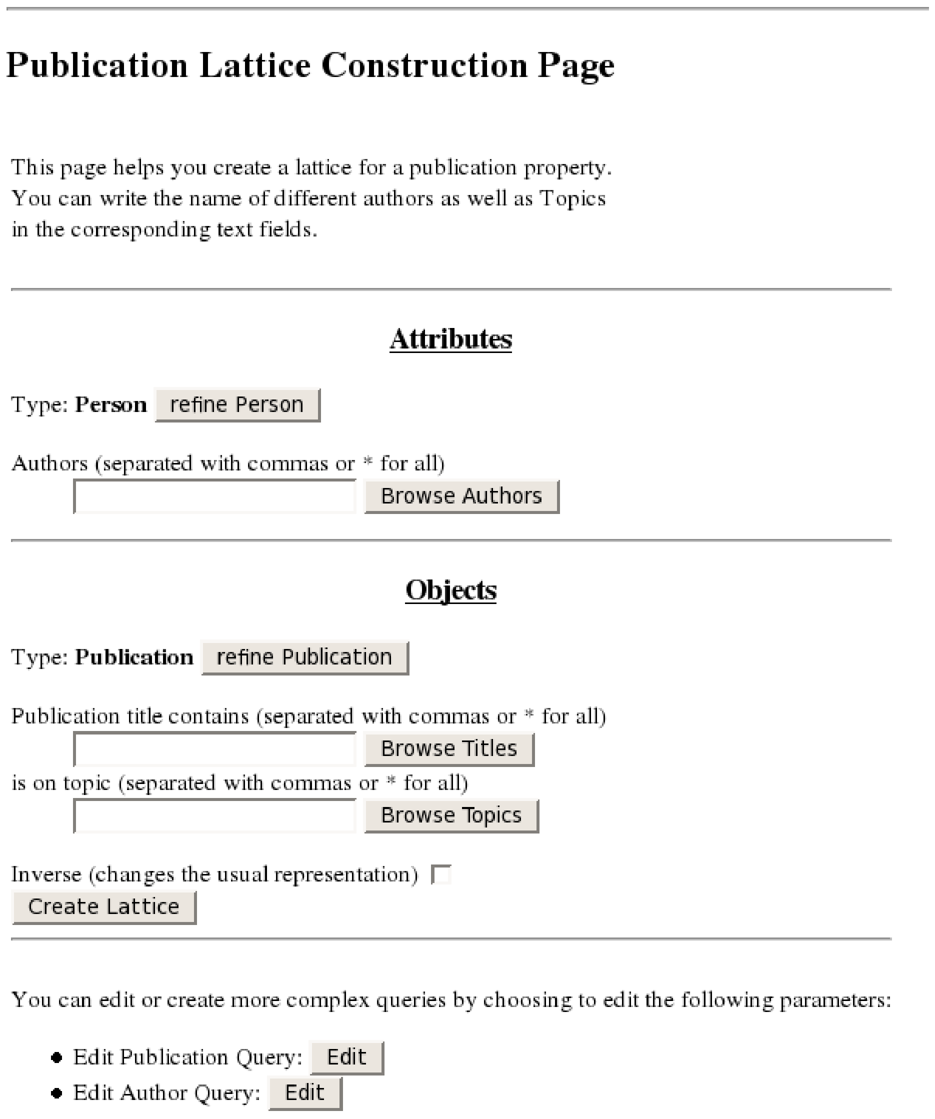
The small form presented in Figure 4.14 allows the user to define a lattice of publications for a number of authors which interest him. The page is divided into four parts. First a general explanation is given to the user of the diverse possibilities.

The second part is an attribute section allowing him to either refine the type Person (using the “refine Person” button) or write a number of author names to be selected as attributes for the lattice. The names of the authors can also be selected using the “browse Person” button.

The third part is the object section which allow the user to choose the publication which interest him. As for the attribute type, the type of publication can be refined. He can also specify words which can occur in the title of the publications. He can also choose to set some topics or to browse the titles or topics to set the query.

Finally the user can decide to edit the queries individually using specific query pages. Using the entries given, corresponding queries can be defined, for example a query $q_{authors}$ as well as a query q_{topics}.

By pressing the “Create Lattice” button, the corresponding context is generated using the names of the authors to retrieve publications and organised


Publication Lattice Construction Page

This page helps you create a lattice for a publication property.
You can write the name of different authors as well as Topics
in the corresponding text fields.

Attributes

Type: **Person** refine Person

Authors (separated with commas or * for all)

Browse Authors

Objects

Type: **Publication** refine Publication

Publication title contains (separated with commas or * for all)

Browse Titles

is on topic (separated with commas or * for all)

Browse Topics

Inverse (changes the usual representation) ☐

Create Lattice

You can edit or create more complex queries by choosing to edit the following parameters:

- Edit Publication Query: Edit
- Edit Author Query: Edit

Figure 4.14: An example of web formular to create a simple context of author publications.

according to their authors. The publications retrieved can then be used to restrict the set of publications to a number of desired topics.

The resulting context index is an application dependant constructor (see Chapter 3, Section 3.5.7, page 101) for the principle behind this kind of constructors.

This application constructor is a mapping returning

$$R(q_{\mathsf{authors}}, <\mathsf{publication}>, \mathsf{role}(<\mathsf{hasTopic}>, q_{\mathsf{topics}}))$$

if both queries q_{authors} and q_{topics} have been given and

$$R(q_{\mathsf{authors}}, <\mathsf{publication}>)$$

if only the query q_{authors} and finally

$$R(<\mathsf{publication}>, \mathsf{role}(<\mathsf{hasTopic}>, q_{\mathsf{topics}}))$$

if only q_{topics} has been filled in the entries.

The example given shows that the form fillin approach is particularly interesting in combination with an application dependent constructor. It is possible to create customised forms which can be easily filled in order to display customised views. These forms can also be intelligently designed by proposing different completions and informations to help the user select the elements.

Direct Manipulation

Direct manipulation can be used in different ways to create queries and context indices. The Drag-and-Drop action of the user is common to these different methods. The semantics of a direct manipulation action depend on the elements selected and the receiving area of the drop part of an action. A Drag-and-Drop action is applied to selections. In our applications, most selections are sets of elements. This enables the user to select more than one element in a list of elements. For example, the researcher of our scenario may retrieve the publications of a group of researchers and selects the ones which seem most relevant for his research and drag and drop them on the target panel of the interface.

In the parameter oriented parametrisation, the receiving part of the drag is the parameter to be set, whereas in the view guided parametrisation, the user drags the elements to a given constructor of the view he wishes to build.

Parameter Oriented Parametrisation

Parameter oriented parametrisation is the simplest mean of parametrisation. After the user selected some elements, he can use this selection on one of the parameters of a constructor. Depending on the type of the parameter, and on whether this parameter has already been set, different result may occur. For example, when dropping a set of relations on a relation parameter, the relations can be added to the existing relations if it is not empty or a new relation query can be created. If the parameter is not a relation query parameter, then the interface suggest the domain or the range of the relation.

To illustrate this, we consider the CoRelation constructor presented in Chapter 3, Section 3.5.6. It takes two parameters:

- a query returning the entities to start from (i.e. a query $q_1 \in L_1$), and
- a relation query.

The user selected a set of instances q_I of the concept Person. For example, the user selects $\{\mathsf{Julien\ Tane}, \mathsf{Philipp\ Cimiano}\}$ and drags this selection on the start entities parameter. The empty parameter is immediately set with q_I. The user then selects the concept Person and drags it to the relation query parameter. There a popup window is opened suggesting a list of alternative relations which have been obtained using the $\mathsf{relations}(\{\mathsf{Person}\})$ query operator. This operator returns the set of relations which have a concept or a subconcept of Person in their signature. The user may then choose the relation isWorkedOnBy and set the parameter. Note that no check is performed, that this parameter is not contradictory to the other parameter, only the parameter itself is taken in consideration.

View Guided Parametrisation

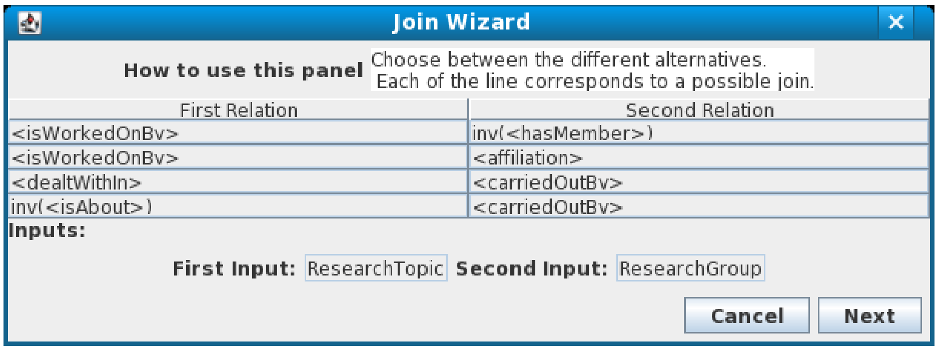
View Guided parametrisation consists in offering guidance to the parametrisation of constructors. In Chapter 3, Section 3.5, we introduced a number of constructors as well as a more generic definition of application dependent constructors. We first explain the principle using an example with a join constructor.

Suppose a user is interested in visualising a view which needs to be defined with the join constructor (see Chapter 3, Section 3.5.5). To simplify its construction only two relation queries must be provided by the user, the underlying queries are assigned default values which can later be refined by users.

The constructor definition from Chapter 3, Section 3.5, defines the join constructor with five parameters: two relation queries and three set queries. In our implementation, only the two relation queries of these five queries are needed, the other queries get default values, defined as follows:

- the object set query: the domain of the first relation
- the attribute set query: the range of the second relation
- the intersection of the range of the first relation with the domain of the second relation

We now illustrate how the join constructor can be parametrised.



How to use this panel: Choose between the different alternatives. Each of the line corresponds to a possible join.

First Relation	Second Relation
<isWorkedOnBv>	inv(<hasMember>)
<isWorkedOnBv>	<affiliation>
<dealtWithIn>	<carriedOutBv>
inv(<isAbout>)	<carriedOutBv>

Inputs:

First Input: ResearchTopic Second Input: ResearchGroup

Figure 4.15: The wizard allowing the creation of the join.

Example 18 *The user is interested in the research topics of the AIFB research groups. There is no relation between the two concepts* ResearchTopic *and* ResearchGroup*. However, these two concepts can be linked together using the join of two relations. To perform this, the user can select the two concepts and ask for the join construction wizard. This new window proposes a number of alternative joins for the given input In the present case, there are four alternatives which are presented to the user in the wizard shown in Figure 4.15. The distribution of the research topics per research group according to the two relations:*

<isWorkedOnBy> - <affiliation> is shown in Figure 4.16. Note that by pressing the next button the view oriented parametrisation continues, where the user can refine the middle query of the join.

More details on these processes is given in Chapter 5, Section 5.5.2. We present now another example of view guided parametrisation which is based on an application dependent constructor for visualising publications.

Example 19 *The researcher of our scenario may be interested in seeing the publications organised per year. We define a new constructor which takes a query returning the years as input, as well as a query returning persons and returns a context classifying the publications of these authors according to the years of publication:*

Input:

$q_{years} \in L_1$ *a query returning a set of years (defined by the user)*
$q_{authors} \in L_1$ *a query returning authors (given by the context of use)*

For example the user could select the pertinent years from a list, while the authors are selected by the topics they work on or the research group they are affiliated to.

Using these queries as input, the following application dependent constructor can be defined

$$(q_{years}, q_{authors}) \mapsto R\left(\begin{array}{c} <\mathsf{publication}> \\ \mathsf{role}(<\mathsf{year}>, q_{\mathsf{year}}) \cap \mathsf{role}(<\mathsf{author}>, q_{\mathsf{authors}}) \\ q_{years} \end{array}\right)$$

The main difference between the default views presented in Section 4.3.2 and this kind of view lies in the importance of the parametrisation. This parametrisation can be achieved in several manners.

4.3.4 View Adaptation

Frequently the view defined at first is not appropriate to be visualised directly as a lattice due to the very large number of formal concepts. For example, a lattice containing more than two or three hundred formal concepts is likely to be unreadable. In such cases, it is desirable to allow the user to refine the view definition so as to focus on a part of the data available in the view. There are two strategies for achieving this purpose:

- changing the view in order to focus on specific parts of the view, or
- hiding (and even sometimes removing) unnecessary elements.

Modifying the View

A user may have different reasons to modify the definition of a view. For example, a user looking at text-mining publications may want to focus more particularly on a number of authors or add supplementary topics.

Changing the view can of course be done by going back and redefine the view altogether. But once the view has been created, it is possible to guide the user in modifying the view in order to reduce or adapt the view of interest. We list here strategies which can be used for this purpose:

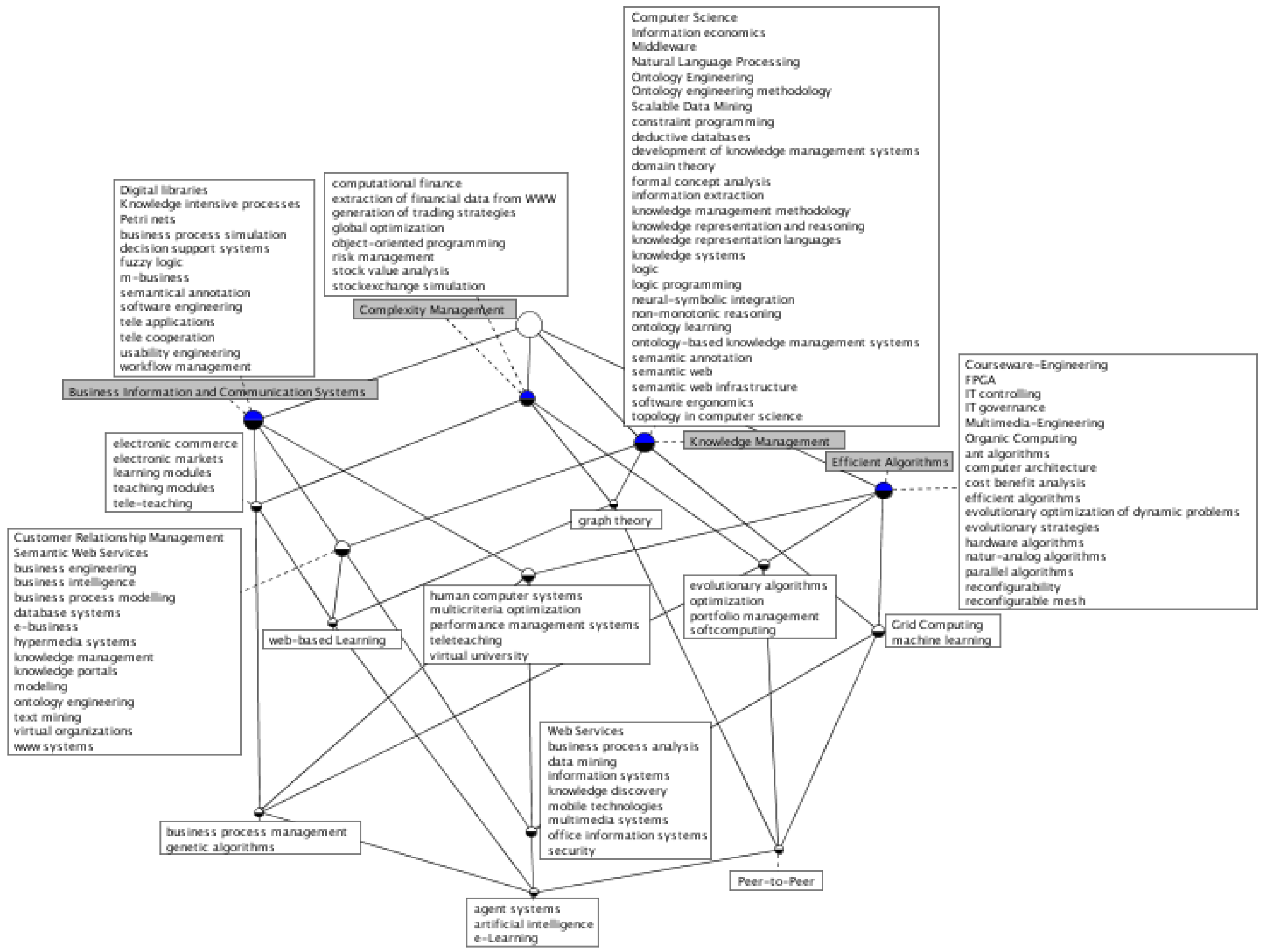


Figure 4.16: The join relation indirectly indicating the research topics of AIFB research groups by looking at which topic its members work on.

- transposing the view (i.e. objects become attributes and attributes objects, and the relation is inversed)
- complementation: the relation is complemented
- changing the object and attribute set
  - Removing elements
    - choosing subconcepts (if the view was defined using concepts)
    - restricting the parameters of the role queries
    - removing instances
    - adding an intersection argument (or create an intersection query with supplementary argument)
    - removing an argument of a disjunction or union query
  - Adding elements
    - choosing superconcepts (if the view was defined using concepts)
    - extending the parameters of the role queries
    - adding instances
    - add disjunction operand (or create a disjunctive query using supplementary operand)
    - removing some intersection arguments from the query
- changing of the relation
  - Removing elements
    - increasing the minimum frequency threshold (per default 1)
    - intersection with another relation
  - Adding elements
    - union with another relation
    - decreasing the frequency threshold (1 is the minimum)

These strategies are relevant because they theoretically and practically improve the usability of the view realisation and view visualisation processes.

We give here a simple example of some of these operations:

Example 20 *Consider the view defined by the three following queries:*

1. q_1 := role(<publishes>$^{-1}$,{Knowledge Management Group, Efficient Algorithm Group})
2. q_2 := inst({Person})
3. q_3 := <author>

In the ontology used by the AIFB portal, the formal context of this view contains 804 objects, 841 attributes and 848 formal concepts. This means that the view cannot be visualised as a concept lattice. [20] *To overcome the problem, the user can either redefine the view or adapt the view. To redefine the view, the user could replace the attribute query by a more specific query like* q_2' := role(<affiliation>, {Knowledge Management Group, Efficient Algorithm Group}).

He could also adapt the view in one of the following manners:

[20]The interface presents a message dialog informing the user to refine the context definition, for example by choosing restricting one of the parameters.

- *choosing subconcepts: replace Person by Professor, Assistant in q_2*
- *restricting the role query: choose only one of the research groups in q_1*
- *removing*[21] *instances: replace q_2 by*

$$q_2'' := \mathit{inst}(\{\mathit{Person}\}) \setminus \mathit{inst}(\{\mathit{PhDStudent}\})$$

The user decides to focus on the publications by only professors or assistants. He chooses the first alternative. The resulting lattice is shown in Figure 4.17. It is a much smaller than the preceding one and can be therefore more easily visualised.[22]

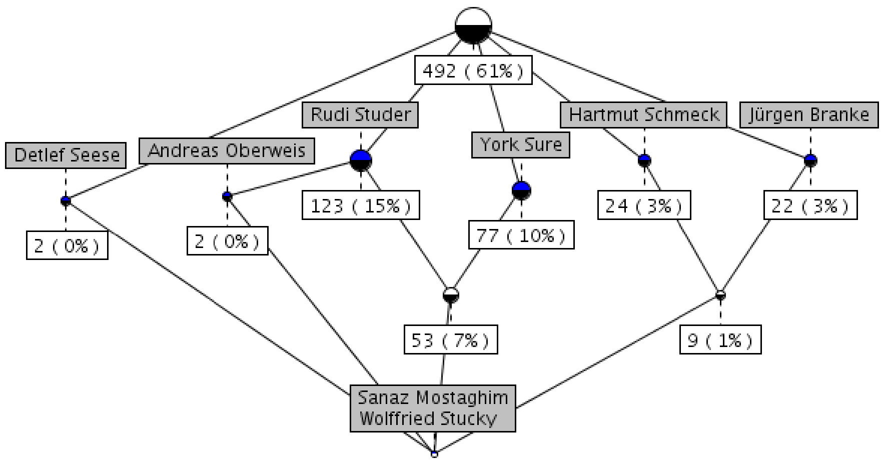


Figure 4.17: The adapted view displaying the publications assigned to the Efficient Algorithm Group or to the Knowledge Management Group and who were written by a Professor or an Assistant.

Other view adaptations have also been envisioned which are closely related to the interaction with the lattice. But their description and implementation is postponed to future work.

Hiding Elements

The idea behind the views presented until now is that the context index of the view describes the content of the view. If the actual view is actually difficult to read, an interesting possibility consists in hiding irrelevant or less relevant elements. By hiding elements, the readability of the view can be improved while still avoiding its redefinition. Depending on the actual visualisation paradigm different strategies can be used for this purpose.

In Chapter 5 Section 5.5.4, we present an algorithm to partition the set of top elements of the lattice. This has the advantage that larger concept lattices

[21]This is an application of the $\setminus$ operator on queries.

[22]Note that the two persons at the bottom at the lattice are not members of any of the two research group. They also have not coauthored any publication with any of the members of the two groups.

can be displayed in a more meaningful way. It is possible to generate the context index for the resulting classes of the partitions.

Other visualisation paradigm also enable the user to hide some elements. The graph view paradigm (see Section 4.2.1), has a feature allowing users to hide elements in order to improve the readability of the graph.

4.4 Summary

We discussed in this chapter different issues with respect to knowledge browsing and presented the novel approach we designed to explore knowledge bases.

We first discussed the main issues regarding knowledge browsing paradigms using views in Section 4.1. then we presented the corresponding browsing process and illustrated it using a scenario.

In Section 4.2, we dealt with the different means of visualising a view. Notably, we considered three different display paradigms.

Finally, we investigated in Section 4.3 different aspects of the view definition process. Notably, we identified different view definition methods and showed the role played by the usual query definition strategies in this process.

Chapter 5

Implementation

Chapter 3 and 4 dealt respectively with the underlying model and the user interaction for the browsing framework we developed. In the present chapter, we address the implementation issues of our framework. We first give an overview of its architecture (see Section 5.1) and then we present in the next three sections the main interacting components: the QBMC infrastructure (see Section 5.2), the context infrastructure (see Section 5.3) and the implementation of a query-based multicontext on top of the KAON API and KAON2 API (see Section 5.4). In the next section, we discuss technical details for the development of the graphical user interface (see Section 5.5). Finally, in the last section of this chapter (see Section 5.6), we discuss the choice of the underlying FCA implementation.

5.1 Architecture Overview

We start the description of our implementation by giving a brief overview of the architecture of our tool. Though the query-based multicontext infrastructure has been designed as a generic framework adaptable for other kinds of applications, we present it here only for the purpose of exploring knowledge bases. This section describes briefly the main modules of the query-based multicontext browser and the interactions they have with each other.

The second part of this section illustrates the way the tool works by giving a simple example of one iteration of the main cycle of the query-based multicontext browser which we already exposed in Chapter 3, Section 3.1.

5.1.1 The Main Components

The query-based multicontext browser has been designed as a modular application. The design of the query-based multicontext browser follows the principles of the *Model-View-Controller* design pattern (see [Burbeck, 1992]). The models of the infrastructure are formal contexts, lattices and query results. The views are the panels displaying these models and allowing interaction with the infrastructure. Finally, the controllers are software components which manage and manipulate diverse resources such as query results and formal contexts. Controllers use context indices and queries not only to identify and retrieve ressources but also as intensional representation to ease the manipulation through the operations we defined in Chapter 3. As mandated by the Model-View-Controller design pattern, the model and controller components are mainly independent of the GUI of the application and could be used for other purposes than knowledge exploration. We focus however in this chapter on the infrastructure for exploration purposes; the interested reader can refer to Chapter 7, Section 7.2.2 for further applications.

Figure 5.1 shows a functional view of the framework. It displays the main relations between the three main modules of the application: the graphical user interface, the QBMC infrastructure and the context infrastructure.

Overall the user interacts with the *graphical user interface* which transmits user request using events with the QBMC infrastructure. The *QBMC infrastructure* manages the resources available in the context infrastructure, i.e. query results, contexts and lattices. Finally, the *context infrastructure* contains the structures manipulated by the controller. The rest of this section gives a general overview of these components.

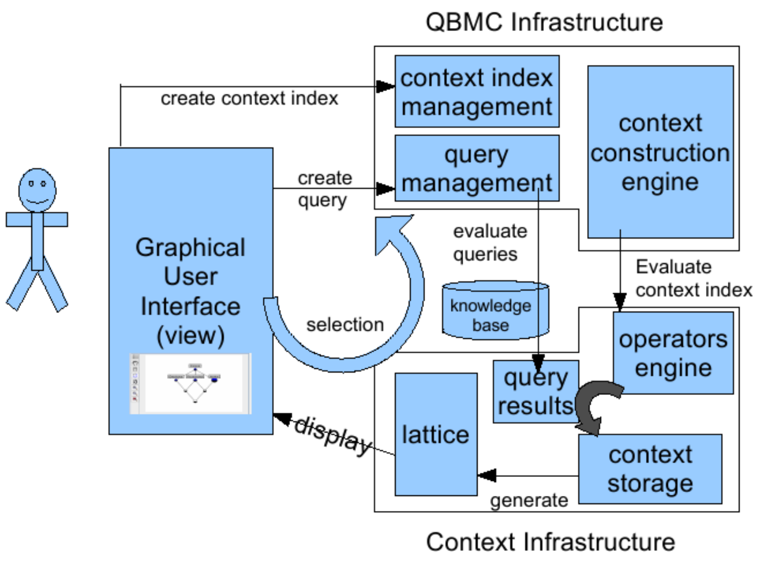


Figure 5.1: An overview of the architecture of the QBMC Browser.

The QBMC Controller

The *QBMC controller module* manages queries and context indices occurring during the browsing process. This module is composed of three submodules:

- the query management submodule,
- the context indices management submodule, and
- the context construction engine.

These three submodules are described in Section 5.2. The first two modules manage diverse intensional representations. The context indices management system stores the already evaluated context indices. These context indices are composed of existing queries. These queries are managed by the query management submodule.

The role of the context construction engine is to guide the construction of the contexts either using available constructors or from context indices. It takes care of the communication with the context infrastructure presented below to build the contexts. It plays a role similar to a query planning module in a database management system.

The Context Infrastructure

The role of the context infrastructure is to manage the construction and storage of formal contexts. We address in Section 5.3 the structure of this module. In particular we discuss the implementation aspects of creation, storage and operators.

This module consists of four submodules:

- the query results submodule,
- the operators engine,
- the context storage, and
- the lattice module.

The first module stores the results of queries while the context storage module stores the contexts. The operators engine takes care of the implementation of some context operations, such as transposition, apposition or subposition so that contexts can be stored as composite structures. Finally the lattice module contains the current lattice of the application.

The Graphical Interface

In Chapter 4, we discussed the issues to be dealt with when developing a query-based multicontext browser. We give in Section 5.5 more technical details on the implementation of the graphical interface of our tool. Some of these aspects have some impact on the usability of the interface but are not crucial for the principle of the user interaction. In Section 5.6, we expose the reasons determining the choice of the Formal Concept Analysis infrastructure underlying our implementation. Due to this design choice, we could not apply the query-based multicontext approach to nested line diagrams (these diagrams were exposed shortly in Chapter 2, Section 2.3.1).

The Knowledge Base Framework

The semantic query infrastructure presented in Chapter 3, Section 3.4.2 can be implemented using different knowledge base frameworks. We describe in Section 5.4 a methodology to implement a semantic query infrastructure using an existing knowledge base framework and discuss the properties of two different implementations.

The Underlying FCA Framework

Finally, there is still another important element which is implicit in Figure 5.1. Since the QBMC browser uses usual Formal Concept Analysis structures such as formal contexts and lattices, the description we give of our implementation would be incomplete without a description of the underlying FCA framework. We cover the different alternatives we had, and then expose the reason behind the choice of a specific Formal Concept Analysis framework.

5.1.2 One Round Example

In order to give an intuition of how the query-based multicontext browser works, we illustrate one round of the cycle we presented in Chapter 3, the instance of query infrastructure used is based on the AIFB portal ontology.

We saw in Chapter 4, Section 4.1.3 that the subsumption hierarchy of the ontology is a good generic entry point for browsing the knowledge base. Therefore, we assume that the current context index is $\mathrm{S}(\leq_C (\{Root\}),\leq_C)$.

The user selected a certain number of concepts which correspond to the elements which interest him. The user is interested in the researchers who are related to these topics through the projects they are working on. It is a typical example of the join constructor (see Chapter 3, Section 3.5.5). The context index corresponding to this task is the following:

$$p = J \begin{pmatrix} inst(\{\mathsf{Topic}\}) \\ < \mathsf{dealtWithIn} > \\ inst(\{\mathsf{Project}\}) \\ < \mathsf{hasMember} > \\ inst(\{\mathsf{Person}\}) \end{pmatrix}$$

The first task of the user is to choose a constructor and parametrise it. We presented in Chapter 4, Section 4.3.3 how to do this. The user selects first the concepts: Person, Publication and Topic, then he chooses the relations worksAtProject and hasTopic.

These three queries are stored by the query management module. The constructor is stored by the constructor management system. When the user presses the button **Construct** on the context index construction panel, a construction command is sent to the context construction module with p as parameter.

The context index p is then analysed. It can be evaluated by composing two contexts composed with one another. These two context indices are:

$$p_1 := (\mathrm{inst}(\{\mathsf{Topic}\}),\mathrm{inst}(\{\mathsf{Project}\}),<\mathsf{dealtWithIn}>)$$

$$p_2 := \mathrm{inst}(\{\mathsf{Project}\}),\mathrm{inst}(\{\mathsf{Person}\}),\ <\mathsf{hasMember}>)$$

If the two composed contexts had already been calculated, it is more efficient to compose them on the spot than performing the whole join again. This construction is performed by checking for each object o of the context $\mathbb{K}_{p_2}$ whether it is an attribute of $\mathbb{K}_{p_1}$ with non empty extent.[1] For each element of the extent of such an attribute o_2 in $\mathbb{K}_{p_1}$, its intent in the new binary relation contains the intent of o_2.

[1]This is even more efficient if the existing context is in its dual form, i.e the context binary relation is represented as a set of object sets indexed by their attribute index.

Once the context has been evaluated, the event **Current Context Changed** is sent from the current context model to the lattice model. This event triggers the evaluation of the lattice on the new existing current context index.

Once the lattice model has been updated, it dispatches also an event **Current Lattice Modified**. The lattice panel and other elements of the Graphical User Interface are notified and update themselves, the line diagram of the concept lattice is displayed. In order to appear under the form presented in Figure 5.2,[2] the user must edit the diagram by moving the labels and nodes at more readable places. The program helps the user in this process by preventing him from breaking the lattice conventions, i.e the order of the concept lattice is preserved.

5.2 QBMC Infrastructure

We already mentioned in the previous section that the *QBMC infrastructure module* plays the role of the controller of a Model-View-Controller architecture and that it was itself split into three submodules: the query management module, the context index management module and the context construction module.

Their role has also been briefly exposed. This section explains in detail the way these constructors work as well as the dependencies between these submodules.

5.2.1 Query Management Module

The management of queries is crucial in the context of the query-based multicontext. Before we describe further the implementation, we comment briefly on the requirements it should fulfill.

Requirements

This module plays an important role in the query-based multicontext architecture. It serves as an interface to the evaluation of queries for the two other submodules. We list here the properties the *query management module* should have:

1. high performance: queries need to be evaluated quickly.

2. evaluation delegation strategies: if possible, the evaluation of queries should be evaluated on the back-end.

3. capable of composing queries: it should be possible to use the results of queries as input for others.

4. query equivalence test: the query infrastructure should be able to recognize equivalent queries, in order to prevent a new evaluation.

5. efficient containment testing: containment testing should be as quick as possible.

[2]The reader may be interested in comparing this lattice with the one found in Chapter 4, Figure 4.16, page 144.

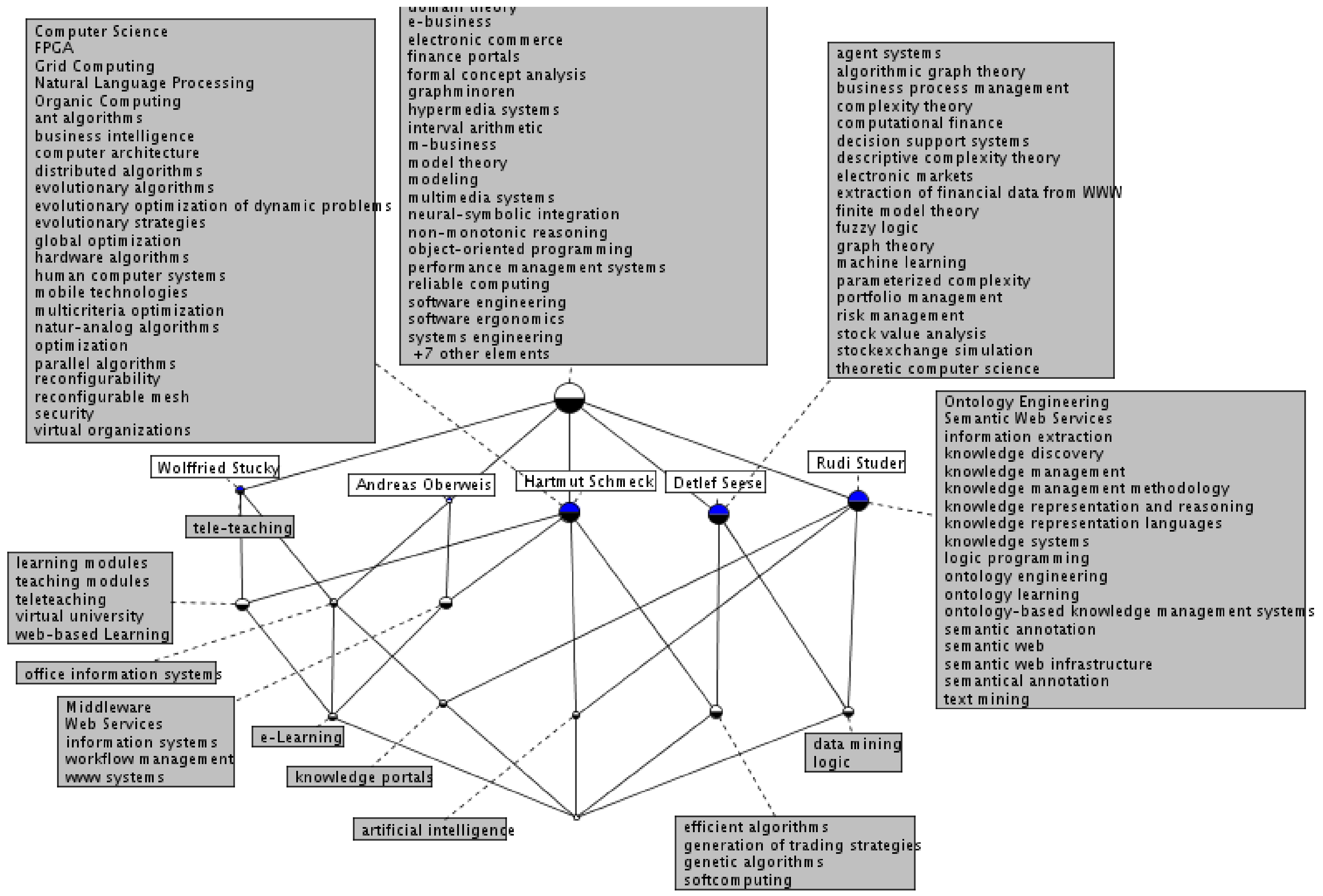


Figure 5.2: The lattice of the join which shows which research topics are dealt by some AIFB professors through one of the projects they coordinate. The topics at the top are not coordinated through some projects.

The performance of the evaluation should have the highest priority, in order to ensure that the context are presented quickly to the user. The implementation must make a trade-off between performance and reliable testing of the equivalence or containment relation between queries. Requirements (4) and (5) should therefore be implemented in such a way that the test for (4) and (5) never exceeds the time needed for the evaluation of the query itself.

The Query Evaluation Process

In [Vossen, 1994], Chapter 17, page 428, Gottfried Vossen describes the sequence of the query evaluation process as well as the optimisation techniques which occur at these different steps. In our implementation, we use the same kind of order to evaluate queries. Table 5.1 gives an overview of the steps used during the evaluation process:

Table 5.1: Typical steps for the evaluation of queries in a database system (adapted from [Vossen, 1994], Chapter 17, page 428).

Structure used	Activities at the given step
declarative query	high level optimisation
$\downarrow$	transform to a normalised form
logical algebra	logical optimisation and high level query planning
$\downarrow$	delegate to context infrastructure module
physical algebra	physical optimisation and high level query planning
$\downarrow$	delegate and compose evaluation
(query,query result)	evaluation on the back-end or in main memory

The declarative query is the query as returned by the interface or from a context index. Some high level optimisation techniques can be used at this step. For example, empty strings, incorrectly typed queries or syntactically incorrect queries return empty sets or empty relations.

The declarative form is transformed into a normal logical form. This transformation is highly dependent on the language and the kinds of optimisation which can be done on such queries. For the query language primitives which we proposed in Sections 3.3 and 3.4.2, we mainly sort the element set definitions. This prevents the costly checks which could occur when testing that large sets are equal or contained in one another. Moreover, some interesting information about the queries are gathered during this step, such that high level query plans can be created.

Once the query plans have been created, a decision procedure picks the most preferable one. The evaluation of the logical normalised representation is then delegated to the context infrastructure module, which creates a query plan to evaluate the query in the most efficient manner on the underlying instance of the query infrastructure. Depending on the query and the instance of query infrastructure concerned, this step may correspond to evaluating two steps such as:

- translating the query infrastructure language into the one used by the back-end of this instance of query infrastructure,

- use the results of already computed queries, and
- evaluate subqueries before using their results for the operators higher in the query tree.

Query Languages

In order to implement this evaluation process, a formalisation of queries is necessary. The queries presented in Sections 3.3 and 3.4 of Chapter 3 can be written as trees in a term algebra. A query can be seen as a term in a *Term Algebra* and can be stored as a tree having operators as internal nodes and identifiers as leaves. In our approach, the identifiers correspond to the identifiers of the typed QBMC presented in Chapter 3, Section 3.3.2. The different operators are the operators found in the subsequent sections of Chapter 3, Section 3.3 and Section 3.4.2.

In most cases, the query trees are just left as they are. The rewriting of the queries for the actual data sources is executed in the context infrastructure module.

Query Evaluation

In our architecture, the actual evaluation of queries is delegated to the Query Evaluation and Storage module of the context infrastructure module. The implementation of this module is described in Section 5.3.1.

The evaluation module maintains an up-to-date list of the query results it stores. The equivalent list is kept in the query management module where evaluated queries are stored with a pointer to their evaluations.[3] The evaluation of a query being dependent on the instance of the query infrastructrure in which it has been evaluated, this pointer also includes information on the instance of query infrastructure used.

In order to delegate complex queries and to be able to reuse the results, the delegation of composed queries (i.e. the query consists of other queries, for example the union of queries) is done by sending the composed query as a block. The query plan establishes the order of the diverse evaluations of the queries. This is particularly useful for the evaluation of context indices, where some query results might occur more than once in the evaluation of an index.

5.2.2 Context Index Management Module

The *context index management module* has a role similar to the query management module. Instead of managing queries, it manages context indices. It is also the interface needed to access contexts. The manipulation of context indices corresponds to the manipulation of contexts.

Contrary to the query management module, however, this module does not take care of the evaluation of context indices. This task is delegated to the context construction engine. The evaluation of queries has been well studied in the literature and the hope is to delegate most of the query evaluation to the actual underlying instance of the query infrastructure.

[3]This means that this cached results should completely updated if the data changes.

Let us now describe how the context index management module works. For this, we first discuss the structure used to code context indices. Then we deal with the issue of its storage. Finally, we deal with the methods used for the context index comparisons.

Context Index Definition

In our implementation, context indices are objects implementing the ContextIndex interface (i.e. in the object oriented sense). The actual implementation of a given ContextIndex may take diverse forms (i.e. diverse classes):

- the default context index implementation,
- one of the constructor context index implementations,
- one of the context index operator implementations.

Whereas the first of the three only implements the ContextIndex interface, the implementation classes used for the other kinds depend on the constructor and operators chosen. This offers the possibility to manipulate the context indices using an API instead of string manipulations. In that way, it is also possible to use keys to refer to given context indices. These keys can contain specific information about the context indices. For instance, it is possible to refer to another context index. A typical example would be operators such as dual and complement operations which wrap the underlying context indices (in the case of the dual: swapping objects and attributes while transposing the relation can be done by wrapping the binary relation).

Context Index Storage

In order to use context indices efficiently, we define a storage mechanism which allows to compare context indices quickly. For this, we use a sorted map structure called P where for any given context index, a context index key is created. The keys are ordered along the following criteria in decreasing order:

1. type: either normal, operator, or constructor,
2. their parameters (the queries involved),
3. some supplementary properties.

Using these sorting parameters it is possible to check context indices which are similar to the one looked for. This data structure is not only useful for finding context indices in this module. This storage mechanism is designed to allow the context index comparison algorithm and the context construction module to test only a small number of context indices.

5.2.3 Context Construction Engine

The evaluation of a given context index is done in three phases: construction plan generation, construction plan selection, construction plan execution. The following paragraphs describe the role of these three phases in the evaluation of a context index.

Construction Plan Generation

The first step in evaluating a context index is to compare the existing contexts with the context index of the view we want to construct. This is done in order to reuse existing contexts or parts of them.

Diverse cases can occur:

- The context index has never been evaluated.
- The context index has already been created and evaluated.
- Part of the context index has already been calculated.

The next step is using diverse strategies to generate construction plans. Our current implementation considers only simple plans. But, the literature of databases and information retrieval shows that a great number of optimisation techniques could greatly improve the efficiency of the query answering by generating query plans optimised for certain kind of query infrastructures. Since the breadth of this topic goes beyond the scope of this thesis, we do not go into further detail, though we acknowledge some relevant source of inspiration such as: [Abello et al., 2002, Garcia-Molina et al., 1999, Baeza-Yates and Ribeiro Neto, 1999].

Construction Plan Selection

Once diverse evaluation strategies or plans have been suggested at the previous phase, one strategy must be selected in order to evaluate the new context index. The selection of the correct plan depends on the diverse resources available. For example, if a larger context exists, then there is no need to reevaluate the context from the back-end.

In the realised system, the selection of the construction relies on some heuristics. For instance, the construction of the join is done by constructing the two contexts and joining the relations. But other techniques are preferable. Database literature presents a great number of evaluation techniques. Some of these techniques necessitate to gather prior information about the data. Since the ontology framework we used did not offer such means, the optimisation methods were limited.

Construction Plan Execution

In order to execute the plan, a composite strategy[4] is created, which evaluates its parts and then construct the whole. For this, it necessitates some composite data structures. These composite structures correspond to the context index operators and constructors presented in the previous chapter. These structures use references from the query and context index management modules in order to point to the needed structures.

To conclude this section, note that the planning for the construction of contexts using context index very efficiently has not yet fully investigated. In our implementation, we considered only very simple approaches to ease the evaluation.

[4]The word composite and strategy should be understood as references to the respective design patterns (see [Gamma et al., 1995] for more information).

Table 5.2: The naive algorithm for the evaluation of a context index.

```
**Input:** an istance of a query infrastructure $\mathbb{Q}$, a context index $p := (q_1, q_2, q_3) \in \mathbb{P}$
**Output:** the formal context for the context index $p$
    $S_{q_1} \leftarrow$ evaluate $q_1$
    $S_{q_2} \leftarrow$ evaluate $q_2$
    $R_{q_3} \leftarrow$ evaluate $q_3$
    compute $R_p := (S_{q_1} \times S_{q_2}) \cup R_{q_3}$
    **return** $(S_{q_1}, S_{q_2}, R_p)$
```

5.3 Context Infrastructure

Formal contexts are one of the basic elements of any Formal Concept Analysis based framework, their generation should be implemented as efficiently as possible. The use of queries to represent intensionally the formal contexts makes it possible to optimise the creation process.

For every query infrastructure, it is possible to generate the formal context of a given context index using the naive algorithm found in Algorithm 5.2.

In many cases, however, the efficiency of the construction of the context index can be improved, namely by reusing existing contexts or by using optimisation techniques.

Let us first describe the workflow for the evaluation of contexts. We saw in Section 5.2.3 that the construction engine creates a plan for the evaluation of a given context. This plan is then presented to the context infrastructure for evaluation. As for the query evaluation process presented in 5.2.1, this is the first step in the evaluation of the context. The plan organises the order of evaluation of the diverse elements necessary for the construction of the whole context. For example, the union of two contexts with context indices $p_1 := (q_1, q_2, q_3)$ and $p_2 := (q_4, q_5, q_6)$ can have different plans:

Plan 1:

1. evaluate p_1 (evaluate the first context index)
2. evaluate p_2 (evaluate the second context index)
3. merge p_1 and p_2

Plan 2:

1. evaluate $q_1 \cup q_4$ (evaluate the object queries)
2. evaluate $q_2 \cup q_5$ (evaluate the attribute queries)
3. evaluate $((q_1 \times q_2) \cap q_3) \cup (q_4 \times q_5) \cap q_6)$

In general, it is not possible to tell which kind of evaluation is the best, but note that in plan 1 the merging of the two contexts is done in the context infrastructure module. The evaluation of the queries for the contexts $\kappa(p_1)$ and $\kappa(p_2)$ can be delegated to the instance of the query infrastructure. If the number of elements which are not in the intersection of the results of q_1 and q_4 is small (i.e. only a few elements are not in $eval_1(q_1 \cap q_4)$), then evaluating $q_1 \cup q_4$ on the query infrastructure, instead of evaluating the two separately, greatly reduces the amount of data to be transported.

In order to discuss the implementation of our architecture, we first give an overview of the main data structures used in our implementation. We then discuss important aspects of the evaluation and storage of query results. The next step deals with the storage of contexts and present the data structures we use to do that. In the following section, we comment on the lattice generation step which has not really been investigated in this thesis, but we nonetheless discuss the advantage and the use of context indices for the creation of lattices and line diagrams. Finally, we discuss some supplementary aspects regarding context containment.

Main Classes

Figure 5.3 shows the most important interfaces of the query-based multicontext infrastructure. These classes are grouped into five main packages: contextindex, queries, context, fcamodel and queryresult.

The contextindex module contains the classes and interfaces for representing context indices. Since context indices represent intensional representation using queries. There are mainly two kinds of context indices: constructor-based and operator-based context indices. This reflects that operators combine context indices which may already have been implemented, whereas constructors usually[5] consist in an evaluation process of the queries which are used as parameters. All context indices could be of course implemented using the fully specified relation constructor (see Chapter 3, Section 3.5), since all context indices consists of three queries: (q_1, q_2, q_3). This module depends mainly on the queries package which we now briefly describe.

The queries package is the package used to represent queries. Since there are mainly two kinds of queries used in the query-based multicontext, the main interfaces are the SetQuery and RelationQuery interfaces. These interfaces represent respectively queries representing sets and relation results. A suplementary type of useful query which is needed to create selections and which is therefore extremely useful for the creation of queries is the ElementSetQuery.

The context package contains the main formal context interface used by the underlying FCA infrastructure as well as an interface for contexts generated from context indices. The main interface ContextInterface implements the main methods necessary for the creation of lattices from formal contexts. This mainly consists in the objects, attributes and relation retrieval methods returning the FCA elements required for these algorithms. The interface ContextIndexContext extends the main interface by referencing the intensional representation of this context: the context index. It allows an access to the underlying query results, i.e the object set query results, the attribute set query result and the relation query results. These accces methods allow the reuse of the results of the context for operations that may be performed on parts of an existing instance of a ContextIndexContext.

The fcamodel package contains some model element classes necessary for the

[5]The subsumption/instance constructor consists actually of a subposition of two contexts.

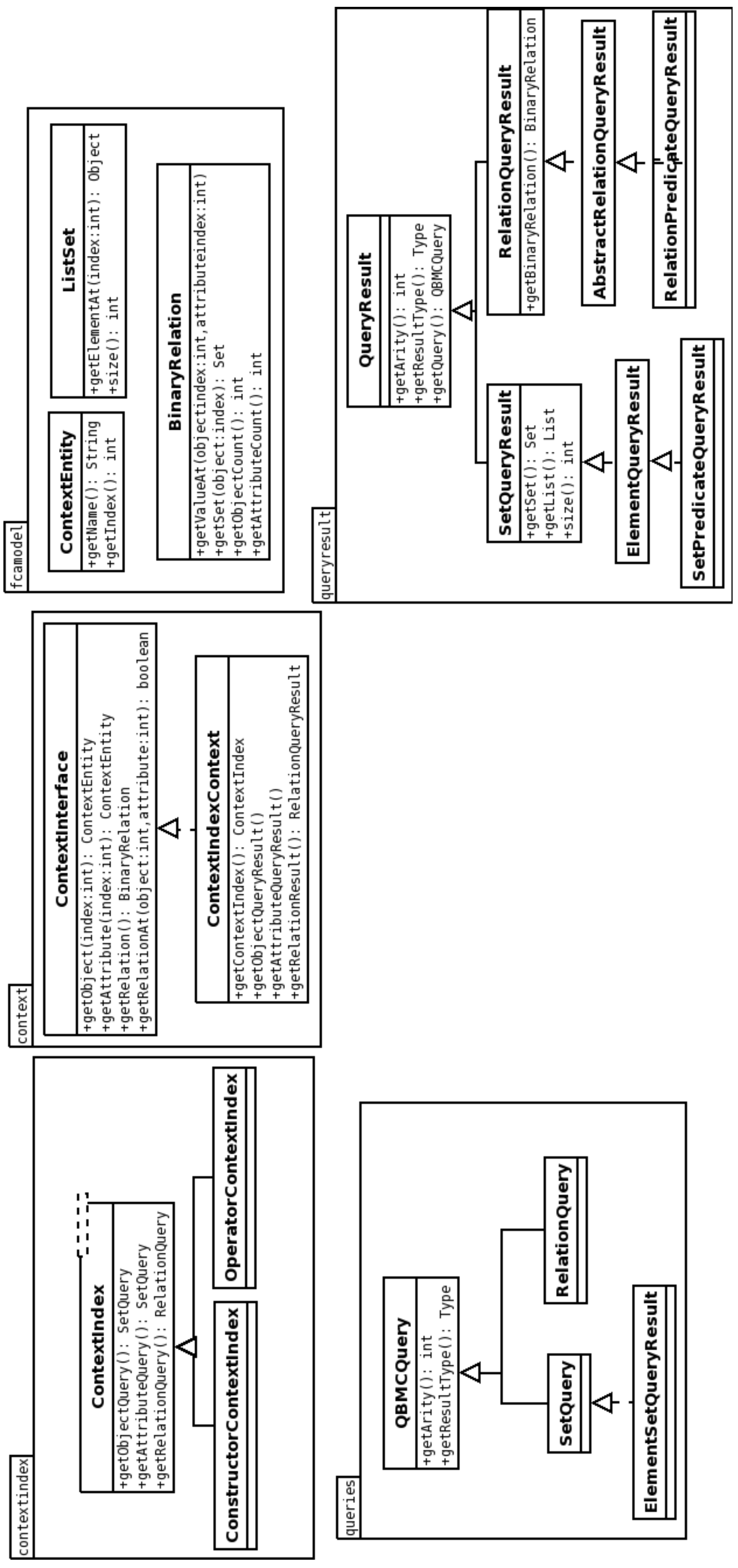

fcamodel
ContextEntity
+getName(): String
+getIndex(): int
ListSet
+getElementAt(index:int): Object
+size(): int
BinaryRelation
+getValueAt(objectindex:int,attributeindex:int)
+getSet(object:index): Set
+getObjectCount(): int
+getAttributeCount(): int
queryresult
QueryResult
+getArity(): int
+getResultType(): Type
+getQuery(): QBMCQuery
RelationQueryResult
+getBinaryRelation(): BinaryRelation
AbstractRelationQueryResult
RelationPredicateQueryResult
SetQueryResult
+getSet(): Set
+getList(): List
+size(): int
ElementQueryResult
SetPredicateQueryResult
context
ContextInterface
+getObject(index:int): ContextEntity
+getAttribute(index:int): ContextEntity
+getRelation(): BinaryRelation
+getRelationAt(object:int,attribute:int): boolean
ContextIndexContext
+getContextIndex(): ContextIndex
+getObjectQueryResult()
+getAttributeQueryResult()
+getRelationResult(): RelationQueryResult
contextindex
ContextIndex
+getObjectQuery(): SetQuery
+getAttributeQuery(): SetQuery
+getRelationQuery(): RelationQuery
ConstructorContextIndex
OperatorContextIndex
queries
QBMCQuery
+getArity(): int
+getResultType(): Type
SetQuery
RelationQuery
ElementSetQueryResult

Figure 5.3: The class diagram for the implementation of the context infrastructure.

implementation of the FCA algorithms. The three main classes of this package are ContextEntity, ListSet and BinaryRelation. Instances of the first represent the objects and attributes of the formal context. ListSet are lists which also implement the interface java.util.Set. The indexation of the elements of the list enables a faster access for the display of information on the elements of the lattice. The instances of the last class, the BinaryRelation, represent binary relation as used by the Formal Concept Analysis algorithms.

Finally the queryresult package contains the data structures for storing the results of the queries. The main class QueryResult allows to retrieve generic information such as the type of the result, the query used as well as the arity. This interface is extented by the SetQueryResult and the RelationQueryResult. The former returns sets or list as well as indicates the size of represented set. The latter stores a binary relation. This binary relation can then be used by the Formal Concept Analysis algorithms.

5.3.1 Query Evaluation and Query Results Storage

Since queries play a central role in our architecture, it is crucial to devise efficient means of evaluation. Moreover, the evaluation of contexts is usually a complex process where the results of the diverse queries interact greatly. One of the main ideas behind the query-based multicontext infrastructure is to combine the results of queries and operators. In order to achieve this goal, the underlying context and query infrastructure should have data structures adapted to the possible combinations. While context data structures are the topic of the next section, we now discuss the query infrastructure details.

Queries and Query Operators

From the definition, a query infrastructure consists of two languages. Many different implementations are possible for a given language pair, depending on the kind of data source. For example, KAON has both an API for manipulating entities and a query language. In addition to the normal API, the KAON2 infrastructure offers different ways of querying the knowledge stored: conjunctive queries, DL-safe rules and SPARQL queries.

This heterogeneity of the datasources implies the necessity of mapping the query-based multicontext query primitives on the primitives of the data source language. We address here aspects corresponding to the generic implementations of the queries. In Section 5.4, we describe a methodology and an implementation for each of the semantic query infrastructures based on KAON and KAON2.

The query-based multicontext queries store mainly two types of information: the type and the arity of the result of the query.

Each of the types of the typed query infrastructures is associated with certain special query operators as defined in Chapter 3, Section 3.4.2. When the query infrastructure is initialised, the implementing classes are registered by the query evaluation manager with the supported types. The registered types are accessible to the GUI and the QBMC architecture. During the query definition process, only the queries with correct types are considered.

The arity of the result is important since it allows to differentiate between set queries and relation queries. This information is in particular needed for the creation of queries. The query creation methods require type and arity information in order to help the user in creating meaningful queries. For example, the creation of a query such as $\mathsf{inst}(\{\text{Julien Tane}\})$ should be prevented because the query $\{\text{Julien Tane}\}$ is not a concept query.

Storage of Query Results

The query results come in two types of implementations. These types correspond to the type of query which has been evaluated.

The sets are stored using ListSets, which are implementations of Sets based on lists. In our implementation, we sorted the sets in order to improve the efficiency of accesses on these lists, since binary search can be performed efficiently on sorted lists.

The results sets store a binary relation. The implementation of such a binary relation is based on a list of bit sets. This is described in further detail in Section 5.6 which describes the underlying FCA infrastructure.

As we described the query management module, we saw that the query evaluation module can cache a certain number of existing query results. Actually in our implementation, we store only a reference to a small number of results, but plans are to enable a better caching using these queries. The current purpose is to minimise the storage so that if a query occurs more than once, its result is not stored more than once.

Evaluation of Query Results

The evaluation of the query result is best done in the way presented in Section 5.2.1. The query is submitted to a series of transformations. These transformations allow to compare it or its parts to the queries which have already been evaluated and also regroups elements belonging together. Some actions or heuristics can also be used to evaluate the query more efficiently. We give here the list of transformations:

- $\{i_1, ..., i_n\}$ are sorted
- $\text{inst}(q_1) \cup \text{inst}(q_2) \longrightarrow \text{inst}(q_1 \cup q_2)$
- $\mathsf{role}(< q_3 >, q_1) \cup \mathsf{role}(< q_3 >, q_2) \longrightarrow \mathsf{role}(< q_3 >, q_1 \cup q_2)$
- $(\text{inst}(q_1) \backslash \text{inst}(q_2)) \backslash \text{inst}(q_4) \longrightarrow \text{inst}(q_1) \backslash \text{inst}(q_2 \cup q_4)$
- $q_1 \cup \cdots \cup q_n \longrightarrow \bigcup_{i=1}^{n} q_i$

The possible optimisations of queries is extremely dependant on the capabilities of the semantic query infrastructure. For example, if the union or intersection operators cannot be performed on the data source, the generic query infrastructure of the query-based multicontext performs these operations.

In order to ensure that the query-based multicontext is generic enough, all the generic query operators presented in Section 3.3 have been implemented in the query module.

5.3.2 Context Storage

Formal contexts in the query-based multicontext infrastructure are indexed using the context index which has been used for its generation. These formal contexts must implement two different interfaces: the context interface of the underlying infrastructure and the context index context interface which allows the query-based multicontext to access the query results of the parts of the generated formal context.

Context Interface

The current implementation of the formal contexts should be seen as an example of facade (i.e a more complex data structure is hidden behind a simple generic interface, see [Gamma et al., 1995]). Actually, the use of the facade enables the use of the query results as a basic storage data structure, we can then use the interfaces required by the underlying Formal Concept Analysis infrastructure, since they are needed in order to perform the generation of the lattice. We implemented this for two different[6] underlying Formal Concept Analysis infrastructures, but we only describe the Concept Explorer implementation in Section 5.6, this is the only one for which the GUI can be used.

Context Index Context

The context index context interface is used in the query-based multicontext infrastructure in order to manipulate and generate new formal contexts using the results of existing formal contexts. In particular, it is used to implement different operator-based constructions such as apposition, subposition, transposition or complementation.

In such cases, the resulting formal context can be efficiently generated from the existing contexts which were used as parameters to the operator-based context. Complementation is implemented by using a simple gate inverting the truth values of the relation. The apposition and subposition operators are implemented using specific disjoint union query results for set and relation query results. Finally, the transposition is performed on the binary relation returning a new transposed binary relation.

5.3.3 Context Creation

One of the reasons to introduce context indices was to help the process of creating meaningful contexts. In this section, we address some issues about the creation of contexts from a query-based multicontext infrastructure.

General Method

The method used to generate a formal context from a context index depends on two aspects.

First, when some part of the context has already been computed (and is still accessible), it should not be recomputed. This is particularly true of the formal context being displayed to the user during the new context generation.

[6]We also did an implementation on top of the Toscana context infrastructure.

The second important information is whether it is possible to perform the operations on the data source instead of the Query-Based Multicontext query engine. In particular, it is also important to determine whether the context index can be evaluated using datalog queries. This means that in the first phase each of the queries necessary for the evaluation of the context index or constructor is submitted to a test to check whether it can be implemented as a datalog query.

This test takes into account the type of query. For example, for some engines, it is not possible to use concepts or relations as constants of the datalog program. A solution must therefore be found for these cases. In Section 5.4.1, we present the technique which can be used in that case.

We use in this section the notations and the implementations of the generic queries presented in Chapter 3, Section 3.3.1, that is, a predicate pred-q_i is used to evaluate a query q_i.

If the context index or constructor cannot be evaluated using datalog queries, then the evaluation is performed using queries from the QBMC query engine and the results are merged in the context storage using the appropriate operators.

Context Operators

In this section, we describe the implementation of the diverse context operators in the context of the query-based multicontext.

As we saw in Section 5.2.3, we presented a module creating a plan for the evaluation of context indices. We saw that it gives a context evaluation plan to the operator engine. Each step of the plan is performed.

For example, a plan for an apposition $p_1|p_2$ may be presented informally as:

- evaluate the object query for both contexts,
- evaluate each context individually but making sure the indices are the same.

Such a plan requires data structures or control mechanisms ensuring that the indices of the objects are the same, because the objects of the resulting context is the same as the sets of objects of both contexts.

Table 5.3: Features of the context operators implementations.

Constructor	Datalog	Composite
dual	×	×
complement	×	×
union	×	×
intersection	×	×
apposition		×
subposition		×

Dual:

The dual operator is very simple, since it can be obtained by swapping the variable from the interpretation of the program of any constructor.

dual(y,x)$\leftarrow$ pred-$q_1(x)$,pred-$q_2(y)$,pred-$q_3(x,y)$.

When the underlying context index has already been evaluated, the dual context index can be simply evaluated by creating a new formal context, setting the attributes of the old formal context as objects of the new, and the objects of the old one as attributes of the new. Finally the binary relation is inverted.

Note that the binary relation can be inverted in two possible manners. First, a new binary relation is created which is transposed. The second method is to use a facade **InverseBinaryRelation** which implements the **BinaryRelation** interface but accesses the data the opposite way. Depending on the BinaryRelation implementation, this may be efficient or not. If the underlying relation consists of lists of sets of attributes for all the objects, this is not likely to be efficient. However, if the underlying data structure is a graph, then the approach may be interesting. Since our implementation uses ListsSets we used the first of the two alternatives.

Complement:

The complement operator can be evaluated using the following rule:

complement(x,y)$\leftarrow$ pred-q_1(x),pred-q_2(y), complement-q_3(x,y).

This rule uses the complement-q_3 predicate presented in Chapter 3, Section 3.3.1. A rule that may be more efficient though we could not perform tests flattens this query in the following way:

complement(x,y)$\leftarrow$
pred-q_1(x),dom-q_3(x), range-q_3(y), pred-q_2(y), $\sim$pred-q_3(x,y).

If the queries cannot be performed in datalog, or if the context index to be complemented has already been computed, the two alternatives are possible just as in the dual case. The first alternatives consists in complementing the binary relation of the underlying context. The second alternative consists in setting a facade on top of this relation which returns the opposite value of this binary relation.

Both methods are possible though we have not tested which provides the best results.

Union:

The union operator is trivial, since it can be implemented by simply evaluating each of the queries in arguments of the resulting context index. This is easily done using the following datalog rule.

union(x,y)$\leftarrow$ union-q_1-q_4(x),union-q_3-q_6(x,y),union-q_2-q_5(y).

If the context index or the constructor cannot be evaluated using datalog, then the result is computed in the QBMC engine merging the results using an union operator.

Intersection:

The intersection operator is as trivial as the union operator, since it can be implemented by simply evaluating each of the queries in argument of the resulting context index. The intersection can then be evaluated using the following datalog rule.

intersection(x,y)$\leftarrow$
intersection-q_1-q_4(x),intersection-q_2-q_5(y),intersection-q_3-q_6(x,y).

If the context index or the constructor cannot be evaluated using datalog, then the result is computed in the QBMC engine merging the results using the intersection operator. The intersection operator checks stops its evaluation in case it can be certain that the computed part is empty.

Constructors

Constructors are also evaluated using plans. There are usually two different kinds of plans. The first kind is only accessible, if the query used to parametrise the constructor can be evaluated using datalog, then the datalog plan is used.[7] Otherwise, the other type of plan is used. In other word, the construction is performed in the QBMC framework.

Table 5.4: Features of the constructors implementations.

Constructor	Datalog	Composite
Subsumption	×	×
Instantiation	×	×
Subsumption-Instantiation	×	×
Join	×	×
CoRelation	×	×

Subsumption Constructor: $S(q_1, q_3)$
The predicate subelement must be the predicate obtained from the subsumption hierarchy query.
subsumption(x,y)$\leftarrow$ pred-$q_1(x)$, pred-$q_1(y)$, subelement(x,y).

Note that if the logic language underlying the ontology does not contain any complex names, it is possible to generate the hierarchy using the neighbouring concept relation. This is in particular the case of RDF ontologies. The information is typically available using the rdf:subConceptOf RDF property with a statement like: x rdf:subConceptOf y. The subsumption can then be constructed as:
subsumption(x,y)$\leftarrow$ pred-$q_1(x)$, pred-$q_1(y)$, subelement(x,y).
subelement(x,y)$\leftarrow$ subconceptOf(x,y).
subelement(x,y)$\leftarrow$ subconceptOf(x,z), subelement(z,y).

Instantiation Constructor: $I(q_1, q_3)$
The predicate instantiation-rel must return the instantiation relation between concepts and instances.
instantiation(x,y)$\leftarrow$ pred-$q_1(x)$,instantiation-rel(x,y).

With the instantiation constructor, the neighbouring relation can be used in the same manner as for the subsumption constructor. Actually it uses the subsumption constructor. In RDF, the instantiation is denoted using the rdf:type RDF property with a statement like: x rdf:type y. The instantiation constructor can then be constructed as:
instantiation(x,y)$\leftarrow$ pred-$q_1(x)$, rdftype(x,y).
instantiation(x,y)$\leftarrow$ pred-$q_1(x)$, rdftype(x,y), subsumption(x,y).

Subsumption-Instantiation Constructor: $I(q_1, q_3)$

[7] In our implementation, we actually use the datalog implementations of the data sources.

In Chapter 3, Section 3.5, we explained that subsumption-instantiation constructors can be constructed by the subposition of the results of a subsumption constructor and an instantiation constructor. This implies that the two construction given above may be used to compute the subsumption-instantiation constructor.

Join Constructor: $J(q_1, q_3, q_2, q_6, q_4)$
As we show in the following rule, the join constructor can be implemented using the join query implementation for datalog which was presented in Chapter 3, Section 3.3.1.
join(x,y)← pred-q_1(x),pred-q_4(y),join-q_3-q_1-q_6(x,y).

If the query used cannot be implemented in datalog, then the join constructor can also be implemented by constructing two formal contexts from the context indices:

- $p_1 := (q_1, q_2 \cap q_3(q_1) \cap q_6^{-1}(q_4), q_3)$ and
- $p_2 := (q_2 \cap q_3(q_1) \cap q_6^{-1}(q_4), q_4, q_6)$

Note that the attribute query of p_1 and the object query of p_2 are equal.

CoRelation Constructor: CoR(q_1,q_3)
The CoRelation constructor is also easily implemented using the two following datalog rules:
corelation(x,y)← pred-q_1(x), pred-q_3(x,y).
corelation(x,y)← pred-q_1(z), pred-q_3(z,y), pred-q_3(x,y).

5.3.4 Lattice Generation

Until now, we have not used the context indices to improve the lattice generation. This is still on going work. We conjecture that using context indices should help in improving the lattice generation task. The reason behind this is that queries often correspond to blocks of data which are highly corelated. Chapter 7, Section 7.2.1 discusses briefly this topic.

5.4 Semantic Query Infrastructure Implementation

In the previous section, we discussed the implementation of the QBMC using some underlying Query Infrastructure. The present section focuses on the implementation of semantic query infrastructures which were presented in Chapter 3, Section 3.4.2).

5.4.1 Methodology for Adapting Data Sources

The query and operator languages presented in Sections 3.3 and 3.4.2 allow many possible underlying infrastructures. We first want to present the methodology used to create a query infrastructure suitable for a query-based multicontext.

Main Steps

The method we use consists of the following steps:

- 1st Step: identify the possible types of elements of Ω (i.e the elements which can be objects or attributes of a formal context),
- 2nd Step: identify the existing language operators of the underlying query infrastructure,
- 3rd Step: define a wrapper for these operations or for specific combinations of these operations,
- 4th Step: define or adapt existing query result data structures to the specific data source results.

Step 1 is a crucial step because the kind of formal context which can be created using a query-based multicontext depends much on which elements can occur as entities of the context. Step 2 is a phase of analysis of the features of the data source. Step 3 consists in the wrapping of the data source language as a query infrastructure. For example, the union operation for concepts should be wrapped so that the evaluation of a query $q_C \cup q_D \in L_C$, is mapped properly to the corresponding union operation on the data source. The operations which do not exist in the data source language but are needed in the query infrastructures have to be implemented by the query evaluation module presented in Section 5.3.1. While Step 3 takes care of the more procedural aspects of implementing the query infrastructure on top of the data source language, Step 4 deals with the data structures needed for this. An important aspect of Step 4 is that it is important to incorporate in the data structure a means of getting identifiers and labels. Identifiers play an important role in our implementation. They allow the checking of whether a given element is already in a set or not. It is also crucial to implement a labelling mechanism since the labels are used in the visualisation. It is therefore important to keep a mechanism to retrieve labels for given elements.

Implementation Issue: Nominals

Nominal queries are frequently used in the query-based multicontext browser, this requirement comes from the importance for users to select particular entities. For example, from a list of research topics only two or three may be relevant for the user. This means that queries based on this kind of queries must be evaluated in the query infrastructure. While it is possible to evaluate these queries by simply creating a number of queries and merging the result in the query-based multicontext query result module (or performing an intersection of the results if the operator used is an intersection), it is more appropriate to delegate the interpretation to the data source.

If the nominal query returns instances then the delegation to the data source can often be performed by adding a new predicate and its instances to the data source so that its evaluation returns the instances.[8] The queries can then use this predicate for calculation on the back-end.

[8]This may not be apriori possible in every knowledge base infrastructure.

Implementation Issue: Concept and Relation Queries

The second issue to take into account for the implementation of the query infrastructure is caused by the presence of queries returning sets of concepts or relations. The problems occurring are similar to nominals, since they can be solved using a brute force approach multiplying the queries to the knowledge base by the number of concepts or relations that such a query returns. Depending on the available query primitives of the data source, certain optimisations are possible. In particular, little optimisations are possible in the RDF(S) case whereas the OWL ontology language provides useful operators. Furthermore, data sources providing query languages returning concepts or relations greatly simplify the implementation efforts.

In [Motik, 2006] and [Chen et al., 1993] metamodelling semantics are discussed in order to deal with statements dealing with groups of individuals. Using these techniques as well as techniques from F-Logic to deal with concepts would simplify the implementation.

Now that we discussed the general methodology as well as the main issues for the implementation of the query infrastructure, we describe in the two following sections the particular implementation of the semantic query infrastructure on top of the two ontology framework KAON and KAON2.

5.4.2 Adapting KAON

Following the methodology mentioned above, we present the diverse steps followed for the creation of the semantic query infrastructure on top of the KAON ontology framework. We call this query infrastructure: KAON query infrastructure, or KAON infrastructure if it is clear from the context.

Elements

In the KAON framework, seven types of elements play an important role: concepts, properties, property instances, instances, literals, lexical entries and OImodels. It is possible to map the three types: C, R, Ri with the corresponding entities of KAON model (see Section 3.4.2). Literals can be seen as values of some attributes of an instance. In our approach and particularly in our implementation, we did not make any distinction between instances and literals, and literals are considered as instances and are therefore mapped to the semantic query type: I. Lexical entries are kinds of instances which reference some elements of the language in order to annotate these elements with language dependent labels or documentation information. These were only used to provide a label to instances. OImodels are the knowledge bases of the system. For the time being, we did not consider OIModels as elements of the query-based multicontext browser. However, models were used in the browsing mechanism of the Courseware Watchdog that we briefly mentioned in Chapter 4, Section 4.1. This preliminary experience led us to conjecture that many applications would benefit from their integration as a supplementary type in the query infrastructure. In Chapter 7, we comment briefly on this (see Chapter 7, Section 7.2.1).

KAON API

The KAON Ontology framework consists of an API for accessing elements of the framework as well as a query language to specify a query. Our implementation of the KAON infrastructure uses both the API and KAON's query possibilities. Table 5.5 presents the functionalities of the KAON API.

While the KAON API does not provide any method for the usual set operations such as union, intersection or difference, the KAON query language provides them. But some operations could not be defined in the language.

For a more detailed description of the KAON framework, we refer to the documentation of the query framework which can be found in [Motik, 2002].

Wrapping the KAON API and Query languages

The wrapping of the methods of the KAON API and of the operators of its query language is a straight forward implementation. It uses loops over the sets of elements or over the results of certain queries.

Note that we only make a limited use of the query language offered by KAON, because not all the needed language primitives needed by the infrastructure could be defined using the KAON language.

Data Structures

We have performed little optimisation of the KAON semantic infrastructure. The implementation used is mainly functional and does not use directly the datalog engine of the KAON infrastructure. This means that all the evaluation work of the generic query operators from Chapter 3 Section 3.3 is actually performed by the QBMC engine and not by the KAON data source.

The implementation uses the KAON objects as underlying objects of the KAON infrastructure. This allows to make direct calls on the methods of the interface.

5.4.3 Adapting OWL Using KAON2

As for the implementation of the semantic query infrastructure for KAON, the adaptation of KAON2 followed the step suggested in Section 5.4.1. The KAON2 ontology framework consists of an API for managing OWL knowledge bases. This API provides a reasoner capable of reasoning in the $\mathcal{SHIQ}(D)$ description logic. This means that nominals are not available when using the KAON2 API. The KAON2 framework is based on a disjunctive datalog engine. This engine provides efficient processing of a large number of A-box axioms.

Elements

To use the functionalities of KAON2, it is relevant to consider the potentially relevant elements. The denomination follows closely the one used by the OWL syntax (see [MacGuiness and van Harmelen (eds), 2004]).

In the KAON2 system (and OWL in general), the concept entities may take two different forms: *OWL classes* or *complex concept descriptions* composed the

Concept	methods
getPropertiesFromConcept	Returns all properties starting from this concept
getAllPropertiesFromConcept	Returns all properties (including inherited ones) starting from this concept
getPropertiesToConcept	Returns all properties pointing to this concept
getAllPropertiesToConcept	Returns all properties (including inherited ones) pointing to this concept
getSubConcepts	Returns direct subconcepts of this concept
getAllSubConcepts	Returns all subconcepts (inherited as well) of this concept
getSuperConcepts	Returns direct superconcepts of this concept
getAllSuperConcepts	Returns all superconcepts (inherited as well) of this concept
getInstances	Returns the set of direct instances of this concept
getAllInstances	Returns the set of all instances (even of subconcepts) of a given concept in the pool
Instance	methods
getPropertiesFromInstance	Returns all property starting from this instance
getPropertiesToInstance	Returns all properties pointing to this instance
getParentConcepts	Returns the set of direct parents of this instance
getAllParentConcepts	Returns the set of all parent concepts of a given instance in the pool
getValuesFromInstance	Returns all properties starting from this instance
getValuesToInstance	Returns all properties pointing to this instance
Property	methods
getSubproperties	Returns the subproperties which are directly subsumed by this property
getAllSubproperties	Returns all the subproperties which are subsumed by this property
getSuperproperties	Returns the superproperties which subsume directly this property
getAllSuperproperties	Returns all the superproperties which subsume this property
getAllPropertyInstances	Returns all the relation instances of this property
getDomainConcepts	Returns the domain concepts of this property
getAllDomainConcepts	Returns all the domain concepts of this property
getRangeConcepts	Returns the range concepts of this property
getRangeConcepts	Returns all the range concepts of this property
Property Instance	methods
getProperty	Returns the property from which this relation instance is an instance of
getAllProperty	Returns all the properties from which this relation is an instance of
getTargetValue	Returns the value of this property
getSourceInstance	Returns the instance which is the source of this relation instance

Table 5.5: List of the methods of the main entities of KAON API

diverse constructors presented in Chapter 2, Section 2.2.1. For example Person is a named concept whereas

$$\mathsf{PhDStudent} \sqcap \exists \mathsf{affiliated}.\{KnowledgeManagementGroup\}$$

is a complex description. In the current version of the implementation, we did not allow the user to manipulate complex concept descriptions. Two important remarks should be done on their use. First, note that many of the query operators we introduced allow to express these complex concepts. Second, complex concept descriptions are particularly useful for representing a number of queries and are therefore used when querying a KAON2 knowledge base. In other words, mixing our complex descriptions and the one found in OWL would have added much complexity. However, it is useful to map our complex queries onto complex descriptions.

For the current implementation of the KAON2 query infrastructure, we used the same strategy as in the KAON one, i.e. the instances in an OWL knowledge base can be of two types: *individuals* and *datatypes elements* and we consider both kinds as instances. Since the types of instances are considered together, we also consider that *object properties*, and *data properties* are both properties. Finally, *object property members* and *data property members* are also considered together as relation instances.

The KAON2 API

The KAON2 API offers a number of possibilities to implement the semantic query infrastructure presented in Chapter 3, Section 3.4.2. In particular, it offers the possibility to use a rule mechanism to query the knowledge base. This means that the most complex queries of our query infrastructure can be performed using appropriate datalog programs. Moreover, another useful feature of the KAON2 API is the capacity to add a number of statements which are considered valid only when evaluating a particular query. This is then used for instance nominals, for which a new predicate name is created which returns exactly these instances and this new concept name is used in the final version of the query.

The construction uses the algorithm shown in Table 5.6:

Data Structure

A promising feature of the KAON2 query interface is the possibility to iterate over the query. Though we have not yet finished the implementation of stream-based query results, such an implementation is likely to reduce the amount of processing to be performed.

5.5 The Graphical User Interface

In Chapter 4, we presented the principle behind the user interface. We now discuss the issues concerning the implementation of the graphical user interface of the query-based multicontext browser. We first describe the main panels with which a user interacts to explore the knowledge base. Then we briefly describe the mechanisms of parametrisation using direct manipulation through drag and drop. Finally, we note the important aspects of visualisation using the concept lattice for the visualisation of views.

Table 5.6: The algorithm used to compute queries with nominals.

```
Input: query $q_1$
Output: the merged result set of the query
  // initialise set of nominal predicates
  N:=$\emptyset$
  // initialise mapping of nominal predicates extensions
  M:= new Map
  // Initialise result set or relation
  $R := \emptyset$
  // Collect nominals
  if $q_1$ contains nominal queries then
    for all each occurrence of a nominal set $\{i_1, \dots, i_n\}$ do
      create new nominal predicate $p_n$
      $N := N \cup \{p_n\}$ create new nominal predicate
      $M := N \cup (p_n, \{i_1, \dots, i_n\})$
    end for
  end if
  // Create templates for queries depending on nominals
  if $q_1$ contains a concept query $q_c$ or a relation query $q_r$ then
    for all occurrence of nominal $n$ in concept query $q_c$ do
      create a query template $temp(n, q_c)$ replacing the occurrence of the con-
      cept $n$
    end for
    // Create templates for queries depending on nominals
    evaluate each concept parts or relation query parts independant of nominals
    for all template queries do
      for all bindings of depending queries do
        create query $q$ to evaluate
        R $\longleftarrow$ evaluate($q$)
      end for
    end for
  end if
  return $R$
```

5.5.1 Main Components

The design of our framework required a minimal of four main components, two auxilliary components and a number of wizards (specialised graphical elements appearing in a supplementary window to inform the user or helping him to perform specific complex actions).

The main components used in our implementations are

- the view panel,
- the context index panel,
- the constructor parametrisation panel, and
- the selection panel.

The layout of these different components is shown in Figure 5.4, while Figure 5.5 gives a better idea of the general appearance of the GUI.

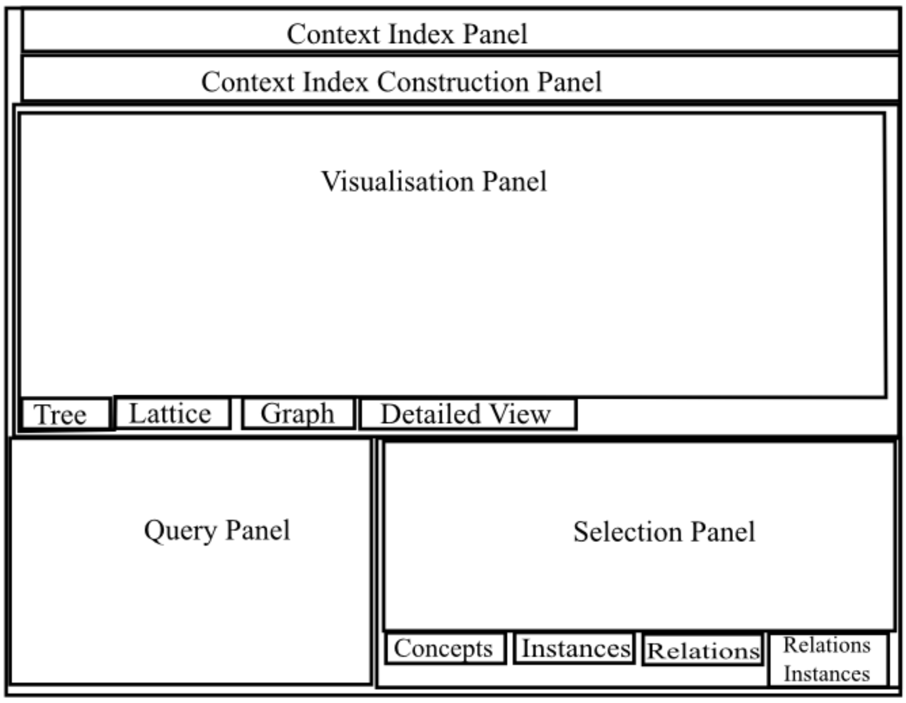


Figure 5.4: The layout of the main components of the query-based multicontext browser.

We now describe briefly each component, starting with the visualisation panel.

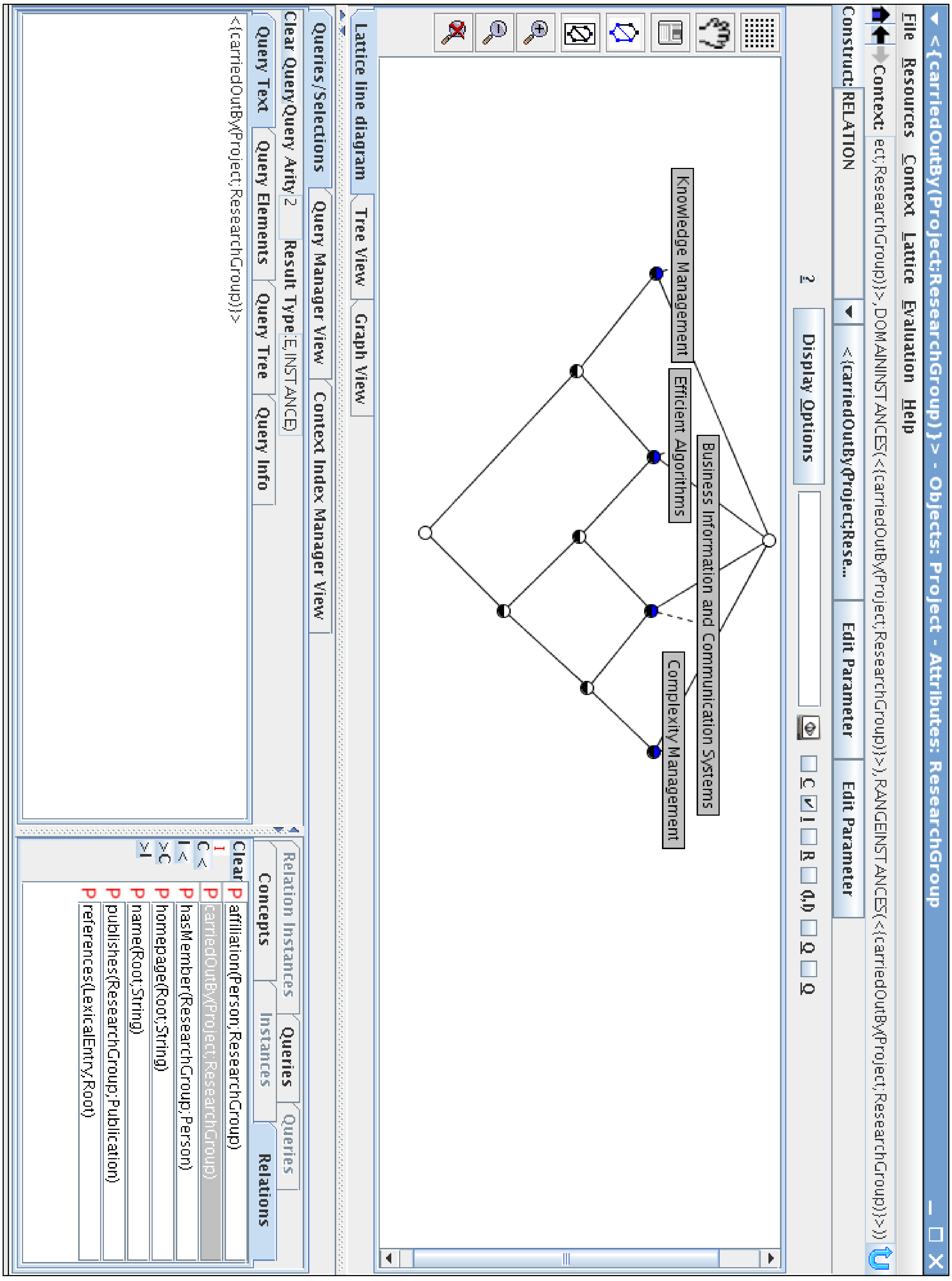
Visualisation Panel

The visualisation panel is the component used to display the current view. In order to accommodate diverse visualisation paradigms, the view panel is a tabbed panel, where each tab consists of the appropriate visualisation panel for one of the three paradigms: graph, tree or lattice. A supplementary tab also contains the detailed view of entities. It allows a user to see the properties of a collection of elements. This is typically interesting to take a closer look at the properties of the elements of the extent of a relevant formal concept.

The selection mechanisms described in Chapter 4, Section 4.2.2 have been implemented for each of the visualisation paradigms. Using a context menu, the selection can be added to the selection panel or used to add the selection to the currently selected query of the constructor panel.

Context Index Panel

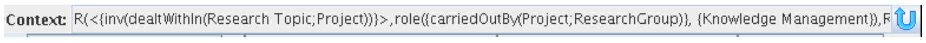
The purpose of *context index panel* is to present to the user the current context index, that is the context index representing the content presented in the view



<{carriedOutBy(Project;ResearchGroup)}> - Objects: Project - Attributes: ResearchGroup
File Resources Context Lattice Evaluation Help
Context: ect;ResearchGroup)}>,DOMAININSTANCES(<{carriedOutBy(Project;ResearchGroup)}>),RANGEINSTANCES(<{carriedOutBy(Project;ResearchGroup)}>))
Construct RELATION
<{carriedOutBy(Project;Rese...
Edit Parameter
Edit Parameter
Display Options
C I R (),() Q Q
Knowledge Management
Efficient Algorithms
Business Information and Communication Systems
Complexity Management
Lattice line diagram
Tree View
Graph View
Queries/Selections
Query Manager View
Context Index Manager View
Clear Query
Query Arity 2
Result Type E,INSTANCE
Query Text
Query Elements
Query Tree
Query Info
<{carriedOutBy(Project;ResearchGroup)}>
Relation Instances
Queries
Queries
Concepts
Instances
Relations
Clear
I
C <
I <
>C
>I
P affiliation(Person;ResearchGroup)
P carriedOutBy(Project;ResearchGroup)
P hasMember(ResearchGroup;Person)
P homepage(Root;String)
P name(Root;String)
P publishes(ResearchGroup;Publication)
P references(LexicalEntry;Root)

Figure 5.5: The main panels while exploring the different projects of the AIFB research groups.

panel. Figure 5.6 shows the appearance of this context index for the context index. A number of simple actions can be performed on the current context index in order to define a new one. There are currently two types of actions: the application of an operator such as transposition and the refining of the current context index.



R(<{inv(dealtWithIn(Research Topic;Project))}>,role({carriedOutBy(Project;ResearchGroup)}, {Knowledge Management}),R

Figure 5.6: The context index panel for the context showing the research topics for the projects of the Knowledge Management Chair.

Constructor Parametrisation Panel

The purpose of the *constructor parametrisation panel* or shorter constructor panel, is to allow the user to define a new view by parametrising the current constructor.

As shown in Figure 5.7, the constructor panel is composed of a button **Construct**, a combo list and a number of buttons representing the parameters of the current constructor. By pressing the button, a user may start the evaluation of the resulting context index. Using the combo list (i.e. a widget used to display a list as well as its unique currently element), a user may choose the type of constructor he is currently interested in parametrising. Finally, the user may press the parameters buttons to select one particular parameter as the current one. The query used for this parameter is then displayed in the query panel.

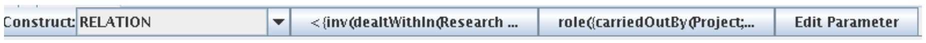

Construct:
RELATION
<{inv(dealtWithIn(Research ...
role({carriedOutBy(Project;...
Edit Parameter

Figure 5.7: The constructor parametrisation panel with the parameters for the view displaying the distribution of the research topics of the projects carried out by the Knowledge Management Group.

Selection Panel

Since selections are crucial in the query-based multicontext exploration framework, the selection panel should be seen as one of the main interaction panels. Once an element or a collection of elements have been selected from the view panel, these elements appear in the selection panel.

To ease the interaction, the selection panel used in our implementation separates the selected elements per type. This separation has two main purposes. The first one is to simplify the visualisation. User may more easily focus on a given type of elements. The second purpose was to allow the use of type oriented actions.

Figure 5.8 shows the selection panel with the three concepts: Person, Publication and ResearchTopic. The three concepts are on a dark background. The darker elements are the ones on which actions are performed. Using the menu **Show Entities**, the user can select these elements to be seen in the item properties panel. In the figure, it can be noted that the tab for **Queries** is not grayed out, indicating that diverse preselected queries may also be selected. Using these queries, the user may create more complex queries. For example, in order to create the query returning the research topic of the persons working in the PADLR project, the user first creates the query $\mathsf{role}(<\mathsf{memberOf}>, \{\mathsf{PADLR}\})$, then uses it together with the relation $\mathsf{isWorkedOnBy}$ to create the query:

$$\mathsf{role}(<\mathsf{isWorkedOnBy}>, \mathsf{role}(<\mathsf{memberOf}>, \{\mathsf{PADLR}\}))$$

.

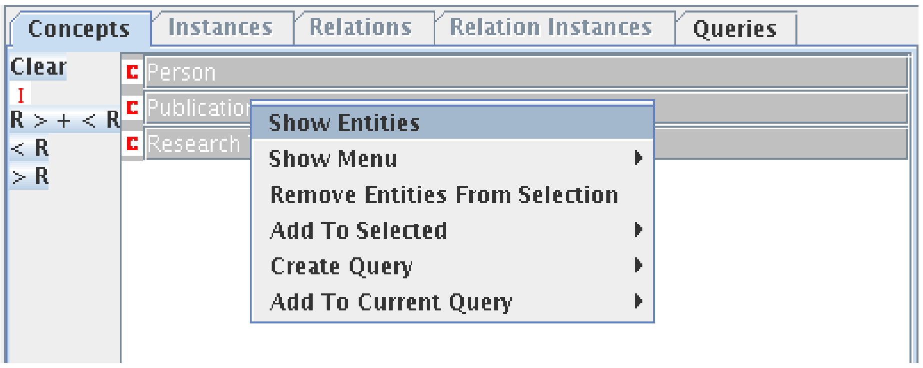


Figure 5.8: The constructor parametrisation panel with the parameters for the view displaying the distribution of the research topics of the projects carried out by the Knowledge Management Group.

Since entities are selected by types, it is possible to present to the user actions which are appropriate for the user. For example, a user may retrieve the subconcepts of the concept selected in the concept panel. A number of default context index constructions (see Chapter 4, Section 4.3.2) can also be proposed to the user if the type of the selected entities is known in advance.

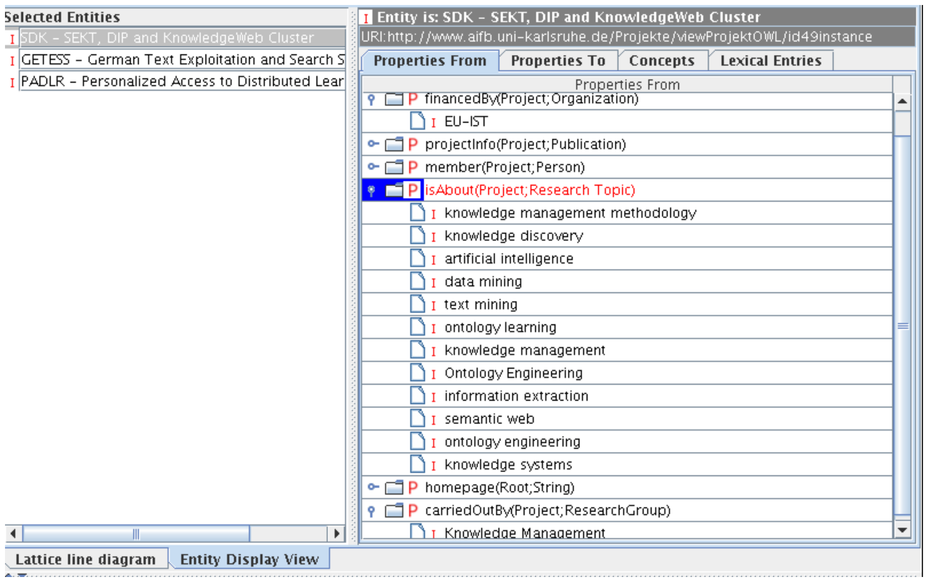
The same principle applies to direct manipulation. Once a collection of relation is dropped on a set query parameter, the type of its result is determined in order to display the appropriate wizard. Using this wizard allows the user to create more complex queries from the dropped query.

Supplementary Panels

Two further panels play an important role: the item property panel and the query panel.

The first of the two is designed to give a detailed representation of a given entity. We added a list containing a number of entities to be visualised so that the user may study the properties of a list of elements one at a time. This panel is shown in Figure 5.9. It displays the three projects of the Knowledge Management Group which deal with text-mining. It was obtained from the extent of the formal concept labeled with text-mining.

The query panel is a special panel displaying the properties of a given query. This allow the user to perform a number of operations on the queries themselves. For example, user can delete some parameters of queries. Direct manipulation is also possible on the query panel. This panel is optional because most of its functionalities can be provided by wizards.


Selected Entities

I SDK – SEKT, DIP and KnowledgeWeb Cluster
I GETESS – German Text Exploitation and Search S
I PADLR – Personalized Access to Distributed Lear

I Entity is: SDK – SEKT, DIP and KnowledgeWeb Cluster
URI:http://www.aifb.uni-karlsruhe.de/Projekte/viewProjektOWL/id49instance

Properties From

P financedBy(Project;Organization)
  I EU-IST
P projectInfo(Project;Publication)
P member(Project;Person)
P isAbout(Project;Research Topic)
  I knowledge management methodology
  I knowledge discovery
  I artificial intelligence
  I data mining
  I text mining
  I ontology learning
  I knowledge management
  I Ontology Engineering
  I information extraction
  I semantic web
  I ontology engineering
  I knowledge systems
P homepage(Root;String)
P carriedOutBy(Project;ResearchGroup)
  I Knowledge Management

Figure 5.9: The projects of the Knowledge Management Group dealing with text-mining displayed in the entity panel.

Moreover, different wizards are presented to the user at diverse stages of the use of the framework. A wizard is a component designed to help the user in completing a complex process by offering a number of clear, simple and controlled steps to perform. Some of the means of parametrising the queries and constructors rely heavily on appropriate wizards. Important wizards are described in the next section dealing with the parametrisation issues.

5.5.2 Parametrisation using Direct Manipulation

Now that we presented the relevant panels, we can describe in more detail the process of parametrising constructors. As explained in Chapter 4, Section 4.3.3, one of the main means of creating relevant views is the use of constructor parametrisation. We also showed that there are two possible ways of parametrising a constructor. Either the parametrisation is parameter oriented or it is constructor-oriented.

For both types of parametrisation some selected elements are dropped on the interface of the constructor parametrisation panel (see page 177). If the selection is dropped over one of the query button, the parameter oriented approach is used, whereas the constructor parametrisation is used if the selection is dropped on the label of the constructor, in the Figure 5.7 the selection has to be dropped over the name: RELATION.

Parameter-Oriented Parametrisation Wizards

As seen in Chapter 4 ,Section 4.3.3, this type of parametrisation only considers the query on which the drag-and-drop operation is being performed. Depending on a number of possibilities, different wizards are presented to the user. In our implementation we used eight different parametrisation panels, depending on three variables.

First, the arity of the parameter to be parametrised is taken into consideration. A different kind of panel is presented to the user whether the query represents a relation or a set.

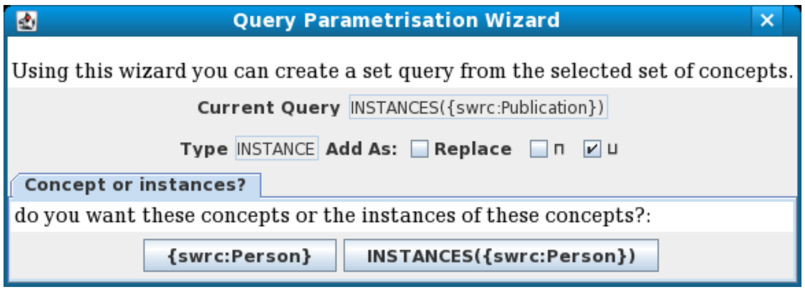
Second, the type of the elements that are being dropped on the given parameter also influences the choice of the panel. These elements may be of one the following predefined type: concept, instances, relation, relation instances.

In case the selected element is a query, the result type of this query is then used. Finally, the last variable depends on whether this parameter has already been parametrised, in other words, whether this parameter already contains a query. In such a case, a choice between replacing this parameter, performing an intersection of the present query with the one which will be created, or performing the union of the results is given to the user.

We present in Table 5.7 a summary of the main possibilities for the creation of queries using a drag-and-drop. The first column informs on the arity of the query on which the drop action occurs. The second column tells the type of elements which this query returns. The third columns specifies the type of the elements which have been dropped. The fourth column specifies one type of operation which can be performed on the input, for example the query $\mathsf{parentconcepts}(q_I)$ returns an instance query. The Operators which operators can be used to combine the two queries of column 2 and 4. These operators are typically $\cup, \cap, \setminus$. Finally, the last column indicates the result of the construction.

Note that the queries with role queries are somewhat more complex. They require the selection of supplementary input. But this can be obtained from the selection panel.

The panel shown in Figure 5.10 is presented to the user if he drops the selection Person over the query parameter which was set to $\mathsf{inst}(\{\mathsf{Publication}\})$. The user can choose whether he wants to create an instance query or an element set: $\{\mathsf{Person}\}$. This new query can either replace or be combined with the preceding query using the following operators. Under the information on the current query, the user can choose one of the three alternatives: replace or creating an union query, for example $\mathsf{inst}(\{\mathsf{Publication}\} \cup \mathsf{inst}(\{\mathsf{Person}\}$.


Query Parametrisation Wizard

Using this wizard you can create a set query from the selected set of concepts.

Current Query INSTANCES({swrc:Publication})

Type INSTANCE Add As: ☐ Replace ☐ ∩ ☑ ∪

Concept or instances?

do you want these concepts or the instances of these concepts?:

{swrc:Person} INSTANCES({swrc:Person})

Figure 5.10: A panel to refine a query by dropping Person over the query parameter $\mathsf{inst}(\{\mathsf{Publication}\})$.

Table 5.7: A summary of the main possibilities for the creation of queries using a drag-and-drop.

arity	Existing Query	Elements added	Input transform	Operators	Result
1	Concept:q_c	Concept Set: c_1	element set q_{c_1}	$* = \cup, \cap, \setminus$	$q_c * q_{c_1}$
1	Concept:q_c	Instance Set:i_1	$\mathsf{parentconcepts}(q_{i_1})$	$* = \cup, \cap, \setminus$	$q_c * \mathsf{parentconcepts}(q_{i_1})$
1	Concept:q_c	Relation Set:r_1	$\mathsf{domainconcepts}(q_{r_1})$	$* = \cup, \cap, \setminus$	$q_c * \mathsf{domainconcepts}(q_{r_1})$
1	Concept:q_c	Relation Set:r_1	$\mathsf{rangeconcepts}(q_{r_1})$	$* = \cup, \cap, \setminus$	$q_c * \mathsf{rangeconcepts}(q_{r_1})$
1	Instance:q_I	Instance Set: i_1	element set (q_{i_1})	$* = \cup, \cap, \setminus$	$q_I * q_{i_1}$
1	Instance:q_I	Concept Set: c_1	$\mathsf{inst}(q_{c_1})$	$* = \cup, \cap, \setminus$	$q_I * \mathsf{inst}(q_{c_1})$
1	Instance:q_I	Relation Set:r_1	$\mathsf{range}(q_{r_1})$	$* = \cup, \cap, \setminus$	$q_I * \mathsf{range}(q_{r_1})$
1	Instance:q_I	Relation Set:r_1	$\mathsf{domain}(q_{r_1})$	$* = \cup, \cap, \setminus$	$q_I * \mathsf{domain}(q_{r_1})$
1	Instance:q_I	Relation Set:r_1	$\mathsf{role}(q_{r_1}, \mathsf{range}(q_{r_1}))$	$* = \cup, \cap, \setminus$	$q_I * \mathsf{role}(q_{r_1}, \mathsf{range}(q_{r_1})$
2	Relation:q_r	Relation Set: r_1	element set q_{r_1}	$* = \cup, \cap, \setminus$	$q_r * q_{r_1}$
2	Relation:q_r	Concept Set: c_1	$\mathsf{related}(q_{c_1})$	$* = \cup, \cap, \setminus$	$q_c * \mathsf{related}(q_{c_1})$
2	Relation:q_r	Instance Set: i_1	$\mathsf{related}(q_{i_1})$	$* = \cup, \cap, \setminus$	$q_c * \mathsf{related}(q_{i_1})$

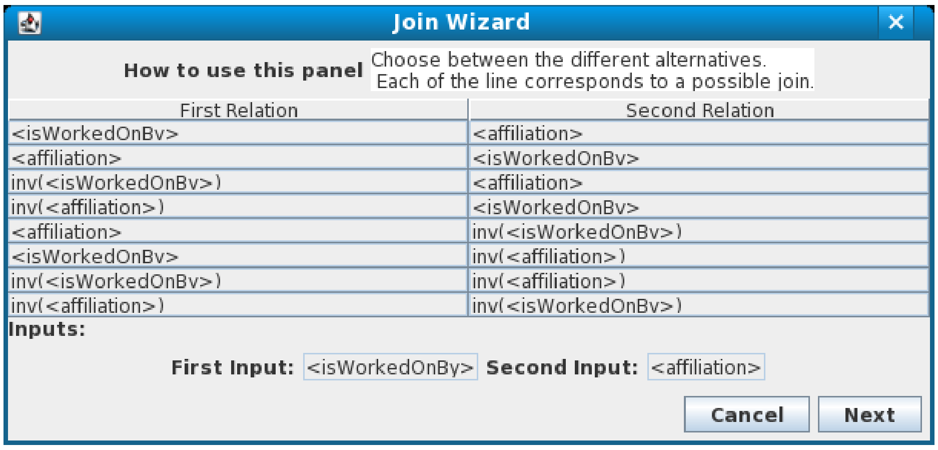
Constructor-Oriented Parametrisation Wizard

The constructor-oriented parametrisation mainly depends on the type of elements selected as well as the type of constructor to be parametrised. Since the diverse subsumption and hierarchy constructors require little parametrisation, no constructor wizard has been implemented.

For the relations constructors, this kind of parametrisation is extremely useful because they usually require more information during the parametrisation process. For example the join constructor necessitates at least two relation queries to be parametrised. Moreover, though the relation constructor is capable of building the view with only a relation query, the concept lattice of such a view maybe too large to be visualised, therefore a more refined parametrisation is useful.

For the constructor oriented parametrisation, mainly two kinds of parametrisation are important.

The first kind consists in filling as many parameters as possible. A user trying to parametrise the join constructor by dropping a selection containing two relations may choose between eight alternatives as to which kind of join is desired. The wizard displayed in Figure 5.11 is shown to the user to let him select one of the possible joins.



How to use this panel Choose between the different alternatives. Each of the line corresponds to a possible join.

First Relation	Second Relation
<isWorkedOnBv>	<affiliation>
<affiliation>	<isWorkedOnBv>
inv(<isWorkedOnBv>)	<affiliation>
inv(<affiliation>)	<isWorkedOnBv>
<affiliation>	inv(<isWorkedOnBv>)
<isWorkedOnBv>	inv(<affiliation>)
inv(<isWorkedOnBv>)	inv(<affiliation>)
inv(<affiliation>)	inv(<isWorkedOnBv>)

Inputs:

First Input: <isWorkedOnBy> **Second Input:** <affiliation>

Figure 5.11: The wizard obtained when dropping the relations and on the join context construction panel.

Once he has chosen an alternative, he can keep refine the definition of this join by using the next button. This corresponds to the other approach which consists in a refinement process where each parameters is refined for constructing a join approach. Given the selection used for the parametrisation, a number of alternatives are proposed to the user. For example, if a set of instances is dropped on a CoRelation constructor, the interface suggests setting these as the start elements. In case of a positive answer, the interface suggests a number of relevant relations for this set of instances from which he can select the one he wishes to visualise.

5.5.3 The Lattice Panel

Since visualisation and selection are two main task for the whole application, the lattice panel plays a central role in the query-based multicontext browser. This panel was implemented using the lattice panel of the Concept Explorer which is described in Section 5.6.1. We added a number of features have been added to the original panel (i.e. the one delivered with the Concept Explorer).

Context Menu Interaction

We added a context menu allowing the user to selection parts of the concept lattice as well as to perform other actions on formal concepts such as saving the elements in their extent or intent. The context menu allows users to select each of the parts of the formal concepts (i.e. intent, extent, attribute contingent, extent contingent) and apply different actions on them. Typical actions are creating queries using the elements, or adding the elements to the current selection or visualising the elements in the entity panel. Finally, default constructors can be employed on these elements to generate a new view. For example, if the selected elements are concepts, the user can generate a subsumption/instance hierarchy for this set of concepts.

Double Middle Click Selection

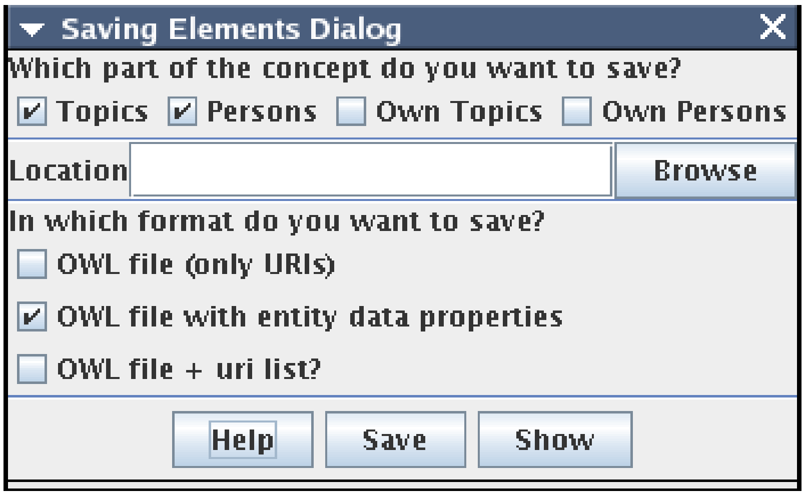
The second feature is a selection method based on a double middle click of the mouse. This is used to select the part of the formal lattice on which the double click has been performed. This method is one of the particular advantages of using the line diagram of a concept lattice because it is a quick and intuitive way of selecting a great amount of elements sharing common properties. This selection mechanism is coupled with the filtering mechanism presented in Chapter 4, Section 4.2.2.

Display Strategies

Among the important feature we added are new strategies or extensions of existing ones for displaying the concept lattice elements. One of the main adaptations consists in hiding a number of irrelevant edges from the visualisation so that the lattice diagram is easier to read. Another important adaptation is providing an alternative algorithm for the layout lattice diagram. This alternative algorithm allows to change from an object-based minimal intersection algorithm to one based on attributes. This alternative implementation is a simple reimplementation of an existing display algorithm found in Concept Explorer.

Saving Models of Relations

We also implemented the possibility to save models of parts of the ontology by using the selection possibilities of the query-based multicontext browser. The user may select a formal concept with a label which interest him and save an OWL mode. In Figure 5.12, the panel used to save parts of a formal concept of a lattice with context index: $(\mathsf{inst}(\{\mathsf{Topic}\}),\mathsf{inst}(\{\mathsf{Person}\}),< \mathsf{isWorkedOnBy} >)$. The user may select to save all objects (here topics) or all attributes (here persons) or only the object contingent and attribute contingent of the formal concept.


Saving Elements Dialog

Which part of the concept do you want to save?

[✓ Topics] [✓ Persons] Own Topics Own Persons

Location

In which format do you want to save?

OWL file (only URIs)

[✓ OWL file with entity data properties]

OWL file + uri list?

Figure 5.12: A panel to save parts of a formal concept of the lattice for the context index (inst({Topic}),inst({Person}),< isWorkedOnBy >).

5.5.4 Lattice Partitioning

In this paragraph, we discuss an algorithm to partition the lattice top formal concepts of the lattice. The fundamental idea is to partition the top elements of the lattice in such a way, that the intersection of the extents of two formal concepts belonging to two different partitions is equal to the extent of the bottom element of the lattice.

The idea behind this algorithm is to be able to separate independent parts of the lattice. This can be used in diverse manners. One of the main applications, is that it allows the improvement of the lattice display.

Anticomplementation Relation

This can be formulated using the following definition:

Definition 41 (Lattice Separation Classes) *Let L be a lattice $(L, \wedge, \vee, \bot, \top)$ We define the anticomplementation relation as:*

$$x \sim y :\Longleftrightarrow x \wedge y \neq \bot$$

We define the antiseparation relation as the transitive closure of the anticomplementation relation. It is an equivalence relation on $L \backslash \{\bot, \top\}$ (it is clear that the anticomplementation relation is reflexive and symmetric on $L \backslash \{\bot, \top\}$).

For x in L, we denote the equivalence class of x by $[x]_\sim$.

It should be noted that the idea comes from reading [Ganter and Kwuida, 2005] on pseudo-com–plemented lattices. Unfortunately most of the lattices we encounter are not pseudo complemented. This means we cannot use the results presented in the article.

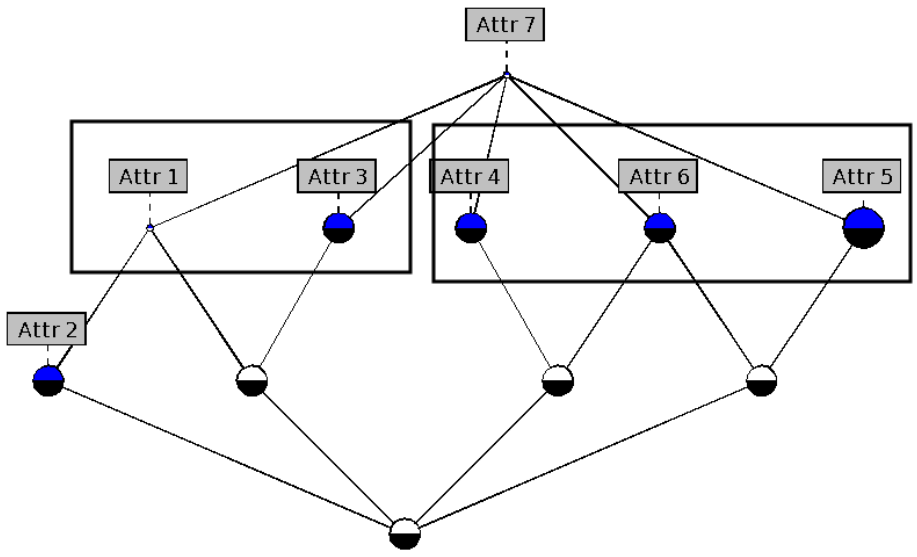


Figure 5.13: A lattice with a partition of the top elements in two classes: {Att1,Att3} and {Att4,Att5,Att6}).

To illustrate this, consider the lattice presented in Figure 5.13. There are two top elements partitions: {Att1,Att3} and {Att4,Att5,Att6}. Note also that there is an attribute for the top concept and there is an object in the bottom concept.

It is possible to use the partitioning to display the two parts of the lattice separately as in Figure 5.14. This simplifies the visualisation of the larger lattices which can be split into much smaller lattices. But it is important to note that not every lattice can be partitioned in more easily displayed lattices. The publication lattice for example cannot be split easily in useful lattices, because there are only four classes in the partition. Three classes are extremly small covering each one object, while the last one has more than thousand formal concepts and covers 843 of the 846 objects. In comparison, the lattice of the worksAtProject relation which contains 94 formal concepts, can be partitioned in 12 classes. One of these classes is shown in Figure 5.15. It is much smaller, and the other classes can be displayed in the same way.

Partitioning algorithm

We provide here a simple algorithm to perform this task in Algorithm 5.8. As it can easily be seen the presented algorithm is greedy. The first part of the algorithm are two simple checks. First of all, if the height of the lattice is two or less, then the extents of the subelements of top are all disjoint pairwise (the lattice is flat, so the top concepts are naturally partitionable). The second criteria checks whether the sum of the size of the union of all the partitions is larger than the size of the extent of the top elements (without the elements of the contingent of top). The partitions are ordered by decreasing size, this means that as soon as the sum of the size of the partitions equals the size of the extent

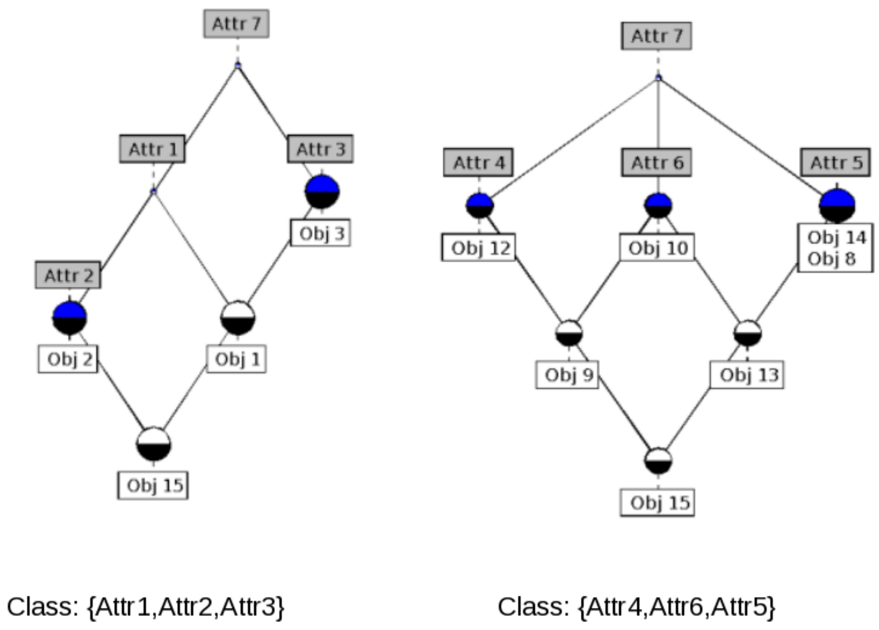


Figure 5.14: The concept lattices of the the two classes {Att1,Att3} and {Att4,Att5,Att6} of the partition of the lattice of Figure 5.13.

of top minus the size of the contingent of top, the algorithm terminates.

Using this algorithm, it is possible to present the partitioned data to the user, thus simplifying the structure of the display of large lattices. However, this algorithm has not been used to for the visualisation evaluation in Chapter 6, Section 6.2.

5.6 Underlying FCA Framework

In order to build our browsing framework, we used an available open source Formal Concept Analysis tool named Concept Explorer (sometimes also called ConExp). We first examine the alternatives we had when choosing this tool as a basis framework. Then we describe the main features already available in the tool. Then we discuss how we use the code base of the Concept Explorer in order to build our graphical interface.

Why reuse an Existing Framework?

We considered different kinds of underlying infrastructures for our tool. We examined the advantages and disadvantages for these different infrastructures. The results of our considerations can be summed up through the list of requirements that our tool infrastructure should have. Once we had created such a list, we studied the positive and negative aspects of the three alternatives which we discuss in the next paragraph.

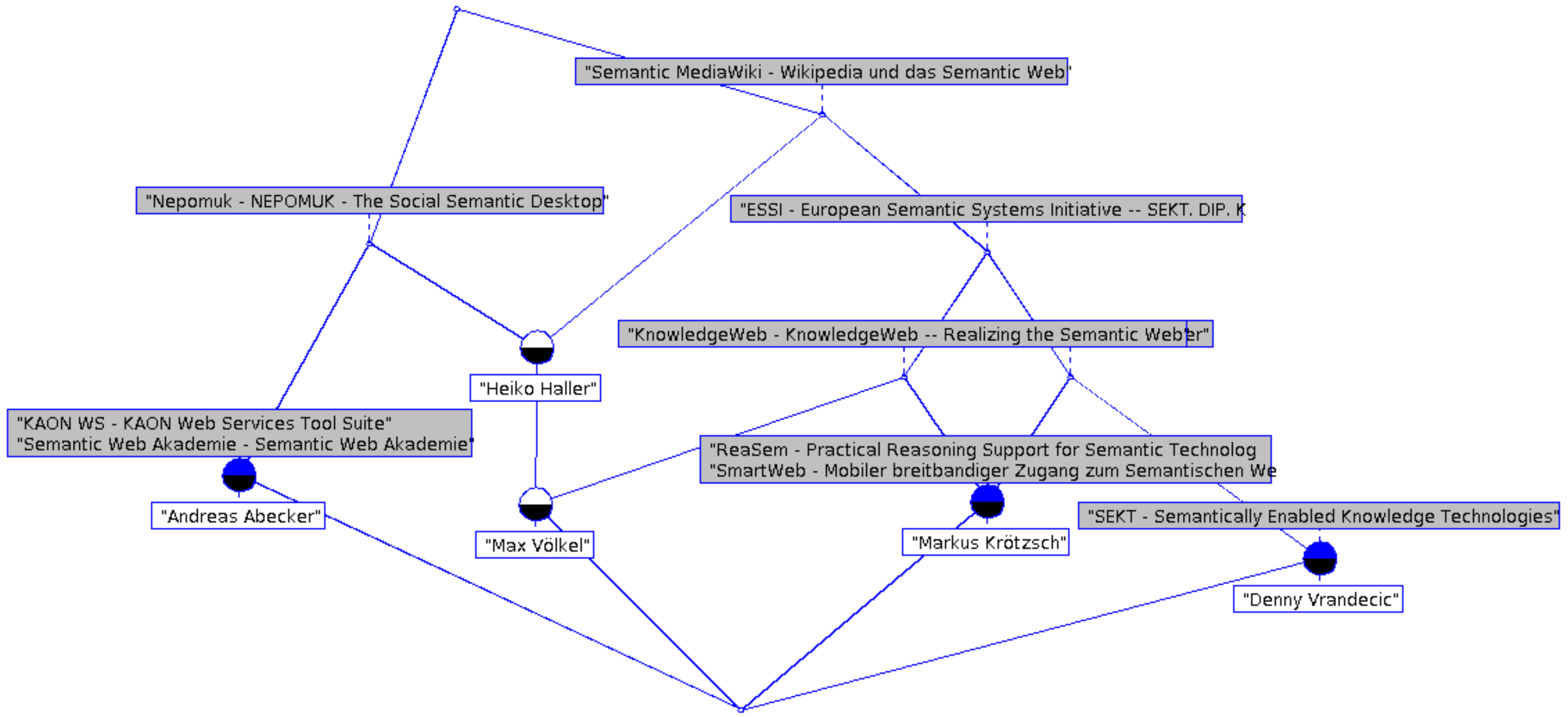

"Semantic MediaWiki - Wikipedia und das Semantic Web"
"Nepomuk - NEPOMUK - The Social Semantic Desktop"
"ESSI - European Semantic Systems Initiative -- SEKT. DIP. K
"KnowledgeWeb - KnowledgeWeb -- Realizing the Semantic Weber"
"Heiko Haller"
"KAON WS - KAON Web Services Tool Suite"
"Semantic Web Akademie - Semantic Web Akademie"
"ReaSem - Practical Reasoning Support for Semantic Technolog
"SmartWeb - Mobiler breitbandiger Zugang zum Semantischen We
"Andreas Abecker"
"Max Völkel"
"Markus Krötzsch"
"SEKT - Semantically Enabled Knowledge Technologies"
"Denny Vrandecic"

Figure 5.15: The concept lattice of a class of a partitioning from the concept lattice of < workedAt >.

Table 5.8: Calculate the partition of the top elements of the lattice.

```
Input: formal concept lattice $L$
Output: a partition $P$ of the top concepts
  // Define functions:
  function part(c) creates a partition object
  function height($L$):= length of the longest chain in $L$
  function topsubconcepts(L):= return the top subconcepts of $L$
  function extent(p):= $\Sigma_{c \in p} |extent(c)|$
  // End of function definitions
  $P := \emptyset$
  for all c in topsubconcepts($L$) do
    $P$:= $P \cup$ part($\{c\}$)
  end for
  if height($L$)= 2 then
    return $P$
  end if
  sumpartition:= $\Sigma_{p \in P}$ extent($p$) - ($|P| - 1$) * |extent($bottom$)|
  if sumpartition = |extent(top)| - |contingent(top)| then
    return $P$
  end if
  for all a pair ($p_1$,$p_2$) of partitions in $P$ do
    if |extent($p_1$) $\cap$ extent($p_2$)| $>$ |extent($bottom$)| then
      $p_3$ := merge($p_1$,$p_2$)
      $P$:= $P \setminus \{p_1, p_2\}$
      $P$:= $P \cup p_3$
      sumpartition = sumpartition + size($p_3$)- size($p_2$) -size($p_1$)
    end if
    if sumpartition = |extent(top)| - |contingent(top)| then
      return $P$
    end if
  end for
```

We now list our requirements regarding the implementation of such an underlying infrastructure:

- an open source framework
- a Java-based FCA framework and portability (most current RDF(S) and OWL implementations provide a Java API),
- efficient context infrastructure,
- efficient lattice calculation,
- good and stable code base, and
- suitable for experimentation with labels and browsing.

5.6.1 Alternatives

The list of requirement lead us to look at three solutions amongst the alternatives we investigated:[9]

- building our own architecture,
- using the ToscanaJ architecture, or
- using the ConExp architecture.

Own Architecture

The implementation of the whole infrastructure from scratch has some interesting aspects, it would have allowed a greater implementation freedom. But reusing an existing infrastructure allowed to make sure the Formal Concept Analysis framework was stable and the software engineering quality of the Concet Explorer or ToscanaJ convinced us very early that reusing an existing framework is a better alternative. Rebuilding the whole architecture would have required much more skill and time than we could allow.

ToscanaJ

While we resolved our choice on the Concept Explorer, another framework had many features which made it attractive as an underlying infrastructure for our browser. This framework is actually a suite of tools called ToscanaJ.[10] It consists of three tools, namely, Toscana, Elba and Sienna. The tool suite was a reimplementation in Java of an older framework called Toscana. Toscana has been described in [Kollewe et al., 1994] and ToscanaJ was presented in [Hereth-Correira and Kaiser, 2004]. It has many interesting features, such as nested line diagrams (see Chapter 2, Section 2.3.3). The software engineering quality of the ToscanaJ framework makes it also attractive as an underlying Formal Concept Analysis framework. However, the ToscanaJ toolsuite was not as developed as it is currently at the time of our choice and the workflow of ToscanaJ using three different tools instead of one made it less suitable for our purposes.

The Concept Explorer

The Concept Explorer[11] is a tool developed by Serhiy Yevtushenko. It is a small tool implementing many interesting functionalities of Formal Concept Analysis frameworks such as:

- context editing,
- lattice drawing panel (using different layout algorithms),

[9]Early January 2005, a team from Montreal released another open source FCA alternative framework: Galicia. Time and some design choices made it unsuitable to switch the already implemented architecture to this platform. The tool can be found under sourceforge at: http://galicia.sourceforge.net.

[10]The software can be downloaded at: http://www.sourceforge.net/project/toscanaj.

[11]The Concept Explorer can be downloaded from http://www.sourceforge.net/projects/conexp.

- association rules framework, and
- attribute exploration.

The Concept Explorer offers a well designed architecture for experimenting with Formal Concept Analysis algorithms with an efficient bitset implementation of binary relations. This was the main reason of our choice. The next section gives more details about the tool.

5.6.2 Concept Explorer

We now describe in more details the main features of the Concept Explorer. We address four topics which have consequences on our implementation.

Context

An important aspect of the Concept Explorer is the underlying context data structure. which uses bitsets to code the lines of the contexts. Given an ordered set A of elements, a bitset is a binary word which encodes the presence or absence of an object at a given position in A. For example for the ordered set $A := \{$Cat, Dog, Fish$\}$, 010 encodes the subset $\{$Dog$\}$ of A. The use of bitsets allows for an efficient implementation of set comparisons and has practical consequences on the way the whole infrastructure is designed. The performance of context algorithms is influenced by the form of the context since the use of bitsets implies that object sets comparison can be performed more efficiently than attribute sets. This means that it is sometimes better to use the transposed context and then inverse the lattice.

Lattice Algorithms

For the generation of lattices, the Concept Explorer uses the commonly used Next Closure algorithm (presented in [Ganter and Reuter, 1991]) which is perfectly adapted to bitset implementation of binary relations. A number of other algorithms exist to compute the formal concepts (see for example, [Bordat, 1986, Godin and Mili, 1993, Stumme et al., 2002]). Next closure is especially suitable for the bit set implementation found in the Concept Explorer, but it might be particularly interesting to experiment with other algorithms such as TITANIC (see [Stumme et al., 2002]). This would require the introduction of some primitive in the query languages used by the Query-Based Multicontext infrastructure.

Lattice Layout Algorithms Concept Explorer provides a number of lattice layout algorithms: minimal intersection layout, Freese layout, chain decomposition, force layout, layered layout. A description of some of these algorithms can be found in [Cole, 2001b].

Decoration Settings Concept Explorer provides a number of line diagram decoration options. For example, it is possible to change the size of the nodes or the highlighting mechanism. All these techniques greatly enhance the browsing experience and are implemented in most advanced Formal Concept Analysis tools. However Concept Explorer seemed at the time to offer the best possibilities for experimentation of these settings.

5.7 Summary

In this chapter, we described the implementation of the query-based multicontext browser. We first presented an overview of the architecture.

We have shown in Section 5.3 the role played by queries, context indices and constructors in the generation and composition of formal contexts. In particular, we saw how the QBMC context engine may use different strategies to evaluate context indices, constructors and context index operators depending on the particular elements of their definition. We described the method used to implement our approach on an engine using datalog. Moreover, we presented a solution in order to be able to use nominals and queries returning sets of relations and concepts.

We also presented in Section 5.4 a methodology to adapt existing ontology frameworks to our application. This methodology has been applied to two different types of data source: KAON and KAON2.[12]

The basic GUI elements of our interface have been described in Section 5.5, where we also discussed the role played by Wizards in the context index definition mechanism using direct manipulation.

Finally, we discussed in Section 5.6 the choice of the Formal Concept Analysis infrastructure we used to implement our approach.

To conclude this description of our implementation, it is important to note that the actual framework is still in a very primitive state. Much work should be performed to have it fully functional for an enterprise. This is partly due to the appearance of the user interface which is not very attractive. Though we tried to create an intuitive interface, the learning curve is still a major drawback. Though we try to guide the user with tool tips and or with a tutorial, the entry level of the interface may remain too high for many users.

Many optimisations could still be performed. But the real bottleneck of the application is the slow layout mechanism. Though the formal contexts are usually created quite quickly, the layout of complex concept lattices remains somewhat of a problem.

The next chapter discusses the evaluation of our approach.

[12]Since KAON is based on a RDF(S) ontology language, whereas KAON2 is based on a language derived from description logics, their choice depends on the models of the data used and whether some of the more complex primitives of OWL are needed for the exploration.

Chapter 6

Evaluation

The three preceeding chapters presented the theoretical background (Chapter 3), the user interaction (Chapter 4) and the implementation (Chapter 5) of the query-based multicontext framework. We now turn to the evaluation of our approach.

In this chapter, we first present a comparison of our approach with existing ones in Section 6.1. In Section 6.2, we describe an evaluation we performed to compare the performance of users on three different paradigms when visualising generated views.

6.1 Comparison with other Approaches

In this section, we discuss the difference between our approach and other related approaches. In particular, we focus on other knowledge or document browsing approaches which are based on Formal Concept Analysis.

6.1.1 ClusterMap

A number of graphical approaches to visualisation have been proposed in the literature. Many of these approaches are based on graphs since they are capable of displaying many-to-many relations. In this section, we discuss a visualisation technique for knowledge bases which has many similarities with the one we propose.

Description of the Approach

Fluit et al (see [Fluit et al., 2003a, Fluit et al., 2003b]) introduced a novel graph-based visualisation paradigm called *Cluster Map* and used it in an ontology-based visualisation system named *Spectacle*. The purpose of this visualisation paradigm is to show the distribution of objects among a number of selected classes from a hierarchy. It uses the spring-embedder algorithm from [Eades, 1984]. An example of Cluster Map is found in Figure 6.1.[1]

This map shows available job vacancies classified by economic sector. It can be read according to the following conventions:

[1]This example and others can be found in [Fluit et al., 2003a]. The corresponding software can be downloaded from `http://aduna-software.com/products/autofocus/download.html`.

- the labelled nodes represent the various economic sectors,
- the bubbles with the multiple spheres represent the job vacancies available,
- the numbers between braces in the labels indicate the number of total vacancies available within the sector.

Using this visualisation paradigm, shared elements can be easily identified. For example, it shows that the sectors IT and Technology share five vacancies or that Finance & Administration and Creative Professions share one. Finally, the interface of Spectacle offers a way of querying the ontology by selecting the classes or terms to be displayed.

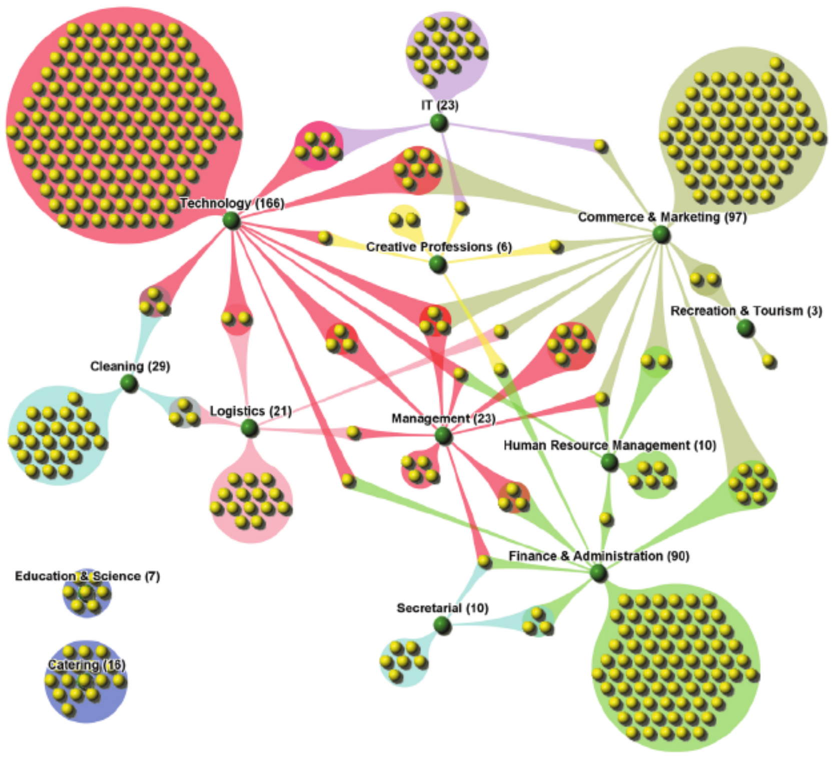


Figure 6.1: Job vacancies organised by economic sector.

Comparison

The ClusterMap approach has many similarities with our approach. The purpose of this visualisation paradigm is to display the relation between two groups of entities: objects and classes. It is similar to our notion of view, since the underlying datamodel can also be seen as a formal context, where classes are attributes. In other words, the formal contexts generated from context indices could also be used as an underlying structure for a Cluster Map.

The difference between the Spectacle approach and the one we propose lies in two aspects: the graph visualisation paradigm and the graph creation methodology. While our approach relies on the line diagrams of a concept lattice, Spectacle uses graphs to display the intersections of diverse classes. Selecting all the elements sharing diverse attributes is facilitated using the lattice view as it can be done in one step while cluster maps would require to select one after another all the possible intersections. A cluster map, however, may be more intuitive to users at first.

We conclude this comparison by pointing on the necessity of further investigations. Both the query-based multicontext and the Spectacle approach rely on some kind of user interaction to create relevant views to be displayed. The techniques presented in this thesis could be useful to increase the expressivity of Spectacle. In particular, the view model in Chapter 3 (in particular Sections 3.3.3, 3.4.2 and 3.5) is much more expressive than the one underlying Spectacle which does not allow the use of complex concept constructions. Moreover, the view definition methods presented in Chapter 4, Section 4.3.3 can be reused for Spectacle. We believe that using some constructor parametrisation method for creating ClusterMap would give the user more freedom of exploration and better possibilities to the user to focus on particular groups than the current Spectacle interface.

6.1.2 Toscana Framework

We now turn to one of the reference frameworks in Formal Concept Analysis, *Toscana* and compare it with our approach.

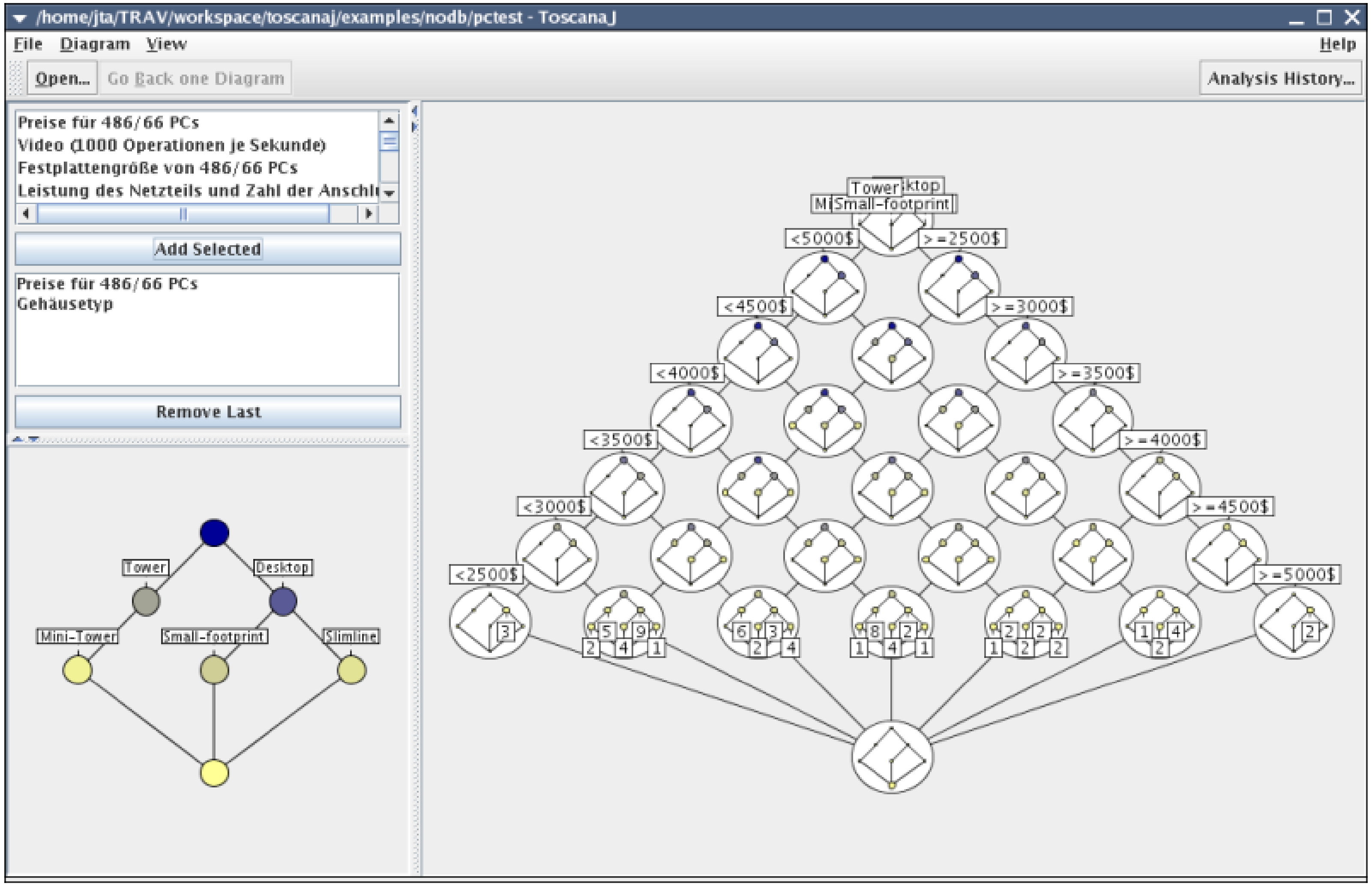
Tool Description

The Toscana framework (and its latest implementation *ToscanaJ*) is based on the idea of conceptual scaling as introduced in [Ganter and Wille, 1989] and relies on a specific usage workflow to define appropriate scales.

The workflow is as follows. A knowledge engineer defines scales which can be seen as logical views on the data. In order to enhance the visualisation of these views, the line diagrams of these scales are edited.

Each of these scales takes the form of a lattice line diagram and addresses a specific aspect. For example, to study the distribution of the price of 486/66 personal computers, one scale is a scale of the price of the computers. Another scale can be the type of hard drive sold with the computer or the type of case of this specific computer. Figure 6.2 shows the graphical user interface of the ToscanaJ tool. In this particular case, the user can compare the relations between two orthogonal views on the data using a nested-line diagram which combines two scales. The first scale is an interordinal scale over the price of 486/66 personal computers, while the second one compares the type of cases for the computers, e.g. whether the case is a tower or a desktop. This visualisation allows to see that the price of most towers is mainly located between \$2500 and \$4000, whereas the desktop cases are distributed over the whole price range.

Once the scales have been displayed, the extent of a node is computed by means of queries over a database. Finally, a domain expert can combine dynamically the scales to discover relevant aspects or confirm hypotheses he has about the data.



/home/jta/TRAV/workspace/toscanaj/examples/nodb/pctest - ToscanaJ

Preise für 486/66 PCs
Video (1000 Operationen je Sekunde)
Festplattengröße von 486/66 PCs
Leistung des Netzteils und Zahl der Anschl

Preise für 486/66 PCs
Gehäusetyp

Figure 6.2: The nested diagram of the lattice for comparing PC price with type of case.

An important feature of the multiscale approach of the Toscana framework is the zooming approach. The idea is that using the different views one gradually restricts the object sets considered by each new scale. In the 486 PC example, a user can restrict the object set to the one with case type Desktop and look at their price distribution. Figure 6.3 shows the resulting view. In the lattice in the left bottom corner, the Desktop node is selected, and in the view on the right only Desktop computers are displayed.

Comparison

The scaling mechanism coupled with the zooming approach (with or without the nested line diagrams) can help summarise certain configuration of the data as was shown in a number of projects where Toscana has been used (see for instance [Wille, 2005] for description of Toscana projects). For example, the diagram in Figure 6.2 clearly shows the distribution of the diverse computer case types among the diverse price ranges. Though our current implementation of the query-based multicontext browser does not yet support nested line diagrams,[2] our approach is compatible with the nested diagrams approach. The main difference between the two approaches lies in the expressivity of the view definition process. While it is possible to focus on specific groups of objects using the zooming approach of the Toscana framework, our view mechanism allows the creation of views which cannot be created using the Toscana framework. Toscana views are based on a modelisation and categorisation of sets of objects, for example, computers have been described according to their price, their types or the processing speed of their CPU. In contrast, Query-Based Multicontext views allow to look at the groups of entities which are formed by the relations they have with other entities. For example, using the Query-Based Multicontext, it is possible to study the groups of persons who coauthored some article.

6.1.3 Docco Framework

In this section, we compare our approach to the Docco[3] application (for technical details see [Becker and Cole, 2003]) which is a simple but useful personal search tool.

Description of the Approach

Docco is a personal document management system capable of visualising the results of keyword-like queries using concept lattices.

Users can submit queries like Java, "Formal Concept Analysis", "Conceptual Graphs" to the system and visualise how well the documents of the collection match the requirements of the query. Figure 6.4 shows that 21 documents share the strings "Formal Concept Analysis" and Java whereas only 4 contain all three keywords. Note that "Conceptual Graphs" and "Formal Concept Analysis" are taken as compound keywords and not as five different keywords. This is one of

[2]The underlying Formal Concept Analysis framework we used did not support nested diagrams and we plan implementing.

[3]See `http://tockit.sourceforge.ner/docco` to download this application.

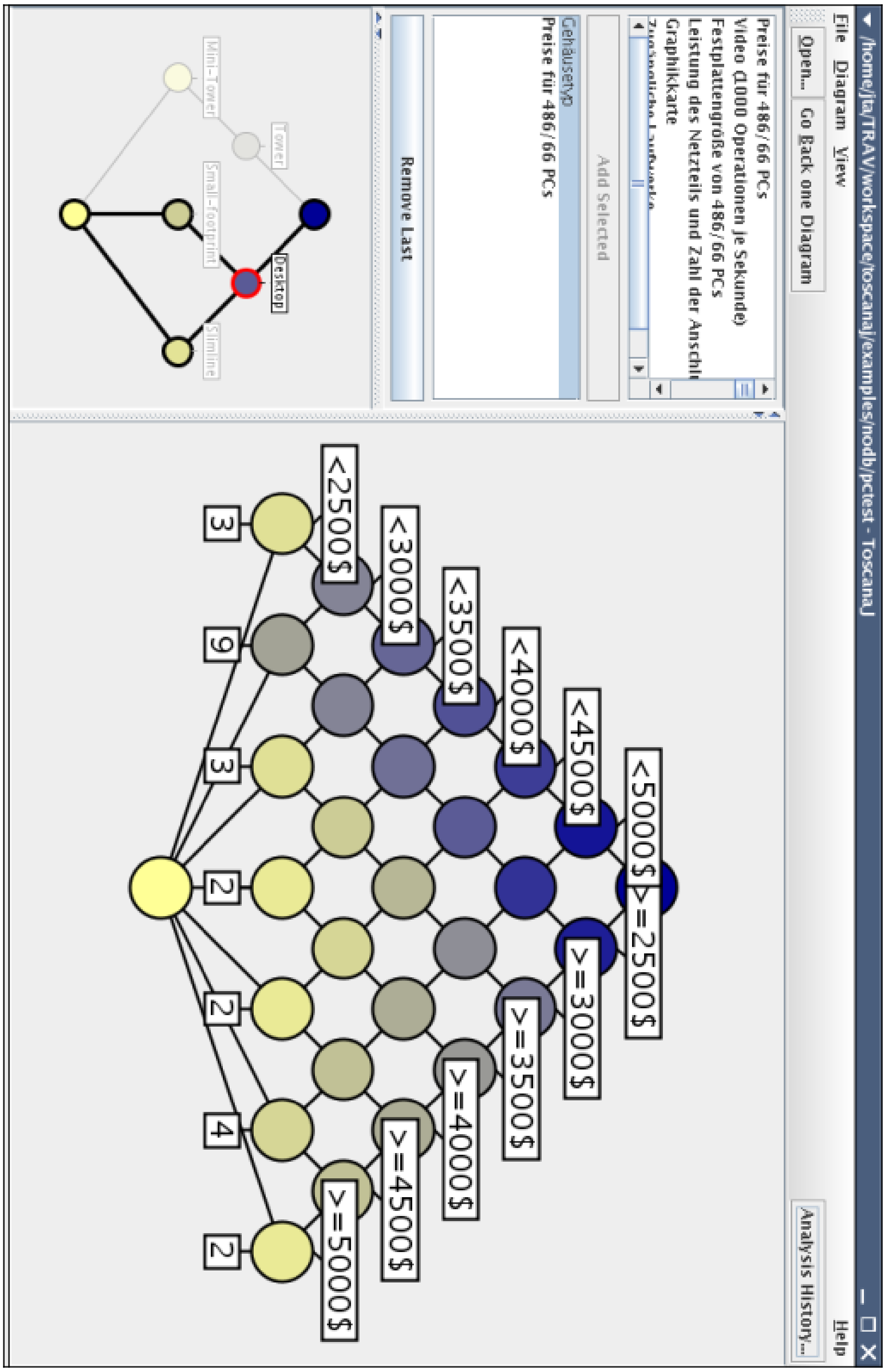
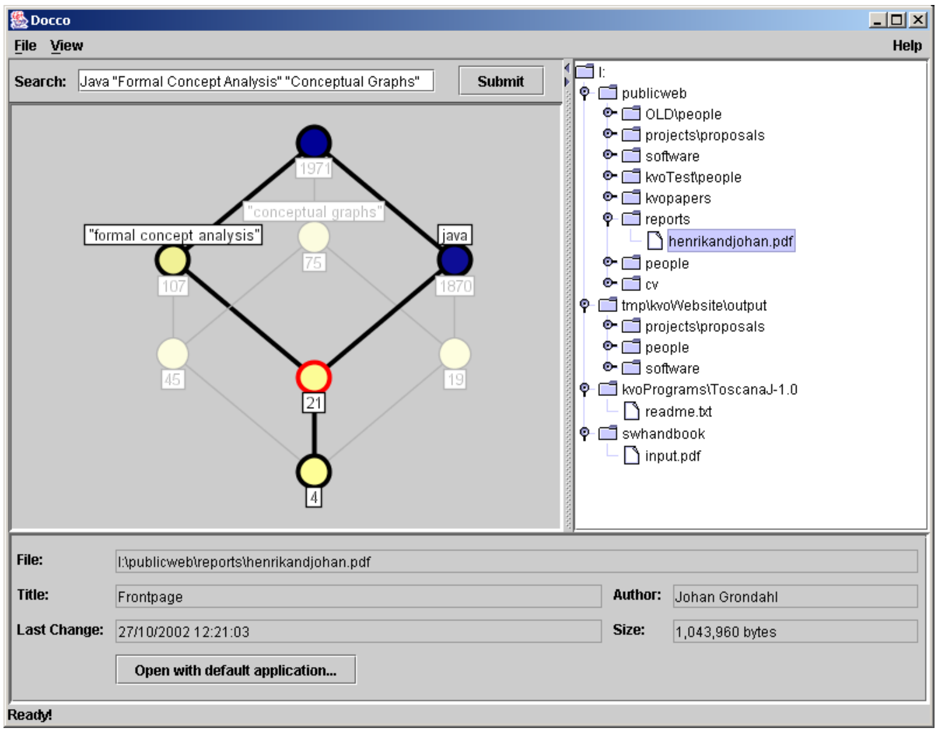


Figure 6.3: The zoomed diagram latice of desktop computer.

the features of the indexing and query engine Lucene which is used as a back end to Docco.



Search: Java "Formal Concept Analysis" "Conceptual Graphs"

File: I:\publicweb\reports\henrikandjohan.pdf

Title: Frontpage Author: Johan Grondahl

Last Change: 27/10/2002 12:21:03 Size: 1,043,960 bytes

Ready!

Figure 6.4: The lattice for the query: Java "Formal Concept Analysis" "Conceptual Graphs" in Docco.

We now explain the principle behind this approach. Every query is then split in literals, which can be seen as atomic propositions. For a given literal, the underlying search engine Lucene returns the set of documents which match the literal, i.e. a literal can be seen as a Boolean test on the document. For example, the literal Java actually test whether the word Java occurs in a document. For the purpose of our explanation, we call match the relation between literals and documents indicating whether a document satisfies a literal. It can then be used as incidence relation of a formal context. In other words, given the set L of literals of the user query and a document collection D, Docco generates the formal context (D, L, match) which can be displayed as a concept lattice.

Interestingly, the search engine Lucene[4] provides other kinds of query primitives. For instance, it is possible to specify that the keywords given should occur in the title or some other field (for example documents keywords). Table 6.1 illustrates different query primitives used in Docco. Observe that the query primitives used by the Lucene indexer are very similar to the CQL[5] standard used by the Library of Congress.

[4]More information on the Lucene search engine can be found at: http://lucene.apache.org/java/docs/.

[5]See http://www.loc.gov/standards/sru/cql/.

Table 6.1: Illustration of different types of queries which can be formulated when using the Docco framework.

Example	Meaning
formal concept analysis	finds the documents containing one of the three words
"Semantic Web" AIFB	finds the documents containing "Semantic Web" or AIFB
ontology AND AIFB	finds the documents containing both terms: ontology and AIFB
title: "Semantic Web"	looks for the document where Semantic or Web appears in the title

Comparison

Docco is an elegant and useful tool to search relevant documents using keywords. Our approach is in fact inspired by the usage of queries in Docco. However, the Docco approach remains limited to document search. It does not allow the use of other kinds of objects or offering a treatment of other background knowledge. Interestingly, it is easy to define a query infrastructure capable of reimplementing the features of Docco. In that way, it can be said that the query-based multicontext architecture is more expressive than the Docco framework.

The approach of Docco could be implemented on top of the query-based multicontext infrastructure. To demonstrate this, the following default query infrastructure could be used as a basis for a framework similar to the Docco framework.

Definition 42 (Docco-like Query Infrastructure) *Let Ω_d be the set of documents accessible in the document collection. Let L_{query} be the set of queries in the syntax used by Docco. We call Ω_l the set of literals of the Docco queries L_{query}. Let $\mathcal{I}$ be a set of Lucene indices with a map called docs from $\mathcal{I}$ to Ω_d which returns the documents of an index $i \in \mathcal{I}$. We define $\Omega := \Omega_l \cup \Omega_d$, in other words the element of the universe are either Docco query literals or documents. Then, we define the query language for documents: $L_{\mathcal{I}}$, it consists of sets of indices, i.e. $L_{\mathcal{I}} := \{I | I \subset \mathcal{I}\}$. Finally, the language L_2 consists of pairs consisting of a set of indices $L_2 := \{(I, q) : q \in L_{query} \text{ and } I \subset \mathcal{I}\}$.*

Then the query infrastructure $(L_{query} \cup L_I, L_2, \Omega, eval_1, eval_2)$ corresponds to the Docco infrastructure, where for $I \in \mathcal{I}$, $eval_1(I) \subset \Omega_d$, for $q \in L_{query}$, $eval_1(q) \subset \Omega_l$ and for $(I, q) \in L_2$, $eval_2((I, q)) = \{(d, l) : d \in docs(I) \text{ and } d \text{ satisfies } l\}$

This approach can be extended to allow semantic indexing of corpora.

Example 21 *For example, for a publication database, if publications are indexed by year and by author, a query like ({2003, 2004, 2005}, title:" Watchdog" DLP ontology) would return the pairs of documents together with the literal they match (i.e. one of title:" Watchdog", DLP or ontology).*

The context index ({Julien Tane, Rudi Studer }, title:" Watchdog" DLP ontology, ({2003, 2004, 2005}, title:" Watchdog" DLP ontology)) would return the context composed of the publications of Julien Tane or Rudi Studer as object *set, classified using the literals: title:" Watchdog", DLP and ontology if their publication year was one of 2003, 2004, 2005.*

6.1.4 The Conceptual Email Manager

In this paragraph, we describe the Conceptual Email Manager and its follow-up applications. Then we compare it to our approach.

Description of the Approach

The Conceptual Email Manager (exposed in [Cole and Stumme, 2000, Cole et al., 2003a]), henceforth CEM, is somewhat similar to Docco in that the type of the objects is fixed. Instead of focusing on documents, the CEM is a tool to organise and interact with large mail collections. Figure 6.5 shows the appearance of a follow-up implementation of the Conceptual Email Manager called Mail-Sleuth[6] and which is sold as a Microsoft Outlook plugin.

CEM extracts relevant terms (words or groups of words) from emails and presents them to the user as descriptors of the emails. These descriptors are then used as attributes of the scales. Some of these descriptors can be the sender of the mail or some words occurring in the subject. The user can choose to accept these descriptors or add some descriptors of his own. He can organise the descriptors in scales. The CEM offers means of adapting the scales to the users needs. For example, in Figure 6.5 the realised scale for the catchword "Clients" is presented.

The CEM uses the same techniques as ToscanaJ for browsing: nested line diagrams and zooming. However, it offers interesting approaches for the definition of the scales. Just like in most mail clients, the CEM lets the user organise the mails into folders. But it introduces the concept of virtual folders (this approach is also present in many recent mail client interfaces, where you can save searches or filters as folders).

Comparison

Like the two previous approaches, the CEM has a fixed type for the objects: Mails. It uses interesting techniques to allow the users to adapt the hierarchies of folders.

In [Cole et al., 2003a], the authors discuss the approach of the Conceptual Email Manager as a knowledge discovery tool. They argue that it is possible to see the interest of Richard Cole for the ECA project can be read from Figure 6.6. However, the nested diagram is quite complex. Using a join constructor, it would be possible to see the correlation between persons and projects. To illustrate this the realised context could then be shown as the lattice in Figure 6.7. It would summarise the project/sender relationship. Of course, this lattice does not contain all the information contained in the nested diagram. For example, many mails of these authors do not deal with any of the four projects but it would make it clearer that Francois Modave did not send any mail on any of the four problems. The join context would not inform on this status.

An interesting aspect of the CEM is that it allows the integration of the user judgement for the classification of mails and for the construction of the scales.

[6]To download a free trial version of Mail-Sleuth, see http://www.mailsleuth.org.

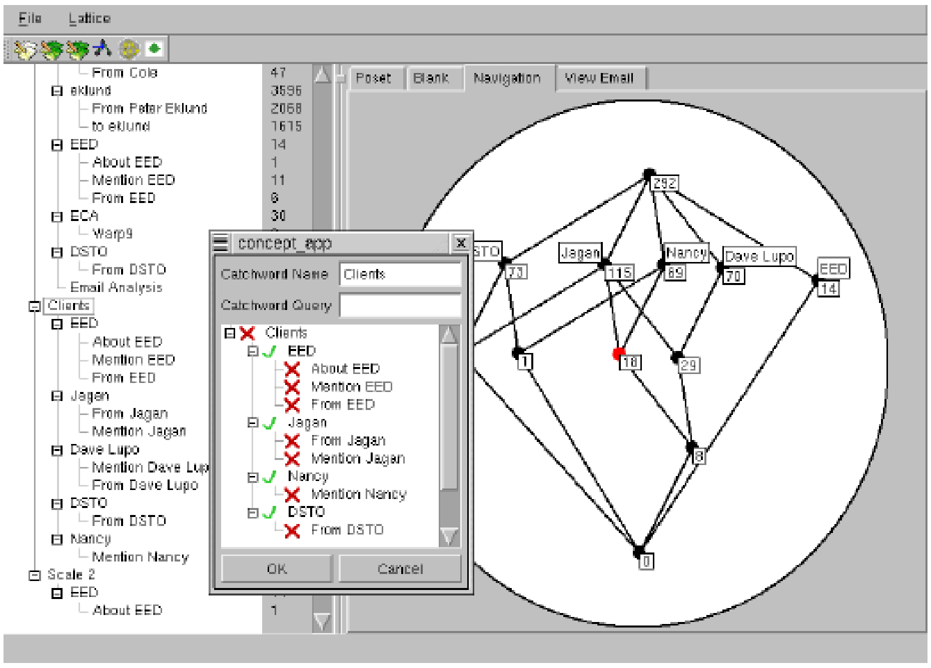


Figure 6.5: The Mail-Sleuth user interface (source [Cole et al., 2003a]).

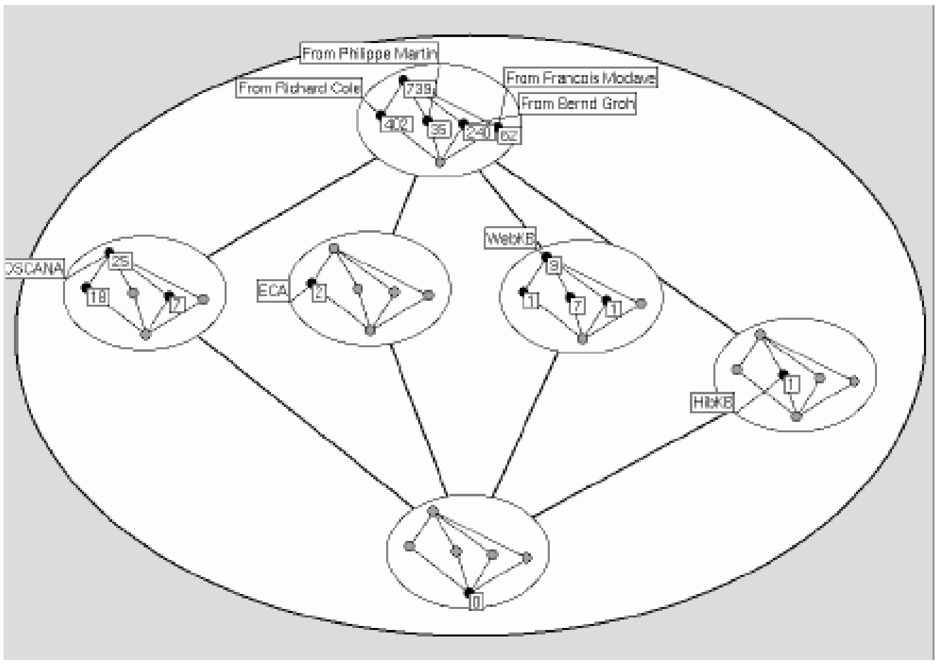


Figure 6.6: The nested diagrams showing the project subject for the members of the KVO Institute (source: [Cole et al., 2003a]).

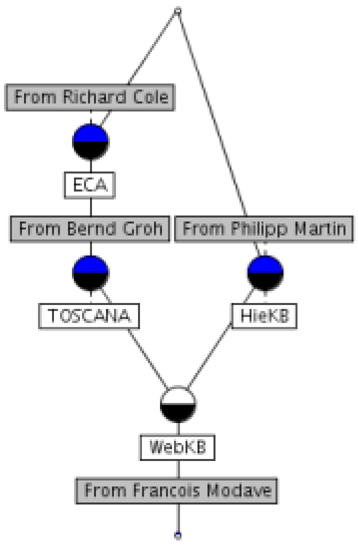


Figure 6.7: The lattice showing the mail project topic relation of the KVO work group.

In Section 7.2, we discuss future steps for the implementation of the approach on top of the query-based multicontext.

6.2 Visualisation Evaluation

This section describes the evaluation we performed in order to compare the three visualisation paradigms presented in Chapter 4, Section 4.2: the tree view, the graph view and the lattice view displaying the line diagram. This evaluation was presented in [Tane et al., 2006], and we describe it here in detail and summarise the lessons learned from it.

Our comparison started with a preliminary evaluation to see how the different testers reacted to the interface. While performing this preliminary evaluation, we noticed quickly that users had difficulties answering the questions asked using the lattice paradigm. We conjectured that these difficulties were due to the lack of familiarity with this way of displaying information. To remedy to this problem, we decided to give testers some training in the reading of the line diagram. This training consisted of three questions with a growing complexity. In addition, another training question allowed users to familiarise themselves with the features of the graph view. We considered that testers were already acquainted with the tree view since it is very frequently used in the design of user interfaces (for example file browsers, etc.).

The evaluation consisted in answering three questions based on three different views of an anonymised AIFB dataset.[7] While the questions were identical

[7]The data has been anonymised in order to ensure that testers could not use their background knowledge to answer the questions.

for each user, the view paradigm presented to them in order to answer each question differed. Moreover, every tester was asked to answer one question in each of the different paradigms. To ensure that the data was distributed evenly, questions and paradigms were combined equitably. Since most users were not used to the view displaying a line diagram, a very short presentation on Formal Concept Analysis was given to them, thus making sure that each tester understood the basic principle behind the visualisation.

Before describing the results of the evaluation, we introduce in the next section the three view paradigms used to display a specific view.

6.2.1 Description of the View Paradigms

In Chapter 4, Section 4.2, we discussed the issues related to the visualisation of information for the three chosen paradigms. While all the approaches can be used to display other types of relations (for example hierarchical), the views chosen for the evaluation were based on context indices representing binary relations. The tree and graph view paradigms have been chosen as they represent two usual ways of displaying information to the user. These paradigms are possible alternatives to visualise the content of a formal context. As discussed in Chapter 4, Section 4.2, in some cases the tree and graph approaches have their advantages. However, we were mainly interested in evaluating how well user perform on some tasks using the lattice paradigm in comparison of the other paradigms, especially since most users are not familiar with lattices.

Graph View

The Graph view used for the evaluation is adapted from a number of components of the graph-based ontology browser of the KAON project.[8] This browser called OIModeller uses a spring-based graph layout algorithm which is a layout technique simulating a set of objects linked by springs (see [Eades, 1984]). An edge of the graph has an attraction value while each node have a repulsion level. The layout algorithm tries to find a configuration for the nodes of the graph such that an equibrium between repulsion and attraction can be found. This means that the graph updates itself every time the configuration of the nodes has been changed.

The graph panel has also some supplementary features to help the user in visualising and interacting with its content. These supplementary features were restricted to:

- hide nodes,
- pin down nodes,
- rotate,
- zoom, and
- fish eye view.

[8]See http://sf.net/projects/kaon.

In our evaluation, the graph view shows the relations between instances. For the purpose of the evaluation, the components were adapted so that attributes and objects of the realised context could be distinguished using their colour. Figure 6.8 shows the graph for one of the realised contexts used in the preliminary experiment. The graph view offered the possibility to hide nodes[9] or to pin them (i.e. preventing the layout algorithm to update the position of the node).

For the purpose of the evaluation, three different means of selecting elements for the result were implemented. First of all, double-clicking on a given node added the corresponding element to the result set. Another way was to use one the corresponding actions

- add a particular node to the result set, or
- adding all the currently selected elements in the graph

from a context menu. Finally, to create complex selection (i.e. with more than one element) the user could either use a combination of the mouse and the Control key or draw a rectangle over the elements to select.[10]

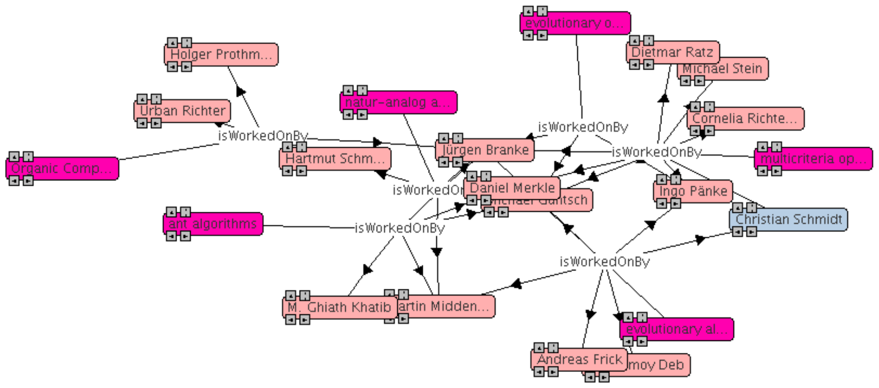


Figure 6.8: The Graph view

Tree View

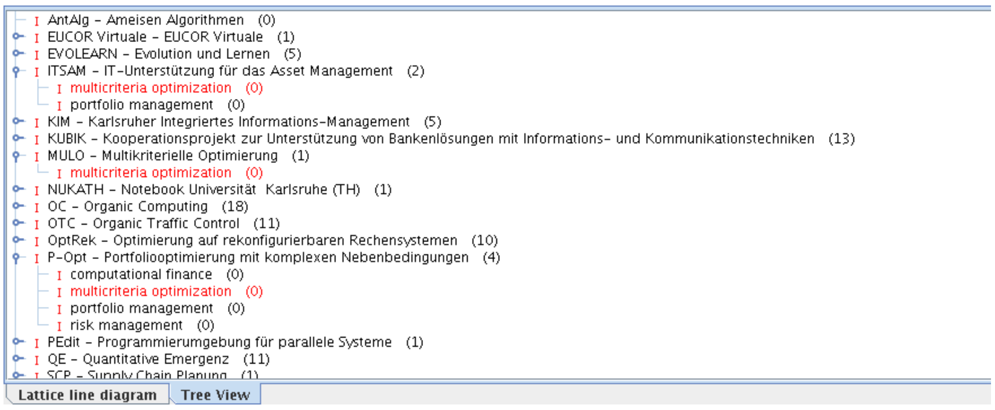
The tree view is a representation paradigm most users are familiar with. In the context of the evaluation process, we have only considered binary relations between attributes and objects (hence the tree has only a height of two[11]). Figure 6.9 shows the appearance of the tree view when used to display a relation context index.

The tree view respected the traditional conventions used. Siblings in the tree were sorted alphabetically and the usual multiple elements selection techniques in tree view enabled the user to select elements for the answer set.

[9]Once a node was hidden, it could not be retrieved anymore.

[10]This is done in the traditional manner: selecting a point and dragging the mouse to some other point. This creates a rectangle with the diagonal being the segment between the two points and the sides are parallel to the edges of the screen.

[11]The root node does not carry any information and is therefore hidden.



AntAlg – Ameisen Algorithmen (0)
EUCOR Virtuale – EUCOR Virtuale (1)
EVOLEARN – Evolution und Lernen (5)
ITSAM – IT–Unterstützung für das Asset Management (2)
multicriteria optimization (0)
portfolio management (0)
KIM – Karlsruher Integriertes Informations–Management (5)
KUBIK – Kooperationsprojekt zur Unterstützung von Bankenlösungen mit Informations– und Kommunikationstechniken (13)
MULO – Multikriterielle Optimierung (1)
multicriteria optimization (0)
NUKATH – Notebook Universität Karlsruhe (TH) (1)
OC – Organic Computing (18)
OTC – Organic Traffic Control (11)
OptRek – Optimierung auf rekonfigurierbaren Rechensystemen (10)
P–Opt – Portfoliooptimierung mit komplexen Nebenbedingungen (4)
computational finance (0)
multicriteria optimization (0)
portfolio management (0)
risk management (0)
PEdit – Programmierumgebung für parallele Systeme (1)
QE – Quantitative Emergenz (11)
SCP – Supply Chain Planung (1)

Figure 6.9: The Tree View

Lattice View

The lattice view shows the line diagram of a certain context index. Different types of interaction are possible. A user can select the elements of the extension of a node using a double click with the middle button. He can also move nodes and labels or display the objects and/or attributes in a specific panel (in [Tane, 2005], we presented the diverse selection and interaction modes for the lattice). Figure 6.10 shows the appearance of the lattice view.

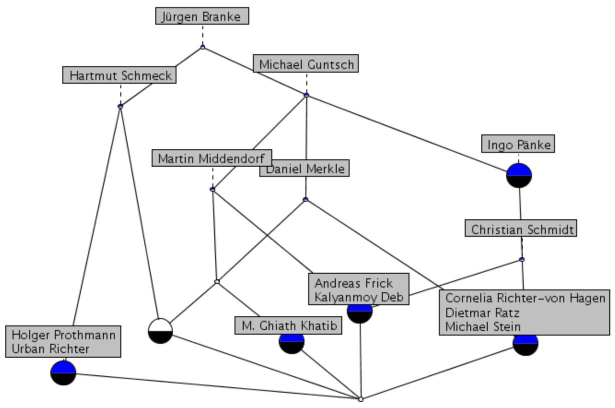


Figure 6.10: The Lattice View

6.2.2 Question Types

Three types of questions were asked to the user as part of the evaluation. Each of these types is explained in further detail in this section. To give a better idea of the evaluation, we reproduce here the original wording of these questions together with the corresponding context indices in exactly the same order as they were presented to each tester. Note that the questions were on the screen and phrases such as the panel above referred to the view visualised using one of the three visualisation paradigms.

Conjunctive Queries

A conjunctive query is a query which has the goal of retrieving all the elements which all satisfy a certain set of criteria.[12] In the case of views, the goal is usually to visualise the elements sharing some common properties. For example, given a group of authors and their publications, the problem is to select a certain subgroup of authors who published together. Another of typical query could be to retrieve the projects share the research topics Formal Concept Analysis and Text-Mining from a view displaying projects and their research topics. We made the hypothesis for the evaluation that the lattice paradigm is the most suitable to visualise this kind of questions.

In order to prove this we compared the time taken by the testers to answer conjunctive queries. This was made under the assumption that if user could answer the conjunctive queries using the lattice quickly they could certainly visualise the conjunctive results.

Finally the question used in the evaluation was:

Question 1 ***Content: the research groups' projects***
The panel above displays the relation between the projects and research groups of the Institute.
Task:
Please select the projects carried out by the two research groups "Efficient Algorithms" and "Knowledge Management" at the same time.

Context Index: The corresponding context index of this view[13] is:

$$\begin{pmatrix} \mathsf{role}(<\mathsf{isCarriedOutBy}>, \{\mathsf{Research\ Group}\}) \\ inst(\{\mathsf{Research\ Group}\}) \\ <\mathsf{isCarriedOutBy}> \end{pmatrix}$$

Finding Outliers

In many cases, it is relevant to visualise elements which do not share properties with others. These elements can be seen as outliers. For example, looking at the research topics of the publications of a conference, it may be relevant to the user to note which topics seem to have not been treated by the authors of this conference.

For the evaluation we used the following question:

Question 2 ***Content: Distribution of the publications of the text mining researchers***
This panels shows the persons working on the text mining field at the institute as well as their publications.
Task:
Please select the publications of the author who did not share any publication with any others of this group of authors.

[12] Since a conjunction of attributes corresponds to positive propositional query (i.e. without negated literals), we can consider that the conjunctive queries involved are positive propositional queries

[13] This context index can easily be constructed by dropping the concept Research Group in the relation parameter.

Context Index: The context used to generate the view[14] of this question is:

$$\begin{pmatrix} \mathsf{role}(<\mathsf{author}>, \mathsf{role}(<\mathsf{isWorkedOn}>^{-1}, \{\mathsf{text\text{-}mining}\})) \\ \mathsf{role}(<\mathsf{isWorkedOn}>^{-1}, \{\mathsf{text\ mining}\}) \\ <\mathsf{author}> \end{pmatrix}$$

The question contained a hint to ease the burden of the task in the tree view. Indeed, the question mentions that there is only one author who did not share any publication with any others. This would have forced the users to look at all the publications in the view. As we show in Section 6.2.4, even with this simplification, the tree view performed much worse than the other approaches.

Disjunctive Queries

Disjunctive queries are queries which have the goal of retrieving all the elements which satisfy at least one of a given set of conjunctive queries. The need to retrieve objects responding to disjunctive sets of criteria is frequent. For example, the authors of a given publication may have published other papers together but not all in common. It is interesting to select publications shared by at least two of these authors.

During the evaluation the following question has been asked to the test participants.

Question 3 ***Content: SEKT project publications and members of the Knowledge Management research group***

The panel above displays the distribution of the publications of the SEKT project among the members of the "Knowledge Management" group.

Task:

Select all the publications where at least two of the following persons are authors:

- *Arthur Judson Brown*
- *Roger Wilson*
- *Arthur Lehning*

Context Index: The context used to generate the view[15] of this question is:

$$\begin{pmatrix} \mathsf{role}(<\mathsf{projectInfo}>, \{\mathsf{SEKT}\}) \\ \mathsf{role}(<\mathsf{memberOf}>, \{\mathsf{Knowledge\ Management\ Group}\}) \\ <\mathsf{isAuthorOf}> \end{pmatrix}$$

6.2.3 Context of the Evaluation

We first describe the main hypothesis for the evaluation as well as the parameters used in the evaluation.

[14]This context index is a simple example of the use of a Corelation constructor.

[15]This context index is more complex than the two others since each argument is chosen independently of the others.

Hypothesis

The goal of our experiment is to compare the performance of the participants of our test for different types of questions using the lattice compared to one of the other paradigms. Due to the small amount of data, we used the T-test distribution (see [Rutsch, 1987]) to determine the significance of the difference in testers' performance. For each of these tests, we formulate the null-hypothesis as:

The time needed by users to answer the given question does not differ in a significant manner between the two paradigms (i.e. lattice and tree or lattice and graphs).

Test Procedure

We now present the circumstances and procedure through which each tester had to go. The evaluation occur ed over the course of a week. 18 academic test users participated in the evaluation. Each of these test participants had to go through the following process steps.

1. short presentation of the context of the evaluation
   (a) short introduction of the three paradigms and their functionalities
   (b) a crash course in the reading of concept lattices
2. 4 preliminary questions
   (a) three lattice-based questions
   (b) one graph-based question
3. three questions with different[16] paradigms in the following order:
   (a) a conjunctive query on a view with a small number of attributes
   (b) an outlier query (small number of attributes but large number of objects)
   (c) a disjunctive query on a large view (many attributes and much larger amount of objects)

For each question, the view was created while the test participant read the question. As soon as he was ready to answer, the user had to press a button to start. The test person could then interact with the view and try to answer the question. Once he had selected all the elements he considered as the answer, he could press the answer button.

[16] All tester answered three questions in three different paradigms. Three testers answered first a question with a lattice first, then a graph, and finally a tree, while others had a tree first, then a lattice and a graph. The six permutations occurred each three times.

6.2.4 Results

After the evaluation, we gathered the results of each test participant as well as the time each of the participants took to achieve the tasks. Figure 6.11 shows the average time in seconds needed to answer the questions in the three paradigms. Question 1 corresponds to the conjunctive query, Question 2 to the outlier query and Question 3 to the disjunctive query.

Our goal was to show that the differences in performance times of the different users were not due to chance, in other words we wanted to invalidate the Null-hypothesis. To do this, it is customary to compute the t-value of a given pair of paradigms. The t-value depends on the means and variance of the values of the list of results of the paradigms.

The following equation states the way our experiment was computed.

$$t = \frac{x_o - x_l}{\sqrt{var(\overline{x}_o) + var(\overline{x}_l)}/\sqrt{5}} \tag{6.1}$$

where $\overline{x}_o$ is the mean of the other paradigm and $\overline{x}_l$ is the mean of the time taken by users to perform the required tasks using the lattice paradigm. The value 5 in the formula corresponds to the *degrees of freedom* of our experiment minus 1. For our experiment, the degree of freedom was 6 since each question with a given paradigm was asked six times to six different test persons.

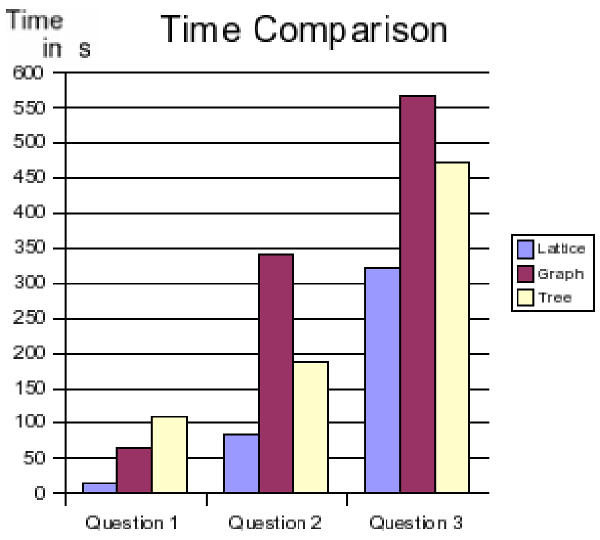


Figure 6.11: Average Time in seconds needed to answer the three questions in the three para–digms.

We also computed the precision and recall for each of the paradigms and each query. However, with such a small dataset of tests, we think that it is better to classify the few errors which occurred. These errors are of three types:

- task misunderstanding,
- search errors, and
- manipulation mistakes.

The only misunderstanding of the task occurred in the lattice view paradigm while trying to answer the disjunctive query. The user misunderstood the task and gave only a partial answer. However, when asked to reread the question after the question had been answered, the test person could immediately give the correct answer.

Search errors occurred a few times in the tree view. These errors can be explained by the high number of comparison necessary when answering the questions with this paradigm. For all the three questions, the extension for a given attribute (i.e. the column of the context) contained between 4 to 60 elements. Though the elements were ordered alphabetically, a large number of comparisons was necessary to answer the questions. All these comparisons were cumbersome for the user. In two cases in question 1, the test participant answered `no` to the question though the two research groups shared one common project.

The manipulation mistakes occurred mainly on the graph view where elements not belonging to the correct answer were selected. The mistakes occurred because the user needed to select a large amount of entities and the group selection mechanism from the graph view was used (see Section 6.2.1 for details on the selection). A selection mistake occurred also in the lattice view, where the user selected the attribute contingent of the node, while the correct answer was the extent. However, the test person noticed his mistake and corrected it before validating the answer. This shows that the distinction between the diverse parts of a formal concept are not immediately understood by the users, thus resulting in manipulation errors.

Interpretation of the Results

Diverse conclusions could be drawn from this evaluation. First, as shown in Figure 6.11, the users performed in average better with the lattice paradigm than with the other paradigms. The computed t-value results are shown in Table 6.2. They show that the Null-hypothesis is rejected in all cases with a chance of errors of maximal 10 %.

Moreover, this performance also proves that the training restricted to four preliminary questions is enough for users to perform the tasks while without the training users tended to be slower.

Table 6.2: Confidence and t-values level of lattice-tree and lattice-graph time comparisons.

Question	Tree	Graph
Question 1	3.94 ($\alpha = 0.05$)	3.69 ($\alpha = 0.05$)
Question 2	1.99 ($\alpha = 0.10$)	2.29 ($\alpha = 0.05$)
Question 3	1.86 ($\alpha = 0.10$)	2.27 ($\alpha = 0.05$)

Another important result is that there were more errors using the two other paradigms. This is mainly due to the cumbersome nature of the chosen tasks for the tree and graph paradigms. In all views, the amount of interaction needed to answer the questions with the lattice view was much smaller than with the other paradigms. The evaluation confirms the intuition that users perform much better with the lattice paradigm if the number of elements to select is large or if the number of elements which need to examined in order to select an element is large.

Note that for the diverse tasks to be performed, the performance of the selection using the lattice could be greatly improved if an incremental highlighting mechanism had been available. Therefore, there seems to be still room for improvement. But, other graph layout implementations may also increase the performance of the user. Finally, it should be made clear that the question asked are not representative of all the possible tasks occurring when visualising a view. However, for these kinds of tasks, the lattice approach seems more advantageous.

6.3 Summary

To sum up the evaluations presented in this chapter, we first discussed the similarities and differences between our approach for knowledge browsing and others. In particular, we positioned our approach regarding the state-of-the-art of Formal Concept Analysis knowledge explorations tools.

Finally, we also presented the results of an evaluation we performed in order to see how well users coped with the concept lattice visualisation paradigm in comparison with the tree and graph paradigms.

Chapter 7

Discussion

To conclude this thesis, we recall our contributions and outline open issues as well as other potential applications of our approach.

7.1 Contributions

The work we presented in this thesis can be split up in three main contributions, which we detail further in the course of this section. We first introduced a new framework for contextualised semantic views. Then we used this framework as underlying model for a novel approach to the exploration of knowledge bases. Finally, we discussed the relevance of our approach in comparison with other exploration methods.

A Contextualised Semantic View Framework

We introduced a theory for the definition and construction of contextualised views to be generated from a query infrastructure. Central to this theory is a new structure we called the *Query-based Multicontext*, which can be seen as the formal space consisting of all the contextualised views which can be generated from a data source. The purpose of this structure provides a generic model for the definition, manipulation and generation of contextualised views using appropriate intensional descriptions: *context indices*. To enable simple operations on these descriptions, we first introduced a selection generic query and context index operators which can be easily implemented for any query infrastructure and query-based multicontext.

We also proposed an extension of this model allowing to generate views from knowledge bases as used in the Semantic Web. To achieve this goal we investigated the necessary query primitives required for the implementation of a query-based multicontext over a knowledge base. We showed how these primitives mostly correspond to primitives of the OWL-DL language. As a proof of concept, we describe the infrastructure of such an ontology infrastructure on top of two software components capable of providing knowledge bases: KAON and KAON2.

Finally, we introduced a high-level template mechanism which is specially adapted to simplify the view definition process presented in Chapter 4.

A Semantic View Exploration Framework

Using our framework of contextualised semantic views, we have proposed a novel approach for the exploration of knowledge bases. We have discussed the advantages of our approach compared to an entity-centered approach using an abstract model of information spaces. We investigated a number of important issues which had to be addressed for the design of our approach. As mentioned in Chapter 4, these issues correspond to specific phases of interaction with the framework, namely the view visualisation, the elements selection and finally the view definition.

Regarding the visualisation of views, we investigated the properties of three visualisation paradigms based on different structural assumptions. We also studied the means of selecting elements using these three paradigms and described how the line diagrams of concept lattices are particularly suitable for the selection of sets of entities for our approach.

Finally, we also investigated the crucial view definition process and proposed different approaches to support it. In particular, we described a mechanism to ease the definition of views based on the constructor mechanism we defined.

An Evaluation of Diverse Visualisation Paradigms

We evaluated our approach in two manners. First, we compared it with other knowledge browsing approaches. Second, we performed a user evaluation comparing how well user perform on the different kinds of views on certain kind of questions.

7.2 Future Work

Beyond the contributions of our thesis, we outline in this section some open questions or relevant theoretical aspects which could not be addressed in this work. We also discuss other application domains which could benefit from the query-based multicontext approach.

7.2.1 Open Questions

The approach presented in this thesis is only a step in the development of the query-based multicontext theory and its potential applications. In the following paragraphs, we discuss different open issues and directions for the query-based multicontext theory.

Investigations on Interaction

During the elaboration of our browsing framework, we noted that the interaction with the lattice could be greatly improved. A number of aspects need to be further investigated. In particular, the visualisation of large views remains an unsolved problem. Our approach helps selecting the relevant information to display with a lattice, but some views can still not be displayed in a completely suitable manner. None of the visualisation paradigms proposed until now allows users to cope with the size and/or the complexity of all the structures displayed. Though the user can define more suitable views using the adaptation techniques described in Chapter 4, Section 4.3.4, the process remains complex. The query-based multicontext approach provides a ground framework, but appropriate interaction workflows should be designed carefully. The generic refinement approaches we proposed did not take into consideration the nature of the view nor the type interaction.

Depending on the type of view, i.e. the constructor or operator used in its definition and the visualisation paradigm chosen, a number of operations depending on the situation could be proposed to the user. In particular, a stronger use of the order relation underlying the lattice is likely to offer many suitable means of refining views, but these means may not be suitable for all users.

Since the content index specifies the type of elements of the objects or attribute sets, this information may be used to offer type dependent means of interaction with the view. In particular, it seems promising in the case of the lattice paradigm,[1] for which different highlighting and diagram decoration strategies should be developed to reflect further intrinsic semantic properties of the data.

We also mentioned in Chapter 4, Section 4.2.2 that other selection means could also provide new ways of selecting elements in the lattice. We have not investigated the possibilities of selecting sets of formal concepts instead of knowledge base entities because of the difficulty of creating an intuitive interface using these types of selections.

Nested Diagram Implementation

We also extended the query-based multicontext theory to many-valued multicontexts. However, we have not yet developed the corresponding extension for our knowledge browsing approach. A number of design decisions have to be taken in order to extend this theory. One of these decisions is related to the choice of the appropriate scale to display to the user. Another question concerns the appropriate data model for a nested diagram implementation.

Investigations on Informative Feedback

An important aspect to consider when developing an application is to provide an informative feedback (see [Shneiderman and Plaisant, 2005]). In particular, the graphical interface should provide useful hints helping the users in exploring the knowledge base. In this thesis, we did not deal with this aspect. However, we envision diverse supplementary additions to the framework.

We believe that using well known techniques from the field of *Inductive Logic Programming* (hence ILP), would allow us to provide informative feedback to a user. We recommend [Dzeroski and Lavrac, 2001] for a an overview of the theory behind ILP as well as an idea of the state-of-the-art in this field.

To illustrate how methods from the field of ILP could be used by the query-based multicontext browser, let us first recall that the goal of ILP is to find a suitable logical description of a set of examples. Traditionally, this set of example is separated into two disjoint sets E^+ and E^-, the sets of positive and negative examples. In the context of the query-based multicontext browser,

[1]Similar techniques may also be used for other paradigms, but the hierarchical and clustering nature of the lattice seems more appropriate to act on groups of elements.

suppose a user selects a particular node in a lattice. The elements of the extension of the node could be used as positive examples, whereas the elements in the extensions of concepts of the rest of the lattice could be taken as negative examples. An ILP program could then provide a logical description of the positive examples. This logical description could be presented to the user as supplementary feedback about the selected elements.

Extending the Semantic Query Infrastructure

The semantic query infrastructure introduced in Chapter 3, Section 3.4.2 was limited to 4 types of objects: concepts, instances, relations and relation instances. We mentioned in Chapter 5, Section 5.4.2 a possible extension of this query infrastructure concerning contextualised information. Diverse approaches may help in that aspect. First of all, OWL and RDF(S) documents on the web are identified using URLs. These unique identifiers could serve to determine the provenance of the information. The URLs could be used as contextualised information by stating in which document a fact is stated. Moreover, the import mechanisms creates an inclusion structures on the documents. The facts and the import structure could be easily displayed using a subsumption/instance constructor (see Chapter 3, Section 3.5). As a matter of fact, the Courseware Watchdog (see [Tane et al., 2003, Tane et al., 2004]) implements a similar functionality.

Another promising approach could rely on techniques similar to *Named Graphs* (see [Carroll et al., 2005]), it would allow an even greater expressivity and flexibility for the query-based multicontext infrastructures. The main idea behind Named Graphs is extremely similar in diverse manners to the ideas behind the views of a query-based multicontext, i.e. Named Graphs offer the possibility to name a subgraph of a given RDF graph. Using these names, it is possible to define operations on these graphs. Moreover, a combination of Named Graphs and the query-based multicontext framework is likely to provide supplementary expressive power. We describe briefly a similar idea in the next paragraph.

Query-Based Graphs

The discussion of Named-Graphs in the previous paragraph leads to another interesting parallel with the present approach: replacing formal contexts by graphs. Like formal contexts, graphs have been widely studied in the literature. We imagine that an approach to the query-based multicontext approach, using graphs instead of lattices to display the information would also provide interesting results.

Particularly for applications in the Web, an approach based on query-based graphs seems promising. Imagine that the web pages are considered as the nodes of the graph and that the links between them are considered as the edges of the graphs and provided with a certain kind of annotation (the terms, concepts or ontological annotation used in a certain window around the links). Using this model, we conjecture that it would be possible to create a framework capable of creating contextualised views of parts of the Web.

As discussed in Chapter 4, Section 4.2.1, the graph paradigm can be used to display query-based multicontext views. Context indices are suitable for creating interesting graphs, but since edges can be labeled in a graph, the context index model as presented in Chapter 3, 3.2.1 may not be appropriate without any modification.

To adapt to graph-based views, the underlying model should also contain names for edges. Binary formal contexts can be seen as a special case of a more generic model where the pairs resulting from the relation query are named. This model corresponds to the idea of many-valued multicontexts (see Chapter 2, Section 2.3.3). But the many-valued multicontexts are still not sufficient to express arbitrary graphs since they require that the labels are functionally dependent of the pair of entities selected. In addition to the adaptation of the current indices, displaying information as a graph seems also to require other kinds of constructor mechanisms as well as strategies for visualisation and definition.

We envision further investigations in this direction, especially since we conjecture a few practical benefits for the Web context. Moreover, the literature on graph theory is vast and many existing approaches seem relevant for these investigations. In particular, we believe that many promising results could be drawn from the study of graph grammars (see [Nagl, 1979]) and graph theory in general.

Improve Formal Concept Analysis Algorithms

In our opinion, an interesting consequence of describing the content of a formal context using queries is that it also provides structural information which could be used to develop incremental algorithms for the context and lattice generation phases as well as the layout phase. Such a generation algorithm would generate a context or a lattice using known logical properties of the queries used.

7.2.2 Other Applications

As we developed the query-based multicontext theory, we also aimed at addressing two other kinds of application fields. Both of these application fields can be seen as a part of a greater field, i.e. knowledge discovery, but we discuss them separately because they require different approaches and techniques to address their issues.

Query-Based Multicontext for Data Mining

In addition to the relevant research topics of optimising the query-based multicontext for navigating databases, our framework of semantic contextualised views could be used in Data Mining applications. The goal of Data Mining is to extract new relevant patterns out of large quantities of information usually stored in databases. While knowledge bases usually offer a rich conceptual structure, Data Mining applications usually consider large amounts of data in a few tables, that is, the conceptual schemas which are used are somewhat limited. It is therefore an open issue to see how the query-based multicontext theory could be used to address some of the issues occurring in Data Mining applications.

In recent years, the fields of Data Mining and Knowledge Discovery in Databases (henceforth DM and KDD) have seen a great number of applications being developed (see [Maimon and Rokrach, 2005, Fayyad et al., 1996]).

Techniques from the Formal Concept Analysis literature have proved useful to solve some of the relevant problems (see for example [Stumme et al., 2002, Valtchev et al., 2004, Stumme et al., 2004]). Since the query-based multicontext theory can be seen as an extension of traditional Formal Concept Analysis, it seems promising to investigate how it could be used in these applications.

As one step in this direction, we propose the use of the query-based multicontext approach in conjunction with iceberg lattices algorithms. These algorithms such as TITANIC presented in [Stumme et al., 2002, Stumme, 2004], compute lattices representing the most frequent formal concepts of a given formal context. The combination of the two approaches would provide a new framework for the visualisation of data by using an appropriate query infrastructure to focus on some part of the data. Instead of displaying the traditional concept lattice, a user of such a framework could obtain an overview by specifying an appropriate frequency threshold. Moreover, following an idea of Hannu Toivonen (see [Toivonen, 1996]), it might be possible to replace the frequency counting by some other criteria in the algorithms.

Another potential area of research for the query-based multicontext theory is the development of application dependent query operators and constructors. The use of appropriate feature extraction mechanisms as query operators is in particular extremely relevant from that perspective. We believe that query-based multicontexts related techniques could be used to mine association rules on top of databases. [Tsur et al., 1998] described a technique called *query flocks* which also seems to be a promising technique to be combined with the query-based multicontext approach.

Query-Based Multicontext for Text Mining

In Chapter 6, Section 6.1.3 we presented the tool Docco. In our discussion, we also showed that the Docco approach could be integrated in the query-based multicontext approach. We believe that this integration would open new possibilities for text-mining. However, such an approach would require the definition of more expressive query infrastructures. In particular, we believe that a combination of semantic markup in documents with traditional Information Extraction would be required (see [Pazienza, 1999] for an overview of relevant information extraction techniques).

Different approaches have been proposed for the automatic extraction of semantic markup or the population of knowledge bases from text (see for example [Cimiano, 2006, Ciravegna et al., 2004]). We believe that the use of a query-based multicontext oriented approach could help in ensuring the correctness of the extracted data as well as provide useful insights in the actual content of document collections.

7.3 Final Words

The work presented all along this thesis opens new perspectives for the combination of Formal Concept Analysis-based applications with ontologies or databases. Future extensions to more document and data mining oriented approaches also seem promising and we hope that this work serves as a starting point for further research.

Bibliography

[Abello et al., 2002] Abello, J., Pardalos, P. M., and Resende, M. G., editors (2002). *Handbook of Massive Data Sets*. Kluwer Academic Publishers.

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison Wesley, Reading, Mass. U.S.A.

[Aeschlimann and Schmid, 1992] Aeschlimann, A. and Schmid, J. (1992). Drawing orders using less ink. *ORDER*, 9:5–13.

[Ahlberg et al., 1992] Ahlberg, C., Williamson, C., and Shneiderman, B. (1992). Dynamic queries for information exploration: An implementation and evaluation. In Bauersfeld, P., Bennett, J., and Lynch, G., editors, *Proceedings of the ACM CHI 92 Human Factors in Computing Systems Conference*, pages 619–626, New York, U.S.A. ACM Press.

[Antoniou and van Harmelen, 2004] Antoniou, G. and van Harmelen, F. (2004). *A Semantic Web Primer*. MIT Press, London, U.K.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, Cambridge, U.K.

[Baeza-Yates and Ribeiro Neto, 1999] Baeza-Yates, R. and Ribeiro Neto, B., editors (1999). *Modern Information Retrieval*. ACM Press, New York, U.S.A.

[Battista et al., 1999] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G. (1999). *Graph Drawing - Algorithms for the Visualization of Graphs*. Prentice Hall, Upper-Saddle River, NJ, U.S.A.

[Battle et al., 2005] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., and Tabet, S. (2005). Swsf application scenarios. Found at http://www.daml.org/services/swsf/1.0/applications/#service-discovery-example. Version 1.0.

[Becker and Cole, 2003] Becker, P. and Cole, R. J. (2003). Querying and analysing document collections with formal concept analysis. In *Proceedings of the 8th Australasian Computing Symposium*. Camberra, Australia.

[Berners Lee et al., 2001] Berners Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*.

[Bordat, 1986] Bordat, J.-P. (1986). Calcul pratique du treillis de galois d' une correspondance. *Math. Sci. Hum*, 96:31–37.

[Borg, 1992] Borg, I. (1992). *Grundlagen und Ergebnisse der Facettentheorie*. Number 13 in Methoden der Psychologie. Verlag Hans Huber, Bern.

[Boyle et al., 1993] Boyle, J., Eick, S. G., Hemmje, M., Keim, D. A., Lee, J. P., and Summer, E. (1993). Database issues for data visualization: Interaction, user interfaces, and presentation. In Lee, J. P. and Grinstein, G. G., editors, *Database Issues for Data Visualization, IEEE Visualization '93 Workshop*, number 871 in Lecture Notes in Computer Science, pages 25–34. Springer Verlag, Berlin, Germany.

[Brickley and Guha (eds), 2004] Brickley, D. and Guha (eds), R. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. Available from http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[Burbeck, 1992] Burbeck, S. (1992). Application programming in smalltalk-80: How to use model-view-controller (mvc). University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive. Available at: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

[Calvanese et al., 1997] Calvanese, D., DeGiacomo, G., and Lenzerini, M. (1997). Conjunctive query containment in description logics with n-ary relations. In *International Workshop on Description Logics, Paris, 1997*, pages 5–9, Published electronically at http://www.lri.fr/~mcr/ps/dl97.html.

[Carpineto and Romano, 2005] Carpineto, C. and Romano, G. (2005). Using concept lattices for retrieval and mining. In Ganter, B., Stumme, G., and Wille, R., editors, *Formal Concept Analysis: Foundations and Applications*, number 3626 in Lecture Notes in Artificial Intelligence - State-of-the-Art Survey, pages 161–179. Springer.

[Carroll et al., 2005] Carroll, J. J., Bizer, C., Hayes, P., and Stickler, P. (2005). Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA. ACM Press.

[Chen, 2006] Chen, C. (2006). *Information Visualization*. Springer Verlag, London.

[Chen et al., 1993] Chen, W., Kifer, M., and Warren, D. S. (1993). A foundation for higher-order logic programming. *Journal of Logic Programming*, 3(15):187–230.

[Cimiano, 2004] Cimiano, P. (2004). ORAKEL: A natural language interface to an F-logic knowledge base. In *Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems*, number 3136 in Lecture Notes in Computer Science, pages 401–406, Berlin, Germany. Springer.

[Cimiano, 2006] Cimiano, P. (2006). *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. PhD thesis, Fakultät Wirtschaftwissenscahften der Universität Fredericiana zu Karlsruhe.

[Cimiano et al., 2003] Cimiano, P., Staab, S., and Tane, J. (2003). Deriving Concept Hierarchies from Text by Smooth Formal Concept Analysis. In *Proceedings of the GI Workshop "Lehren - Lernen - Wissen - Adaptivität" (LLWA), Fachgruppe Maschinelles Lernen, Wissenentdeckung, Data Mining, Karlsruhe, Germany.*

[Ciravegna et al., 2004] Ciravegna, F., Chapman, S., Dingli, A., and Wilks, Y. (2004). Learning to harvest information for the semantic web. In *Proceedings of the 1st European Semantic Web Symposium, Heraklion, Greece, May 10-12*, volume 3053 of *Lecture Notes in Computer Science*, pages 312–326. Springer.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 16(6):377–387.

[Cole, 2001a] Cole, R. (2001a). Automatic layout of concept lattices using layer diagrams and additive diagrams. In Oudshoorn, M., editor, *24th Australiasian Computer Science Conference, Australian Computer Science Communications 1, IEEE Computer Society*, pages 47–53.

[Cole and Stumme, 2000] Cole, R. and Stumme, G. (2000). CEM - a Conceptual Email Manager. In Ganter, B. and Mineau, G. W., editors, *Proceedings of the International Conference on Conceptual Structure 2000*, volume 1867 of *Lecture Notes in Artificial Intelligence*, pages 438–452, Berlin, Germany. Springer.

[Cole, 2001b] Cole, R. J. (2001b). *Document Retrieval using Formal Concept Analysis.* PhD thesis, School of Information Technology, Griffith University.

[Cole et al., 2003a] Cole, R. J., Eklund, P., and Stumme, G. (2003a). Document retrieval for email search and discovery using formal concept analysis. *Journal of Applied Artificial Intelligence (AAI)*, (17).

[Cole et al., 2003b] Cole, R. J., Eklund, P. W., and Stumme, G. (2003b). Document retrieval for email search and discovery using formal concept analysis. *Journal of Applied Artificial Intelligence (AAI)*, 17(3):257–280.

[Correia, 2002] Correia, J. H. (2002). Relational scaling and databases. In Priss, U., Corbett, D., and Angelova, G., editors, *ICCS*, volume 2393 of *Lecture Notes in Computer Science*, pages 62–76. Springer.

[Davey and Priestley, 1994] Davey, B. A. and Priestley, H. A. (1994). *Introduction to lattices and order.* Cambridge Univ. Press, repr. edition.

[Davis et al., 1993] Davis, R., Shrobe, H., and Szolovits, P. (1993). What is a knowledge representation? *AI Magazine*, 14(1):17–33.

[de Bruijn and Fensel, 2005] de Bruijn, J. and Fensel, D. (2005). Ontology definitions. In Bates, M. J., Maack, M. N., and Drake, M., editors, *Encyclopedia of Library and Information Science.* Marcel Dekker, inc, Boca Raton, FL, U.S.A.

[de Bruijn et al., 2005] de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2005). OWL DL vs. OWL Flight: Conceptual modeling and reasoning on the semantic web. In *Proceedings of the 14th International World Wide Web Conference (WWW2005)*, pages 623–632, Chiba, Japan. ACM.

[Dzeroski and Lavrac, 2001] Dzeroski, S. and Lavrac, N., editors (2001). *Relational Data Mining*. Springer, Heidelberg.

[E. Bozsak et al., 2002] E. Bozsak et al. (2002). KAON - Towards a large scale Semantic Web. In *Proceedings of the Third International Conference on E-Commerce and Web Technologies (EC-Web)*. Springer Lecture Notes in Computer Science.

[Eades, 1984] Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160.

[Eiter et al., 1997] Eiter, T., Gottlob, G., and Mannila, H. (1997). Disjunctive datalog. *ACM Transactions on Database Systems*, 22:364–418.

[Eklund, 2004] Eklund, P. W., editor (2004). *Concept Lattices, Second International Conference on Formal Concept Analysis, ICFCA 2004, Sydney, Australia, February 23-26, 2004, Proceedings*, volume 2961 of *Lecture Notes in Computer Science*. Springer.

[Eklund et al., 2004] Eklund, P. W., Ducrou, J., and Brawn, P. (2004). Concept Lattices for Information Visualization: Can Novices Read Line-Diagrams? In [Eklund, 2004], pages 57–73.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors (1996). *Advances in Knowledge Discovery and Data Mining*. AAAI Press and MIT Press, Menlo Park and Cambridge, MA, USA.

[Fensel et al., 2003] Fensel, D., Hendler, J. A., Lieberman, H., and Wahlster, W., editors (2003). *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential [outcome of a Dagstuhl seminar]*. MIT Press, London, U.K.

[Fluit et al., 2003a] Fluit, C., Sabou, M., and van Harmelen, F. (2003a). *Visualising the Semantic Web*, chapter Ontology-based Information Visualisation, pages 36–48. Springer Verlag, London.

[Fluit et al., 2002] Fluit, C., ter Horst, H., and van der Meer, J. (2002). Ontoknowledge: Visualization facility. Found at `http://www.ontoknowledge.org/downl/del13.pdf`.

[Fluit et al., 2003b] Fluit, C., ter Horst, H., van der Meer, J., Sabou, M., and Mika, P. (2003b). *Toward the Semantic Web: Ontology Driven Knowledge Management*, chapter Spectacle, pages 145–159. Number 9. Wiley, Chichester, England.

[Frasincar et al., 2003] Frasincar, F., Telea, A., and Houben, G.-J. (2003). *Visualizing the Semantic Web: XML-based Internet and Information Visualization*, chapter Adapting graph visualization techniques for the visualization of RDF data. Springer. Chapter 9.

[Furnas and Zacks, 1994] Furnas, G. and Zacks, J. (1994). Multitrees: Enriching and reusing hierarchical structure. In *Human Factors in Computing Systems: Proceedings of the CHI'94 Conference*, New York, U.S.A. ACM.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

[Ganter and Kwuida, 2005] Ganter, B. and Kwuida, L. (2005). Which concept lattices are pseudocomplemented? In Ganter, B. and Godin, R., editors, *ICFCA*, volume 3403 of *Lecture Notes in Computer Science*, pages 408–416. Springer.

[Ganter and Reuter, 1991] Ganter, B. and Reuter, K. (1991). Finding all closed sets: A general approach. *Order*, (8):283–290.

[Ganter and Wille, 1989] Ganter, B. and Wille, R. (1989). Conceptual Scaling. In Roberts, F. S., editor, *Applications of combinatorics and graph theory to the biological science*, pages 139–167, New York. IMA Volumes in Mathematics and Its Applications.

[Ganter and Wille, 1999] Ganter, B. and Wille, R. (1999). *Formal Concept Analysis – Mathematical Foundations*. Springer Verlag, Berlin – Heidelberg.

[Garcia-Molina et al., 1999] Garcia-Molina, H., Widom, J., and Ullman, J. D. (1999). *Database System Implementation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Gast, 1996] Gast, P. (1996). Begriffliche Strukturen mehrwertiger Multikontexte. Master's thesis, Technische Hochschule Darmstadt - Fachbereich Mathematik - Arbeitsgruppe Formale Begriffsanalyse, Darmstadt, Germany. Diplomarbeit.

[Godin and Mili, 1993] Godin, R. and Mili, H. (1993). Building and maintaining analysis-level class hierarchies using Galois Lattices. In *Proc. of OOPSLA'93, Washington (DC), U.S.A.*, pages 394–410, New York, U.S.A. Special issue of ACM SIGPLAN Notices.

[Grädel, 1999] Grädel, E. (1999). The restraining power of guards. *Journal of Symbolic Logic*, 64(4):1719–1742.

[Grimm and Motik, 2005] Grimm, S. and Motik, B. (2005). Closed world reasoning in the semantic web through epistemic operators. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-188/. published electronically.

[Hayes (editor), 2004] Hayes (editor), P. (2004). RDF Semantics. Available from `http://www.w3.org/TR/rdf-mt/`.

[Hearst, 1999] Hearst, M. (1999). *Modern Information Retrieval*, chapter User Interfaces and Visualization, pages 257–323. In [Baeza-Yates and Ribeiro Neto, 1999].

[Hereth-Correira and Kaiser, 2004] Hereth-Correira, J. and Kaiser, T. B. (2004). A Mathematical Model for TOSCANA-Systems: Conceptual Data Systems. In [Eklund, 2004], pages 39–46.

[Herman et al., 2000] Herman, I., Melançon, G., and Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43.

[Hitzler et al., 2005a] Hitzler, P., Haase, P., Krötzsch, M., Sure, Y., and Studer, R. (2005a). DLP isn't so bad after all. In Grau, B. C., Horrocks, I., Parsia, B., , and Patel-Schneider, P., editors, *Proceedings of the Workshop OWL - Experiences and Directions, Galway, Ireland*.

[Hitzler et al., 2005b] Hitzler, P., Studer, R., and Sure, Y. (2005b). Description logic programs: A practical choice for the modelling of ontologies. In *1st Workshop on Formal Ontologies Meet Industry, FOMI'05, Verona, Italy, June 2005*.

[Horrocks et al., 2006] Horrocks, I., Kutz, O., and Sattler, U. (2006). The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67, Menlo Park, California. AAAI Press.

[Horrocks et al., 2005] Horrocks, I., Parsia, B., Patel-Schneider, P., and Hendler, J. (2005). Semantic web architecture: Stack or two towers? In Fages, F. and Soliman, S., editors, *Principles and Practice of Semantic Web Reasoning (PPSWR 2005)*, number 3703 in Lecture Notes in Computer Science, pages 37–41, Heidelberg. Springer.

[Horrocks and Sattler, 2004] Horrocks, I. and Sattler, U. (2004). Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence*, 160(1–2):79–104.

[Hotho and Stumme, 2002] Hotho, A. and Stumme, G. (2002). Conceptual clustering of text clusters. In Kkai, G. and (Eds.), J. Z., editors, *Proc. Fachgruppentreffen Maschinelles Lernen (FGML 2002), 7.-9.10.2002, Hannover*, pages 37–45.

[Huchard et al., 2002] Huchard, M., Roume, C., and Valtchev, P. (2002). When concepts point at other concepts: the case of UML diagram reconstruction. In Liquiere, M., editor, *ECAI2002 Workshop: Advances in Formal Concept Analysis for Knowledge Discovery in Databases*, pages 32–43, Lyon, France.

[Klyne and Carroll (eds), 2004] Klyne, G. and Carroll (eds), J. J. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. Available from `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

[Knöpfel et al., 2006] Knöpfel, A., Gröne, B., and Tabeling, P. (2006). *Fundamental modeling concepts*. Wiley and Sons, Weinheim, Germany.

[Kollewe et al., 1994] Kollewe, W., Skorsky, M., Vogt, F., and Wille, R. (1994). *Begriffliche Wissensverarbeitung - Grundfragen und Aufgaben*, chapter TOSCANA - ein Werkzeug zur begrifflichen Analyse und Erkundung von Daten, pages 267–288. B.I.-Wissenschaftsverlag, Mannheim.

[Kosba, 2004] Kosba, A. (2004). User experiments with tree visualization systems. In *Proceedings of InfoVis 2004, IEEE Symposium on Information Visualization*, pages 9–16, Austin, TX, USA.

[Kuhlthau, 2005] Kuhlthau, C. C. (2005). *Theories of Information Behavior*, chapter Information Search Process. Information Today.

[Lamping et al., 1995] Lamping, J., Rao, R., and Pirolli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proc. ACM Conf. Human Factors in Computing Systems, CHI*, pages 401–408. ACM.

[Leone et al., 2002] Leone, N., Pfeifer, G., Faber, W., Calimeri, F., Dell'Armi, T., Eiter, T., Gottlob, G., Ianni, G., Ielpa, G., Koch, C., Perri, S., , and Polleres, A. (2002). The dlv system. In Flesca, S., Greco, S., Ianni, G., , and Leone, N., editors, *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA)*, number 2424 in Lecture Notes in Computer Science, pages 537–540, Heidelberg. Springer Verlag. System Description.

[MacGuiness and van Harmelen (eds), 2004] MacGuiness, D. L. and van Harmelen (eds), F. (2004). OWL Web Ontology Language: Overview. Available from `http://www.w3.org/TR/rdf-mt/`.

[Maimon and Rokrach, 2005] Maimon, O. Z. and Rokrach, L., editors (2005). *Data Mining and Knowledge Discovery Handbook*. Springer.

[Marchionini, 1995] Marchionini, G. (1995). *Information seeking in electronic environments*. Cambridge University Press, Cambridge, U.K.

[Motik, 2002] Motik, B. (2002). *KAON Developper Manual*. FZI-Forschungszentrum. Available at: `http://sf.net/projects/kaon`.

[Motik, 2006] Motik, B. (2006). *Reasoning in description logics using resolution and deductive databases*. PhD thesis, Fakultät Wirtschaftwissenscahften der Universität Fredericiana zu Karlsruhe.

[Munzner, 2000] Munzner, T. (2000). *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University. `http://graphics.stanford.edu/papers/munzner_thesis/index.html`.

[Nagl, 1979] Nagl, M. (1979). *Graph-Grammatiken*. Vieweg, Braunschweig, Germany.

[Nardi and Brachman, 2003] Nardi, D. and Brachman, R. J. (2003). *An Introduction to Description Logics*, chapter 1, pages 1–40. In [Baader et al., 2003].

[Pazienza, 1999] Pazienza, M. T., editor (1999). *Information Extraction: Towards Scalable, Adaptable Systems*. Springer, Berlin.

[Priss, 2005] Priss, U. (2005). Establishing connections between formal concept analysis and relational databases. In Dau, F., Mugnier, M., and Stumme, G., editors, *Common Semantics for Sharing Knowledge: Contributions to ICCS*, pages 132–145.

[Rector, 2003] Rector, A. (2003). *Medical informatics*, chapter 13, pages 406–426. In [Baader et al., 2003].

[Rector and Rogers, 2006] Rector, A. and Rogers, J. (2006). Ontological & practical issues in using a description logic to represent medical concepts: Experience from galen. School of Computer Science PrePrint, University of Manchester: CSPP-35:1-35.

[Reiter, 1980] Reiter, R. (1980). Equality and domain closure in first order data bases. *J. ACM*, (27):235–249.

[Rutsch, 1987] Rutsch, M. (1987). *Statistik 2: Daten modellieren.* Birkhäuser Verlag, Basel.

[Schmitz et al., 2002] Schmitz, C., Staab, S., Studer, R., Stumme, G., and Tane, J. (2002). Accessing distributed learning repositories through a courseware watchdog. In *Proceedings of E-Learn 2002: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, October 15-19 2002, Montreal, Canada*, pages 909–915.

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

[Shneiderman, 1996] Shneiderman, B. (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *IEEE Visual Languages*, number UMCP-CSD CS-TR-3665, pages 336–343, College Park, Maryland 20742, U.S.A.

[Shneiderman, 1997a] Shneiderman, B. (1997a). Designing information-abundant web sites: issues and recommendations. *Int. J. Hum.-Comput. Stud.*, 47(1):5–29.

[Shneiderman, 1997b] Shneiderman, B. (1997b). *Designing the User Interface: Strategies for Effective Human-computer Interaction.* Addison-Wesley, Reading, MA.

[Shneiderman and Plaisant, 2005] Shneiderman, B. and Plaisant, C. (2005). *Designing the User Interface.* Pearson/Addison-Wesley, 4. ed., internat. ed. edition.

[Smith and Ceusters, 2006] Smith, B. and Ceusters, W. (2006). *Computing, Philosophy and Cognitive Science*, chapter Ontology as the core Discipline of Biomedical Informatics: Legacies of the Past and Recommendations for the Furture Direction of Research. Cambridge Scholar Press. forthcoming.

[Sowa, 1984] Sowa, J. F. (1984). *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley Publishing Company, Reading, MA.

[Sowa, 2000] Sowa, J. F. (2000). *Knowldege Representation - Logical, Philosphical and Computational Foundations.* Brooks/Cole, Pacific Grove, CA, U.S.A.

[Studer and Staab, 2003] Studer, R. and Staab, S., editors (2003). *Handbook on Ontologies in Infornation Systems.* Springer Verlag, Berlin.

[Stumme, 2004] Stumme, G. (2004). Iceberg Query Lattices for Datalog. In *ICCS*, volume 3127 of *Lecture Notes in Computer Science*, pages 109–125. Springer.

[Stumme et al., 2004] Stumme, G., Cimiano, P., Hotho, A., and Tane, J. (2004). Conceptual Knowledge Processing with Formal Concept Analysis and Ontologies. In Ecklund, P., editor, *Proceedings of the 2nd Intl. Conf. on Formal Concept Analysis. LN 2961*, Berlin, Germany. Springer Verlag.

[Stumme and Mädche, 2001] Stumme, G. and Mädche, A. (2001). FCA-Merge: Bottom-up merging of ontologies. In *In Proceedings of the 7th Intl. Conf. on Artificial Intelligence (IJCAI '01), Seattle, WA*, pages 225–230.

[Stumme et al., 2002] Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., and Lakhal, L. (2002). Computing iceberg concept lattices with TITANIC. *Data Knowl. Eng.*, 42(2):189–222.

[Sure et al., 2005] Sure, Y., Bloehdorn, S., Haase, P., Hartmann, J., and Oberle, D. (2005). The SWRC Ontology - Semantic Web for Research Communities. In *Workshop on Building and Applying Ontologies for the Semantic Web (BAOSW 2005) at the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005).*

[Tane, 2004] Tane, J. (2004). Query Based Multicontext based browsing: a technical report. Technical report, Research Unit Knowledge and Data Understanding. http://www.kde.cs.uni-kassel.de/tane/techreport.

[Tane, 2005] Tane, J. (2005). Using a Query-Based Multicontext For Knowledge Base Browsing. In *Formal Concept Analysis, Third International Conf., ICFCA 2005-Supplementary Volume*, pages 62–78, Lens, France. IUT de Lens, Universite d'Artois.

[Tane et al., 2006] Tane, J., Cimiano, P., and Hitzler, P. (2006). Query-Based Multicontexts for Knowledge Base Browsing: An Evaluation. In Schärfe, H., , Hitzler, P., and Ohrstrom, P., editors, *Conceptual Structures: Inspiration and Application 14th International Conference on Conceptual Structures, ICCS 2006, Aalborg, Denmark, July 16-21, 2006*, number 4068 in Lecture Notes in Computer Science, Heidelberg. Springer.

[Tane et al., 2004] Tane, J., Schmitz, C., and Stumme, G. (Mai 2004). Semantic Resource Management for the Web: An ELearning Application. In *Proc. 13th International World Wide Web Conference (WWW 2004).*

[Tane et al., 2003] Tane, J., Schmitz, C., Stumme, G., Staab, S., and Studer, R. (2003). The Courseware Watchdog: an Ontology-based tool for Finding and Organizing Learning Material. In Klaus David, I. W., editor, *Fachtagung Mobiles Lernen und Forschen, 6.11.2003, Universität Kassel*, Kassel, Germany. Kassel University Press.

[Tempich et al., 2004] Tempich, C., Ehrig, M., Fluit, C., Haase, P., Marti, E. L., Plechawski, M., and Staab, S. (2004). XAROP: A midterm report in introducing a decentralized semantics-based knowledge sharing application. In Karagiannis, D. and Reimer, U., editors, *Proceedings of the 5th International Conference on Practical Aspects of Knowledge Management (PAKM 2004)*, number 3336 in Lecture Notes in Computer Science, Vienna, Austria. Springer.

[Toivonen, 1996] Toivonen, H. (1996). *Discovery of frequent patterns in large data collections*. PhD thesis, Department of Computer Science, University of Helsinki. Report-A-1006-5.

[Tsur et al., 1998] Tsur, D., Ullman, J. D., Abiteboul, S., Clifton, C., Motwani, R., Nestorov, S., and Rosenthal, A. (1998). Query flocks: a generalization of association-rule mining. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA. ACM Press.

[Valtchev et al., 2004] Valtchev, P., Missaoui, R., and Godin, R. (2004). Formal concept analysis for knowledge discovery and data mining: The new challenges. In [Eklund, 2004], pages 352–371.

[van Harmelen and Horrock, 2000] van Harmelen, F. and Horrock, I. (2000). Questions and answers on OIL: the ontology inference layer for the semantic web. *IEEE Intelligent Systems*, 15(6):69–72.

[Vossen, 1994] Vossen, G. (1994). *Datenmodelle, Datenbanksprachen und Datenbank–Management–Systeme*. Addison–Wesley, 2nd edition edition.

[Wille, 1982] Wille, R. (1982). Restructuring lattice theory: an approach based on hierarchies of concepts. In Rival, I., editor, *Ordered Sets: proceedings of the NATO advanced study institute held at Banff, Canada Aug. 28 to Sept. 12, 1981*, number 83 in NATO ASI Series, pages 445–470. Reidel. NATO advanced study institutes series ; 83.

[Wille, 1996] Wille, R. (1996). Conceptual Structures of Multicontexts. In *Conceptual Structures: Knowledge Representation as Interlingua, Proceedings of the 4th International Conference on Conceptual Structures, ICCS'96*, volume 1115 of *Lecture Notes*, pages 23–39, Sydney, Australia. Springer Lecture Notes in Computer Science.

[Wille, 1997] Wille, R. (1997). Conceptual Graphs and Formal Concept Analysis. In Lukose, D., Delugach, H., Keeler, M., Searle, L., and Sowa, J., editors, *Conceptual Structures: Fulfilling Peirce's Dream, Proceedings of the 5th International Conference on Conceptual Structures, ICCS'97*, volume 1257 of *Lecture Notes in Artificial Intelligence*, pages 290–303, Seattle,WA, U.S.A. Springer Lecture Notes in Computer Science.

[Wille, 2005] Wille, R. (2005). Conceptual knowledge processing in the field of economics. In *Formal Concept Analysis: Foundations and Applications*, number 3626 in Lecture Notes in Artificial Intelligence, pages 226–249. Springer.

[Ziegler et al., 2002] Ziegler, J., Kunz, C., and Botsch, V. (2002). Matrix browser - visualizing and exploring large networked information spaces. In *Conference on Human Factors in Computing Systems CHI '02 extended abstracts on Human factors in computing systems*, pages 602–603, Minneapolis, Minnesota, USA.

Appendix A

Ontology

http://swrc.ontoware.org/ontology

$$\begin{aligned}
\top &\sqsubseteq \forall \mathsf{citedBy}.\top\\
\mathsf{ResearchGroup} &\sqsubseteq \forall \mathsf{head}.\mathsf{Employee}\\
\mathsf{ResearchGroup} &\sqsubseteq \forall \mathsf{member}.\mathsf{Employee}\\
\mathsf{ResearchGroup} &\sqsubseteq \mathsf{Organization}\\
\mathsf{Article} &\sqsubseteq \forall \mathsf{author}.\mathsf{Person}\\
\mathsf{Article} &\sqsubseteq \mathsf{Publication}\\
\mathsf{Unpublished} &\sqsubseteq \forall \mathsf{author}.\mathsf{Person}\\
\mathsf{Unpublished} &\sqsubseteq \mathsf{Publication}\\
\mathsf{Exhibition} &\sqsubseteq \mathsf{Event}\\
\mathsf{ProjectReport} &\sqsubseteq \forall \mathsf{describesProject}.\mathsf{Project}\\
\mathsf{ProjectReport} &\sqsubseteq \mathsf{Report}\\
\mathsf{Manager} &\sqsubseteq \mathsf{Employee}\\
\mathsf{Undergraduate} &\sqsubseteq \mathsf{Student}\\
\mathsf{Meeting} &\sqsubseteq \forall \mathsf{participant}.\mathsf{Person}\\
\mathsf{Meeting} &\sqsubseteq \mathsf{Event}\\
\mathsf{Student} &\sqsubseteq \forall \mathsf{studiesAt}.\mathsf{University}\\
\mathsf{Student} &\sqsubseteq \mathsf{Person}\\
\mathsf{TechnicalStaff} &\sqsubseteq \mathsf{Employee}\\
\mathsf{Misc} &\sqsubseteq \mathsf{Publication}\\
\mathsf{TechnicalReport} &\sqsubseteq \forall \mathsf{organization}.\mathsf{Organization}\\
\mathsf{TechnicalReport} &\sqsubseteq \mathsf{Report}\\
\mathsf{InProceedings} &\sqsubseteq \forall \mathsf{publisher}.\mathsf{Organization}\\
\mathsf{InProceedings} &\sqsubseteq \forall \mathsf{organization}.\mathsf{Organization}\\
\mathsf{InProceedings} &\sqsubseteq \forall \mathsf{editor}.\mathsf{Person}\\
\mathsf{InProceedings} &\sqsubseteq \forall \mathsf{author}.\mathsf{Person}\\
\mathsf{InProceedings} &\sqsubseteq \mathsf{Publication}\\
\mathsf{MasterThesis} &\sqsubseteq \mathsf{Thesis}\\
\mathsf{ResearchProject} &\sqsubseteq \mathsf{Project}\\
\mathsf{Product} &\sqsubseteq \forall \mathsf{developedBy}.\mathsf{Organization}\\
\mathsf{University} &\sqsubseteq \forall \mathsf{student}.\mathsf{Student}\\
\mathsf{University} &\sqsubseteq \forall \mathsf{hasParts}.\mathsf{Department}\\
\mathsf{University} &\sqsubseteq \mathsf{Organization}
\end{aligned}$$

$$\begin{aligned}
\mathsf{AdministrativeStaff} &\sqsubseteq \mathsf{Employee}\\
\mathsf{Manual} &\sqsubseteq \forall \mathsf{organization.Organization}\\
\mathsf{Manual} &\sqsubseteq \forall \mathsf{author.Person}\\
\mathsf{Manual} &\sqsubseteq \mathsf{Publication}\\
\mathsf{SoftwareProject} &\sqsubseteq \forall \mathsf{product.Product}\\
\mathsf{SoftwareProject} &\sqsubseteq \mathsf{DevelopmentProject}\\
\mathsf{Employee} &\sqsubseteq \forall \mathsf{affiliation.Organization}\\
\mathsf{Employee} &\sqsubseteq \mathsf{Person}\\
\mathsf{Lecture} &\sqsubseteq \forall \mathsf{givenBy.Person}\\
\mathsf{Lecture} &\sqsubseteq \mathsf{Event}\\
\mathsf{AssociateProfessor} &\sqsubseteq \mathsf{FacultyMember}\\
\mathsf{InCollection} &\sqsubseteq \forall \mathsf{author.Person}\\
\mathsf{InCollection} &\sqsubseteq \forall \mathsf{publisher.Organization}\\
\mathsf{InCollection} &\sqsubseteq \forall \mathsf{editor.Person}\\
\mathsf{InCollection} &\sqsubseteq \mathsf{Publication}\\
\mathsf{Workshop} &\sqsubseteq \mathsf{Event}\\
\mathsf{FacultyMember} &\sqsubseteq \mathsf{AcademicStaff}\\
\mathsf{Lecturer} &\sqsubseteq \mathsf{AcademicStaff}\\
\mathsf{Publication} &\sqsubseteq \forall \mathsf{cite.Publication}\\
\mathsf{Proceedings} &\sqsubseteq \forall \mathsf{editor.Person}\\
\mathsf{Proceedings} &\sqsubseteq \forall \mathsf{publisher.Organization}\\
\mathsf{Proceedings} &\sqsubseteq \forall \mathsf{organization.Organization}\\
\mathsf{Proceedings} &\sqsubseteq \mathsf{Publication}\\
\mathsf{PhDThesis} &\sqsubseteq \mathsf{Thesis}\\
\mathsf{Association} &\sqsubseteq \mathsf{Organization}\\
\mathsf{Institute} &\sqsubseteq \forall \mathsf{hasParts.ResearchGroup}\\
\mathsf{Institute} &\sqsubseteq \forall \mathsf{cooperateWith.Institute}\\
\mathsf{Institute} &\sqsubseteq \mathsf{Organization}\\
\mathsf{Graduate} &\sqsubseteq \mathsf{Student}\\
\mathsf{Project} &\sqsubseteq \forall \mathsf{isAbout.ResearchTopic}\\
\mathsf{Project} &\sqsubseteq \forall \mathsf{financedBy.Organization}\\
\mathsf{Project} &\sqsubseteq \forall \mathsf{projectInfo.Publication}\\
\mathsf{Project} &\sqsubseteq \forall \mathsf{head.Employee}\\
\mathsf{Project} &\sqsubseteq \forall \mathsf{member.Person}\\
\mathsf{Project} &\sqsubseteq \forall \mathsf{carriedOutBy.Organization}\\
\mathsf{Event} &\sqsubseteq \forall \mathsf{atEvent.Event}\\
\mathsf{Event} &\sqsubseteq \forall \mathsf{hasPartEvent.Event}\\
\mathsf{ResearchTopic} &\sqsubseteq \forall \mathsf{dealtWithIn.Project}\\
\mathsf{ResearchTopic} &\sqsubseteq \forall \mathsf{isWorkedOnBy.AcademicStaff}\\
\mathsf{ResearchTopic} &\sqsubseteq \mathsf{Topic}\\
\mathsf{Conference} &\sqsubseteq \mathsf{Event}\\
\mathsf{ProjectMeeting} &\sqsubseteq \mathsf{Meeting}\\
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{headOf.Project}\\
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{headOfGroup.ResearchGroup}\\
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{publication.Publication}\\
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{organizerOrChairOf.Event}\\
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{memberOfPC.Event}\\
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{cooperateWith.AcademicStaff}\\
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{editor.Publication}\\
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{supervises.PhDStudent}
\end{aligned}$$

$$\begin{aligned}
\mathsf{AcademicStaff} &\sqsubseteq \forall \mathsf{worksAtProject.Project}\\
\mathsf{AcademicStaff} &\sqsubseteq \mathsf{Employee}\\
\mathsf{InBook} &\sqsubseteq \forall \mathsf{editor.Person}\\
\mathsf{InBook} &\sqsubseteq \forall \mathsf{author.Person}\\
\mathsf{InBook} &\sqsubseteq \forall \mathsf{publisher.Organization}\\
\mathsf{InBook} &\sqsubseteq \mathsf{Publication}\\
\mathsf{Report} &\sqsubseteq \forall \mathsf{author.Person}\\
\mathsf{Report} &\sqsubseteq \mathsf{Publication}\\
\mathsf{SoftwareComponent} &\sqsubseteq \mathsf{Product}\\
\mathsf{Organization} &\sqsubseteq \forall \mathsf{develops.Product}\\
\mathsf{Organization} &\sqsubseteq \forall \mathsf{publishes.Publication}\\
\mathsf{Organization} &\sqsubseteq \forall \mathsf{carriesOut.Project}\\
\mathsf{Organization} &\sqsubseteq \forall \mathsf{finances.Project}\\
\mathsf{Organization} &\sqsubseteq \forall \mathsf{technicalReport.TechnicalReport}\\
\mathsf{Organization} &\sqsubseteq \forall \mathsf{employs.Person}\\
\mathsf{Thesis} &\sqsubseteq \forall \mathsf{author.Person}\\
\mathsf{Thesis} &\sqsubseteq \forall \mathsf{school.University}\\
\mathsf{Thesis} &\sqsubseteq \mathsf{Publication}\\
\mathsf{Booklet} &\sqsubseteq \forall \mathsf{author.Person}\\
\mathsf{Booklet} &\sqsubseteq \mathsf{Publication}\\
\mathsf{Department} &\sqsubseteq \forall \mathsf{hasParts.Institute}\\
\mathsf{Department} &\sqsubseteq \mathsf{Organization}\\
\mathsf{AssistantProfessor} &\sqsubseteq \mathsf{FacultyMember}\\
\mathsf{Book} &\sqsubseteq \forall \mathsf{publisher.Organization}\\
\mathsf{Book} &\sqsubseteq \forall \mathsf{editor.Person}\\
\mathsf{Book} &\sqsubseteq \forall \mathsf{author.Person}\\
\mathsf{Book} &\sqsubseteq \mathsf{Publication}\\
\mathsf{FullProfessor} &\sqsubseteq \mathsf{FacultyMember}\\
\mathsf{DevelopmentProject} &\sqsubseteq \mathsf{Project}\\
\mathsf{Enterprise} &\sqsubseteq \mathsf{Organization}\\
\mathsf{PhDStudent} &\sqsubseteq \forall \mathsf{publication.Publication}\\
\mathsf{PhDStudent} &\sqsubseteq \forall \mathsf{supervisor.AcademicStaff}\\
\mathsf{PhDStudent} &\sqsubseteq \forall \mathsf{worksAtProject.Project}\\
\mathsf{PhDStudent} &\sqsubseteq \mathsf{Graduate}\\
\mathsf{cite} &\equiv \mathsf{citedBy}^{-1}\\
\mathsf{develops} &\equiv \mathsf{developedBy}^{-1}\\
\mathsf{carriesOut} &\equiv \mathsf{carriedOutBy}^{-1}\\
\mathsf{publisher} &\equiv \mathsf{publishes}^{-1}\\
\mathsf{citedBy} &\equiv \mathsf{cite}^{-1}\\
\mathsf{employs} &\equiv \mathsf{affiliation}^{-1}\\
\mathsf{supervisor} &\equiv \mathsf{supervises}^{-1}\\
\mathsf{headOfGroup} &\equiv \mathsf{head}^{-1}\\
\mathsf{projectInfo} &\equiv \mathsf{describesProject}^{-1}\\
\mathsf{hasPartEvent} &\equiv \mathsf{atEvent}^{-1}
\end{aligned}$$