A Meaning Driven Querying Methodology: End-user Oriented Querying

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY (ETH) Zurich

for the degree of Doctor of Technical Sciences

presented by

EPAMINONDAS KAPETANIOS

Diploma in Computer Science,
Univ. of Athens, Hellas
born October 8, 1963
citizen of Athens, Hellas

accepted on the recommendation of

Prof. Dr. M. C. Norrie, examiner
Prof. Dr. H. Hinterberger, co-examiner

2000
This thesis is dedicated to the lovers of science and truth.
Acknowledgments

I am grateful to Prof. Dr. Moira C. Norrie, Institute of Information Systems, as well as to Prof. Dr. Hans Hinterberger, Institute of Scientific Computing, at the Department of Computer Science, Swiss Federal Institute of Technology (ETH) Zurich, for accepting this thesis.

I would also like to express my gratitude to all those students who helped, in one way or another, in building up the system(s) based on the theory presented in this thesis, by realizing components of the system(s). In particular, I would like to acknowledge the contribution of the following students: Robert Ritter, for implementing an HTML based user interface to the Meaning Driven Automaton for querying classification rules such as medical guidelines on the Web, Dijana Fuhrer-Stakic, for the Java based implementation of the graph-based visual query interface, Sabine Keuser and Paul Groenewoud, for the Java based implementation of the collection/querying graphical user interface for simple instances descriptions, and Patric Müller, for his help in implementing the storage transformation component for data collection processes.

Without their willingness to work with the Application Programming Interface underlying the implementation of the inference engine of the specified automaton as well as their kind cooperation, no working system would have been possible. My special thanks also are devoted to Andrea Lombardoni, Andreas Frei, and Ljiljana Vukelja for their efforts and contributions in examining various implementation aspects of the system(s).

I am also indebted to Dr. med. Julian Schilling, Institute of Social and Preventive Medicine, University of Zürich, Reto Haeni, Project Manager of IMSMA within the Centre of Security Studies and Conflicts research, ETH-Zürich, Bernhard Brabec, Federal Institute of Snow and Avalanche Research, Rudolf Maag, from the pharmaceutical company GlaxoSmithKline, for their financial support and their trust in using the developed system(s) which resulted from the theory.

My special thanks are also devoted to Dr. med. Peter Gerstl, Dr. med. Pius Wyss, Dr. med. Michael Oberson, Dr. med. Fabrizio Barazzoni, Prof. Dr. med. Ticiano Moccetti, director of
the Cardiocentro Ticino (CCT), for their feedback and valuable discussions we had. Furthermore, I would like to express my thanks to the company SAG (Software AG) Switzerland for showing interest in this querying approach and providing me with their product XML (Tamino) Server free of charge for experimentation purposes.

Last but not least, I was always delighted to meet and play together with Dr. phil. Pierfrancesco Basile in the same basketball team and especially for the numerous philosophical discussions we had and his guiding of a naive user of philosophy, like myself, through the literature. Discussions and recommended books were an enlightenment for the many questions I had.
## Contents

Acknowledgments v  
List of Figures xiii  

1. INTRODUCTION 1  
   1. Problem statement and motivation 1  
   2. Recommended problem solution: Meaning Driven Querying Methodology 3  
   3. Contribution 5  
   4. Structure of the thesis 7  

2. THEORETICAL BACKGROUND 9  
   1. Graphs, binary relations and propositional logic 10  
      1.1 Graphs and binary relations 10  
      1.2 Propositional logic 11  
   2. Formal language syntax 12  
      2.1 Regular languages and expressions 13  
      2.2 Turing machines 13  
      2.3 Context freedom and sensitivity in formal language syntax 14  
   3. Query languages 16  
   4. Theories on semantics 17  

3. RELATED WORK 21  
   1. From meta-data to knowledge bases 22  
      1.1 About meta-data 22  

1.2 Terminological and Knowledge Based Systems
1.3 Meta-data standards and models

2. Database querying approaches
2.1 Instances description oriented query languages
2.2 Navigation style oriented querying
   2.2.1 Visual Query Systems and Languages
   2.2.2 Graph based formalisms and traversal paths/links

3. Question Answering Systems

4. End-user database interfaces
   4.1 Graphical database querying interfaces
   4.2 Natural language interfaces for databases

5. Human-Computer Interaction (HCI)

6. Summary

4. QUERY CONSTRUCTION PARADIGMS
   1. Simple instances description oriented queries
   2. Advanced instances description oriented queries
      2.1 Expressing operations
   3. Querying of classification rules
      3.1 Guidelines and classification rules

5. THE REPRESENTATION MODEL OF MEANING OF TERMS
   1. Representation of intentional meaning of terms
      1.1 The domain terms
      1.2 The operational terms
   2. Representation of context of terms
      2.1 The connectionism model
      2.2 Representation of constraints
         2.2.1 Constraints over domain terms
         2.2.2 Constraints over operational terms

6. THE STATE AUTOMATON SPECIFICATION AND THE MDDQL
1. The definition of the meaning driven querying automaton
2. The state transition logic and its algorithms
   2.1 Exclusion of operational terms
   2.2 Inclusion of operational terms
7. EXPRESSIVENESS AND COMPUTABILITY OF MDDQL
   1. Expressiveness
   2. Computability
8. TOWARDS A COMPREHENSIVE QUERY ANSWERING SYSTEM
   1. A general overview of the system architecture
   2. Implementation of the automaton as an inference engine
   3. The query execution and/or transformation perspective
   4. The perspective of generating answers
9. EPILOGUE
   1. Summary
   2. Lessons learned
   3. From prototypes to operational use
   4. Future outlook
10. BIBLIOGRAPHY
Appendices
A– GLOSSARY
B– Application paradigms
   1. A Second Opinion System (SOS) in Medicine
   2. Analysis of Hospitalisation Infections
   3. An Information Management System for Mine Action (IMSMA)
   4. A Regional Avalanche Information and Forecasting System
C– Implementation issues
   1. Java based implementation of the state transition algorithms
2. OMSJava based implementation of the representation model of meaning 162
   2.1 The model specification: An example 162
   2.2 The model instantiation: An example 167
List of Figures

4.1 Natural language selection. 48

4.2 System suggestion for refinement of the concept “patient”. 49

4.3 System suggestion for refinement of the concept “postmenopausal patient”. 50

4.4 System suggestion for query refinement based on unconditioned properties and/or values. 51

4.5 System suggestion for query refinement including the preconditioned property “major impairment”. 51

4.6 Exclusion of property “age” when a different query context is given. 52

4.7 Addressing values within the dynamically inferred finite value domain for “age”. 52

4.8 A German language terms based query construction 53

4.9 Current query state and/or potential refinement referring to the concept term “Automatically delivered measurement data”. 55

4.10 Potential query refinement through exploitation of acyclic graph structures 56

4.11 Potential query refinement through exploitation of acyclic graph structures 57

4.12 Constraints based assignment of numerical values for the property (variable) "temperature". 58

4.13 Semantics based assignment of univariate functions (categorical values) to a query. 59

4.14 Semantics based assignment of univariate functions (numerical values) to a query. 60
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3</td>
<td>The multi-lingual, constraints based specification of conceptualization of the terminology and its meaning.</td>
<td>122</td>
</tr>
<tr>
<td>8.4</td>
<td>The query execution perspective.</td>
<td>124</td>
</tr>
<tr>
<td>8.5</td>
<td>The query transformation and execution perspective</td>
<td>125</td>
</tr>
<tr>
<td>8.6</td>
<td>The query result (answer) generation perspective</td>
<td>126</td>
</tr>
<tr>
<td>C.1</td>
<td>The Java based implementation of the end-user guidance mechanism (state transition functions) for simple instances description oriented queries and classification rules</td>
<td>160</td>
</tr>
<tr>
<td>C.2</td>
<td>The Java based implementation of the end-user guidance mechanism (state transition functions) for advanced instances description oriented queries</td>
<td>161</td>
</tr>
</tbody>
</table>
Abstract

Providing information out of data has always been the major concern in information science. Addressing and disseminating the acquired information, however, has been the major focus of research activities in query languages and question answering systems. Despite the variety of approaches taken so far, two central questions can be raised when we ask for information: how to ask for information and what kind of queries/questions the system can answer.

Although the second question has been exhaustively examined by theories and/or practical solutions referring to model- or proof-theoretic semantics, closed versus open world assumptions, fuzziness or uncertainty of query results, etc., the way how to ask for available information still remains a major challenge, especially when end-users are involved in the querying process. The major presupposition of formulating a reasonable query in terms of database or Web data query languages used as programming tools is a) the familiarity with the syntax formalisms of the query languages which are rather complex for an end-user, b) understanding of the semantics of the model underlying the data/knowledge repository.

In order to improve usability and alleviate end-users from syntax-based formulation of queries, advanced (visual) query interfaces and/or languages as well as interactive query formulation techniques have been proposed which provide a syntax-free (visual) formalism for query formulation. The latter mostly relies on graphical presentations of conceptual models which turn out to be very tedious to handle and difficult to understand, especially when large knowledge repositories and/or complex data schemas are concerned. Additionally, the role of semantics defined as the relationship of linguistic symbols and their meaning as well as a constraints-based usage of query terms and results has not been the case. Furthermore, in many cases the end-user does not really know what is relevant to her/him or how to make use of the available knowledge in order to formulate a query.

In this sense, the meaning driven querying methodology presented in this thesis changes the perspective of end-user oriented query construction in that the system guides the end-user to the construction of a reasonable query through consideration of semantics/meaning of vocabulary terms. The meaning of queries is expressed in terms of a) constraints-based consideration of terms within a query, i.e. rejection of mutually exclusive terms, b) the intensional meaning or interpretation of query terms, i.e. talking about the same things but with different interpretations, c) the appearance of query terms in a particular natural language which enhance the understanding of the query terms. Meaning is represented in a knowledge base and provide the set of alphabet terms for the designated query language MDDQL (Meaning Driven Data Query Language). In addition to the knowledge base as a repository of the query alphabet terms, an inference engine and visual query interface(s) are used as an interaction platform or communication blackboard between end-user and machine.

The inference engine incorporates the interaction mode between system and end-user conceived as an abstract machine which receives as input the current query context (current state of a query) and produces a semantically meaningful output (subsequent/extended state of the query). The decision on which terms should be further considered for a potential extension of the query expression is based on the semantics/meaning of the application domain and/or operational terms which is represented in the knowledge base for the vocabulary. Furthermore, since terms might appear in more than one natural language, it is possible to construct a query expression in two or more natural languages and still receive the same query results. This is due to the separation between the world of symbols as used for the
representation and structuring of data/knowledge repositories and the world of multi-lingual vocabularies as used for the formulation of queries in MDDQL.

Given that the construction mode of a query resembles the definition of an abstract machine, a constructed query is an already parsed one. This is in contrast with other parsing approaches where a query needs to be formulated first and subsequently parsed. Therefore, no additional parsing after the interactive construction is needed, a feature which is particularly valuable when multi-lingual environments of end-users are considered. Otherwise, more than one parser needs to be implemented, according to the number of natural languages supported.

Abstract


Um die Nutzbarkeit eines Systems zu erhöhen und die Aufgabe der syntax-basierten Formulierung einer Abfrage von Endbenutzern zu erleichtern, es wurden sowohl visuelle Abfrageschnittstellen und/oder Programmiersprachen als auch interaktive Abfrageformulierungstechniken vorgeschlagen und entwickelt, die auf einen syntaxfreien (visuellen) Formalismus für die Abfrageformulierung basieren. Diese beziehen sich, in den meisten Fällen, auf graphische Darstellungen des konzeptionellen Modells, die relativ schwierig zu handhaben und zu interpretieren sind, insbesondere wenn große Wissensrepositorien und/oder komplexe Datenschemata zu berücksichtigen sind. Darüber hinaus ist die Rolle der Semantiken, die sowohl durch die Beziehung linguistischer Symbole und deren Bedeutung als auch durch die constraints-basierte Benutzung der Abfrageterme und Abfrageresultate definiert ist, nicht von zentraler Bedeutung. Diese Problematik ist auch durch die Tatsache verstärkt, dass Endbenutzer meistens nicht wissen, was ist relevant oder wie sie verfügbares Wissen zu gebrauchen ist, um eine Abfrage zu formulieren.

wird ausgedrückt durch a) die constraints-basierte Einbettung der Begriffe in einer Abfrage, z.B. Ablehnung von bedeutungslosen Abfragen, b) die intensionale Bedeutung oder Interpretation der Abfragebegriffe, d.h. ähnliche Begriffe mit unterschiedlichen Interpretationen behandeln, c) die Erscheinung der Begriffe in der Abfrage in einer vorausgewählten natürlichen Sprache, die das Verstehen der verwendeten Symbole des Implementierungsmodells ermöglicht oder erhöht.

Begriffe und derer Bedeutung ist in einer Wissensbasis repräsentiert, die die Menge aller dem Alphabet der Abfragesprache MDDQL gehörenden Begriffe bestimmt. Darüber hinaus existieren zwei weitere Komponenten, die den Abfragekonstruktionsmechanismus von MDDQL realisieren: ein Inferenzmechanismus und die visuellen Abfrageschnittstellen als Kommunikationsplattform zwischen einem Endbenutzer und dem System.


Eine durch den als abstrakte Maschine spezifizierten Interaktionsmodus eine in MDDQL konstruierte Abfrage gilt als eine Abfrage, die schon grammatikalisch analysiert ist. Diese Eigenschaft steht im Gegensatz zu den üblichen Parsingmechanismen, die eine Abfrage erst nach dem Formulierungsabschluss parsen. Diesbezüglich, es wird kein zusätzlicher Parsen der Abfrage benotigt, das, im Fall einer mehrsprachigen Benutzersprache, die Verfügbarkeit mehrerer Parsingmechanismen, d.h. ein Parsingmechanismus pro natürliche Sprache, fuer die selbe Abfrage erfordert hättte.
Chapter 1

INTRODUCTION

A more searching argument against the axiom of internal relations is derived from a consideration of what is meant by the "nature" of a term. Is this the same as the term itself, or is it different? If it is different, it must be related to the term and the relation of a term to its nature cannot, without an endless regress, be reduced to something other than a relation. Thus if the axiom is to be adhered to, we must suppose that a term is not other than its nature... But in that case, what is the bond that unites predicates into predicates of one subject? Any causal collection of predicates might be supposed to compose a subject, if subjects are not other than the system of their own predicates... We cannot attempt to introduce a relation of coherence between predicates, in virtue of which they may be called predicates of one subject; for this would base predications upon a relation, instead of reducing relations to predications. Thus we get into equal difficulties whether we affirm or deny that a subject is other than its "nature".


1. PROBLEM STATEMENT AND MOTIVATION

Providing information out of data has always been the major concern in information science. Addressing and disseminating the acquired information, however, has been the major focus of research activities in query languages and question answering systems. Despite the variety of approaches taken so far, two central questions can be raised when we ask for information: how to ask for information and what kind of queries the system can answer.

Although the second question has been exhaustively examined by theories and/or practical solutions referring to model- or proof-theoretic semantics, closed versus open world assumptions, fuzziness or uncertainty of query results, etc., the way how to ask for available information still remains a major challenge, especially when end-users are involved in the querying process.

The roots of the challenge can be found at the assumption we make that an end-user:
1. learns the query language syntax

2. understands the meaning of the underlying database schema in terms of adequate interpretations of data constructs such as relations, classes and collections,

3. understands the meaning of attributes and/or values,

4. is aware of the context which relates to the data as expressed in terms of measurement units, explanation or definitions, etc.

In this sense, the key answer to the question of how to ask for the available information goes through the role of semantics the lack of which in existing information systems becomes more and more apparent, i.e. a semantic gap. Furthermore, the end-user does not really know what he is looking for or how to ask for available information, unless she/he fully understands it, i.e. seeing, interpreting it. Even if the user knows what he wants, he is confronted with complex syntax formalisms, since typical systems rely only on low-level features as expressed by the data model on which the data repository relies.

The problem becomes more acute when the semantic complexity of a particular application domain increases. Acquisition of scientific information, for example, requires great familiarity with the domain science which is prohibiting to end-users other than domain experts. Even within a particular domain science, using an information system as communication platform for knowledge/information exchange among domain experts becomes very difficult, or even impossible, without an appropriate interpretation of data and/or its context. Additionally, querying information systems which rely on large data repositories in terms of large or complex database schemas becomes cumbersome, since most of the querying interfaces they provide are oriented towards application programmers and not end-users.

On the other side, the growth of the Internet as an easy-to-access information repository providing various types of information ranging from unstructured data to traditional record-oriented one poses an additional querying challenge. Proposals coming from different areas, namely databases, artificial intelligence and human-computer interaction, share the final goal of making the Web a huge, easy-to-access, information repository. Therefore, the problem definition resembles the definition of a bridge of the semantic gap in terms of moving from low-level features to high-level semantics.

In order to meet this requirement and overcome the end-user/queried system communication problem, various advanced database or information system query interfaces have been elaborated which mainly aimed at meeting the challenges (1) and partly (2). However, the challenges (3) and (4) could not be adequately met by these query techniques, since they still lack the exploitation of meaning of vocabulary terms during formulation of a query expression. Furthermore, these solutions shifted the problem from understanding the data to understanding the user interface, and in
most cases, they are still bound to the world of symbols chosen for the underlying data/knowledge representation model. Thus interpretation of an underlying database schema or knowledge model in terms of linguistic elements of a particular natural language (partly expressed by challenge (2)) such as English or German has not been the case.

Since construction of reasonable or meaningful queries in more than one natural language which meets all the challenges given above without adding complexity to the human-computer querying interaction has been declared to be the major goal in order to succeed in retrieving valuable information, a more “intelligent” querying paradigm must be provided. It is expected that the system actually helps in formulating a reasonable query by taking into consideration the meaning and context of data. This presupposes that a considerable amount of knowledge concerning the nature of terms to be used within a query must be provided by the system.

2. RECOMMENDED PROBLEM SOLUTION: MEANING DRIVEN QUERYING METHODOLOGY

The meaning driven querying methodology as proposed in this thesis relies upon the interactive guidance of end-users in constructing meaningful or reasonable multi-lingual queries through system suggestions of terms to accomplish a query statement. This is achieved through the exploration of the knowledge space of vocabulary terms as given by a particular application domain.

Having the system guide the end-user to the construction of a reasonable query (meeting challenges (1) and (2)) presupposes that only reasonable suggestions of query terms are made for further consideration in the query construction process (meeting challenges (3) and (4)). This has been conceived as a transition from a particular query state (context) - set of terms currently being members of the query state - to a new query state. Thereby, the new terms which can extend the current query context are determined by the meaning of terms and/or the query context. In order to meet challenge (3), the meaning of terms is defined by

- the natural language based interpretation of symbols used by the data repository system,
- the data context such as measurement units, explanations or definitions, images, etc.,

where, in order to meet challenge (4), the meaning of the query context is defined by

- the semantic dependencies holding among terms, i.e. which terms can be put together in a reasonable query statement.
The meaning of terms determines the view of terms, and consequently, the degree of relevance for further consideration by an end-user selection out of the set of system suggested terms. On the other hand, the semantic dependencies determine which terms should appear (suggested) to the end-user when a particular query context is given. Therefore, the suggested terms are inferred by checking the semantic consistency between the current query context and the terms to extend it such that

- only those attributes and/or values are suggested which are related to those already participating in the current query context,
- only those attributes and/or values are suggested for which no semantic conflicts with the already participating attribute/value terms are detected,
- only those operations are suggested which are applicable to the terms according to the data contents which they represent.

Therefore, the query construction process through an end-user/system interaction can be formally defined as a state automaton, where a state is equivalent to a current query context - set of terms already participating in the query state - and a state transition function where the meaning of terms and/or query context determines the next query state. It turns out that the specification of a meaning driven state automaton leads to the specification of a Meaning Driven Data Querying Language MDDQL where the automaton acts as an acceptor of the query statements. Since the final query state is determined on the basis of already checked semantic consistencies, the query is regarded as an already accepted one, which means that syntactic and semantic parsing of the query become part of the query construction process. This is particularly useful when more than one natural language is considered for representing the interpretation of symbols in the data repository system.

A system based on the meaning driven querying methodology has been implemented as follows. Its main components are a) the knowledge base represented by an undirected graph expressing a semantic space for the representation of meaning of vocabulary (query) terms and b) an inference engine operating upon the knowledge space and providing the state transition (inference) algorithms for suggestions of semantically consistent terms. The first is provided by an object-oriented view to query terms, i.e. nodes of the graph represent objects of terms, as well as a conditioned connectionism model, whereas the latter is provided by a library of implemented algorithms in the Java programming language.

The conditioned connectionism model within the formed semantic space enables a (recursive) structural definition of the conceptualization of the domain of discourse. This enables the assignment of properties/attributes to concepts as well as the assignment of well-defined value domains to properties/attributes in a multi-lingual mode. In order to exclude values and/or attributes from
system suggestions which are not consistent with the current set of attribute/value pairs, however, preconditions have also been expressed and assigned to nodes which must be satisfied.

The specified and implemented algorithms infer the set of terms to be suggested to the end-user given a particular query context and the representation model of meaning. They implement all possible state transition functions applying for the construction of specific families of query statements according to the wished complexity, and therefore, having an impact on the expressiveness of the query language.

In addition, the system consists of add-on components such as a) various visual query interfaces with which the end-user/system interaction takes place, b) an application server where transformations of the constructed queries with MDDQL towards database specific query languages such as SQL take place, c) a statistical evaluation server where statistical operations as included in the query take place. Given that more than one data or knowledge repository are given, the system can be extended by semantic mediators which enable the construction of an answer from more than one repository.

3. CONTRIBUTION

The meaning driven querying methodology presented in this thesis changes the perspective of end-user oriented query construction in that the system guides the end-user to the construction of a reasonable query through the represented semantics of a given vocabulary. Simple visual query interfaces are used as an interaction platform or communication blackboard between end-user and machine. This blackboard is where a partly constructed query appears and where the system makes reasonable suggestions to the end-user for further refinement of the query depending on the current query state. The meaning of domain and operational terms as well as of the formulated query itself is the central part of the end-user/machine interaction.

Therefore, it is not necessary to force the end-user to learn a particular query language and/or understand the semantics of the implemented data model which is a very tedious task or even impossible when advanced or large application domains are considered - for example, in domain sciences such as medicine, physics, and/or large data repositories for decision support systems. Moreover, there is no need of providing a complex visual query interface to cope with. But still the ability of an end-user formulating a complex query with operational/analytical aspects must be the goal.

The meaning of queries is expressed in terms of a) constraints-based suggestions to end-users holding among query terms, i.e. rejection of meaningless queries, b) the intensional meaning or interpretation of query terms, i.e. talking about the same things, c) the appearance of query terms
in a particular natural language which enhance the understanding of the query terms. This is a crucial issue in data or information exchange, particularly when terminology coming from other domain sciences is used. Constructing queries through a common understanding of data supports interdisciplinary research and makes knowledge more applicable and useful to the public.

A further consequence of the meaning driven querying methodology is that a submitted query is an already parsed one, in contrast with other parsing approaches where a query needs to be formulated first and subsequently parsed. This holds due to the fact that the end-user/system interaction mode is specified in terms of a state transition automaton which enables the construction of a reasonable or meaningful query interactively. Therefore, no additional parsing after the interactive construction is needed, a feature which is particularly valuable when multi-lingual environments of end-users are considered. Otherwise, more than one parser needs to be implemented, according to the number of natural languages supported.

Additionally, the interactive parsing mechanism makes implicit use of meaning of terms in order to decide which will be the next state to move into. In this sense, embedding of syntax and semantics within the parsing mechanism of a query language could alleviate difficulties arising with the task of machine understanding of users’ or application domain semantics.

Finally, the meaning driven querying methodology supported in

- providing and disseminating large sets of medical guidelines concerning interventions in Cardiology and Hysterectomy over the Internet,
- establishing and introducing the underlying query answering system as an evidence based medical decision support system, at a first stage, for the Cardiocentro Ticino (CCT)
- evaluating medical data, and consequently, decisions concerning cardiological interventions in the past, in a series of doctoral studies at the Ospedale Civico di Lugano and Triemlis Hospital Zurich,
- triggering a pilot project with the participation of 4-5 clinical institutions in Switzerland in order to establish an evidence-based decision support system for hysterectomy,
- gathering and evaluating data referring to infections during hospitalization in order to increase quality of medical services in hospitals,
- gathering and querying observers’ data referring to snow and weather related physical parameters for the anticipation, warning and study purposes of avalanches in the Swiss Alps,
- querying data originated from cleaning actions of fields of scattered mines in countries involved in war.
References to the theoretical as well practical aspects of the query methodology are given in [KNS98, Kap98, KNB99, KNF99, SKF+99, KNFS00, SKM+00, SFK+00, Kap00].

4. STRUCTURE OF THE THESIS

This thesis has been structured as follows. Chapter 2 presents the theoretical background underlying the problem solution. Since embedding of meaning in expressed queries is one of the central issues of the querying methodology, the focus has been put on the syntactic and semantic issues characterizing languages as syntactic objects, particularly, query languages. Consideration of semantic theories sheds light into the role of semantics as a means of representation of meaning in order to understand a syntactic structure. Furthermore, since represented meaning is a driving mechanism of query construction, a brief overview of the theory underlying the representational model of meaning in terms of binary relations, graphs and propositional logic for expressing preconditions is given.

Chapter 3 deals with the related work referring to those approaches which try to alleviate the end-user task of querying data without confrontation with the implementation model peculiarities such as database schema and data interpretation as well as learning of database specific query languages. The approaches taken so far vary from browsing of meta-data or ontology bases to advanced human-computer interaction and natural language interfaces to databases. The major part of the related work refers to currently available query languages as querying interfaces of databases, since their relation to meaning, as being syntactic objects, is examined.

Chapter 4 provides paradigms of system-guided, meaning-driven query construction sessions which are characterized by different degrees of complexity. What they all have in common is the meaning-based query construction philosophy which relies on system guidance of the end-user through semantically consistent query states. This system-guided, incremental query specification or refinement is depicted in all examples given in this chapter. In order to illustrate the methodology, examples from query construction sessions from several application domains, particularly scientific applications, where the querying methodology is currently applied and/or used as a test bed are given.

Chapter 5 describes the model for the representation of meaning of the querying terminology. It includes both the domain and the operational terms. The model also provides the expression of exceptions in terms of preconditions as a major part of expressing meaning in terms of constraints and/or context. Preconditions characterize not only the consideration of particular domain terms within a constructed query as a syntactic object but also that of operation terms. Therefore, addressing of operations within a constructed query is bound to the meaning of domain terms on which they apply.
Chapter 6 covers the formal specification of the meaning driven querying methodology in terms of a *Meaning Driven Data Querying Automaton* MDDQA underlying the query construction mechanism. Since moving from one given query state to another one can be formally specified in terms of a finite state automaton which makes use of the meaning of terms constituting a query as well as the end-user reactions to system suggestions, the specification of a *Meaning Driven Data Query Language* MDDQL has been given by having the MDDQA as a syntactic and semantic parsing mechanism of the language.

The state transition algorithms underlying the automaton are also described in this chapter. Note that, given a particular query state (context), the state transition is partly decided upon the represented meaning of terms, covered by chapter 5, and partly upon the end-user choices. Therefore, the system end-user/machine is the machine realizing the automaton rather than the machine alone.

Chapter 7 illustrates the expressiveness and computability of MDDQL, the query language accepted by the *Meaning Driven Data Querying Automaton* MDDQA. The expressiveness is formulated in terms of a set of production rules, since MDDQL falls into the category of generative language. These rules do not intend to give a complete grammar of the language, since the semantic contents cannot be covered by the rules. They give only an overview of what can be formulated as potential query in MDDQL. Moreover, it is shown that given this specification there are always some algorithms which guarantee the computability of acceptance of any formulated query.

Chapter 8 gives an overview of the query answering system *SAQAS - Semantically Advanced Query Answering System* - architecture, the development of which is driven by the implementation of the automaton in both terms of representation model of meaning and of the inference engine for the query construction guidance. Both are positioned at the middle layer of the query answering system designed and implemented as a client-sided, database supported Web application relying on the basis of a three-tier system architecture.

Finally, chapter 9 gives a summary of the main concepts and the recommended problem solution as well as *lessons learned* from a system development and usability point of view. This chapter also covers the operational part of the system developed so far and gives an overview of the future work to be done.
Chapter 2

THEORETICAL BACKGROUND

Alice: I thought this was a theory book.

Vittorio: Yes, but good theory needs the big picture.

Sergio: Besides, what will you tell your grandfather when he asks what you study?

Riccardo: You can’t tell him that you’re studying the fundamental implications of genericity in database queries.


Since query languages is a means of communication between information systems and humans, all three levels, syntax, semantics and pragmatics, need to be addressed in order to provide an intuitive and human oriented way of dealing with information. Of all three levels, syntax is the best understood. Semantics and Pragmatics are still considered a critical issue when a communication process between humans and information systems is going to be established.

In particular, when (data analysis) queries must be posed to information systems dealing with data underlying definitions and/or intensional meaning, semantics and/or pragmatics become crucial in order to avoid meaningless queries, and therefore, increase the overall usability and throughput of the system. Since semantics is oriented towards the intensional meaning itself, posing queries relying on pragmatics, in terms of how the basic meaning is related to the current context and the user’s expectations, has been addressed in this thesis.

In the following, a brief overview is given about some theoretical tools and results that are used in this thesis. In particular, section 1. deals with the basics about binary relations, graphs and propositional logic. Section 2. deals with syntax and (query) languages. Subsequently, the reader is guided through semantic theories (section 4.) dealing with meaning and interpretation in (natural) languages as known in the field of computational linguistics.
1. GRAPHS, BINARY RELATIONS AND PROPOSITIONAL LOGIC

1.1 GRAPHS AND BINARY RELATIONS

A (directed) graph is a pair \( G = (V, E) \), where \( V \) is a finite set of vertices and \( E \subseteq V \times V \) the set of edges. A directed path in \( G \) is a nonempty sequence \( p = (v_0, \ldots, v_n) \) of vertices such that \( (v_i, v_{i+1}) \in E \) for each \( i \in [0, n - 1] \). A path from \( v_0 \) to \( v_n \) has length \( n \). An undirected path in \( G \) is an nonempty sequence \( p = (v_0, \ldots, v_n) \) such that \( (v_i, v_{i+1}) \in E \) or \( (v_{i+1}, v_i) \in E \) for each \( i \in [0, n - 1] \). A directed or undirected path is proper, if \( v_i \neq v_j \) for each \( i \neq j \).

A directed or undirected cycle is a directed or undirected path \( (v_0, \ldots, v_n) \) such that \( v_n = v_0 \) and \( n > 0 \). A directed cycle is proper if \( (v_0, \ldots, v_{n-1}) \) is a proper path. An undirected cycle is proper if \( (v_0, \ldots, v_{n-1}) \) is a proper path and \( n > 2 \). If \( G \) has a cycle from \( v \), then \( G \) has a proper cycle from \( v \). A graph \( G = (V, E) \) is acyclic if it has no cycles or, equivalently, the transitive closure of \( E \) is irreflexive.

The latter states that \( E \) can be viewed as a binary relation over a finite set \( V \). Consequently, any binary relation defined over a finite set \( Z \) can be viewed as a graph. For instance, for any finite set \( Z \) the graph denoted by \( P^{fin}(Z), \subseteq \) is acyclic, where \( (\subseteq) \) is the binary relation inclusion defined over the finitary set of all subsets of \( Z \).

Generalizing, a binary relation over a (finite or infinite) set \( S \) is a subset \( R \) of \( S \times S \), the cross-product of \( S \) with itself. We sometimes write \( R(x, y) \) or \( xRy \) to denote that \( (x, y) \in R \). In case of n-ary relations over \( S, R \subseteq S^n \), where \( S^n \) the cross-product of \( S \) with itself \( n \) times. A binary relation might be assigned one or more of the following properties:

- \( R \) is reflexive if it holds that \( (x, x) \in R \) for each \( x \in S \),
- \( R \) is irreflexive if it holds that \( (x, x) \notin R \) for each \( x \in S \),
- \( R \) is symmetric if \( (x, y) \in R \) implies that \( (y, x) \in R \) for each \( x, y \in S \),
- \( R \) is antisymmetric if \( (y, x) \notin R \) whenever \( x \neq y \) and \( (x, y) \in R \),
- \( R \) is transitive if \( (x, y) \in R \) and \( (y, z) \in R \) implies that \( (x, z) \in R \) for each \( x, y, z \in S \).

A binary relation \( R \) that is reflexive, symmetric and transitive is called an equivalence relation. A binary relation \( R \) that is reflexive, antisymmetric and transitive is called a partial order of \( S \). A total order is a partial order \( R \) over \( S \) such that for each \( x, y \in S \), either \( (x, y) \in R \) or \( (y, x) \in R \). For instance, the relation \( \subseteq \) over \( P(Z) \) is a partially ordered set when the cardinality of \( Z \) is greater than 1.
1.2 PROPOSITIONAL LOGIC

The field of mathematical logic is one of the important foundations for database theory. It serves as
the basics for query languages. The reader is referred to [Min88] for a comprehensive introduction
to mathematical logic. However, we briefly review the basic notions of propositional logic or
 calculus required for this thesis.

In propositional calculus, we assume an infinite set of propositional variables, typically denoted
 p, q, r, ..., possibly with subscripts. The special propositional constants true and false are also
 permitted. Well-formed propositional formulas are constructed from the propositional variables
 and constants using the unary connective negation (¬) and the binary connectives disjunction (V),
 conjunction (∨), implication (→) and equivalence (↔).

A truth assignment for a set \( V \) of propositional variables is a function \( \xi : V \rightarrow \{ \text{true}, \text{false} \} \). The
 truth value \( \phi[\xi] \) of a propositional formula \( \phi \) under truth assignment \( \xi \) for the variables occurring
 in \( \phi \) is defined by induction on the structure of \( \phi \) in the natural manner. Therefore, we get,

- \( \text{true}[\xi] = \text{true} \);
- if \( \phi = p \) for some variable \( p \), then \( \phi[\xi] = \xi(p) \);
- if \( \phi = (\neg \psi) \) then \( \phi[\xi] = \text{true} \) iff \( \psi[\xi] = \text{false} \);
- \( (\psi_1 \lor \psi_2)[\xi] = \text{true} \) iff at least one of \( \psi_1[\xi] = \text{true} \) or \( \psi_2[\xi] = \text{true} \).

A formula \( \phi \) is satisfiable if there is at least one truth assignment that makes it true, unsatisfiable
 otherwise. Moreover, it is valid if each truth assignment for the variables in \( \phi \) makes it true.
 For instance, the formula \( (p \lor q) \) is satisfiable but not valid, whereas the formula \( (p \land (\neg p)) \) is
 unsatisfiable and the formula \( (p \lor (\neg p)) \) is valid.

A literal is a formula of the form \( p \lor \neg p \), where \( p \) is a propositional variable. A propositional
 formula is in conjunctive normal form (CNF), if it has the form \( \psi_1 \land ... \land \psi_n \), where each formula
 \( \psi_i \) is a disjunction of literals. Disjunctive normal form (DNF) is defined analogously. It is known
 that if \( \phi \) is a propositional formula, then there is some formula \( \psi \) equivalent to \( \phi \) that is in CNF
 (respectively DNF).

We also say that formula \( \phi \) logically implies formula \( \psi \) or \( \psi \) is logical consequence of \( \phi \), denoted
 \( \phi \vdash \psi \), if for each truth assignment \( \xi \), if \( \phi[\xi] \) is true, then \( \psi[\xi] \) is true. Formulas \( \phi, \psi \) are logically
 equivalent, denoted \( \phi \equiv \psi \), if \( \phi \vdash \psi \) and \( \psi \vdash \phi \).
2. FORMAL LANGUAGE SYNTAX

Since database specific query languages underlie a formal syntax specification, we will embark, in the following, on the theoretical considerations of language syntax formalisms. This is meant to show that languages and their formal specification are mostly observed from a syntactical point of view only.

Formal languages and computability are part of the foundations of theoretical computer science. A general reference of this area can be found in [HU79, LP81]. In particular, we define a language over a finite set $\Sigma$ called alphabet to be a subset of $\Sigma^*$, where $\Sigma^*$ it the set of all words which can be constructed out of the elements of $\Sigma$. For example, if $\Sigma = \{a, b\}$, then $\{a^n b^n | n \geq 0\}$ is a language over $\Sigma$. A word over alphabet $\Sigma$ is a finite sequence $a_1, \ldots, a_n$, where $a_i \in \Sigma$, $1 \leq i \leq n$, $n \geq 0$.

An important type of computation over words is acceptance. The objective is to accept precisely the words that belong to some language of interest. Specifications of languages rely on the definitions of different kinds of acceptors. The simplest form of an acceptor is a finite-state automaton (fsa). An fsa processes words by scanning the word and remembering only a bounded amount of information about what has already been scanned. This can be formalized by computation allowing a finite set of states and transitions among the states, driven by the input. In other words, an fsa $M$ over alphabet $\Sigma$ is a 5-tuple $\{S, \Sigma, \delta, s_0, F\}$, where

- $S$ is a finite set of states;
- $\Sigma$ is an alphabet called the input alphabet;
- $\delta$, the transition function, is a mapping $S \times \Sigma \rightarrow S$;
- $s_0$ is a particular state of $S$ called the start state;
- $F \subseteq S$ is the set of accepting or final states.

Given an input word $\omega = a_1, \ldots, a_n$, an fsa reads one symbol at a time from left to right. This can be visualized as a tape on which the input word is written and the fsa with a head that reads symbols from the tape one at a time. The fsa starts in state $s_0$. A move from state $s_i$ to a state $s_{i+1}$, i.e. $\delta(s_i, a)$, is done by reading the current symbol $a$ and moving the head to the next symbol on the right. If the fsa is in an accepting state after the last symbol in $\omega$ has been read, $\omega$ is accepted. We also say that the fsa has a finite control. The language accepted by the fsa $M$ is denoted $L(M)$. 
2.1 REGULAR LANGUAGES AND EXPRESSIONS

A language accepted by some \(f\sa\) is called a regular language. Not all languages are regular, since for some languages there might be no \(f\sa\) accepting them. For instance, there is no \(f\sa\) which can be specified for the language \(\{a^n b^n | n \geq 0\}\), since no \(f\sa\) can remember the number of \(a\)'s scanned in order to compare it to the number of \(b\)'s due to the boundedness of the memory.

An alternative to specify regular languages is by using so-called regular expressions [MY60]. Such an expression over \(\Sigma\) is written using the symbols in \(\Sigma\) and the operations concatenation, union and repeat. This alternative is more convenient, if the alphabet is small. However, one of the most useful features of regular languages is that they have a dual characterization, one using \(f\sa\) and another using regular expressions. Indeed, Kleene's theorem [Kle56] says that a language \(L\) is regular iff it can be specified by a regular expression.

There are two important variations of \(f\sa\) that do not change their accepting power. The first allows scanning the input back and forth any number of times, yielding two-way automata [RS59, She59]. The second is non-determinism. A non-deterministic \(f\sa\) allows several possible next states in a given move, i.e. the transition function \(\delta\) takes the form \(S \times \Sigma \rightarrow P(S)\), where \(P(S)\) is the power set of the set of states \(S\). Thus several computations are possible on a given input and a word is accepted, if there is at least one computation that ends in an accepting state.

Non-deterministic \(f\sa\) (n\(f\sa\)) accept the same set of languages as \(f\sa\). However, the number of states in the equivalent deterministic \(f\sa\) may be exponential in the number of states in a non-deterministic one. Therefore, non-determinism can be viewed as a convenience allowing much more succinct specification of some regular languages. The equivalence between \(f\sa\) and n\(f\sa\) has been proved by Rabin and Scott [RS59].

2.2 TURING MACHINES

A Turing Machine (TM) can be defined as an acceptor of a language \(L(TM)\) [Tur36, Dav58], if we change the functionality of the head of a \(f\sa\) in that not only symbols are being read but also written - overwriting of symbols with elements of the given alphabet, when the head moves in both directions, and when is given that the amount of tape is infinite. Formally, a Turing machine is denoted \(M = (S, \Sigma, \Gamma, \delta, s_0, B, F)\), where

- \(S\) is a finite set of states,
- \(\Gamma\) is the finite set of allowable tape symbols,
- \(B \in \Gamma\) is the blank symbol,
A MEANING DRIVEN QUERYING METHODOLOGY

- \( \Sigma \subset \Gamma \) not including \( B \) is the set of input symbols.
- \( \delta \) is the next move function, a mapping \( S \times \Gamma \rightarrow S \times \Gamma \times \{L, R\} \), with \( \delta \) maybe undefined for some arguments and \( L \) is the move of the head to the left, \( R \) is the move of the head to the right.
- \( s_0 \) is a particular state of \( S \) called the initial state.
- \( F \subset S \) is the set of accepting or final states.

TMs can also be viewed as generators of words rather than simple acceptors. Typically, this is a non-terminating computation generating an infinite language. The set of words generated by a TM is denoted \( G(TM) \). A language that can be accepted by a Turing machine is said to be recursively enumerable (r.e.). In other words, if \( L(M) \) is such a language, then any Turing machine recognizing \( L(M) \) must fail to halt on some input not in \( L(M) \). As long as \( M \) is running on some input \( \omega \), we can never tell whether \( M \) will eventually accept \( \omega \) if we let it run long enough or whether \( M \) runs for ever.

Finally, TMs are viewed as computing a function from input to output and, therefore, TMs provide the classical formalization of computation and they are also used to develop classical complexity theory. Thus a function \( f \) from \( \Sigma^* \) to \( \Sigma^* \) is computable iff there exists some TM computing it. Church’s thesis states that any function computable by some reasonable computing device is also computable by a turing machine. Therefore, the definition of computability by TMs is robust. Variations such as non-deterministic TM and allowing multiple tapes make no changes to the accepting power.

2.3 CONTEXT FREEDOM AND SENSITIVITY IN FORMAL LANGUAGE SYNTAX

A further specification mode of languages is provided by a different approach to \( \text{f}sa \) and \( \text{T}M \), where the generation of words is emphasized rather than acceptance, despite the fact that this can be turned into an accepting mechanism by parsing. The major representative of this approach is context-free grammars (CFG) [Cho56]. It is defined as a 4-tuple \( \{N, \Sigma, S, P\} \), where

- \( N \) is a finite set of non-terminal symbols;
- \( \Sigma \) is a finite alphabet of terminal symbols, disjoint from \( N \);
- \( S \) is a distinguished symbol of \( N \), called the start symbol;
- \( P \) is a finite set of productions of the form \( \xi \rightarrow \omega \), where \( \xi \in N \) and \( \omega \in (N \cup \Sigma)^* \).
Thus a CFG $G = \{N, \Sigma, S, P\}$ defines a language $L(G)$ consisting of all words in $\Sigma^*$ that can be derived from $S$ by repeated applications of the productions $P$. An application of the production $\xi \rightarrow \omega$ to a word $v$ containing $\xi$ consists of replacing one occurrence of $\xi$ by $\omega$. The specification power of CFGs lies between that of fsa's and of TM's. All regular languages are context-free and all context-free languages are recursive. The opposite does not always hold. For example, $\{a^n b^n | n \geq 0\}$ is context-free but not regular, whereas $\{a^n b^n c^n | n \geq 0\}$ is recursive but not context-free.

Similar to regular expressions which have an equivalent automaton, i.e. the finite automaton, context-free grammars have their machine counterpart, the push-down automaton (PDA). The equivalence, in this case, is less satisfactory, since the PDA is a non-deterministic device and the deterministic version accepts only a subset of all context-free languages (CFL's). The PDA is actually a finite state automaton with control of both an input tape, from which the input symbols are read, and a stack or "first-in-last-out" list, where symbols, from some alphabet possibly disjoint from the input alphabet, may be entered or removed from the top of the list. Depending on the input symbol, the top symbol on the stack and the state of the finite control, a number of choices are possible.

Formally speaking, the PDA $M$ can be defined as a system $\{S, \Sigma, \Gamma, \delta, s_0, Z_0, F\}$, where

- $S$ is a finite set of states;
- $\Sigma$ is an alphabet called the input alphabet;
- $\Gamma$ is an alphabet called the stack alphabet;
- $\delta$, the transition function, is a mapping $S \times (\Sigma \cup \{\epsilon\} \times \Gamma \rightarrow S \times \Gamma^*$;
- $s_0$ is a particular state of $S$ called the initial state;
- $Z_0 \in \Gamma$ is a particular stack symbol called the start symbol;
- $F \subseteq S$ is the set of accepting or final states.

Placing restrictions on productions $\xi \rightarrow \omega$ of a phrase structure grammar, we receive a grammar which is called context-sensitive and its language context-sensitive language (CSL). For instance, a production rule of the form $a_1 A a_2 \rightarrow a_1 \beta a_2$ with $\beta \neq \epsilon$ (where $\epsilon$ is the empty set) looks almost like a context-free production rule, but it permits replacement of $A$ only in the context $a_1 A a_2$. The counterpart automaton characterizing CSL is a linear bounded automaton (LBA).

Finally, the four classes of recursive enumerable (r.e.) languages, CSL, CFL and regular are characterized as languages of type 0,1,2,3, respectively. It has been shown that except for the empty string, the type-$i$ languages properly include the type-$(i + 1)$ languages, for $i = 0, 1, 2$. This means that a CFL is equivalent to a CSL, but not every CSL is equivalent to a CFL. The same
relation holds between CFL's and regular languages, as discussed previously, and between \( r.e. \) and CSL. This is due to the hierarchy theorem and Chomsky's hierarchy which defines these classes of languages as the only potential models of natural languages.

3. QUERY LANGUAGES

A query language is a (sub)language which is a prerequisite for a syntax formalism. As such, an automaton \( M \) can be specified which underlies the query language \( QL(M) \) to be accepted by \( M \) over a finite set \( S_q \) as input alphabet. Elements of the alphabet are set-theoretic operations such as union, difference, intersection, inclusion as well as functions mapping a particular domain of interest into subsets of it. Since query languages are thought of as generally providing access to stored data, there are some primary differences between the general field of mathematical logic and specializations made in database theory. The most obvious specialization is the focus on the use of functions on data values. Two other specializations are the focus on finite models and the special use of constant symbols.

Two major families of query languages have been developed [AHV95]: a) those dealing with instances which are given on the basis of the specification of their properties, b) those adopting a navigation style. The first family consists of query languages based on three evaluation methods

- on algebraic operations, such as relational or collection algebra with operators \( \sigma, \pi, \times, \bowtie \), etc.,
- on logic with an expressive power equivalent to the algebra such as relational calculus, and
- those based on logic programming such as datalog which can be viewed as logic programming without function symbols.

They are all set-at-a-time oriented, in the sense that they focus on identifying and uniformly manipulating sets of instances rather than identifying instances individually and using loops to manipulate groups of instances. However, since instances in the answer are specified by properties they satisfy, with no reference to the algorithm producing them, relational calculus and datalog are conceptually declarative, whereas algebra-based languages are conceptually procedural.

In such a formal framework, we must distinguish between a query as a syntactic object and a query mapping which is a function defined by a query interpreted under specified semantics. For example, in the relational model, query mappings generally have as domain the family of all instances of a specified relation or database schema called the input schema and as range the family of instances of an output schema. We generally say that a query (mapping) is from or over an input schema to its output schema, and, consequently, two queries \( q_1, q_2 \) are equivalent \( q_1 \equiv q_2 \), if they have the same output schema and \( q_1(I) = q_2(I) \) for each instance \( I \) over \( R \).
In the following, we will mainly concentrate on the queries as *syntactic objects* rather than as query mappings. This is due to the fact that we emphasize the role of data meaning and interpretation when queries are being constructed. Meaning and interpretation mainly refers to attributes and values as elements of a query. However, meaning and interpretation based assignment of (analytical) operations to queries have also been addressed. For this reason, we assume that a countably finite set $\text{att}$ of attributes is fixed. We also assume that a countably finite set $\text{dom}$ disjoint from $\text{att}$, called the *domain* also exists, with a *constant* representing a value being an element of $\text{dom}$. When different attributes should have distinct domains, we assume a mapping $\text{Dom}$ on $\text{att}$ such that $\text{Dom}(A)$ is a set called the domain of attribute $A$.

Since semantics of query languages are conceived as meaning and interpretation of query elements as well as queries themselves based on some natural language, we embark, in the following section, on some semantic theories as known from the field of computational linguistics and artificial intelligence.

### 4. THEORIES ON SEMANTICS

The term *semantics* stands for *meaning* or the *study of meaning*. The meaning of a natural language word or sentence is the entity or action it denotes. In computer understanding of language, *semantics* is the process of determining the meaning of the input. Traditionally, this presupposes that one has, first, a computational representation for meaning, and, second, a method for deriving the representation for a given input.

The nature of meaning and the means for its representation have been topics of interest in *linguistics* and the *philosophy of language*. Scholars in the field of *artificial intelligence* developed *semantic theories* which are more or less influenced by scholars of *philosophy of language*. One particular concern in all semantic theories is *compositionality*. The principle of compositionality is that the meaning of the whole sentence is *some systematic function of the meaning of its components*. This is intuitively reasonable, but there are two alternatives: a) the meaning of the whole is a systematic function of not only the meaning of the parts but also of the *context and situation* in which it is formulated - ideally a semantic theory should be able to account for this as well, b) the meaning of the whole is a *non-systematic* function of the meaning or form of the parts.

Another major debating subject has been the separation of *syntactic* and *semantic* parsing processes. Usually, systems dealing with simple input [Wei66] do not consider separation between *syntactic* and *semantic* parsing processes. It is clear, however, that *syntactic knowledge* interacts with *semantic knowledge* - semantic analysis requires information about input structure but in order to provide this, the parser requires information about the meaning of the constituting parts.
This is because many sentences in natural language are structurally ambiguous. A complete separation of the two processes has been experienced in the system LUNAR [Woo73], where the disadvantage was that the semantic interpreter was forced to make a decision on semantic well-formedness without knowing what alternatives there are (possibly the second best).

However, in all semantic theories, the major focus has been the semantic interpretation during or after syntactic parsing of a sentence or question and not how to exploit represented knowledge (meaning) concerning a particular application domain in order to construct a meaningful query. Moreover, since different natural languages are supposed to be used for expressing the same query, providing syntactic or semantic parsing for each natural language is rather impractical or even impossible.

The semantic theories developed in the part focussed primarily on the means of representation of meaning in order to understand a syntactic structure. For example, decompositional semantics [Mar84] attempted to represent the meaning of each word by decomposing it into a set of semantic primitives. It turned out that such a lexical decomposition is problematic, since it is extremely difficult, even impossible, to find a suitable, linguistically universal collection of semantically primitive elements in which all words (of all languages) can be decomposed into their necessary properties. Furthermore, decompositional semantics is also problematic in its notion of how a sentence is represented; it is necessary to decide how such lexical representations must be combined into the representation of the whole sentence. Consequently, there must be corresponding methods to infer the context of the resultant structure.

Attempts to make decompositional semantics usable have been made by [Sch73, Sow84]. In particular, [Sch73] injected his primitives in the form of conceptual-dependencies into a structure that represented a sentence and then performed inference upon it. However, the set of conceptual-dependency primitives is incomplete and unable to capture particular nuances of meaning. [Sow84] proposed a system of conceptual graphs where a decompositional approach is combined with that of frames in order to gain advantage of both.

Montague semantics [Tho74] refers to truth-conditional and model-theoretic semantics, where the first is related to the meaning of a sentence as a set of necessary and sufficient conditions for the sentence to be true, i.e. to correspond to a state of affairs in the world. Model-theoretic means that the theory uses a formal mathematical model of the world in order to set up relationships between linguistic elements and their meanings. Montague employs not just one model of the world but rather a set of possible worlds. The truth of a sentence is then relative to a chosen possible world at a particular time, where a world-time pair constitutes an index.

However, Montague’s system contains a set of syntactic rules and a set of semantic rules in one-to-one correspondence. Semantic interpretation is done in terms of semantic objects such as individuals in (the model) of the world, individual concepts, properties of individual concepts and
higher-order functions. The meaning of a sentence is a truth condition relative to a world-time index.

Situation semantics [BP83] attempts to formalise the idea of a situation in the real world as a suitable semantic object. They see linguistic semantics as just a special case of meaning in the world. The viewpoint taken is that truth-conditional semantics is inadequate because the content of the sentence is lost, since only truth values in a possible world are of relevance. In procedural semantics [JWS81], production rules translate the parsed input into procedure calls that operate upon the database. The rules have been very specific and very powerful. They are triggered by looking at specific words, and therefore, the whole system is not readily adaptable.

However, the meaning of a sentence is the procedure into which the sentence is compiled, either in the computer or in the mind. The procedure itself can be seen as the intension of the sentence, the concept or idea behind it, and the result of the execution as the extension, the particular entity denoted. The manipulated items are still uninterpreted symbols.

Knowledge-based semantics [BS85, Win77] has been an approach in order to avoid many of the problems of decompositional and procedural semantics. Frames based representation of meaning has been used in order to resolve ambiguities of structured input when a sentence is being parsed. Therefore, syntactic and semantic analysis work very closely together. However, knowledge-based semantics is more powerful for full services of a retrieval or inference system. It is thus more powerful but also less self-contained; it is more suited in a large, general AI system and less in a natural-language oriented interface to a database.

In 1957 an event occurred that not only revolutionized the world of linguistics but left a lasting impression on philosophy, psychology and other areas. It was the publication of a short monograph by Noam Chomsky entitled Syntactic Structures [Cho57]. This monograph explored the implications of automata theory for natural languages. Noam Chomsky argued that the sentences of a natural language cannot be meaningfully generated by a finite-state machine or by any context-free grammar, or at least that "...any grammar that can be constructed, will be extremely complex, ad hoc, and unrevealing." He then proposed a theory of Transformational Grammar (TG) [Cho57, Cho65].

At the most abstract level, the theory of TG involves specifying a set of "kernel" sentences of a language; an assortment of "transformations" such as verb tenses and passive voice as well as the ordering in which transformations should be carried out. Despite the fact that TG has been proposed due to its efficacy of a transformational component, Chomsky also recognized that TG would have to be "formulated properly in terms that must be developed in a full scale theory of transformations".

Although the TG had an impact on Computational Linguistics (CL), it centered mostly around matters of syntax. In the long term, the hypothesis of TG most significant for work in CL is that
an understanding of the syntax or structure of natural language sentences can be arrived at on a solely grammatical basis without considering the real world properties, e.g. meanings, of the terms being discussed. This notion, sometimes known as the "autonomy of syntax" continues to provide a useful division in categorizing current work in CL as the debate continues as to what interactions are desirable or necessary, between the structural (syntactic) and interpretive (semantic, pragmatic) components of a theory or implementation.
Chapter 3

RELATED WORK

Some kind of knowledge of logical forms, though with most people it is not explicit, is involved in all understanding of discourse. It is the business of philosophical logic to extract this knowledge from its concrete integument, and to render it explicit and pure.

—Bertrand Russell. Our Knowledge of the External World

Information of data interpretation and inference of meaning has always been the focus of research and interest in computational linguistics, semantic theories, question answering systems and other AI application areas. However, since this thesis is oriented towards the exploitation of meaning of terms in order to guide the end-user to a reasonable query, the sections in this chapter refer to the relevant literature and approaches taken so far in order to contribute to the problem of querying data through meaningful interpretations.

Section 1. refers to related work in the area of meta-data and knowledge bases as central issue in order to cope with, manage and provide interpretation and understanding of data. Section 2. refers to the currently available data query languages that are both instances description oriented and navigational style oriented. Section 3. deals with query answering systems as an attempt to provide answers to end-users through dialogues based on natural language. Section 4. refers to the related work in the area of user interfaces to database systems as an effort to alleviate the task of database querying through user friendly interfaces to database systems. Finally, section 5. refers to human-computer interaction issues addressing the problem of designing and analyzing human-computer interaction methods.

In particular, section 1. addresses the issue that existing approaches for representation and management of meta-data and knowledge are not primarily concerned with meaning driven generation of complex or analytical queries. They are rather conceived as additional repositories which can be queried or browsed in order to increase understandability. Working with the actual data is shifted to the level of syntax formalisms and implementation models.
On the other hand, section 2. addresses the statement that database query languages based on instances oriented descriptions are conceived as syntactic objects underlying syntax formalisms and are strongly bound to data implementation models without any consideration of semantics in terms of meaning representation. Navigational style oriented querying frees the end-user from syntax formalisms, but they are still bound to the underlying data implementation model and do not exploit the meaning of terms when a particular query is constructed. Additionally, in most cases they end up with complex user interfaces which are tedious to work with.

Moving to the other extreme of having end-users communicate with the systems in some kind of natural language-based interaction mode, query-answering systems (section 3.) add more complexity to the question parsing and understanding parsing process which makes multi-lingual-based querying a cumbersome task. Furthermore, semantics-based parsing of natural (sub)language still poses some challenges.

The more friendly and ergonomic user interfaces have been used as front-end to databases and information systems either lack the guidance facilities of the end-user towards a query (section 4.) by ending up with complex parsing techniques for natural language based interfaces, or are not suited for query construction (section 5.). If we treat the human-computer interaction of a query construction session as a state automaton, however, the constructed query can be simultaneously considered as an already parsed query where meaning has been taken into account.

1. FROM META-DATA TO KNOWLEDGE BASES

1.1 ABOUT META-DATA

Increasing understandability of data has always been the major concern of developers of information systems, especially when large and complex data repositories are at hand. Suggested solutions have been provided aiming at different user communities and purposes. They vary from models representing the contents of file directories and databases - to be examined in this section - to knowledge based (terminological) systems - to be examined in the subsequent section.

In particular, the term meta-data has been used extensively in the literature, in order to refer to schema information (meta-data), such as class definitions from object databases or relations describing other relations in relational databases, documents describing documents, etc.

Useful conceptualizations of meta-information range from relatively early and simple ideas, typified for example by research on meta-information for geographic information [MSea91], to more recent efforts to develop expressive meta-information structures that are capable of handling a wide variety
of services such as an analysis of the data described by the meta-data [KK95] and content based searching over a variety of media [JH94, CIT94].

This has resulted in partitions of the types of information that meta-data should represent. One fairly basic example of such a partition is given by [BR94, BK95b]. On the other hand, some researchers are promoting meta-information structures limited to a relatively small number of core elements [Wei95]. Concomitant with the increase in the functions and applications of meta-data is an increasing sophistication in views or models of meta-data and the types of information that meta-data should represent.

For example, Lopez and Saacks view meta-data as a theory about the underlying data sets. This theory would be useful in "predicting" the need for a particular data set [LS92]. Lopez and Saacks also describe meta-data as providing a knowledge model capturing the semantics of a particular domain. Similarly [Hea91] suggests that meta-data should provide a representation of various data and knowledge models in a distributed environment.

These extended conceptions of meta-information require more powerful languages for representing the meta-information. For example, Jarke, et al., [Jea95] discuss languages explicitly in terms of database programming and knowledge representation languages applied to managing meta-information. The InfoHarness system [SSB95] specifies a meta-information language that describes catalogs. Other research on algorithmic generation of language translations [CYFS95], languages for interoperability [HMR95], and defining semantics of expressions in other languages [SSR94], are also salient for the language view of meta-information that we take. Of particular interest to this paper is Tamir and Kandel's specification of languages as formal systems [TK95].

1.2 TERMINOLOGICAL AND KNOWLEDGE BASED SYSTEMS

On the other side, the term meta-data has been brought into a close relation to the term knowledge base [IM96, MKBC96, MPC96, TA96, KMHP96, BA92, CHS91] and ontology [SNC96, GG95, GCS94, ACHK93, LG90], which led to the development of terminological systems. These systems are mainly based on complex formalisms such as description logics which are used in order to automatically draw conclusions or inferences concerning the classification of particular terms given a set of already known terms.

A series of terminological or, in a broader sense, knowledge based system approaches have been proposed aiming at a common terminology or knowledge to be used for information access over more than one available data repository. In particular, terminological systems underly a complex formalism based on description logics and aim at answering questions such as 'what are the kinds of this' or more excitingly 'what can I say about this' and by no way address the role of meaning
in query languages or the system guiding the construction of queries as syntactic and semantic objects.

For example, one of the goals of GALEN [ZRS+95, RSNR95], as a research and development project funded by the European Commission as part of its Framework programmes, is a semantically valid model of clinical terminology, represented in a formal language (GRAIL), and associated with sophisticated support for different natural languages and conversion between different coding schemes. The main modules (concept, multi-lingual and code conversion) are integrated into a single multi-user, networked, software system, the GALEN Terminology Server (TeS). The TeS combines the functionality of the three modules to provide sophisticated but uniform terminology services to client applications. It embodies GALEN’s view of terminologies as dynamic functional systems, rather than the traditional view as static data files.

The problem of using a common query language as well as a common terminology has also been addressed within the framework of intelligent system integration such as SIMS [AKS96, AKL97]. It is an intermediate layer - a mediator - between information sources and human users or applications programs. Queries to SIMS are in a uniform language, independent of the distribution of information over sources, of the various query languages, the location of sources, etc. SIMS determines which data sources to use, how to obtain the desired information, how and where to temporarily store and manipulate data, and how to maintain an acceptable level of efficiency in performing its task.

Within SIMS, a model is constructed to describe the domain about which information is stored in the information sources, as well the structure and contents of the sources themselves. The domain model is a declarative description of the objects and activities possible in the application domain as seen by a typical user. For each information source the model indicates the data-model used, query language, network location, size estimates, etc., and describes the contents of its fields in relation to the domain model. The user formulates queries using terms from the domain, without needing to know about specific information sources. SIMS’ models of different information sources are independent, greatly easing the process of extending the system.

Furthermore, SIMS uses a planner to generate a sequence of queries to individual information sources that satisfy a user’s query. The planner first selects information sources to be used in answering a query. It then orders sub-queries to the appropriate information sources, selects the location for processing intermediate data, and determines which sub-queries can be executed in parallel. Change to information sources is handled by changing models only. The changes will be considered by the planner in producing future plans that utilize information from the modified sources. This greatly facilitates extensibility.

The focus of SIMS, however, is put on the role of meta-data information and/or semantics in order to generate information source specific queries and not of how to exploit the semantics in order to drive
the construction process of meaningful queries. The same argumentation and perspective of view also holds for a variety of other more or less application domain specific system implementations as presented in the following.

The objectives of [MSD+98] consist of building a conceptual interface by means of the Internet technology inside an enterprise Intranet and to propose a method to realize it. This method is based on the knowledge sources provided by the Unified Medical Language System (UMLS) project of the US National Library of Medicine. Experiments concern queries to three different information servers: PubMed, a Medline server of the NLM; Theriaque, a French database on drugs implemented in a Hospital Intranet; and a Web site dedicated to Internet resources in gastroenterology and nutrition, located at the Faculty of Medicine of Nice (France). Access to each of these servers is different according to the kind of information delivered and according to the technology used to query it.

[MMW98] is concerned with a Knowledge-Based Systems approach to solving this problem for clearly bounded situations, in which both the domain and the types of query are constrained. At the user interface, a dialogue is conducted in terms of concepts with which the user is familiar, and these are then mapped into appropriate database queries. To achieve this, a model for query decomposition and answer construction has been used. This model is based around the development of an Intensional Structure containing information necessary for the recapture of semantic information lost in the query decomposition process and required in the answer construction process. The model has been successfully implemented in combination with an embedded KBS, within a five-layer representation model.

The distributed Inter-operable Object Model (DIOM) [LC97] promotes an adaptive approach to inter-operation via intelligent mediation aimed at enhancing the robustness and scalability of the services provided for integrating and accessing heterogeneous information sources. DIOM’s main features include (1) the recursive construction and organization of information access through a network of application-specific mediators, (2) the explicit use of interface composition meta operations (such as specialization, generalization, aggregation, import and hide) to support the incremental design and construction of consumer’s domain query model, (3) the deferment of semantic heterogeneity resolution to the query result assembly time instead of before or at the time of query formulation, and (4) the systematic development of the query mediation framework and the procedure of each query processing step from query routing, query decomposition, parallel access planning, query translation to query result assembly.

In [DLP96], the MDBMS interface (or mediator interface) that describes a CIS could be different from the union of the local interfaces that describe each local database. In particular, the mediator interface may be defined by semantic knowledge that includes views over particular local databases, integrity constraints and knowledge about data replication in local databases. We present
A MEANING DRIVEN QUERYING METHODOLOGY

A methodology for query reformulation which is based on the uniform representation of all semantic knowledge in the form of integrity assertions and mapping rules. A reformulation algorithm exploits this semantic knowledge, and performs semantic rewriting based on pattern-matching, to obtain a query on the union of the local interfaces.

The approach [SR95] that has been taken to the design and implementation of a prototype high-level interface to Geographical Information Systems (GIS) is based on the functional style of programming. Functional languages appear to offer some important properties, for example: the strong and polymorphic typing and the ease with which new types can be defined; the ability to order the knowledge base within functions; and the facility to create functional hierarchies composed of compound or higher-order functions which allow high-level operations to be manipulated as units.

It is described how a functional solution to the problems can be represented by a command based approach and how this can be improved upon through the use of a graphical user interface with direct manipulation of objects or icons.

The objectives of the Knowledge-Based Multimedia Medical Distributed Database System (KMeD) [CCT95] are to: query medical multimedia distributed databases by both image content and alphanumeric content; model the temporal, spatial and evolutionary nature of medical objects; formulate queries using conceptual and imprecise medical terms and support cooperative processing; develop a domain-independent, high-level query language and a medical domain user interface to support KMeD functionality; and provide analysis and presentation methods for visualization of knowledge and data models. Using rules derived from application and domain knowledge, approximate and conceptual queries may be answered. These concepts are validated in a test-bed linked with radiology image databases. The joint research between the UCLA Computer Science Department and the School of Medicine illustrates that the prototype system is of direct interest to medical research and practice. The results of this research are extensible to other multimedia database applications.

A knowledge-based database assistant as an expert system designed to help novice users formulate correct and complete database queries has been presented in [WU93]. This paper describes a knowledge-based database assistant with advanced facilities such as (1) a menu-based query-making guidance, (2) a menu-based natural-language user-interface, and (3) a database-commands generator which formulates formal database queries with the SQL language. The system works as an intelligent front-end to an SQL database system or a computer-aided SQL tutorial-system.

In this paper, a semantic-network model, named S-Net, is discussed, which is used to represent the knowledge for formal database-query formulating processes. The menu-based English user-interface allows end-users to make a query by filling a certain query pattern with appropriate words. The query-pattern filling process is guided by pop-up menus provided by the system. The query-pattern instances thus obtained are then translated into formal database queries. The translation
Related work

is carried out by evaluating operations on S-Net knowledge-base which conveys knowledge about application domain, and the underlying database schema.

Furthermore, representation of meaning/knowledge by using graphs has become a central issue in AI supporting the interaction between people and machines [Fin79, Sow84]. They appeared in different forms such as conceptual graphs [Sow79], existential graphs, conceptual dependency graphs [SNW75], correlational nets [Cec61], and implemented as semantic nets [Sow91], partitioned nets, structured inheritance nets. There was not always a clear distinction between semantics and syntax on a graph, however, since most of these graphs were committed to transformational grammars. Furthermore, they all have been investigated within the area of linguistics, particularly, in parsing and understanding natural language and not as a model driving the query construction by exploitation of meaning of terms.

1.3 META-DATA STANDARDS AND MODELS

Standards and models, such as those described in the following, have been elaborated and proposed which more or less lead to increased understandability of data, processes, etc. These standards are conceived as an agreement upon documentation of data sources, conditions and processes in order to increase interoperability of systems and the exchange of data.

Directory Interchange Format (DIF): The objectives of this standard are to provide a common set of terminology and definitions for the documentation of digital geospatial data. The standard establishes the names of data elements and compound elements (groups of data elements) to be used for these purposes, the definitions of these compound elements and data elements, and information about the values that are to be provided for the data elements.

The standard was developed from the perspective of defining the information required by a prospective user to determine the availability of a set of geospatial data, the fitness of the set of geospatial data for an intended use, the means of accessing the set of geospatial data and to successfully transfer the set of geospatial data. The standard does not specify the means by which this information is organized in a computer system or in a data transfer, nor the means by which this information is transmitted, communicated, or presented to the user.

DIF is mainly used by those user communities which are interested in geo-spatially referenced data as provided by various data collection centres or satellites and reside on file-based data repositories such as the Global Change Master Directory (GCMD) [Ols].

Resource Description Framework (RDF) is a standard that was designed by the World Wide Web Consortium (W3C) to enable Web applications, which depend on machine-understandable meta-data, and to support interoperability between such applications. It targets a number of impor-
tant areas that include resource discovery, intelligent software agents, content rating, intellectual property rights and privacy preferences. RDF is used to create models of meta-data that may be understood by processing agents. It is complementary to XML, which is used to encode and transport RDF models. XML does not have an exclusive right on representing RDF models; other mechanisms may be used to serve the same purpose in the future.

**Knowledge Interchange Format (KIF)** is defined in the introduction of KIF [Ins98] as follows:

"a language designed for use in the interchange of knowledge among disparate computer systems (created by different programmers, at different times, in different languages, and so forth). KIF is not intended as a primary language for interaction with human users (though it can be used for this purpose). Different computer systems can interact with their users in whatever forms are most appropriate to their applications (for example Prolog, conceptual graphs, natural language, and so forth)."

KIF is also not intended as an internal representation of knowledge for computer systems or within closely related sets of computer systems (though the language can be used for this purpose as well). Typically, when a computer system reads a knowledge base in KIF, it converts the data into its own internal form (specialized pointer structures, arrays, etc.). All computation is done using these internal forms. When the computer system needs to communicate with another computer system, it maps its internal data structures into KIF.

The purpose of KIF is roughly analogous to that of Postscript. Postscript is commonly used by text and graphics formatting systems to send information about documents to printers. Although it is not as efficient as a specialized representation for documents and not as perspicuous as a specialized WYSIWYG (What You See Is What You Get) display, Postscript is a programmer-readable representation that facilitates the independent development of formatting programs and printers. While KIF is not as efficient as a specialized representation for knowledge nor as perspicuous as a specialized display (when printed in its list form), it too is a programmer-readable language and thereby facilitates the independent development of knowledge-manipulation programs.

**Knowledge Querying and Manipulation/Modelling Language (KQML)** is a language and protocol for exchanging information and knowledge [FMFM94]. It is part of a larger effort, the ARPA Knowledge Sharing Effort, which is aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving.

KQML focuses on an extensible set of performatives, which defines the permissible operations that agents may attempt on each other’s knowledge and goal stores. The performatives comprise a
Related work

Substrate on which to develop higher-level models of inter-agent interaction such as contract nets and negotiation. In addition, KQML provides a basic architecture for knowledge sharing through a special class of agents called communication facilitators which coordinate the interactions of other agents. The ideas which underlie the evolving design of KQML have been explored through experimental prototype systems which are being used to support several test-beds in such areas as concurrent engineering, intelligent design and intelligent planning and scheduling.

Open Knowledge Base Connectivity (OKBC) is an application programming interface for accessing knowledge bases stored in knowledge representation systems (KRSs) [CFF+98]. OKBC is being developed under the sponsorship of DARPA's High Performance Knowledge Base program (HPKB), where it is being used as an initial protocol for the integration of various technology components.

OKBC is a successor of Generic Frame Protocol (GFP) which was primarily aimed at systems that can be viewed as frame representation systems and was jointly developed by Artificial Intelligence Center of SRI International and Knowledge Systems Laboratory of Stanford University. OKBC provides a uniform model of KRSs based on a common conceptualization of classes, individuals, slots, facets, and inheritance. OKBC is defined in a programming language independent fashion, and has existing implementations in Common Lisp, Java, and C. The protocol transparently supports networked as well as direct access to KRSs and knowledge bases.

OKBC consists of a set of operations that provide a generic interface to underlying KRSs. This interface isolates an application from many of the idiosyncrasies of a specific KRS and enables the development of tools (e.g., graphical browsers, frame editors, analysis tools, inference tools) that operate on many KRSs. It has been successfully used in several ongoing projects at SRI and Stanford University.

XML-Based Ontology Exchange Language (XOL) is a language for ontology exchange [KCT99]. It is designed to provide a format for exchanging ontology definitions among a set of interested parties. The ontology definitions that XOL is designed to encode include both schema information (meta-data), such as class definitions from object databases – as well as non-schema information (ground facts) such as object definitions from object databases. XOL is similar to other past ontology-exchange languages; its development was inspired by Ontolingua and OML. XOL differs from Ontolingua in having an XML-based syntax rather than a Lisp-based syntax; the semantics of OKBC-Lite are similar to the semantics of Ontolingua. XOL differs from OML in that the semantics of OML are based on Conceptual Graphs, which differs from OKBC-Lite in several ways.

The syntax of XOL is based on XML, which is a language for authoring documents for the WWW. XML syntax was chosen because it is reasonably simple to parse, its syntax is well defined, it is human readable, it appears that XML will be very widely used, and also because it appears that
many already existing software tools for parsing and manipulating XML will soon be appearing. The semantics of XOL are based on OKBC-Lite, which is a simplified form of the knowledge model for the OKBC (see above).

The Synapses approach [Gea98] is to base the sharing of data (records) on a common data model, the Synapses Object Model. In essence, the model provides a set of abstract "building blocks" that can be used to construct the shared record. This can be achieved by an aggregation mechanism for the record in which the aggregation hierarchy contains the object classes in the model. In addition to their use in the exchange format, the classes form the basis for the Synapses Object Directory (SynOD). The SynOD is an active data dictionary/directory containing definitions and locations of sharable user-defined objects that conform to the rules and constructs of SynOD. The records thus constructed are indeed aggregations of information available from the underlying source information systems but are constrained aggregations in accordance with the requirements of the record architecture.

Summarizing, the representation of information concerning the structural properties of data has been attempted with various approaches which vary considerably in terms of complexity and usability. Despite the fact that they increase the understandability of data and processes, they are all specified with different goals in mind, and therefore, cannot be adopted without customization. Furthermore, since a major concern of the representation model of meaning have been the terminology constraints and the multi-lingual representation of terms, they are not suitable as representation models of the inference engine underlying the query construction mechanism as proposed in this thesis.

Even if queries are posed at a meta-data level [GL96, KST96, GBLP96, AGS95], they are based on assumptions which do not refer to the semantics of values and/or data analysis operations. Therefore, one still needs to make oneself familiar with an underlying query language syntax for a meta-data repository, or must work with browsing oriented or keywords-based searching techniques and not with a full query language. In the following, we will have a look at the nature of database query languages and their lack of consideration of terminological meaning.

2. DATABASE QUERYING APPROACHES

This section provides an overview of traditional and proposed query languages to be used as a means of addressing instances in a database through the specification of criteria and properties rather than using some browsing techniques as happens to be the case in meta-data databases or knowledge/ontology browsers. We mainly distinguish between instance description oriented query languages and navigation style oriented query languages.
2.1 INSTANCES DESCRIPTION ORIENTED QUERY LANGUAGES

This family of query languages [AHV95] deals with instances which are given on the basis of the specification of their properties. The family consists of query languages based 1) on algebraic operations, such as relational or collection algebra with operators $\sigma, \pi, \times, \bowtie$, etc., 2) on logic with an expressive power equivalent to the algebra itself such as relational calculus, and 3) those based on logic programming such as datalog which can be viewed as logic programming without function symbols.

They are all set-at-a-time oriented, in the sense that they focus on identifying and uniformly manipulating sets of instances rather than identifying instances individually and using loops to manipulate groups of instances. However, since instances in the answer are specified by properties they satisfy, with no reference to the algorithm producing them, relational calculus and datalog are conceptually declarative, whereas algebra-based languages are conceptually procedural.

Representative query languages of this family are SQL (Structured Query Language) [GW99], OQL (Object-oriented Query Language) or AQL [CB97, KKS92] for relational and object-oriented databases respectively, as well as the query languages WebSQL [MMM97], WebOQL [AM98], StruQL [FFLS97], Florid [HLLS97] which have been proposed as Web data query (manipulation) languages.

In particular, SQL/OQL-like query languages are suitable for well-structured data and presuppose a full understanding of the data model semantics. Furthermore, natural language based interpretation of model elements such as classes, relations, attributes and values as well as operational semantics are not taken into account during query formulation. Semantic consistency within a query is also not a matter of concern. Therefore, they mostly meet the requirements of application programmers but not those of end-users.

Similarly, the family of Web query (manipulation) languages, as suggested for querying Web data, allow access to the internal structure of Web pages from the query language, and therefore, enable a declarative style of query formulation. For example, the task of extracting a set of tuples from HTML pages requires parsing the HTML and selectively accessing certain subtrees in the parse tree. WebSQL [MMM97] is a representative query language for such purposes which conceives the Web as a single database with two virtual relations: Document and Anchor. WebOQL [AM98], StruQL [FFLS97] and Florid [HLLS97], as representatives of a second generation of Web query languages, go beyond structuring/restructuring of HTML pages in two significant ways: a) they provide access to the structure of the Web objects (modeling of internal structures and internal links of Web documents) that they manipulate, b) they provide the ability of creating new complex structures as a result of a query.
In particular, WebOQL manipulates sets of related hypertrees which are collected into Webs. Hypertrees are ordered arc-labeled trees with two types of arcs, internal and external. Arcs are labeled with records. Despite the fact that WebOQL is a functional language, queries are couched in the familiar select-from-where form. StruQL is a query language proposed for the STRUDEL Web site management system and is considered as a general purpose query language for semi-structured data. StruQL is based on a data model of labeled directed graphs. The query result is a graph in the same data model as the input graphs. StruQL has been mainly used for two tasks: querying heterogeneous sources to integrate them into a site data graph and for querying this data graph to produce a site graph. Finally, FLORID is a prototype implementation of the deductive and object-oriented formalism F-logic [KLW95].

Given the emergence of XML and XML-based Web documents, a generation of query languages such as XML-QL [Mai98], Lorel [AQM+97], XQL (XML Query Language) [RLS98] and XSL (eXtensible Stylesheet Language) [Gro98] appeared. XML-QL has been designed at AT&T Labs as part of the Strudel project. The language extends SQL with an explicit CONSTRUCT clause for building the document resulting from the query and uses the element patterns (patterns built on top of XML syntax) to match data in an XML document. XML-QL can express queries as well as transformations for integrating XML data from different sources. Lorel was originally designed for querying semi-structured data and has now been extended to XML data. It was conceived and implemented at Stanford University and has a user-friendly SQL/OQL style including a strong mechanism for type coercion and permits very powerful path expressions which are extremely useful when the structure of the documents is not known in advance.

XSL consists of a collection of template rules. Each template rule has two parts: a pattern which is matched against nodes in the source tree and a template which is instantiated to form part of the result tree. XSL makes use of the expression language, defined by XPath [Cla99], for selecting elements for processing, for conditional processing and for generating text. XQL can be considered a natural extension to the XSL pattern syntax. It has been designed with the goal of being syntactically very simple and compact (a query could be part of a URL) with a reduced expressive power.

All these query languages are too complex to be used directly by interactive users and they are meant to be used mostly as programming tools. Despite the fact that SQL/OQL-like query languages are suitable for well-structured data and are less complex than Web query languages, they still presuppose a full understanding of the data model semantics. Furthermore, semantics of query expressions in terms of natural language, multi-lingual based interpretation of elements of a query expression such as classes, relations, attributes and operations, as well as semantic constraints is not subject of matter. The latter also holds for the family of the Web query languages as presented in this section.
2.2 NAVIGATION STYLE ORIENTED QUERYING

In the following, we are dealing with the second major family of query languages which adopts a navigational style for the construction of a query or addressing a particular query result. They are classified into two main categories: a) those which are enriched with visual query interfaces or metaphors and b) those which allow path expression formalisms in a query expression.

2.2.1 VISUAL QUERY SYSTEMS AND LANGUAGES

Avoiding a query language formalism when end-users need to pose queries to a database system has received much attention during the last 10-15 years within the database research community [CCS97, DLP96, PK95, MK94, MRR94, GJC94, HGR94, Ber93, OW93, SKK92] and many Visual Query Systems (VQS) or Visual Query Languages (VQL) have been developed to alleviate the end-users' tasks. A survey of these approaches is given in [CCLB97]. A classification of VQSs based on the criteria of expressive power, usability and categories of potential users is given in [BCCL91].

Despite the fact that VQSs can be seen as an evolution of query languages adopted in database management systems in order to improve the effectiveness of Human-Computer Interaction (HCI) [CSW98], the notion of usability as defined in [BCCL91] was mainly specified in terms of the models used in VQSs for denoting both data and queries, their corresponding visual representations and the strategies provided by the system in order to formulate a query.

This definition [BCCL91] does not reflect the Human-Computer Interaction (HCI) view of usability as a software quality related to user perception and acceptability of the software system. In particular the data model has no significant impact on the user perception of the system. VQSs mainly rely on the integration of the data model and query language in a user-database interface [Cha97] as well as presentation and interaction components that together form a graphical user interface [MGP98].

A classification of VQSs based on the criteria of visual representation of queries and query results is given in [CCLB97]. VQSs are mainly classified into form-based, diagrammatic and iconic according to the representation of the domain of interest for query formulation or the query result. They all refer to information repositories dealing with alphanumeric data only, not with semi-structured, video or audio data.

In particular, form-based representation techniques became popular after the appearance of the first example based query language QBE (Query-by-Example) in 1977 [Zlo77]. More than a dozen such languages have been proposed and/or implemented [HP89, SSK89, Weg89, Eps91]. They are designed for different application domains such as statistical and scientific applications, office automation, historical databases and spatial databases and have different capabilities and expressive
powers. Most example-based languages are based on revised versions of Codd’s domain relational calculus.

A survey of example-based query languages is given in [OW93] which compares the features of 12 example-based languages in terms of (1) query specification and interpretation, (2) object manipulation, (3) query language constructs, and (4) query processing techniques. Example-based query languages allow users to specify queries through an example that is constructed graphically, thus utilizing the analogy between a semantically meaningful example and the query. These languages provided a user-friendly graphical interface, especially for relational databases.

At the end of the 80s and beginning of the 90s, we observe a shift of visual querying towards Entity-Relationship (ER) based diagrammatic representation of database schemata such as [EW81, ACS90, DAA+95]. They all rely on typical query operators such as selection of visual elements, traversal on adjacent element and creation of a bridge between disconnected elements. In some systems like [EW81, DAA+95], the navigation path is implicitly expressed by transforming the query schema into a tree whose nodes appear in the order selected in the query. Some systems also allow the users to define their own visualization rather than creating tailored displays of data. A proof of equivalence of navigations in Query By Example — QBD [ACS90] to relational algebra is given in [CS88].

In contrast with Entity-Relationship based diagrams, the iconic query language QBI (Query-By-Icon) [MPSC95] is not concerned with path expressions but it allows users to query and understand the content of a database by manipulating icons. It provides intensional browsing through meta-query tools that assist in the formulation of complete queries in an incremental manner by using icons. A comparison study between using ”diagrams” and using ”icons” for the formulation of queries is given in [BCMS96]. QBI is based on a slight variation of the Graph Model introduced in [CSA93].

Hybrid VQSs have also been proposed which offer an arbitrary combination of the above three visual formalisms, either offering the user various alternative representations of databases and queries, or combining different visual formalisms into a single representation. The major categories have been those VQSs which combine forms and diagrams [KM84], diagrams and icons [GN88], and form, diagrams and icons [CLF91].

Although it was reported that manipulating diagrams is definitely easier than writing SQL commands, it was first an Esprit project, VENUS, which started addressing the real ”meaning” of usability by observing users such as hospitals and research centers. The efficiency of processing queries in the VENUS environment is considered in [CCS97]. The goal of this system was to allow a user to query multiple data sources by interacting with a conceptually simple database, and by providing a query interface based on the use of visual formalisms. This system takes advantage of existing multi-database (MDB) query processing strategies, and deals with optimization issues
by considering the exploitation of inter- and intra-schema semantics. A methodology is presented
which harnesses knowledge of the set-theoretic relationships between classes participating in a
query, and demonstrates that significant savings can be made in MDB query processing.

Aiming at a different category of users, the Vista visual language [MRR94] has been proposed,
which can be easily used by statistical users when they interact with a statistical database in order
to manipulate data directly. This language uses a direct acyclic graph as an internal model for the
database scheme representation. The operators on the statistical data manipulate their descriptive
elements, while the statistical processing is performed through statistical packages. This visual
language permits querying in the database scheme and the selection of subschemas. Vista is
integrated in the statistical database management system Adams (Aggregate Data Management
System). This system allows also an interaction through a keyword-based language. The main
tasks performed by the Adams system are the graphical visualization of the database scheme, the
direct manipulation and the expression of query commands.

Besides VENUS, other visual query languages have also been proposed and applied to multi-
database query facilities. The subject of [MK94] was the architecture and design of a multi-database
query facility. These databases contain structured data, typical for business applications. Problems
addressed are: presenting a uniform interface for retrieving data from multiple databases, providing
autonomy for the component databases, and defining an architecture for semantic services. DIRECT
is a query facility for heterogeneous databases. The databases and their definitions can differ in their
data models, names, types and encoded values. Instead of creating a global schema, descriptions of
different databases are allowed to co-exist. DIRECT has been exercised with operational databases
that are part of an automated business system.

[GJC94] proposes the query language DFQL, which has been designed to mitigate SQL’s ease-
of-use problems. DFQL provides a graphical interface based on the data-flow paradigm in order
to allow a user to construct queries easily and incrementally for a relational database. DFQL
is relationally complete, maintains relational operational closure, and is designed to be easily
extensible by the end user. A prototype DFQL system has been implemented.

The query language Vizla [Ber93] builds up answers to queries by pointing to representations of
sets and functions in a conceptual model of the data base of an application, and to iconic identifiers
of computational operators or control constructs. The primary use of Vizla is in the validation of
conceptual models of information systems, but it is to be developed into a user interface to SF, a
prototyping language for information and control systems. Moreover, it can be regarded as a visual
programming language in its own right. As such it is based on abstract data types.

One of the major lessons learned, however, was the understanding that users had many difficulties
even if ER-based diagrammatic presentations of database schemas have been chosen as a query
interface - the well-known "wires and meshes" problem. Another major difficulty users had was
the combination of selection conditions on attributes using the boolean connectives AND, OR, NOT. Moreover, users locked up in an option. By then, developers suspected that being a database expert was not enough to design "interactive" information systems.

Furthermore, the conceptual or implementation model - mostly relational - is partially integrated into the query paradigm. Query construction takes place in terms of navigation through a graphical representation of the entire model and without any consideration of meaning in terms of semantic constraints holding among query elements, i.e. mutually exclusive properties or values, as well as their interpretations. Moreover, usage of different natural languages has not been the case.

Even if VQSs are characterized from the perspective of end-user/system interaction strategies where mainly top-down, browsing and schema simplification strategies are considered for understanding the domain of interest prior to formulation of a query, meaning is not taken into consideration as a predominant feature of the interaction strategy. Additionally, understanding of domain of interest and formulation of a query have been conceived as two different processes.

2.2.2 GRAPh BASED FORMALISMS AND TRAVERSAL PATHS/LINKS

Graph based formalisms have been proposed for both Object-Oriented DBMSs where traversal paths can be expressed as queries [YM98, CW97], and for Web query systems [CCD+99, WJ98]. In particular, [YM98, CW97] proposed querying object-oriented databases by following paths or links connecting objects to each other. [WJ98] proposes a visual user interface - WebIFQ (Web In-Frame-Query) - which assists users in specifying queries and visualising query criteria by including document meta-data, structures and linkage information.

XML-GL [CCD+99] is conceived as a graphical query interface to XML, playing the same role as graphical query interfaces such as Query-by-Example for the relational world presented above. However, since query formulation is done by means of labelled XML graphs, we classified XML-GL into the family of graph based formalisms. The language has been designed at Politecnico di Milano and the implementation is ongoing. The basic idea is the usage of a graphical representation of XML documents and DTDs (Document Type Definitions) by means of labelled graphs.

Traversal like approaches using graph queries or high level concepts also underly the development of query interfaces for large clinical databases [LCRM98, TJB96, GBJR96, BK95a], or for general purpose systems [CMB93, DPKaQ95, CYC+96, MGP98, MPG98, GGK99, ZCMK99]. The query generators mostly use an object-oriented data model or functional models such as in case of [GGK99].
Despite the fact that in all these approaches query construction is done by navigational issues where the end user does not need to learn a particular query language, it is often hard to operate on complex or large diagrammatic representations, which occurs when large database schemas are considered. Furthermore, semantic constraints within a query, which might lead to semantically incorrect queries, are not taken into account.

Even in cases such as [GGK99, ZCMK99] where data values are considered for the incremental formulation of the final query, these values are not addressed within well-restricted value domains and cannot be the subject of a semantically meaningful consideration of values for conditional statements given the current query context. Without such a meaningful consideration of values, queries might be constructed the results of which are, semantically speaking, not worth addressing and might lead to expensive operations without commensurate results.

In [GGK99], where values can be taken from intermediate results during query construction and further refined to form the final query, we are still faced with the problem of addressing values from pre-calculated results which, in the case of large databases, might exceed several hundreds of rows or tuples.

3. QUESTION ANSWERING SYSTEMS

Attempts to provide natural language based data retrieval have been undertaken in a series of question-answering systems. They mainly provide a parsing mechanism for accepting or rejecting questions formulated in English. However, none of these systems are concerned with multilingual questions nor do they free end-users from syntax formalisms, e.g., natural language or some sublanguage. In some cases, the semantics of terms are not taken into account at all.

In particular, BASEBALL [GWCL63] was designed to interact with a primitive database, stored as attribute-value pairs that contained information about the month, day, place, teams and scores for the American League baseball games. The system could answer questions posed in ordinary English about data and its storage. An example input is "What teams won 10 games in July?".

Another early program was SADSAM [Lin63] designed to parse sentences written in basic English and make inferences about "kinship relations". The system comprised two modules: one for parsing (the syntactic appraiser and diagrammer SAD) and one for semantic analysis (SAM). The basic operation of the semantic module involved searching a previously constructed parse tree for words denoting kinship relationships in order to construct a family tree stored as a linked structure.

SIR [Rap68] was a system which had the goal of "developing a computer program having certain cognitive abilities and exhibiting some human-like conversational behavior". The system was similar to SAD SAM in allowing a user to input new information, then ask questions about it.
However, SIR emphasized relations such as set-subset, part-whole, and ownership, as suggested by the following: Every boy is a person. A finger is part of a hand. Each person has two hands. John is a boy. Every hand has five fingers. How many fingers does John have?

The DEACON system [CBHL66] was designed to answer questions about "a simulated Army environment" and represents an important precursor of the database front-ends of the 1970s. Its internal ring-like data structure could be dynamically updated, thus enabling users to supply new information such as "The 425th will leave Ft. Lewis at 21950" as well as ask questions such as "Is the 638th scheduled to arrive at Ft. Lewis before the 425th leaves Ft. Lewis?”. Reflecting upon their experiences with DEACON, the authors noted that "perhaps the most significant new feature needed is the ability to define vocabulary terms in English, using previously defined terms. This realization led to the REL system and its successors.

REL [TLDD69] had as its primary goal the facility of implementing and, subsequently, the user-based extension and modification of highly idiosyncratic language/data base packages. From a theoretical standpoint, REL was based on the notion that an English language subset could be treated as a formal language "when the subject matter which it talks about is limited to material whose interrelationships are specifiable in a limited number of precisely structured categories". The first sizable application of REL was for an anthropological database at Caltech of over 100,000 items. Work on REL continued well into the 1970s until the system, now quite advanced over its early prototypes, was renamed ASK.

CONVERSE [Kel68] was intended to strike a reasonable compromise between the difficulties of allowing completely free use of ordinary English and the restrictions inherent in existing artificial languages for data base description and querying. An example input was "Which Pan Am flights that are economy class depart for O'Hare from the city of Los Angeles?”.

Until recently, the striking compromise is a matter of discussion. The problem is still recognized as "a large set of theories proposed in recent years, covering one or more aspects of natural language understanding, but few of them could have been integrated successfully into one working system”. Therefore, SAPFO [Pal93] is a paraphrasing and question-answering system, designed to experiment with language as a whole. Based on a psycholinguistic experiment with re-narration of stories, an experimental system for paraphrasing of Slovak has been built.

The paraphraser serves as a computer tool for linguistic analysis. It enables particular linguistic hypotheses or concepts to be tested and their behavior in mutual interaction with other subsystems of language and on large corpora to be observed. Finally, it is claimed that more attention should be paid to non-synonymical relationships in language. An algebra of so called synsemantic transformations enables semantically relevant constructions of quite different surface forms to be matched.
Despite these difficulties, however, question-answering systems have been designed for various question-answering problems. For example, a question answering system based on fuzzy logic [VTK94] provides the capability to assess whether a database contains information pertinent to a subject of interest. This is done by evaluating each annotation in the database via a fuzzy evaluator that attributes a fuzzy membership value indicating its relationship to such subject. An assessment is provided for the database as a whole regarding its pertinence to the subject of interest, and consequently comments that are considered irrelevant to the subject may be discarded.

Furthermore, a framework used to integrate the four major diverse CIM (computer integrated manufacturing) databases in the manufacturing environment is presented, namely, design, planning, manufacturing and accounting databases [WD92]. The framework consists of the existing four databases along with their database management systems (DBMSs), a knowledge-based system for each DB that integrates the local DBMS with the global system, a global manager that coordinates the entire integration effort, and a natural language (NL) user interface that handles global queries. From a firm's executive suite, the role of the NL user interface is critical for the successful application of the method. The NL user interface is composed of the DCG (definite clause grammar) parser, the semantic interpreter and the query generator.

Finally, resolving ambiguities in interpreting the user's utterances has been considered as one of the most fundamental problems in the development of a question-answering system. The process of disambiguating interpretations requires knowledge and inference functions on an objective task field. Therefore, a framework for understanding conversational language has been proposed [UKS+92], using the multi-paradigm knowledge representation ("frames" and "rules") which represents concept hierarchy and causal relationships for an objective field. Knowledge of the objective field is used in the process to interpret input sentences as a model for the objective world.

In interpreting sentences, a procedure judges preferences for interpretation candidates by identifying causal relationship with messages in the preceding context, where the causal relationship is used to supplement some shortage of information and to give either an affirmative or a negative explanation to the interpretation. The procedure has been implemented in an experimental question-answering system, whose current task is to allow user's consultation in operating an electronic device. The experimental results are shown for a concrete problem involving resolving anaphoric references, and characteristics of the knowledge processing system are discussed.

4. END-USER DATABASE INTERFACES

In the following, we consider end-user oriented interfaces to database systems as a means of increasing semantic content when a system is being interrogated. In particular, we consider graphical interfaces to databases which are more or less tied to the underlying database model with many
restrictions which make them specific to particular application domains, and therefore, they lack generality.

On the other side, natural language based interfaces have been introduced as a solution to the problem of learning and understanding particular or application specific data models and query languages. The introduced syntactic/semantic complexity, however, concerning understanding of posed queries themselves by the system led to specific solutions rather than a generally applicable one.

4.1 GRAPHICAL DATABASE QUERYING INTERFACES

Graphical query interfaces use various approaches to support query formulation. For example, menu-based user interfaces such as RABBIT and KARMA [TWF+82, BR85] provide an interactive database query constructing facility for casual users. They maintain the current partial query description and an example individual that satisfies the query for guiding a user in reformulating a partial query.

Another example is the class of co-operative graphical query interfaces such as Kaleidoscope [Cha91] which guides the user's query formulation grammatically and conceptually by pruning syntactically valid but semantically irrelevant choices for SQL queries. Furthermore, intelligent systems that interact with users in a co-operative manner in order to provide correct, non-misleading and useful answers to queries have been addressed in [Kap82, GGM92]. The systems had to handle users' misconceptions, false presuppositions and miscontruals related to queries and their answers.

More user-friendly features in a database query tool have been addressed by co-operative query answering [CD89, CD91, Mot90, GGM91, CC94]. Typical functions include approximate query answers [CYC+96, CC94] when exact answers are not available, associative query answers [CD89, CC92, FCY94], alternative answers [CD89] and intensional answers [CLC91, Mot94].

Three high-level interfaces to database systems are discussed in [Mot89]. These interfaces are concerned with browsing the database, retrieving neighborhood answers (similar-to) and providing a flexible interface where no query is rejected and explanations are given for null answers. The use of database contexts for disambiguating queries is developed in [AFM89], where a dialogue is carried out with the user using a dialogue tree to determine in what additional attributes the user is interested. However, the dialogue tree is restricted to attributes and no semantic inferences are made to determine additional attributes.

In [RWY+97], a pattern of high-level queries can also be seen as a query that retrieves a set of documents. Thus the data mining tools can be used to identify interesting queries which can be used to browse the collection. The main pattern types, the system can search for, are frequent
sets of concepts, association rules, concept distributions and concept graphs. To enable the user to specify some explicit bias, the system provides several types of constraints for searching the vast spaces of patterns that exist in the collection. The patterns which have been verified as interesting are structured and presented in a visual user interface allowing the user to operate on the results to refine and redirect search tasks or to access the associated documents.

WebDB [WJ98] is a Web query system to support more comprehensive database-like query functionality through a visual user interface, WebIFQ (Web In-Frame-Query), assists users in specifying queries and visualizing query criteria including document meta-data, structures and linkage information. WebIFQ automatically generates corresponding query statements for WebDB. As a result, users are not required to be aware of underlying complex schema design and language syntax ties.

Query by Review [GBJR96], with its more constrained user interface, performed somewhat better than AccessMed, a more general tool. Neither tool achieved adequate performance, however, which points to the difficulty of formulating a query for a clinical database and the need for further work.

Multi-modal user interfaces are discussed in [CCC+96, Zha98]. In [CCC+96], an approach for multi-paradigmatic visual access to databases is described, which is proposed to achieve seamless integration of different interaction paradigms. The user is provided with an adaptive interface augmented by a user model, supporting different visual representations of both data and queries. The visual representations are characterized on the basis of the chosen visual formalisms, namely forms, diagrams, and icons. To access different databases, a unified data model, the Graph Model, is used as a common underlying formalism to which databases, expressed in the most popular data models, can be mapped.

In [Zha98], interactive query formulation techniques for databases are presented which aim at bringing together high-level query formulation for simple queries based on a semantic model, incremental query answering for complex queries, associative query answering for relevant information and semantic query guidance for incremental query sessions. The latter suggests further queries which are considered to be relevant to the already submitted ones.

From an application point of view, current digital information systems in radiology are insufficient to accommodate the retrieval needs of scientific applications. Significant efforts are required in retrieving clinical cases for teaching and research. A prototype system that supports intelligent case retrieval based on a combined specification of patient demographics, radiologic findings and pathologic diagnoses is presented in [TJB+96]. The documents for these cases can be distributed among multiple heterogeneous databases. The system features automatic indexing of radiology and pathology reports, a comprehensive lexicon for thoracic radiology, an interface to a hospital information system, radiology information system and picture archiving and communication systems, and a graphical user interface for query formulation and results visualization.
Clinical research involves recording, storage and retrieval of disease-related patient data, typically using a database system [LCRM98]. In order to facilitate ad hoc queries to clinical databases, a query generator has been developed with a graphical interface. The query generator uses an object-oriented data model which is visualized using directed graphs. The main focus of the work was the definition of object-oriented user views to the partly complex data structures of a relational database. Furthermore, an attempt to define graphical abstractions for all common types of queries has been made. Thus, even for non-expert database users such as clinicians, it is easy to assemble highly complex queries for a thorough examination of the content of large research databases [BK95a].

Graph Model databases are queried through the adaptive interface. The semantics of the query operations is formally defined in terms of graphical primitives. Such a formal approach permits us to define the concept of "atomic query", which is the minimal portion of a query that can be transferred from one interaction paradigm to another and processed by the system. Since certain interaction modalities and visual representations are more suitable for certain user classes, the system can suggest to the user the most appropriate interaction modality as well as the visual representation, according to the user model.

4.2 NATURAL LANGUAGE INTERFACES FOR DATABASES

Natural language interfaces to databases [ART94, PG86] usually use pattern-based methods or parsing to translate a natural language input into some logical form and then to a sometimes complex query. A major obstacle to the system is that it is usually difficult to understand an input for complex queries expressed in natural language. The typical process to transform a natural language into a query language in real systems is to translate the natural language query input into a logical form representation by a pattern-based method or parsing. The logical form representation is then transformed into a query language by consulting the database schema.

However, the integration of intensional meaning underlying attributes and values participating in a conditioned query statement has not been addressed within all natural language based approaches. Moreover, the need of parsing posed queries expressed in a natural language or even in a sublanguage is bound to the complexity of the system as well as the failure of formulating a syntactically correct query by the end-user. In addition, some knowledge of the application domain semantics is still required in order to issue a semantically meaningful query.

On the other hand, the problem that casual users of database systems are typically not skilled in structured query languages has already been addressed in [AA93]. The paper challenges the dictum that the usability of a Natural Language Interface (NLI) is enhanced when its linguistic capabilities are extended. It is argued that effective natural language communication needs a naturalistic
sublanguage of English, reduced in complexity, but nevertheless providing the flexibility of natural language input. Two investigations are described, both of which involved real users performing real information retrieval tasks.

The first gives an insight into the detailed characteristics of such a sublanguage, providing a comparison with earlier research. The second compares the effect of inter-sentential linking devices like ellipsis on the usability of an NLI, with that of simple, extra-linguistic editing facilities. The results show that enhanced linguistic capabilities can indeed improve usability under certain circumstances, but that extra-linguistic enhancements can be just as effective. The results also show that usability can actually degrade as both the linguistic and the extra-linguistic capabilities of an interface are improved.

The study reported in [SM94] involved the use of the natural language query system INTELLECT. It evaluated the level of correct interpretation to investigate whether the use of such a system is practical. Two sets of queries generated by two groups of senior-level business students were used. Questions from the first set were generated by “naive” students who were untrained, and not aware that they were providing queries which were to be executed by a computer. Students from the second group attended a short lecture and understood that they were to generate natural language queries to be executed by a computer. INTELLECT’s lexicon was augmented in stages. The level of correct interpretation achieved in this study is far above any previously reported and suggests that existing natural language query systems may be practical. Key features in the accuracy of interpretation were user training and iterative lexicon enhancement.

The problem of mapping natural language (NL) sentences to SQL has been addressed in the paper [Ott92] where an augmentation called SQL + is presented, and it is shown that, when observing some simple rules, a NL sentence can be mapped from its internal representation to SQL + in a straightforward and uniform manner. Though these SQL + expressions may not always be optimal from a performance point of view, they have some advantages which are considered to be more useful. However, in order not to restrict the full power of NL one also has to augment the SQL language in some way, because SQL has some restrictions and shortcomings, of which the lack of a loop capability for handling ordinals (e.g. the 3rd lowest..., the 5th highest...) is the most serious.

In [LW92], a natural language interface has been developed to facilitate the use of statistical packages. Queries are parsed into "case frame" based on statistical primitives. A rule-based expert system uses the case frame to choose a statistical test and generate a batch file that, when executed, answers the query.

In [SKL93], an intelligent database interface system, IDQS, is proposed. This system serves as the interface between users and relational database systems. IDQS enables users to query the database in English, and thus does away with the need to become familiar with a query language. IDQS consists mainly of two big blocks: a natural-language processing unit and a knowledge-base. The
NL processing unit and knowledge-base are two completely separate entities. This system structure provides a very good base for a transportable system. To transport the system to a new domain, one only needs to update the knowledge-base. No modification has to be done on the NL processing unit. The knowledge-base is designed in such a way that knowledge updating is an easy task which can be carried out by a semi-expert database user.

In [WI92], a knowledge-based database assistant (KDA) which integrates a natural language query system with a skeleton-based query guiding facility has been presented. When a user works with the KDA natural language query system, the query guiding facility can supply several kinds of skeletons to guide users in performing database retrieval tasks. A semantic network model called S-Net has been introduced to represent the knowledge for natural language query processing and skeleton generation. The methods for implementing the system are also discussed in their paper. This approach focuses on facilities for guiding novice users in performing the database retrieval tasks, such as formulating valid database queries, refining incomplete database queries, and modifying database query misconstruction in a tutor based way of interaction with the underlying database.

Furthermore, IRUS [BMS86] is also a natural language interface which attempts to resolve several linguistic problems in user system conversation. [GAMP87] provides a menu-based knowledge acquisition tool for acquiring domain-dependent knowledge, making it easy to transport to a different domain. Finally, [BJ84] gives users feedback at successive checkpoints in the natural language translating process, whereas "Fred" [JLNPS86] is an intelligent database assistant which combines database expertise with an intelligent user interface to provide natural language dialogue and menu selection.

5. HUMAN-COMPUTER INTERACTION (HCI)

A user-database interface consists of two key components: the data model and the query language. Database management system designers can manipulate these two components to produce an effective database interface to maximize user task performance. However, in order to do this, the designers first need to know the effects the two components have on end users and the interaction effect of the two components. Despite the acknowledgment of user-database interfaces as a cornerstone in database utilization, there is still a paucity of research in this area.

In the field of HCI there exist many formalisms for analysing, describing and evaluating interactive systems. However, in developing and evaluating user interfaces to databases, we found it necessary to be able to describe presentation and interaction aspects that are catered for poorly or not at all in current formalisms. [MGP98] presents a framework for the systematic description of data model, presentation and interaction components that together form a graphical user interface. The utility of the framework is then demonstrated by showing how it can be used to describe two existing
visual query interfaces. These examples show that the framework provides a systematic method for the concise description of graphical interfaces to databases that can be used either during interface design or as a communication aid.

An empirical study that investigates the effect of entity-relationship versus relational models, and textual versus visual query languages for user-database interfaces is described in [CSW98]. A good understanding of how the data models and query languages affect user performance will enable the database developer to choose and design interfaces that can provide effective and efficient support for end users. To provide guidelines for the design of these new interfaces, many proposed visual query languages (VQLs) are analysed and compared from the user’s viewpoint. The analysed features include ER concepts, interaction modes, model diagram transformations, condition specification, aggregate functions, and output organisation [Cha97].

[AG97] presents a methodology for developing a user interface that combines fourth generation interface tools (SQL forms) with a natural language processor for a database management system. The natural language processor consists of an index, a lexicon and a parser. The index is used to uniquely identify each form in the system through a conceptual representation of its purpose. The form fields specify database or non-database fields whose values are either entered by the user (user-defined) or are derived by the form (system-defined) in response to user input. A set of grammar rules are associated with each form. The lexicon consists of all words recognized by the system, their grammatical categories, roots, their associations (if any) with database objects and forms.

The parser scans a natural language query to identify a form in a bottom-up fashion. The information requested in the user query is determined in a top-down manner by parsing through the grammar rules associated with the identified form. Extra-grammatical inputs with limited deviations from the grammar rules are supported. Combining a natural language processor with SQL forms allows processing data modification tasks without violating any database integrity constraint, having duplicate records, or entering invalid data. A prototype natural language interface is described as a front-end to an ORACLE database for a computer integrated manufacturing system.

6. SUMMARY

Various approaches and mechanisms have been proposed and implemented which aimed at alleviating the task of end-users to understand and, consequently, address data and information represented in a language which can be interpreted by machines and not by humans. In order to help in improving human/computer communication when asking for information, the proposals vary from advanced visual query interfaces to AI-based approaches such as question answering systems and natural language based interfaces with different degrees of complexity.
However, advanced visual query interfaces mostly rely on some diagrammatic presentation of the \textit{domain of discourse} which shift the problem of understanding data and its contents to understanding of visual formalisms, whereas AI-based approaches contributed in understanding the complexity of dealing with natural language based interfaces in that parsing of meaning turned out to be a rather sophisticated task. On the other side, database query languages are tightly coupled with implementation models and do not make use of any semantics/meaning of data and/or database schema when a query is being formulated.

In this thesis, a query formulation methodology is presented which treats the meaning of an application domain as the central part of human/computer interaction for the purposes of formulating complex (analytical) queries. The query formulation process takes place by means of a human/computer interaction mode which relies on the incremental refinement of queries according to system suggestions; the system takes into account the semantic context of a query.

The latter has been conceived and realized as an abstract state machine, and therefore, no additional parsing of the query semantics is needed. With this approach, it is possible to formulate queries which are reasonable and meaningful when using linguistic elements from more than one natural language. Furthermore, no learning of any syntax or visual formalisms is required.
The way to take the world which I have found most tenable is to regard it as a single Experience, superior to relations and containing in the fullest sense everything which is.
—F. H. Bradley, Essays on Truth and Reality, pp. 245-246, 1914

In order to better understand the philosophy behind the querying approach and prior to the formal specification of the Meaning Driven Data Querying Automaton (MDDQA) which underlies the methodology used to guide the end-user to multi-lingual and meaningful queries, we give some examples of query construction paradigms throughout this chapter. The examples rely on end-user/system interaction sessions as illustrated by screenshots of the various graphical user interfaces for the interactive query construction technique.

The querying methodology relies upon movement from a current query state or context to a next, semantically consistent one. Therefore, the methodology is equivalent to the refinement of the current query state, where the user-system interaction is enabled in such a way that particular query terms are selected and/or activated by the end-user out of sets of system suggested (inferred) terms. This is done either for the purposes of looking at the meaning of query terms in order to decide if they should be considered for the next query state (refined query) or for the purposes of selecting particular query terms of interest, e.g. selection of particular attributes and/or values.

In the following discussion, query construction scenarios in the form of end-user/system interaction with the help of Web-based graphical user interfaces (GUIs) are given. The query construction paradigms refer to three different categories of query families: a) simple instances description oriented queries (section 1.) where only conjunctive (AND-connected) queries are enabled, b) advanced instances description oriented queries (section 2.) which additionally allow disjunctive queries as well as (analytical) operations to be posed, c) querying of classification rules (section 3.).
1. SIMPLE INSTANCES DESCRIPTION ORIENTED QUERIES

Simple instances description oriented queries are those which address instances based on their description as given by AND-connected (conjunctive) conditions. The conditions are considered as being \(<\text{attribute}, \text{categorical value}>\) pairs connected by an equality comparison operator. No functions or further comparison operators are considered for the description of instances to be retrieved. The query construction examples in the following refer to patient instances as gathered and queried in the medical application domain of hysterectomy in gynecology.

At the initial stage of the query construction process, the end-user is requested to select a particular natural language (figure 4.1) in which all query terms will appear during the query construction session. The availability of query terms in a particular natural language is a matter of providing the objects standing for the domain alphabet terms with labels from this language.

Having selected a natural language and given that the selected initial state for the query construction is the concept term Patients for Hysterectomy, the MDDQA suggests a set of potential query terms to be further considered for the refinement of the query (figure 4.2). In other words, given that the query discourse is Patients for Hysterectomy, the query might be further refined either by the concept postmenopausal or the concept pre/perimenopausal. Having done so, the MDDQA has moved either to the state \(<\text{patient}, \text{postmenopausal}>\) or the state \(<\text{patient}, \text{pre/perimenopausal}>\). Note that the initial state might also be selected out of a set of initial states (concept terms) such as Patients for Hysterectomy, Patient’s condition before Hysterectomy and Patient’s condition after Hysterectomy.
If the current query (MDDQA) state is \(<\text{patient, postmenopausal}>\), then the system suggests a set of further query terms as potential terms to be considered for the query refinement (figure 4.3). The set of suggested query terms is semantically consistent with the given query state in that it is inferred on the basis of the set of terms constituting the current query state and the connectionism model underlying the representation model (ML-DAG) of the domain alphabet terms. For example, the concept \(\text{leiomyomata}\) would be excluded from the set of suggested terms, if the current query state would have been \(<\text{patient, peri/premenopausal}>\).

Moreover, moving to a subsequent query state is also a matter of end-user selection of one or more terms out of the suggested ones which is based on the underlying meaning of a particular term such as \(\text{Cervical intra-epithelial neoplasia or dysplasia}\) (see also figure 4.3 - left hand side). Thus the query (MDDQA) state transition is not only determined by the meaning as expressed in terms of established associations and preconditioning statements within the representation model for domain alphabet terms but also by the underlying meaning of query terms in the domain of discourse.

The meaning based query (MDDQA) state transition logic also applies to property and value terms as follows. Assuming that the current query (MDDQA) state is given by the set of terms \(<\text{patient, postmenopausal, endiometrosis with involvement of the uterus}>\) (figure 4.4), which means that we are interested in postmenopausal patients for endiometrosis with involvement of the uterus, and given that we are not interested in further refining the concept \text{patient}, the MDDQA suggests a set

---

Figure 4.2. System suggestion for refinement of the concept “patient”.

[Diagram of system suggestions]
of property terms to be considered for a potential new query state including those property and value terms that constitute the AND-connected conditional statements.

For example, as depicted in figure 4.4, the properties Symptoms and Guidelines compliance will be suggested for value instantiation. A third property such as Major impairment will appear, if and only if, the value of property Symptoms is set to yes · finite value domain of Symptoms is \{yes, no\}. This is due to the specification of conceptualisation where the precondition Major impairment is valid only if there are symptoms has been represented within the model of the domain alphabet terms. Since it makes sense to consider both properties within the same query under preconditions, the query (MDDQA) state transition to a query state where Major impairment is also included, such as depicted in figure 4.5, depends on the value assigned to the property Symptoms.

Similarly, age as a property is only relevant given the query state (context) of <patient, peri/premenopausal, endometriosis with involvement of the uterus> as depicted in figure 4.6, since the patient instance is not characterized as postmenopausal anymore (figure 4.4) but as peri/premenopausal.

The assignment of values to properties in order to describe the patient instances in terms of conditional statements is also done by moving from a particular query state to a semantically subsequent one. Thus the new query (MDDQA) state will include value terms as selected out of suggested
Figure 4.4. System suggestion for query refinement based on unconditioned properties and/or values.

Figure 4.5. System suggestion for query refinement including the preconditioned property “major impairment”.
Figure 4.6. Exclusion of property "age" when a different query context is given.

Figure 4.7. Addressing values within the dynamically inferred finite value domain for "age".
(inferred) finite value domains for particular properties. The elements of the finite values domains are also inferred on the basis of the given query context (current query context) and the representational model of the domain alphabet terms. Figure 4.7 depicts the finite value domain for the property age - \{less than 45, equal or more than 45\} - given the particular query state. Note that elements of a predefined finite value domain might be excluded from consideration when specific preconditions are satisfied.

Finally, figure 4.8 depicts the constructed query referring to all pre/perimenopausal patients for endometrosis with involvement of the uterus having age less than 45, no symptoms and the guidelines respected as expressed by using German terms.

Since movement to a subsequent query state is equivalent to a query refinement process and given that the representational model for the domain alphabet terms supports the connection of terms to more abstract ones, a current query state might also include complex (assembled) properties and/or values. For example, it is possible to construct a query in terms of assembled properties such as dismission date and dismission destination which both are conceived as assembled properties of the concept Hospitalization of patient. These properties are assembled by assigning the property terms date and destination to the property term dismission.

Similarly, complex (assembled) values such as never frustrated or almost never frustrated can be assembled out of the assignment of the values never and almost never to the value frustrated within the same finite value domain. Both properties date and destination will be suggested by the...
MDDQA when it might come to the extension of the current query state which already includes the term *dismission*. Accordingly, the values *never* and *almost never* will be suggested by the MDDQA when it might come to the refinement of the value *frustrated* assigned to the property *patient's condition*.

2. **ADVANCED INSTANCES DESCRIPTION ORIENTED QUERIES**

In the following, advanced query construction sessions in a different domain are presented. They are based on the same concept of incremental refinement of a current query context by adding semantically consistent query terms as inferred by the MDDQA. However, the end-user/MDDQA interaction takes place on a blackboard where each query is constructed in terms of a graph based visual formalism. This enables the visual representation of query terms as connected graph nodes, and therefore, it is possible to express the semantics of links holding among query terms. Furthermore, it is easier to distinguish between *conjunctive* and *disjunctive* clauses as well as between *concept terms* and *relationship ones*.

Operations can also be assigned to query terms (nodes) according to the semantic value of the query term standing for a variable. Operation assignment is not a particular feature of the graph based visual formalism. However, we will also refer to operation assignment as potential query refinement with this visual query formalism. Two application paradigms will be used in order to illustrate the end-user query construction session when this visual query interface is used. One application paradigm refers to querying of physical parameters related to avalanche prediction processes (appendix B) and the other one refers to data related to cleaning actions of land mines areas (appendix B).

In this mode, the end-user is also requested to select a preferred natural language in which the query terms (graph elements) are going to appear. For the sake of convenience, we will restrict ourselves to *English* as a query terms presentation language. Assuming that the end-user either has no idea, or does not need to gain some knowledge about the underlying data semantics, she/he is required to select an initial term, i.e. start with an initial query (MDDQA) state.

Having selected *Automatically delivered measurement data* from the set of initial terms \{*Automatically delivered measurement data, Observers data, Measurement network, Measurement station*\} as an initial concept term to start with, the current query (MDDQA) state consists of a single concept term positioned on the blackboard (figure 4.9). In order to further refine the query by considering one or more terms as suggested by the MDDQA to be included in the new query state to move on, the end-user has always the possibility of looking at the underlying specification and/or meaning of a particular term.
Note that this visual query interface distinguishes between concept terms and terms standing for relationships such as measured by (figure 4.9), when the set of suggested terms is presented to the end-user. Given this particular example, the next semantically consistent query state might consist of the concept term Automatically delivered measurement data plus one or more terms selected out of the set of suggested terms \{ENET 10-minutes data, ENET 1-hour data, IMIS-100 data, measured by, time\}. The concept terms are conceived as subconcepts which further refine the concept Automatically delivered measurement data, whereas measured by as a relationship and time as a property hold for all kinds of Automatically delivered measurement data.

Note also that, if the initial query state would have been the concept term Measurement station, the suggested relationship would have been measures, since active and passive voice depend on the relationship direction, e.g. passive, if we move from Automatically delivered measurement data to Measurement station through a relationship expressed by the verb measure and active, if we move from Measurement station to Automatically delivered measurement data.

Let us assume that the current query (MDDQA) state consists of the query terms \{Automatically delivered measurement data, measured by, time, ENET 1-hour data\} (figure 4.10) which indicate the fact that the end-user is interested in retrieving the time of the special kind of Automatically delivered measurement data such as ENET 1-hour data measured by some measurement stations. A potential query state to move on might include one or more concept terms from the set of suggested
terms \{Wind data, Atmospheric data, Snow data\}, since all these kinds of terms are conceived as subconcepts of the concept terms ENET 10-minutes data, ENET 1-hour data or IMIS-100 data, i.e. they can be reached from all these parent concept terms.

![Diagram of query refinement through exploitation of acyclic graph structures](image.png)

*Figure 4.10. Potential query refinement through exploitation of acyclic graph structures*

Having selected Snow data as a concept term to be included in the subsequent query (MDDQA) state which now consists of the query terms \{Automatically delivered measurement data, measured by, Time, ENET 1-hour data, Snow data\}, a potential extension of the current query context could be triggered by including one or both suggested property terms \{height, temperature\}, as depicted in figure 4.14, which refer to snow data given that this particular query context holds. Since the concept term Snow data might also be addressed within the query state as consists of the terms \{Observers data, Time, Snow data\}, more than these two properties would have been suggested by the MDDQA.

Another example of a given query context based inference of query terms to form a new query state is the case where the properties Radiation, Humidity for Atmospheric data would not have been suggested. This happens only if one of the concept terms ENET 10-minutes data, ENET 1-hour data had been considered within the current query state, since it makes no sense to address these properties for the concept term Atmospheric data in conjunction with these concepts, but only in conjunction with the term IMIS-100 data.

Assuming that the current query (MDDQA) state is composed of the query terms \{Automatically delivered measurement data, measured by, Measurement station, Type, Snow station, Time, ENET
1-hour data, Snow data, Temperature, at 25cm, at 50 cm}, where Snow station is the value assigned to the property term (variable) type of measurement station, a potential extension of the query state could be triggered by the selection of a value either for the property temperature at 25cm of snow or for the property temperature at 50cm of snow.

However, similar to finite value domains suggested by the MDDQA for categorical value assignments (see also previous section) to properties (variables), numerical values can also be addressed within suggested numerical value intervals such as \([-50, 50]\) celsius for both properties (see figure 4.14). Any values specified outside this particular range of values will be rejected, since it makes no sense to address another value, when the current query state or context is given. Note also that the numerical interval is expressed in the underlying measurement unit Celsius, and therefore, any value assignment is bound to a particular range of values as stored due to the measurement unit.

Different finite value domains might be inferred by the MDDQA even for the same property, if the query context varies. For example, the property direction of wind is assigned two possible finite value domains such as \([0, 360]\) degrees or the set of categorical values \{North, North-East, North-West, South,...\}. The arithmetic interval will be suggested only if the current query state is something like \{Automatically delivered measurement data, ENET 10-minutes data, Wind data, Direction\}. The set of categorical values will be presented only when the current query context would have been \{Observers data, Wind data, Direction\}. 

![Query View Operations](image)
Figure 4.12. Constraints based assignment of numerical values for the property (variable) "temperature".

Nevertheless, a particular feature of the graph based visual formalism is the possibility of activating the labels of the links which give an insight into the connectionism semantics of the query nodes (terms). For example, we might be interested in having an insight into the underlying specification of the associations among query terms. This might be either definitional when an assignment of particular terms to each other has been done by definition during the specification of conceptualization or assertional indicating the fact that assignment of particular terms to each other has been done by assertions as it happens to be the case when terms are assigned to each other from already existing databases.

Further link labels indicate the fact that two terms might be connected by an is-a or a part-of relationship. It is also possible to assign the logical operator OR – OR-connected conditions – to nodes (terms) - outgoing dashed line (link) - as well as negating a term by assigning it the NOT operator. The latter is currently restricted to nodes standing for specified and/or selected atomic or interval values. Consequently, these values should be excluded from the query result.

### 2.1 EXPRESSING OPERATIONS

A particular query (MDDQA) state (query context) can also be extended by assigning operations to particular query terms according to their roles within the given query context. The set of suggested
Query construction paradigms

operations is also inferred by the MDDQA just as the suggestion of terms from the set of domain alphabet terms. However, the inferences take into account the actual role of a query term and the role of operations as given by their classification within the input alphabet database model, as well as the current query state.

Assuming that the current query context is composed of the query terms \{Measurement station, assigned to, Measurement network, Type, ENET-network, Type, Wind station\}, as depicted in figure 4.13, we could move to a semantically consistent query state by including the statistical operation Relative Frequency assigned to the property type of measurement station which, in turn, has been assigned the value Wind station. This operation can be selected out of the set of MDDQA suggested operations \{Relative Frequency, Frequency\} which are relevant for the property term type of measurement station, since this term internally has been classified as categorical variable, i.e. its finite value domain is defined in terms of categorical values, even if they are internally encoded by numerical ones such as 0, 1, 2,...,.

No arithmetic operations such as average or mean value, deviation will be suggested by the system for consideration. This is only the case when a query term has been classified as a property term and as numerical variable. For example, figure 4.14 depicts an example of assigning an operation to the query term temperature at 25cm. Similarly, comparison operators such as \(\geq\), \(\leq\), \(<\) are only suggested for nodes which stand for atomic arithmetic values, e.g., the atomic value term 30 in the query depicted in figure 4.15.

On the other hand, bi- or multi-variate operations such as time series, scattering diagrams are only considered in conjunction with the presence of more than one property within the current query.
state. Furthermore, preconditions might be assigned to the objects representing operations within the meta-data database. Since operations are objects too, additional attributes might be defined which provide more information about the intended usage of each operation. This can be viewed at the time of selecting a particular operation out of the set of potential operations.

3. QUERYING OF CLASSIFICATION RULES

Classification rules [BFOS84] have been extensively used in decision support systems as classifiers of instances due to the class or classes specified in the conclusion part of the rules. They are mostly represented as an ORed set of rules the antecedent part of which are AND-connected preconditions or premises in terms of $<\text{attribute}, \text{value}>$ pairs. In other words, a particular rule applies to an instance, if the set of premises of the antecedent part is relevant to the particular instance.

In particular, medical guidelines have been specified in order to support decision for medical interventions in various domains. They can be conceived as IF. THEN classification rules concerning patient instances. For example, a large set of such classification rules have been specified for coronary angiography and coronary revascularisation - medical interventions in cardiology - and hysterectomy - medical intervention in gynecology. These rules map instances (patients) into con-
clusion classes of appropriateness or necessity. The classes refer to a range of scores between 1 and 9. In order to classify a particular patient instance due to appropriateness or necessity of a medical intervention, we need, first, to address that particular set of <attribute, value> pairs which represents the patient’s profile.

It turns out that addressing the appropriate classification rule is a matter of addressing the most relevant attribute and value terms out of a large set of <attribute, value> pairs. This becomes a cumbersome task for end-users (patients and medical experts), especially when a large set of attribute and value terms, and consequently, a large set of classification rules has to be taken into consideration. Current knowledge representation techniques for classification rules do not alleviate this task. For example, if decision trees [Qui86] are chosen as knowledge representation technique, then the tree becomes too large to be presented to the user or efficiently maintained.

Alternatively, decision tables [WF00] as knowledge representation technique requires the familiarity of end-users with database specific query languages such as SQL [GW99], when a relational database is used, or object-oriented query languages [CB97, KKS92] (object-oriented database), or more recently, XML oriented query languages such as XML-QL or XQL [ea99, Mai98, AQM+97, FFLS97], when XML based repositories are used for the management of the classification rules (see also section 2. in chapter 3). In all these cases, however, end-users must
cope with the syntax of a repository specific query language and understand the database model and/or data semantics.

In case that first order predicate logic– [MW88] or rule–based [HRWL84, Ign91] approaches are chosen as knowledge representation technique of classification rules, we might avoid formulation of database specific queries or navigation through large decision trees, but we are still bound to the peculiarities of the inference logic which is inadequate for the nature of classification rules, especially in domain sciences such as medicine.

For example, logic–based approaches, where a backward chaining inference engine - proof-theoretic approach with top–down evaluation - such as Prolog or Datalog [MW88], or deductive databases [Min88] are used, rely heavily upon the logic of proving or reaching a conclusion by trying to satisfy the body of rules (antecedent rule part). This is against the nature of classification rules, since we first need to describe the antecedent part and then decide which rule applies for a classification of a patient instance. Given that a particular class can be reached from a variety of antecedent parts of rules, such a query answering technique would result in a large subset of relevant rules which should be further examined by the end–user in order to choose the relevant one.

Even if a forward chaining inference engine - bottom–up evaluation in proof-theoretic, model-theoretic or fixpoint approaches - is used, where the conclusion can be reached from the antecedent part in logic- or rule–based systems in general, we still have to cope with the meaning and/or semantic dependencies among attribute/values combinations. In both approaches, there is no easy separation between rules and data, and therefore, attributes and values are rather terms or symbols directly embedded within rules.

Regardless of how the underlying knowledge representation model has been realised, addressing <attribute, value> pairs by end–users within domain sciences such as medicine, requires a thorough understanding of the domain knowledge, i.e. intensional meaning of attributes and/or categorical values. Furthermore, the usage of a particular natural language for addressing attribute and/or value terms has an impact on the query results. All retrieval techniques are bound to the symbols chosen for the representation of attributes/values.

3.1 GUIDELINES AND CLASSIFICATION RULES

The following definitions should establish a common terminology for the reader to be used throughout this section.
**Definition 1.1.** A classification rule is an argument consisting of a collection of AND-connected propositions where some propositions are singled out to be conclusions, i.e. classes into which a particular instance has to be mapped. An argument cannot have more than one conclusion.

**Definition 1.2.** A proposition is the sort of thing which qualifies as an evidence to establish a reason for an argument. A proposition constitutes an \(<\text{attribute, value}>\) pair with terms within \(T\), where \(T\) is the disjoint set of attributes and categorical values participating in arguments.

**Definition 1.3.** A premise is a proposition offered as a reason. The proposition which is supposed to be supported by premises is called conclusion which stands for the class to be assigned to a particular instance.

**Definition 1.4.** A Domain of Interest is the subject of matter within which an argument holds.

For instance, the following classification rule (guideline) holds within the domain of interest of Patients candidate for Coronary Angiography\(^1\):

**Rule CA-1:**
IF patient is asymptomatic AND risk factors are two or less AND stress test is not done AND risk occupation is high THEN appropriateness of intervention is 1.0

whereas the following guideline holds within the domain of interest of Patients candidate for Coronary Revascularization\(^2\):

**Rule CR-1:**
IF patient is asymptomatic AND stress test is positive AND disease is left main AND ejection fraction is \(\geq 50\%\) THEN appropriateness of intervention is 7.0 AND necessity of intervention is 7.0

---

\(^1\)A medical examination for the assessment of heart diseases  
\(^2\)A medical operation in order to treat heart diseases
In the following, we will use the terms *classification rule* and *guideline* interchangeably. It turns out that a guideline is a meaningful argument (*classification rule*), only if it consists exclusively of propositions (premises and conclusions) which make sense to assemble together within a particular rule. Therefore, having the system suggesting to the end-user attributes and/or values that make sense to be addressed in order to reconstruct the set of premises to be used as key mapping value to the conclusion part of the rule, the semantic dependencies which hold among attribute/value terms must be considered.

For example, the attribute *necessity of intervention*, e.g., rule CR-1, makes sense to be addressed in the conclusion part of guidelines, only if the value of the attribute *appropriateness of intervention* is in the range $[7-9]$. Similarly, it makes sense to consider the attributes of *risk factor*, *stress test* and *risk occupation*, only if a patient has been classified or characterized through the value *asymptomatic* and the given domain of interest is *coronary angiography*. Given another domain of interest such as *coronary revascularization*, another set of attributes and/or finite value domains will be relevant even if the classification of a patient is still the same, namely *asymptomatic* (see also rule CR-1).

In addition, attributes and/or values might be expressed in different natural languages and/or underly a specification/definition which is a common issue when domain science terminology is used. The latter refers also to attributes standing for numerical variables specified by measurement units. All these issues provide an intensional meaning to attributes or values, the absence of which during user/system interaction might lead to wrongly interpreted attributes and/or values, and consequently, to wrong classification rules.

For example, the term *risk factors* underlies the definition of *Risk factors for coronary artery disease* are defined as: *diabetes mellitus, hypertension, dyslipidemia, current smoking and previous smoking more than ten pack years, positive family history* (e.g. *history of a myocardial infarction, previous revascularization in a patient of less than 65 years, known coronary artery disease in patients less than 65 years*) and *male gender*. For patients with atypical angina, male gender is not defined as a risk factor.

As a second example, the term *asymptomatic* underlies the definition *Asymptomatic are those patients with no history of atypical or chronic stable angina with normal physical activity; patients with a history of chronic stable angina who have had no angina on their current medical regimen for more than three months; patients who have sustained a myocardial infarction more than 21 days ago with no recurrence of angina, and patients screened because of risk factors or because of high risk occupation.*

Given that we want
that the system guides the end-user through the information space of attributes and values with respect to semantic dependencies among terms constituting premises of classification rules,

- making use of embedded intensional meaning of attributes and/or values,

- use different natural languages,

the following querying paradigm of a classification rule out of a large set of already specified classification rules in the domain of coronary angiography in medicine illustrates the querying methodology. Figure 4.16 depicts an example of having assigned the value atypical angina to the attribute patient out of the set of suggested (inferred) terms. The meaning underlying this term can be viewed before a decision is made for further consideration.

![Figure 4.16](image)

*Figure 4.16. Moving into the initial query state with the attribute/value pair \( \text{<patient, atypical angina>} \).*

Figure 4.17 depicts the inferred suggestion of the system as a result of the current query state as given by the set of premises \{\text{<patient, atypical angina>}, \text{<stress test, positive>}, \text{<gender, female>}\}. Since age conditionally depends on the value of gender, instantiation of age will follow that of gender. Having selected the value female for gender, the inferred set of terms as finite value domain for age will be \{\text{under 50, 50 to 75, 75 or older}\}. Otherwise, if the value for gender would have been set to male, the value under 50 would have been excluded from the inferred (suggested) set of terms as finite value domain, since the precondition not (gender, male) assigned to the term \text{under 50} is not satisfied.
The end-user is not requested to assign values to irrelevant attributes, because attributes will not be presented to the end-user, if assigned preconditions are not satisfied. For example, given the query state, \{<patient, atypical angina>, <stress test, positive>, <gender, female>, <age, 75 or older>\}, the attribute risk factors will not be presented at all, since the precondition not (age, 75 or older) is not satisfied.

Figure 4.18 depicts the potential extension of the current query state \{<patient atypical angina>, <stress test, positive>, <gender, female>, <age, 50 to 75>\} with an attribute/value pair selected from the suggested attribute/finite value domain Risk factors \(\rightarrow\) \{none, one or more\}. This suggestion is due to the fact the the assigned precondition not (age, 75 or older) is satisfied, since the attribute/value pair <age, 75 or older> does not appear in the current query context or state. Finally, the assigned classes, i.e. the conclusion part, of the appropriate classification rule which matches the set of premises considered in the final query state for a particular patient instance is depicted in figures 4.19 and 4.20.

The classification rule as depicted in figure 4.19 complies with the query state or context where risk factors qualifies as relevant attribute, whereas figure 4.20 depicts the classification rule having risk factors not qualifying as a relevant attribute.
Figure 4.18. Suggestion of the conditioned attribute risk factors and relevant finite value domain.

Figure 4.19. The classification rule without consideration of attribute risk factors.
Figure 4.20. The classification rule with consideration of attribute risk factors.
Chapter 5

THE REPRESENTATION MODEL OF MEANING OF TERMS

A sign is something which stands for something to somebody in some respect or capacity
—Charles S. Peirce, Collected papers, Harvard University Press, 1958

Prior to the specification of the meaning driven querying automaton which guides the end-users to the construction of a reasonable or meaningful query, we will describe the representation model of meaning of terms which constitute the querying alphabet $T$. In particular, it constitutes the input alphabet of the specified automaton. In order to provide an automaton, however, where meaning of terms is taken into account during query construction, a representation model for expressing the meaning of terms needs to be addressed first as knowledge repository of the query language alphabet.

Figure 5.1 gives an overview of the components of the automaton from a systemic point of view. The repository of alphabet terms is made up of two parts: the Query Domain Alphabet DB and the Query Operational Alphabet DB, where the first is subject to the specification of conceptualization, since it is application domain specific. The component for the Inference/State transition Logic represents the inference engine which drives the end-user during construction of a meaningful query by suggesting meaningful terms according to the given query context. This component will be covered in the next chapter. Finally, the Visual Query Interface refers to the blackboard where the interactive session for a query construction takes place.

Meaning is mainly expressed in terms of intentional meaning of terms (section 1.) and in terms of the context of terms (section 2.). In particular, the intentional meaning of terms, both application domain terms and operational terms, is expressed by having all terms conceived and represented as objects rather than symbols. Therefore, symbols with additional properties are considered to be the elements of the finite set of the input alphabet.
The context of terms is expressed by a connectionism model (section 2.1) which provides the basis for connecting/assigning terms to each other, and therefore, forms a linked information space of terms, as well as by constraints holding within it (section 2.2). The latter refers to constraints holding among application domain terms and between operational and domain terms.

The specification of meaning of terms can be either part of the specification of conceptualization from which the application domain terms are derived or part of reverse engineering where semantics are derived from a given data set. On the other hand, operations are also part of conceptualization but they are application domain independent.

Even though the term conceptual model has been coined with many slight differences in meaning [JM00], its essential features have been the representation and understanding of the problem raised by the users or customers in order to reach an agreement on the scope of the solution in terms of providing or building a software system that solves the problem in question. The objective of conceptual models in the field of Data Engineering is to represent the domain of discourse in terms of data involved in the problem and operations on these data [Wie95, LK95, MBJK90]. Embedding meaning in query formulation or construction processes, however, poses additional challenges to conceptual modelling, especially when meaning/interpretation of terms becomes the subject of the specification of conceptualization.
The meaning of the querying alphabet as expressed in such a way that the context of terms is explicitly taken into account by the state transition algorithms, which infer the possible, semantically consistent query states to move into, as described in the following chapter. The intentional meaning of terms has an implicit impact on the state transition algorithms in that the end-user selection of query terms, which determine the current query state, can be done by having a look at the properties of terms before selection.

1. REPRESENTATION OF INTENTIONAL MEANING OF TERMS

The finite set of input alphabet terms is classified into two subsets: $T = T^D \cup T^F$, where $T^D$ is the subset of the domain terms (section 1.1), which refers to all terms relative to a particular application domain, and $T^F$ is the subset of the operational terms (section 1.2), which refers to terms standing for operations. All terms are represented as objects, and therefore, terms can be assigned properties.

In particular, a term is conceived as a unit of thought and is assigned a term unique identifier - TUI. A TUI might refer to one or more natural languages. A word in a particular natural language such as English or German might be assigned to a term as representative of a class of synonyms but not as the identifier of the term.

1.1 THE DOMAIN TERMS

The intentional meaning of domain terms $T^D$ is expressed by properties which might refer to:

- definitions and/or explanations of terms,
- underlying units of measurement,
- images
- synonyms

Therefore, domain terms are presented to the end-user for selection during query construction together with its properties. In this sense, only well-understood domain terms are selected/included in the query, since, for instance, annotation of a term is a property of a term, which can be viewed by the end-user during the query construction process. All domain terms are classified either as concepts, properties or values.
In order to illustrate the approach, an example is given as taken from the medical domain of interest hysterectomy, for which the following subset of domain (alphabet) terms have been specified and expressed in English. Note that everything which appears within single quotes is a value assigned to the properties of a term which is uniquely identified by term unique identifiers (tui’s).

\{tui = 1; name = 'Patient'; symbol = 'P1'}
\{tui = 40; name = 'Premenopausal'; symbol = 'P2'}
\{tui = 50; name = 'with asymptomatic leiomyomata'; symbol = 'P44'}
\{tui = 60; name = 'Abnormal uterine bleeding (no leiomyomata)'; symbol = 'P43';
description = 'Unacceptable social and/or hygienic situation for the patient.'
\{tui = 100; name = 'Major Impairment'; symbol = 'MImpair';
description = 'During the last 3 months, the patient stayed home (missed work or cancelled all activities) for 1 day per month because of pain or discomfort.'
\{tui = 200; name = 'Estimated uterine weight'; symbol = 'EVW';
description = 'Based on clinical experience estimated in grams (280 grams corresponds to 12 weeks of gestation). If the patient is given GnRH agonists prior to proposed surgery, use size prior to treatment with these agents';
measurementUnit = 'grams'}
\{tui = 1000; name = '> 20% within 6 months'; symbol = '> 20'}
\{tui = 2000; name = '> 50% within 6 months'; symbol = '> 50'}
\{tui = 3000; name = 'Positive'; symbol = 'PGC'}
\{tui = 4000; name = 'Negative'; symbol = 'NotPGC'}
\{tui = 5000; name = 'Postmenopausal'; symbol = 'Post';
description = 'Amenorrhea greater or equal 12 months serum:
FSH greater 30IU/l estradiol less 100pmol/l'}

Since all alphabet terms are objects having assigned natural language words as value of the field name, there is a mapping to underlying, natural language word-independent symbols as referred to by the value of the field symbol. In this sense, the structural properties remain the same when we change natural language reference. For example, the same domain terms are represented in German as follows:

\{tui = 1; name = 'Patient'; symbol = 'P1'}
\{tui = 40; name = 'Prämenopausale'; symbol = 'P2'}
\{tui = 50; name = 'mit asymptomatischen Leiomyomata'; symbol = 'P44'}
\{tui = 60; name = 'Abnorme uterine Blutungen (keine Leiomyomata)'; symbol = 'P43';
description = 'Die Blutung ist für die Patientin gesellschaftlich und/oder hygienisch inakzeptabel.'
The representation model of meaning of terms

\{(tui = 100; name = 'Starke Beeinträchtigung'; symbol = 'MImpair';
  description = 'Während eines Zeitraums von 3 Monaten bleibt die Patientin für mindestens
  1 Tag pro Monat wegen Schmerzen oder Unwohlsein zu Hause (d.h. sie geht nicht zur Arbeit
  und/oder nimmt an keinen sozialen Aktivitäten teil).')
\}

\{(tui = 200; name = 'Geschätztes Uterusgewicht'; symbol = 'EVW';
  description = 'Basiert auf klinischer Erfahrung, in Gramm geschätzt.
  280 Gramm entsprechen 12 Schwangerschaftswochen (gemessen vor einer allfälligen GnRH
  Analoga Behandlung).')
\}

\{(tui = 1000; name = '-> 20% innerhalb 6 Monate'; symbol = '-> 20';
\}

\{(tui = 2000; name = '-> 50% innerhalb 6 Monate'; symbol = '-> 50';
\}

\{(tui = 3000; name = 'Positiv'; symbol = 'PGC';
\}

\{(tui = 4000; name = 'Negative'; symbol = 'NotPGC';
\}

\{(tui = 5000; name = 'Postmenopausal'; symbol = 'Post';
  description = 'Der Zeitpunkt 12 Monate nach der letzten spontanen Blutung. Serum:
  FSH mehr als 30 IU/I, Estradiol: weniger als 100 pmol/I'
\}

Given the application domain above, the terms \{1,40,50,60\} are classified as concepts, the terms
\{100,200\} are classified as \{properties\} and the terms \{1000,2000,3000,4000,5000\} are classified
as values.

Domain terms underly a further classification structure which enable an insight into the nature of
terms which is conceived as a contribution to the intensional meaning of terms. This is similar to
operational terms as presented in the next section. The nature of terms is further described by the
names of classes or collections into which terms as information objects have been further classified.
We will call the set of all classes participating in the representation model of meaning of terms \(C_T\).

For example, we might distinguish between Categorical or Numerical Variables as classes into
which terms classified as Properties will be further classified. Complex/Atomic concepts or Un-
ivariate/Bivariate/Multivariate might be further classifications. An example is given in appendix
C concerning instantiation and classification of domain and/or operational terms.

1.2 THE OPERATIONAL TERMS

Similar to domain terms, operational terms are also considered as objects having intensional mean-
ing represented by linguistic elements, explanations stating the purpose of application, etc. Opera-
tional terms are mainly classified as
comparison operators such as

{tui = OP3, name = 'equals to', symbol = '='};
{tui = OP4, name = 'equals to or more than', symbol = '≥'};
{tui = OP5, name = 'equal to or less than', symbol = '≤'};
{tui = OP6, name = 'greater than', symbol = '>'};
{tui = OP7, name = 'less than', symbol = '<'};
{tui = OP9, name = 'not equal to', symbol = '<>'};
{tui = OP10, name = 'between', symbol = 'BETWEEN'};
{tui = OP11, name = 'LIKE', symbol = 'LIKE'};

univariate functions such as descriptive statistical operations

{tui = OP16, name = 'Minimum', symbol = 'MIN'};
{tui = OP17, name = 'Maximum', symbol = 'MAX'};

{tui = OP18, name = 'Frequency', symbol = 'FQ', explanation = 'The number of occurrence of a particular qualitative or categorical value within a total number of observations.'};

{tui = OP19, name = 'Relative Frequency', symbol = 'RFQ', explanation = 'The number of occurrence of a particular qualitative or categorical value divided by the total number of observations. The relative frequencies provide the most relevant information as to the pattern of the data. One should also state the sample size, which serves as an indicator of the credibility of the relative frequencies.'};

{tui = OP20, name = 'Mean Value', symbol = 'AV', explanation = 'The calculated average.'};

{tui = OP21, name = 'Median Value', symbol = 'CV', explanation = 'The value under which fifty percent of all values lie.'};

{tui = OP22, name = 'Summary', symbol = 'SUM', explanation = ' '};
{tui = OP23, name = 'Standard Deviation', symbol = 'SD', explanation = ' '};

etc.,

which are usually applied for the summary description of data concerning a single variable (property term). Further functions for the study of bivariate or multivariate data are also considered, when one wishes to discover relationships which might exist between the variables (property terms), or
how strong the relationships appear to be supported, and whether one variable of primary interest can be effectively predicted from information on the values of the other variables.

\textit{Bivariate functions} operate over two variables such as

\textit{Gender} and \textit{Type of occupation} of patients,

\textit{Height} and \textit{Temperature} of snow, etc,

Typical examples of such operational terms are

\{tui = OP100, name = 'Cross-classification', symbol = 'CC', explanation = 'Tabulation of data related to two variables.';\}

\{tui = OP101, name = 'Scatter diagram', symbol = 'SD', explanation = 'Study of the relationship between two variables.';\}

\{tui = OP102, name = 'Correlation-coefficient', symbol = 'CCF', explanation = 'Calculation of numerical measure of strength of the linear relation between two variables.';\}

\{tui = OP103, name = 'Linear regression', symbol = 'LR', explanation = 'Prediction of one variable from another.';\}

etc.

Accordingly, \textit{multivariate functions} can enter the set $T^F \subset T$ of alphabet terms.

Since operational terms are domain application independent, we refer to the set $T^F \subset T$ which need not be instantiated each time a new application domain is addressed.

\section{REPRESENTATION OF CONTEXT OF TERMS}

Representing the intentional meaning of terms resulted in information objects which form an information space. Connecting these objects to each other forms a contiguity space which is conceived as the context of terms (section 2.1). Additionally, we establish a graph based structure of the MDDQL alphabet model, where contiguity is the only directly experienced elementary relation between two terms in a potential term-sequence formed on the base of a contiguous space of discrete objects. Therefore, this property can be used as the basis of defining a constructive grammar (see also next chapter).

However, the context of terms is also expressed on the base of constraints which can be assigned to information objects (section 2.2). Constraints make the validity of connectionism relevant to a
particular state of affairs. Thus it is possible to express exceptional statements as well as to check for semantic consistency among terms included in the same statement. Currently, representation of constraints is achieved by using propositional logic with all elements being subsets of the set of alphabet terms.

2.1 THE CONNECTIONISM MODEL

The purpose of the connectionism model is to provide a structured, contiguity space within which terms are located. The underlying structure is a graph, where terms are represented as nodes. Alphabet terms are represented, however, as objects rather than symbols forming a large information space of querying alphabet $I_A \equiv I_D \cup I_P$, with $I_D$ the space formed by domain terms and $I_P$ the space formed by operational terms. Note that $I$ is equivalent to $T$. $I$ is chosen as representation symbol of the set of query alphabet terms in order to denote that terms are conceived as Information Objects underlying an Interpretation rather than simple Terms or symbols. Therefore, the nodes of the graph carry the most semantic information of terms.

Connecting alphabet terms together provides an organisational structure for $I_A$ to be used as navigation space when constructing multi-lingual queries by synthesizing terms as suggested by the system rather than syntactic/semantic parsing of an already formulated query. However, an extension of $I_A$ through $C_T$ as the set of all defined classifications of alphabet terms must be considered, since assignment of operational terms to domain ones is done at the classification level of terms. Therefore, the entire information space is given by $I \equiv I_A \cup C_T$.

The connectionism model supports two categories of links: a) the (interconnecting) links, e.g., connecting concepts to properties and properties to values - the latter form well-restricted value domains, and b) the (recursive) links that connect terms of the same class to each other. Links also hold within $C_T$ as well as between $I_A$ and $C_T$.

Figure 5.2 depicts the graph formed by $I_A \cup C_T$ which consists of various layers each of which belongs to one of two categories: a) the category of term instances which refers to concept-, property-, value terms, and operational terms. They all constitute the query language alphabet $I_A$, b) the category of term classes $C_T$ which includes the layers with all known classifications $C_T$ of domain and operational terms, respectively. Interconnecting links between $I_A$ and $C_T$ indicate the membership of term instances to particular classes, whereas the recursive links within $C_T$ indicate the classification structures holding among term classes.

Formally speaking, the information space $I$ is conceived as a multi-layered directed acyclic graph - $ML$-DAG defined as follows:
Definition 1.0: An ML-DAG \((V, E)\) is a directed acyclic graph where \(V\) is the set of vertices \(v_1, v_2, ..., v_n, n \in N\), \(N\) is the set of natural numbers and \(E\) the set of edges \((v_{n-1}, v_n), v_{n-1}, v_n \in V\). \(V\) is partitioned into layers \(L_1, L_2, ..., L_m, m \in N\) such that \(L_i \subseteq V, i = 1, ..., m\) and \(L_i \cap L_j = \emptyset\) for any \(i \neq j\). Additionally, \(E_R \subseteq E\) is the set of recursive links and \(E_I \subseteq E\) the set of interconnecting links, i.e. \(E_R\) consists of edges \((v_i, v_j)\) where \(v_i \in L_k\) and \(v_j \in L_k\) and for \(E_I\) it holds that \(v_i \in L_k, v_j \in L_p\) where \(k \neq p\).

Considerations for the set of domain alphabet terms \(I_D\). The set of interconnecting links \(E_I\) can be defined in database theoretic terms as follows. We assume that there exists a countably finite set of attributes \(att\), i.e. domain terms classified as properties, and that a countably finite set of values \(dom\), disjoint from \(att\), i.e. domain terms classified as value terms.

The interconnecting links holding between the layers of property terms and value terms establish a mapping \(Dom\) on \(att\) such that \(Dom(A)\) is a set called the domain of \(A \in att\). Moreover, the interconnecting links holding between concept terms and property terms establish a mapping \(Pr\) on the countably finite set \(con\) of domain terms standing for concepts such that \(Pr(C), C \in con\), is a set called the characteristic properties of \(C\).

In addition, recursive links enable the recursive structuring, i.e. classification hierarchies or part-of relationships of concepts, properties, and/or values, respectively. This allows recursively structured
(multi-dimensional) properties as well as recursively structured value domains. It is possible to refer to the reflexive mappings $R$ on $Con$, $att$, $dom$, respectively, such that $R(Con)$ is a set of constituent concepts, $R(att)$ is the set of constituent properties, $R(dom)$ is the set of constituent values. The power sets $P(Dom(A))$ and $P(A(C))$ are also allowed as a range of the mappings $Dom$ and $A$, respectively.

Given that $N : M$ mappings are allowed for both kinds of links, it is possible to assign the same property to more than one concept as well as the same value to more than one property. The same holds for the recursive links, e.g. a concept or property might be connected to more than one complex concept or property. Thus a particular term might participate in the refinement of more than one complex term.

The correspondence between domain alphabet terms and layers of the ML-DAG is given by following identities: $T^D \subseteq V$ is the set of all domain terms, $\{t_c \in T_c \subseteq T^D\}$, where $T_c \equiv L_c$ is the set of terms classified as concepts, $\{T_{c,i} \subseteq T_c\}$ the set of initial concept terms, $\{t_p \in T_p \subseteq T^D\}$, where $T_p \equiv L_p$ is the set of terms classified as properties, $\{t_v \in T_v \subseteq T^D\}$, where $T_v \equiv L_v$ is the set of value terms. The three layers $L_c, L_p, L_v$ correspond to the subsets of domain terms standing for Concepts, Properties and Values or Value Domains, respectively. Moving within the same layer, complex terms are refined or composed on the basis of other terms.

Definition 1.1. An assembled concept, property or value term is a proper path $p_a = v_0, ..., v_n, n \in N$ where $v_i, 0 \leq i \leq n$ belong to the same layer $L_{m_0}$ with $m \in \{c, p, v\}$ and all holding connecting edges belong to $E_R \subseteq E$.

For instance, the concept terms with term unique identifiers $\{1,40,50\}$ (see also section 1.1) are connected with recursive links within $L_c$, which, if they are followed, form the assembled concept patients, premenopausal, with asymptomatic leiomyomata to participate in a query. Since the underlying model is a directed acyclic graph, both assembled concepts $\{patients, premenopausal, fear of cancer\}$ and $\{patients, postmenopausal, fear of cancer\}$ would be allowed to participate in a potential query, since there are outgoing recursive links from both concept terms $\{premenopausal, postmenopausal\}$ leading to the concept term $\{fear of cancer\}$.

Finally, a potential query could be refined by adding relevant properties and/or values such as $\{estimated uterine weight, less than 300 grams\}$, since $\{patients, premenopausal, with asymptomatic leiomyomata\}$ is an assembled concept and it holds that: estimated uterine weight in $L_p$ is a property term (inter)connected with concept term with asymptomatic leiomyomata, value term less than 300 in $L_v$ is also (inter)connected with the property term estimated uterine weight.

Note that assembled properties are also allowed in potential queries such as
\{temperature, at surface\},
\{temperature, at 10cm\},
\{temperature, at 50cm\},
\{coordinate, height\},
\{coordinate, longitude\},
\{coordinate, latitude\}

or assembled values such as

\{Visceralehirurgie, Halschirurgie\},
\{Visceralehirurgie, Gallenchirurgie\},
\{Visceralehirurgie, Darmperforation\},
\{Neurochirurgie, intrakranielle Druckmessung\},
\{Neurochirurgie, Kraniotomie\},

as members of the value domain for the property term sort of intervention.

Considerations for the set of operational terms $I_T$. In contrast to the connectionism model presented so far, with which representation of context of domain terms is expressed at an instance level of terms, representation (extension) of the context through the semantic dependencies holding between operational terms and operands (domain terms of the alphabet) is expressed at the classification level of terms $C_T$.

Given that all domain and operational terms underly a further classification based on their nature (an example is given in appendix C concerning instantiation and classification of domain and operational terms), an assignment of operational terms to operands (domain terms) is defined such a way that associations hold among classes of terms.

For example, if an operational term is classified as univariate function, it can be applied only to single properties within a current query context, since there is an association holding between the class \{Single Property\} as subclass of \{Property\} and the class of operational terms \{Univariate Function\}. An operational term which is a member of the class Bivariate Function can be applied only to terms classified as \{Concept\} or \{Complex Property\}. Moreover, operational terms classified as \{Comparison Operator\} can be assigned only to terms classified as \{Domain Values\}.

In general, potential consideration of a particular operation/operator to be assigned to a particular domain alphabet term within a current query state is a matter of classification of both the operational term(s) and the domain alphabet term as well as the association holding among them. In other words, these associations stand for the sense of which operations/operators could be applied over which domain alphabet terms.
Since the nature of operational terms is application domain independent, classification for these
terms as well as the associations holding between operational and domain terms classification layers
(figure 5.2) are constant. An example of classifications of operational terms is given in section 1.2.

In the following, an example of associations holding at the classification layers of operational and
domain terms is given.

Within the domain terms classification layer:

\{Domain Values\} \rightarrow \{Atomic Value, Interval, Complex Value\}
\{Domain Values\} \rightarrow \{Qualitative Value, Quantitative Value\}
\{Property\} \rightarrow \{Single Property, Complex Property\}
\{Property\} \rightarrow \{Categorical Variable, Numerical Variable\}

Within the operational terms classification layer:

\{Interrelationship Function\} \rightarrow \{Bivariate Function, Multivariate Function\}

and between classification layers of operational and domain terms:

\{Domain Values\} \rightarrow \{Comparison Operators\}
\{Single Property\} \rightarrow \{Univariate Function\}
\{Complex Property\} \rightarrow \{Interrelationship Function\}
\{Entity\} \rightarrow \{Interrelationship Function\}

However, it is necessary but not sufficient to consider associations holding at the classification
layers in order to infer (suggest) a set of semantically consistent operational terms (see also next
chapter). For instance, the operator BETWEEN, despite the fact that it is considered a member of
the class \{Comparison Operators\}, cannot be assigned to all domain alphabet terms classified as
\{Domain Values\}, since it only makes sense to apply it over value intervals and not atomic values.
Similarly, the operator LIKE should be applied only to qualitative or categorical values which are
also atomic.

Focusing on operations applying over more than one variable (argument), we should also take into
consideration special conditions which must be fulfilled such as the number of properties (variables)
involved. Hence, despite the fact that bivariate or multivariate operations (functions) are potential
candidates of assignment to entity terms or complex property, a bivariate function should be applied
only if, the number of properties (variables) involved is exactly two (2), for example.

Therefore, extra-ordinary or exceptional behaviour of particular operations/operators within a cer-
tain class/collection of operational terms also needs to be expressed. This can be done by assigning
preconditions to operational terms too, as described in section 2.2.2.
2.2 REPRESENTATION OF CONSTRAINTS

Representation of constraints has been investigated within the context of Constraints Satisfaction Problems (CSP) [Tsa93] for solving a number of problems arising in the areas of operational research, policy making, optimization problems, etc. Constraints are mainly expressed by a set of variables and values connected with arithmetic and/or comparison operators. The problem solution is expressed in terms of \( <\text{variable, value}> \) pairs which satisfy all constraints.

In the context of MDDQL, constraints are currently expressed in terms of information objects as members of the set of alphabet terms. Constraints are conceived as preconditions the satisfaction of which determines their final consideration for inclusion in a potential subsequent query state. Since one of the major focuses of the Meaning Driven Querying Methodology has been the construction of a query in terms of a semantically consistent description of instances, we must exclude semantic inconsistencies in terms of mutually exclusive alphabet terms. Therefore, a further consideration of a particular concept, property, value or operational term is made relevant to the query context, i.e. set of current query terms already considered by the end-user and constituting the current query state.

Preconditions \( p_i \) are assigned to nodes which belong to \( I_A \) as the information space of domain alphabet term instances (see also figure 5.2). Algorithmically navigating through the information space \( I_D \), in order to set up the inferred terms out of the most contiguous information objects to be proposed to the end-user, means that if there are any preconditions assigned to the most contiguous information objects, then they are satisfied due to truth functionality semantics when the current query state (context) is given. If more than one precondition is assigned to an information object, we can specify a circumstance as a set of preconditions assigned to the same information object.

However, in the following, we distinguish between two classes of preconditions; those that apply to domain terms and those that apply to operational terms. Note that preconditions are expressed in terms of term unique identifiers - natural language independent expression - but for the sake of easier understanding, the terms expressed in English have been used.

2.2.1 CONSTRAINTS OVER DOMAIN TERMS

Consider, for instance, the query addressing the instances described as premenopausal patients with asymptomatic leiomyomata and estimated uterine weight less than 300 grams and a growth of \( 20\% \) within 12 months should be rejected as semantically meaningless, since the value term \( 20\% \) within 12 months is not semantically consistent with the value less than 300 grams assigned to the property estimated uterine weight.


**Definition 2.0.** A domain term including precondition \( p_t \) is conceived as a set of literals constituting a propositional formula in conjunctive \( p_t = (p_1 \land p_2 \land ... \land p_m) \) (CNF) or disjunctive \( p_t = (p_1 \lor p_2 \lor ... \lor p_m) \) normal form (DNF), where the propositional variable \( p_i \) is defined as \( p_i = \bigcup(t_n), t_n \in T, i \in 1, 2, ..., m. \) It also holds that \( p_t \in P_T \), where \( P_T \) is the set of all specified domain terms including preconditions over a particular domain of alphabet terms.

**Definition 2.1.** A domain term including circumstance is a subset of \( P_T \) term including preconditions assigned to a particular domain alphabet term.

**Definition 2.2.** Given a query state \( q_{st} \), the truth functionality semantics of \( p_t \) is defined in terms of the mapping of \( p_t \rightarrow true \), due to the appearance of \( p_i = \bigcup(t_n) \) within the current query context \( q_{st} \), otherwise \( p_t \rightarrow false \).

Consequently, a negated propositional variable is interpreted as stating an exclusion of the domain terms of \( p_t = \bigcup(t_n) \) from the current query state (context). If it happens that \( p_t = \bigcup(t_n) \) is part of the current query state, this causes \( p_t \) to be \( false \), since \( p_t \) is negated.

**Definition 2.3.** Given a query state \( q_{st} \), a precondition \( p_t \) is satisfied if and only if \( p_t \rightarrow true \), i.e. \( \forall p_i \in p_t, p_i \rightarrow true \), if \( p_t \) is in conjunctive normal form (CNF), or \( \exists p_i \in p_t, p_i \rightarrow true \), if \( p_t \) is in disjunctive normal form (DNF).

For instance, given that the well-restricted value domain \{20% within 12 months, 50% within 12 months, none\} has been assigned to the property estimated uterine weight through outgoing links from the node representing the object term \{tui = 200, name = estimated uterine weight, ...\}, reachability of a particular node, i.e. inclusion of a particular term is decided upon the truth-conditional semantics of \( P_T \) given the current query context.

In particular, given that the current query context for instances description is premenopausal patients with asymptomatic leiomyomata and estimated uterine weight less than 300 grams, at a further step of refinement of instances description, only the value terms \{50% within 12 months, none\} will be inferred and suggested to the end-user for further consideration, since the precondition \( p_{200} = p \), with \( p = \neg(estimated \ uterine \ weight, \ less \ than \ 300 \ grams) \) assigned to the value term 20% within 12 months turns to be \( false \), i.e. the precondition is not satisfied, since \( (estimated \ uterine \ weight, \ less \ than \ 300 \ grams) \rightarrow true \) (currently appear in the current query context) and \( p \) is negated.
2.2.2 CONSTRAINTS OVER OPERATIONAL TERMS

Since operational terms are also information objects, they become part of the information space considered so far. Therefore, constraints can be assigned to operational terms similarly to those constraints posed for domain terms. However, the definition of preconditions slightly departs from the definition of preconditions defined over domain terms.

Since the assignment of operations to operands is done at the level of classification structures of terms rather than instances of terms, a precondition over operational terms is also defined in terms of classification roles. For example, if the affected domain term is classified as property and as categorical variable, then univariate functions such as calculation of average must be excluded from any suggestion to the end-user. Similarly, if the affected operand is a term classified as domain value and is currently assigned to a property classified as categorical variable, then comparison operators such as greater than or less than must not be suggested.

Definition 2.4: An operational term including precondition \( p_t \) is conceived as a set of literals constituting a propositional formula in conjunctive normal form (CNF) \( p_t = (q_1 \land q_2 \land ... \land q_m) \) or disjunctive normal form (DNF) \( p_t = (q_1 \lor q_2 \lor ... \lor q_m) \), where the propositional variable \( q_i \in C_T \), with \( C_T \) set of all collections (roles) of domain terms, is defined as the collection or class into which a domain (property or value) term has been classified.

For example, the preconditions \( p'_{OP20} = \neg(Categorical\ Variable) \) and \( p'_{OP6} = (Atomic\ Value \land Quantitative\ Value) \) can be defined over the operational terms \( OP20, OP6 \) standing for the calculation of averages and assignment of the comparison operator greater than, respectively.

Definition 2.5: Given a query state \( q_{st} \) and the set of classification roles \( C_{T_q} \in C_T \) of the property/value term which is affected by an operational term, the truth functionality semantics of \( p_t \), is defined in terms of the mapping of \( q_i \rightarrow true \), due to the appearance of \( q_i \) within \( C_{T_q} \), otherwise \( q_i \rightarrow false \).

Definition 2.6: Given a query state \( q_{st} \) and the classification role(s) of the property/value term \( C_{T_q} \in C_T \) which is affected by an operational term, a precondition \( p_t \) is satisfied if and only if \( p_t \rightarrow true \), i.e. \( \forall q_i \in p_t, q_i \rightarrow true \), if \( p_t \) is in conjunctive normal form (CNF), or \( \exists q_i \in p_t, q_i \rightarrow true \), if \( p_t \) is in disjunctive normal form (DNF).

Definition 2.7: An operational term including circumstance is a set of \( P_T \) operational term including preconditions assigned to a particular operational term or class of operational terms.
Each precondition corresponds to exactly one property term. The cardinality of preconditions within $P_T$ determines the number of arguments expected as input arguments.

Satisfiability of $P_T$ is determined on the basis of examining each precondition against the corresponding $C_{T_q}$, which, in turn, becomes an element in the set of contexts of roles $\cup C_{T_q}$. Hence, it is expected that the cardinality of preconditions appearing in $P_T$ equals the cardinality of number of contexts of roles within $\cup C_{T_q}$.

For example, the precondition $p'_{OP20} = \neg(Categorical\ Variable)$ is not satisfied when suggestions of univariate operations have been requested for the property estimated uterine weight - association holding between the class of domain terms $\{Single\ Property\}$ and the class of operational terms $\{univariate\ Function\}$, since it has been classified as $\{Single\ Property\}$ and as $\{Categorical\ Variable\}$, i.e. $C_{T_q} = \{Categorical\ Variable, Single\ Property\}$.

Similarly, given that there is an association holding between the class of domain terms $\{Domain\ value\}$ and the class of operational terms $\{Comparison\ Operators\}$, $p'_{OP6} = (Atomic\ Value \land Quantitative\ Value)$ is only satisfied when the domain term, with which $OP6$ is probably semantically consistent, has been classified as $\{Atomic\ Value\}$ and $\{Quantitative\ Value\}$, i.e. $C_{T_q} = \{Domain\ Value, Atomic\ Value, Quantitative\ Value\}$. Therefore, despite the fact that it is a comparison operator, it would be excluded from the set of suggested operational terms, if it does not happen to be the case that both $\{Atomic\ Value, Quantitative\ Value\}$ are members of $C_{T_q}$.

Moreover, assuming that the set of preconditions $P_{OP100} = \{p_1 = (Single\ Property), p_2 = (Single\ Property)\}$ is assigned to the operation of cross-tabulation or classification, this operation will be semantically valid if there are exactly two terms, e.g., $\{Age, Stress\ test\}$, in a given query state which are associated with the complex property term or entity, e.g., $Patient, Coronary\ angiography, Atypical\ angina$, to which the operation might be assigned and are classified as $Single\ Property$. This means that $\cup C_{T_q} = \{\{Single\ Property\}\}$. In other words, functions which operate over two or more arguments (variables) are assigned a precondition for each argument type. The precondition should depict the nature of the expected argument. Furthermore, the number of preconditions assigned to such functions also pose a restriction to the number or property terms (variables) within a given query state. This must also be taken into account when a particular function should be inferred by the inference engine of the automaton.

A more detailed description of decisions upon satisfaction of preconditions and/or circumstances is given in the following chapter, which describes the inference engine underlying the suggestions of domain and/or operational terms within a given query state.
Chapter 6

THE STATE AUTOMATON SPECIFICATION AND THE MDDQL

If determinism is true, the laws of nature together with a statement of the conditions of the universe before my birth, entail every true statement about my physical movements. I could refrain from making those movements only if I could have falsified the laws of nature or altered those conditions. But even on the hypothetical interpretation I could have done neither of these things.

—Peter van Irwagen, The Incompatibility of Free Will and Determinism, Free Will

Since we had an insight into the representation model of meaning of the alphabet to be used for the construction of queries (see previous chapter), we proceed with the specification of the state automaton (Meaning Driven Data Querying Automaton - MDDQA) which underlies the driving mechanism of construction of meaningful or reasonable queries. The state automaton has been implemented as an inference engine (component Inference/State transition Logic in figure 6.1) operating over the domain alphabet database. It is this component that will be discussed in this chapter.

Given that the state automaton receives as input alphabet not only the set of term symbols but also the represented meaning of terms, the specification of the state transition algorithms (see also figure 6.1) for the automaton has been done by also taking into consideration the represented meaning of terms. Hence, input alphabet and state transition algorithms will constitute an automaton which is not only conceived as an abstract machine guiding the end-user oriented query construction process, but also underlies the specification of a meaning based (sub)language such as the Meaning Driven Data Query Language MDDQL.

As an abstract machine, the $MD\!D\!QA$ is formally defined in section 1. The states are conceived as assemblies of currently included query terms, and therefore, the automaton needs to keep track of what has been read previously. The mapping functions for moving among query states are described

---

1Collected papers, Oxford Readings in Philosophy, 1982
in section 2., where three alternatives are given: a) starting the query construction process by using terms classified as entities, i.e. entry points of the information space of terms (ML-DAG) would be nodes represented entity terms, b) starting the query construction process by typing any term, i.e. entry points of the information space of terms (ML-DAG) would be any nodes, c) construction of set of premises for classification rules.

1. THE DEFINITION OF THE MEANING DRIVEN QUERYING AUTOMATON

Any meaning based constructed query in MDDQL is always a syntactically and semantically correct query. This is due to the fact that queries are synthesized from system suggested terms whose semantic consistency is a consequence of reaching the current query state.

The MDDQA accepting MDDQL queries is defined as an abstract machine

\[ M = \{ S_q, \Sigma_q, \Gamma_q, \delta_q, q_0, P_q, F_q \}, \]

where:

- \( S_q \) is a finite set of query states \( q_i, i \geq 0 \),
- $\Sigma_q \equiv T \equiv I_A$ is the input alphabet, with $\Sigma_{qD} \equiv T^D \subset \Sigma_q$ the subset of domain alphabet terms and $\Sigma_{qF} \equiv T^F \subset \Sigma_q$ the subset of operations alphabet terms.

- $\Gamma_q$ is a subset of $\Sigma_q$, called the current query state, realized by a stack.

- $P_i$ is the set of propositional formulas $p_i$ each one expressing a term-including precondition for which a truth assignment holds (see also chapter 5, section 2.2).

- $\delta_q$, the transition function, is a mapping $\delta_q : (\Sigma_q^* \cup \{\text{strip}\}) \rightarrow S_q$, where $q_{i-1} \xrightarrow{p_{q_{i-1}}} q_i$, with $q_{i-1}, q_i \in S_q, i = 1, \ldots, n$.

- $q_0$ is a particular state of $S_q$ called the initial query state.

- $F_q \subseteq S_q$ is the set of accepting or final query states.

The mapping function $\delta_q$ is defined over $\Sigma_q^*$ and not simply $\Sigma_q$, since more than one input query term can be read before the query is being processed. A query state $q_i \in S_q, i \geq 0$, includes the subset of query terms $q_i \subset \Sigma_q \equiv T \equiv I_A$ for which all $t \in q_i$ are connected by some proper directed path in $I$ with $I_A \subset I$, and $\forall t, t \in q_i, p_t \rightarrow \text{true}$.

A state transition diagram is depicted in figure 6.2 in order to illustrate a state automaton in terms of query states. Since each query state is conceived as a subset of the set of alphabet terms $T \equiv I_A$, all query states $\{q_0, q_1, q_2, q_3, q_4, q_5\}$ include alphabet terms presented with linguistic elements of some natural language. $q_0$ is the initial query state and $q_f$ is the final one. Consequently, each subsequent state includes additional terms (application domain or operational) and is considered to be a more restrictive description of the previous instance(s).

![Figure 6.2. A state transition diagram for MDDQA](image-url)
However, the cardinality of the set of all potential query states is bounded by the countably finite set of all proper directed paths as expressed by the holding links of the graph \( G(ML - DAG) \) representing the space of information objects \( I \) (see also figure 6.3).

The specified automaton can be considered an abstract machine which consists of an input tape, a finite control and a head which is moved by the finite control mechanism to the left or right on the input tape and sets the automaton to a particular (query) state. Stacks are also given, since the automaton needs to remember the set of current terms as well as those of the set of previously visited query states. This is similar to devices specified for push-down automata. Strip move is also allowed where the finite control mechanism sets the automaton to a previous query state and moves the head to the left of the input tape. This is similar to deletion operations over the current set of query terms.

A query construction is a move from a current query state to a more descriptive or refined one which is guided by decisions (state transitions) that depend on the current query context as well as the model used to represent meaning of terms. The end-user, however, is part of the finite control mechanism, and therefore partly determines which next state to move to through choices of terms based on the intentional meaning of terms (see also chapter 5, section 1). Therefore, the end-user can choose only query states, which are semantically consistent with the current one.

Since query state transitions are based on semantically consistent input terms as suggested by the system as well as on end-user choices based on term interpretation and meaning, the MDDQA is conceived as an interactive acceptor of meaningful queries formulated in MDDQL. This allows syntax/semantic parsing of a query to be replaced with end-user/system interaction, whereby the semantics of query alphabet terms are taken into account during query construction.

In the following section, state transition algorithms are described which realize the \( \delta \) function among semantically consistent query states. In particular, we distinguish among three major cases of interaction strategy for the query construction:

- starting query construction (formulation) with entity terms at the concept assembling layer (figure 6.3),
- starting with any arbitrary term from the set of domain alphabet terms \( t \in I_D \),
- attribute/value pairs for premises of classification rules.

General characteristics of the state transition algorithms are that

- all algorithms work with entry points represented by nodes belonging to one of the concepts-properties-value domains layers (figure 6.3), i.e. only terms which belong to the domain alphabet are considered first.
all algorithms work either with or without the set of operational terms depending on the profile of queries needed.

2. THE STATE TRANSITION LOGIC AND ITS ALGORITHMS

In the following, the token \(< SELECT >\) refers to the choices made by the end-user. Note that \(< SELECT >\) tokens indicate the possible choice of the end-user for a term to extend the current context (state) of the query out of a set of suggested (inferred) terms. Since the alphabet terms are represented as information objects, it is also possible to have a look at additional properties of the terms such as definition, measurement unit, icons, etc. which provide a deeper insight into the intentional meaning of a particular term (concept, property, value), before selection.

2.1 EXCLUSION OF OPERATIONAL TERMS

Starting with entities. The first algorithm refers to the function of the automaton when the starting terms (entry points of the graph) are concept terms classified as entities, i.e. possible entry points are those nodes of \(I_a\) which belong to the concepts (assembling) layer. In other words, a top-down navigation strategy of the concepts-properties-value domains assembling layers of the graph (figure 6.3) is realized. The algorithm is illustrated with an example from the application domain of 'Information Management System for Mines Actions' some terms of which will accompany the
algorithm description. A subset of these terms and their conceptualization is depicted in figure 6.4 which represents part of figure 6.3.

BEGIN Algorithm 1.0

Start query construction.

Suggest set of natural languages
e.g., \{English, German\}

\(< SELECT >\) a natural language in which terms should appear.
e.g., \{English\}

Suggest initial entity terms to start with.
e.g., \{Explosive devices, Country, Incidents\}

\(< SELECT >\) an initial entity term
e.g., \{Explosive devices\}

Move to the initial query state \(q_0\).
e.g., \(q_0 = \{Explosive devices\}\)

Set initial query state \(q_0\) to be the current state \(q_c\)

Repeat

\(< SELECT >\) a term in current query state \(t_s \in q_c\)
while there are outgoing edges from \( t \in q_c \) to some nodes (terms) with \( t \in I_D \) do

  get connected node

  if connected node is not already in \( q_c \) then

    if there is a set of preconditions assigned to node then

      if set of preconditions is satisfied

        add node to list of inferred nodes (terms) \( T_I \)

      else ignore node

    else add node to list of suggested nodes

    else ignore node

end of while

Suggest inferred terms \( T_I \) to end-user for selection as information objects (chapter 5, section 1)
e.g., \{caused by, manufactured in, category, nomenclature, material, shape, purpose\} when \( t_s \in q_c \)
is \{Explosive devices\}

< SELECT > term(s) from \( T_I \)
e.g., \{manufactured in, purpose\}

Add selected term(s) to the current query state \( q_c \) by assigning them to \( t_s \)
e.g., \{Explosive devices, manufactured in, purpose\}

Move to \( q_j \) which is a subsequent query state of \( q_c \)

Make \( q_j \) the current query state \( q_c \equiv q_j \)

until final query state is reached
e.g., \{Explosive devices, purpose, manufactured in, country, name, United Kingdom, Switzerland\}

END Algorithm 1.0

Following this example, the term \{country\} belongs to the set of suggested (inferred) terms \( T_I \) when \( t_s = \{manufactured in\}, t_s \in q_c \) and \( q_c \equiv \{Explosive devices, purpose, manufactured in\} \) (see also figure 6.5), whereas the term \{name\} belongs to the set of suggested (inferred) terms \( T_I \) when \( t_s = \{country\}, t_s \in q_c \) and \( q_c \equiv \{Explosive devices, purpose, manufactured in, country\} \). Similarly, the terms \{United Kingdom, Switzerland\} belong to the set of suggested (inferred) terms \( T_I \) when \( t_s = \{name\}, t_s \in q_c \) and \( q_c \equiv \{Explosive devices, purpose, manufactured in, country, name\} \) (see also figure 6.6).

Note that the final query state of the given example, conceived as a set of query terms, is equivalent to the expression of the question What is the purpose of explosive devices manufactured in countries named as United Kingdom and Switzerland? formulated in English. However, starting with terms classified as entities reflects the philosophy of establishing a central theme or subject of discussion such as talking about "explosive devices" before proceeding with further terms which might refine the intended question or query.
In the following, we look at the general case in which any arbitrary term might be a starting point of query construction.
The general case. The second state transition algorithm refers to the function of the automaton when any domain alphabet term can be considered as entry point, i.e. the end-user can start with an arbitrary query term which belongs to the set of domain alphabet terms. In other words, the entry node of the graph (figure 6.3) could be any node which belongs to one of the assembling layers of the graph, i.e. concepts, properties, value domains.

The token \textless TYPE \textgreater also refers to a term typing action of the end-user:

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Start} query construction.
\State \textbf{Suggest} set of natural languages
\hspace{1em} e.g., \{English, German\}
\State \textless SELECT \textgreater a natural language in which terms should appear.
\hspace{1em} e.g., \{English\}
\State \textless TYPE \textgreater the name of the term to be clarified (entry term \(E_t\))
\hspace{1em} e.g., \{purpose\}
\While there is an entry term \(E_t\)
\hspace{2em} \While there are term nodes in \(I_D\) the name of which matches \(E_t\)
\hspace{3em} e.g., \{TUI=500, TUI=600\}
\hspace{3em} \textbf{get} matched term node
\hspace{3em} \textbf{get} all connected nodes with incoming edges - navigating backwards \(I_D\)
\hspace{3em} \textbf{put} into set of clarifying term nodes \(C_T\)
\hspace{1em} \EndWhile
\hspace{1em} \textbf{suggest} set of clarifying term nodes \(C_T\) to the end-user
\hspace{1em} e.g., \{Mines clearing action, Explosive Devices\}
\hspace{1em} \textless SELECT \textgreater a clarifying term \(C_t\) out of set \(C_T\)
\hspace{1em} e.g., \{Explosive Devices\}
\hspace{1em} \textbf{Add} pair of connected term nodes to list of visited path
\hspace{1em} \textbf{Set} \(C_t\) to term to be clarified \(E_t\)
\EndWhile
\State \textbf{Make} out of list of term nodes from visited path the current query state \(q_c\) e.g., \{Explosive Devices, purpose\}
\EndAlgorithm
\end{algorithmic}

and continuing similar to the previous case (algorithm 1.0):

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Repeat}
\hspace{1em} \textless SELECT \textgreater a term in current query state \(t_s \in q_c\)
\hspace{1em} e.g., \{Explosive Devices\}
\hspace{1em} \While there are outgoing edges from \(t \in q_c\) to some nodes (terms) with \(t \in I_D\)
\EndWhile
\EndAlgorithm
\end{algorithmic}
get connected node
if connected node is not already in \( q_c \) then
    if there is a set of preconditions assigned to node then
        if set of preconditions is satisfied
            add node to list of inferred nodes (terms) \( T_I \)
        else ignore node
    else add node to list of suggested nodes
else ignore node
end of while

Suggest inferred terms \( T_I \) to end-user for selection as information objects (chapter 5, section 1)
e.g., \{caused by, manufactured in, category, nomenclature, material, shape\}
< SELECT > term(s) from \( T_I \)
e.g., \{manufactured in\}
Add selected term(s) to the current query state \( q_c \) by assigning them to \( t_s \)
e.g., \{Explosive devices, manufactured in, purpose\}
Move to \( q_j \) which is a subsequent query state of \( q_c \)
Make \( q_j \) the current query state \( q_c \equiv q_j \)

until final query state is reached
e.g., \{Explosive devices, purpose, manufactured in, country, name, United Kingdom, Switzerland\}

END Algorithm 2.0

This algorithm reflects the philosophy of constructing a query when the meaning of the entry term has already been clarified completely. For instance, starting with the term purpose, i.e., let's talk about "purpose", the system suggests Mines clearing action and Explosive devices as potential clarifications of "purpose", i.e., to give an answer to the question are we talking about the purpose of Mines clearing action or that of Explosive devices? before proceeding with the further refinement of the query/question.

The case of premises in classification rules. The third state transition algorithm refers to the function of the automaton when attribute/value pairs constituting a set of premises to be found in the antecedent part of a classification rule are constructed.

Assuming that:

- all terms appearing in the antecedent part of all classification rules have somehow entered the three (concepts-, properties-, value domain assembling) layers which constitute the subset \( I_D \) (domain alphabet terms) of the entire query information space, as depicted in figure 6.7 (part of 6.3) - switch to a medical domain alphabet as an example, and that
- decision tables have been used for the representation of mappings between the antecedent part of classification rules and the conclusions,

the algorithm is similar to algorithms 1.0, 2.0. However, for the sake of clarification, we will refer to the algorithm 1.0 with a different example (classification rules for decision making of interventions in Cardiology).

![Diagram](image)

**Figure 6.7.** Domain alphabet terms for premises of classification rules for appropriateness of interventions in Cardiology.

In the following, the token <SELECT> refers to the choices made by the end-user. To illustrate, we use patients' profiles as the entry point to the $I_D$.

**BEGIN Algorithm 1.1** (as a special case of algorithm 1.0).

**Start** query construction.

**Suggest** set of natural languages

*e.g.,* {English, German}

<SELECT> a natural language in which terms should appear.

*e.g.,* {English}

**Suggest** further entity terms to start with.

*e.g.,* {Coronary angiography, Revascularization}

<SELECT> an entity term

*e.g.,* {Coronary angiography}

**Move** to the initial query state $q_0$.

*e.g.,* $q_0 = \{\text{Patient, Coronary angiography}\}$

**Set** initial query state $q_0$ to be the current state $q_c$.
Repeat
<SELECT> a term in current query state \( t_s \in q_c \)
while there are outgoing edges from \( t \in q_c \) to some nodes (terms) with \( t \in I_D \) do
  get connected node
  if connected node is not already in \( q_c \) then
    if there is a set of preconditions assigned to node then
      if set of preconditions is satisfied
        add node to list of inferred nodes (terms) \( T_I \)
      else ignore node
    else add node to list of suggested nodes
  else ignore node
end of while
Suggest inferred terms \( T_I \) to end-user for selection as information objects (chapter 5, section 1)
e.g., \{atypical angina, chronic stable angina, acute myocardial infarction, ...\} with \( t_s = \text{coronary angiography} \)
<SELECT> term(s) from \( T_I \)
e.g., \{atypical angina\}
Add selected term(s) to the current query state \( q_c \) by assigning them to \( t_s \)
e.g., \{Patient, Coronary angiography, atypical angina\}
Move to \( q_j \) which is a subsequent query state of \( q_c \)
Make \( q_j \) the current query state \( q_c = q_j \)
until final query state is reached
e.g., \{Patient, Coronary angiography, atypical angina, gender, female, stress test, positive\}

END Algorithm 1.1

Following this example, the terms \{gender, stress test\} belong to a set of suggested (inferred) terms \( T_I \) when \( t_s = \{atypical angina\}, t_s \in q_c \) and \( q_c \equiv \{\text{Patient, Coronary angiography, atypical angina}\} \), whereas the term \{female\} belongs to the set of suggested (inferred) terms when \( t_s = \{gender\}, t_s \in q_c \) and \( q_c \equiv \{\text{Patient, Coronary angiography, gender}\} \). Similarly, the term \{positive\} belongs to the set of suggested (inferred) terms when \( t_s = \{stress test\}, t_s \in q_c \) and \( q_c \equiv \{\text{Patient, Coronary angiography, gender, female, stress test}\} \).

The final query state reflects the set of premises the interpretation of which, in English, would be

IF patient for coronary angiography AND
patient has atypical angina AND
gender is female AND
stress test is positive THEN....
Note that the set of final query states is tied to the set of entries in the decision (mapping) tables.

**Satisfaction of preconditions.** The definitions given in chapter 5, section 2.2, concerning "satisfaction" of preconditions eventually assigned to particular nodes require that the state transition algorithms also take into consideration the mappings of preconditions to the truth values \{true, false\}. However, in order to decide which mapping is valid, the state transition algorithm(s) must be extended by a function which receives as input the set of preconditions and the current query state \(q_c\) and returns a truth value.

In the following, a description of the algorithm to support the truth value semantics as related to preconditions assigned to domain alphabet terms is given. Assuming that a node with assigned precondition(s) has been reached during navigation through the space of information objects \(I_D\) as described by algorithm 1.0, consideration of this node within the set of suggested (inferred) nodes (terms) \(T_s\) depends upon the determined truth value semantics of preconditions as follows.

An example will also be given on the basis of a scenario where is assumed that

- the terms \{Patient, Coronary angiography, atypical angina, gender; female\} are within the current query state \(q_c\),
- the precondition \{(Patient, atypical angina) \& \neg (gender, female)\} is assigned to node (term) \{risk factors\},
- the terms \{gender, stress test, risk factors\} have incoming edges from \(T_s = \{atypical angina\}\).

BEGIN Algorithm 3.0

get term circumstances (set of assigned preconditions) \(P_T\)
e.g., \(P_T \equiv \{p_t = (Patient, atypical angina) \& \neg (gender, female)\}\)
get normal form of \(P_T\) (conjunctive or disjunctive)
e.g., \(P_T \rightarrow CNF\)
assign default truth value \textit{true} to \(P_T\), i.e. \(P_T \rightarrow \text{true}\)
while there are preconditions in \(P_T\) (circumstances) assigned to term \(T\) and \textit{extLooping} = \text{true}
do
  get precondition \(p_t\)
e.g., \(p_t = (Patient, atypical angina) \& \neg (gender, female)\)
  get normal form of precondition (conjunctive or disjunctive)
e.g., \(p_t \rightarrow CNF\)
  assign default truth value \textit{true} to \(p_t\), i.e. \(p_t \rightarrow \text{true}\)
while there are literals \(p_i \in p_t, i \geq 0\) and \textit{intLooping} = \text{true}
  get literal \(p_i\)
e.g., \( p_0 = (\text{Patient, atypical angina}) \) (first pass)

\( e.g., p_1 = \neg (\text{gender, female}) \) (second pass)

if it holds that \( t_n \in p_i \) and \( t_n \in q_c \) then

if \( p_i \) is negated then

assign truth value \textit{false} to \( p_i \)

\( e.g., p_1 \rightarrow \text{false} \) (second pass)

else

assign truth value \textit{true} to \( p_i \)

\( e.g., p_0 \rightarrow \text{true} \) (first pass)

if \( p_i \rightarrow \text{false} \) and normal form of precondition \( p_t \) is conjunctive then

set \textit{intLooping} = \textit{false}

assign \( p_t \rightarrow \text{false} \)

\( e.g., p_t \rightarrow \text{false} \) (second pass)

else if \( p_i \rightarrow \text{true} \) and normal form of precondition \( p_t \) is disjunctive then

set \textit{intLooping} = \textit{false}

else

assign to \( p_t \) the truth value of \( p_i \)

end of while

if \( p_t \rightarrow \text{false} \) and normal form of \( P_T \) is conjunctive then

set \textit{extLooping} = \textit{false}

assign \( P_T \rightarrow \text{false} \)

\( e.g., p_t \rightarrow \text{false} \)

else if \( p_t \rightarrow \text{true} \) and normal form of \( P_T \) is disjunctive then

set \textit{extLooping} = \textit{false}

else

assign to \( P_T \) the truth value of \( p_t \)

end of while

\textit{END Algorithm 3.0}

Following the example given above, despite the fact that there is an incoming edge to the term \{\textit{risk factors}\} from the term \{\textit{atypical angina}\}, the term \{\textit{risk factors}\} will not be considered as an element of the set of suggested (inferred) terms. This is due to the fact that \( T_s = \{\text{atypical angina}\} \) is the activated term by the end-user within the current query state \( q_c = \{\text{Patient, Coronary} \}. \)
angiography, atypical angina, gender, female} and the assigned precondition to the term \{risk factors\} is not satisfied within this query state.

This reflects the situation of exceptional handling of assignments as found in the real world. In other words, the assigned precondition as presented previously would express the fact that risk factors is considered as property of patients with atypical angina only if their gender is male or the equivalent expression risk factors is considered as property of patients with atypical angina only if their gender is NOT female.

Terms dependency graphs. From the nature of preconditions as a set of nodes (terms) within the space of information objects $I_D$, it becomes obvious that preconditions render edges or nodes of the multi-layered graph underlying the connectionism model of the representation of context of terms (chapter 5, section 2.1) indeterministic. Their validity depends upon the satisfaction of these preconditions given a particular query state $q_c$.

However, the time of examination of satisfaction of preconditions is a matter of efficiency and terms presentation (suggestion) policy. Since preconditions are examined in order to decide if terms appearing in the same query statement are semantically consistent, there are, in general, two alternatives:

- a progressive strategy where all nodes (terms) are inferred and suggested to the end-user by taking into consideration only the connecting edges, whereby all preconditions related to the query state are examined retrospectively,

- a conservative strategy, where preconditions are examined during inference of nodes (terms) to be suggested, i.e. only those nodes (terms) are inferred (suggested) which have assigned preconditions are satisfied.

In both cases, the final query state includes only terms which are semantically consistent to each other. However, the conservative strategy is more complicated as far as decidability of satisfaction of preconditions is concerned. Since preconditions are expressed in terms of other nodes within the same graph, the time needed to examine the satisfaction of preconditions depends on the size of sub-graphs which are involved as a basis of the current query context.

For example, if a property term is included in the expression of a precondition, then one first needs to know, what the final state of affairs is, which is related to this property term within a given query state before deciding upon satisfiability of the given precondition. For instance, if $\{p_t = (\text{Patient, atypical angina}) \land \neg (\text{gender, female})\}$ is assigned to node (term) \{risk factors\}, then one needs to know what the value of \{gender\} is, which determines satisfaction of precondition upon \{risk factors\}. 


Regardless of the preferred strategy, the "terms dependency graphs" must be elaborated which express all dependencies holding among domain alphabet terms within a final query state. They must be used either to examine preconditions at the final query state or to turn undecidable preconditions into decidable ones during incremental construction of the query.

(a) gender risk factor
   (b) gender risk factor stress test

Figure 6.8. An example of "terms dependency graphs" within the set of domain alphabet terms.

In our example, the "terms dependency graph (path)" of figure 6.8, part (a), has been extended to depict the dependency between \{gender\} and \{risk factors\}. If a further precondition is assigned to the node (term) \{stress test\} such that \(p_{\text{st}} = \{\text{risk factors, less than 2}\}\), the "terms dependency graph (path)" will take the form as depicted in figure 6.8, part (b). If the precondition \(p_{\text{st}} = \{\text{gender, female}\}\) is defined over node (term) \{positive\} as connected to node (term) \{stress test\}, the "terms dependency graph (path)" takes the form of part (c) in figure 6.8.

Formally speaking, all "terms dependency graphs" are defined as set of proper paths (see also definitions in chapter 2, section 1.1) of nodes (terms) \(\{t_1, \ldots, t_n\}\) which belong to some final query state \(q_f \in F_q\), with \(t_i \neq t_j\) for all \(i, j \leq n\), i.e. no cycles are allowed in order to guarantee decidability of satisfaction of preconditions.

However, the decision which nodes (terms) constitute the "terms dependency graphs" is taken on the basis of expression of preconditions. The TDGs provide the basis for the timely decision of satisfaction of preconditions. The following algorithm provides an insight into the construction of "term dependency graphs" (TDGs).

BEGIN Algorithm 4.0

For each node or term in a given final query state to which a precondition has been assigned
**The State Automaton specification and the MDDQL**

**Algorithm 4.0**

In the case that a *progressive* strategy has been chosen, all we need is to traverse the TDGs backwards, i.e., starting from leaf nodes, as related to the final query state. For each leaf or inner node within a particular TDG, we examine the preconditions assigned to them by applying algorithm 3.0.

For example, given the TDGs of figure 6.8, it holds that

- **case (a):** only the precondition assigned to \{risk factors\} needs to be examined against the final query state.
- **case (b):** the precondition assigned to \{stress test\} needs to be assigned first, and then the precondition assigned to \{risk factors\} must be examined.
- **case (c):** the preconditions assigned to \{positive\} and \{risk factors\} can be examined independent from each other (no synchronization is required).

In order to illustrate the effects of examining preconditions retrospectively, consider the scenario of having in a query (final) state the terms \{Patient, Coronary angiography, atypical angina, gender, female, risk factors, less than 2\}. A retrospective examination of the precondition \{(Patient, atypical angina) \land \neg (gender, female)\} will cause the elimination of \{risk factors, less than 2\} from the final query state, since \{risk factors\} is not semantically consistent within this query state.

In the case that a *conservative* strategy has been chosen, we need to consider examination of preconditions in advance. Therefore, in order to avoid undecidability, a "dependency order" of examination of preconditions must be taken into account. The decision upon which preconditions, and therefore, which nodes (terms) are decidable is taken on the basis of the partial graph as resulted by the outgoing edges of concept terms of a particular query state within the information space \(I_D\) and the elaborated TDGs. We will call the resultant partial graph the *scope of relevant preconditions* when a particular query state is given.

For example, figure 6.9 depicts the "scope of related preconditions" as a partial graph of \(I_D\) - exclusion of connecting edges represented by dashed lines - when the concept term \{Patient, Coronary angiography, Atypical angina\} is considered within a given query state. The activation of the term \{Atypical angina\} by the end-user should cause the suggestion of \{Gender, Risk factors,
Stress test}, if a progressive examination strategy would have been chosen for the implementation of the inference engine.

However, having chosen a conservative examination strategy and given that the TDG of figure 6.8, case (b) holds, the inference algorithm will take into consideration these dependencies by suggesting first \{gender\} with the corresponding value domain, and subsequently, examine the precondition of \{risk factors\} for further consideration of this property term. In other words, the examination of the precondition assigned to \{risk factors\} is deferred in that the node (term) becomes a "candidate" or "potential" node (term).

In general, the following algorithm determines "dependency orders" within sets of inferred (suggested) nodes (terms).

BEGIN Algorithm 5.0

Construct partial graph of \( I_D \) as scope of relevant preconditions \( S_p \)
e.g., the partial graph resulted from \{Patient, Coronary angiography, Atypical angina\} (figure 6.9)
get TDGs

while there is a TDG

get TDG
e.g., figure 6.8, part (b)
if all nodes (terms) in TDG are members of \( S_p \) then
e.g., \{Gender \rightarrow Risk factors \rightarrow Stress test\}
exclude all inner or leaf nodes in TDG from set of suggested nodes (terms)
e.g., exclude \{Risk factors, Stress test\}
The State Automaton specification and the MDDQL

**turn** all incoming edges to inner or leaf nodes of $TDG$ within $S_p$ to "red"

e.g., turn edges \{Atypical angina $\rightarrow$ Risk factors\} and
\{Atypical angina $\rightarrow$ Stress test\} to "red"

**while** there are outgoing edges from root nodes of $TDG$

\textbf{set} intermediately connected nodes to potential nodes

e.g., potential node(s) are \{Risk factors\} (first pass)
e.g., potential node(s) are \{Stress test\} (second pass)

\textbf{set} potential nodes to root nodes of $TDG$

e.g., root node is \{Risk factors\} (first pass)
e.g., root node is \{Stress test\} (second pass)

**turn** all "red" incoming edges for potential nodes within $Sp$ to "green"

e.g., edge \{Atypical angina $\rightarrow$ Risk factors\} turned to green (first pass)
e.g., edge \{Atypical angina $\rightarrow$ Stress test\} turned to green (second pass)

i.e., enforcement of node consideration (precondition) checking by
next activation of $t_s = \{\text{atypical angina}\}$

\textbf{end of while}

**end of while**

**END Algorithm 5.0**

Given that the conservative strategy has been chosen for the realization of the terms inference engine, the algorithms 1.0, 2.0, 1.1 need to be adapted in that not simply "outgoing" edges are considered for the inference of terms to be suggested but those "green outgoing" edges, i.e. those edges leading to terms (nodes) which are consistent with the given query context. Therefore, the adaptation applies to the following common part of the information space navigation algorithms:

```
Repeat
  < SELECT > a term in current query state $t_s \in q_c$
  while there are "green" outgoing edges from $t \in q_c$ to some nodes (terms) with $t \in I_D$ do
```

2.2 **INCLUSION OF OPERATIONAL TERMS**

So far, we have described the behaviour of the inference engine and/or meaning driven querying automaton in terms of the query alphabet (application) domain, where inferences concerning operational terms have been excluded. Allowing operational terms (see also chapter 5, section 1.2)
to be members of a particular query state, the inference engine needs to be extended by algorithms operating over a given query state of domain alphabet terms and also infer (suggest) meaningful operations to the end-user.

Inferences of sets of meaningful operations or operators are drawn on the basis of connecting edges holding among classes or collections of term instances (see also chapter 5, section 2). Therefore, the inference of operational term instances is a matter of consideration of connecting edges leading to the classification layers of domain and operational terms (figure 5.2, section 2.1, chapter 5), respectively, as well as the preconditions (chapter 5, section 2.2.2) assigned to operational term instances as information objects.

In the following, we refer to the class or collection of a term instance as the role of term. Since more than one term or role might be relevant for an operation/operator, we refer to the context of roles as the set of all roles assigned to domain terms. Algorithm 6.0 provides the inference mode of sets of operational terms to be suggested for assignment to domain terms. Illustrating examples are given based on the various application domains.
BEGIN Algorithm 6.0

If $q_c$ is the current query state
e.g., \{Patient, Coronary angiography, atypical angina, age, 48, gender, male, stress test, positive, risk factors, less than 2\}
and $t_s$ is the activated term by the end-user
e.g., $t_s = \{48\}$

get the context of roles $r_{t_s}$ of $t_s$
e.g., $r_{t_s} = \{Domain Value, Atomic Value, Quantitative Value\}$
get the roles $R_{t_f} = \bigcup r_{t_f}$ of operational terms associated with $r_{t_s}$
e.g., $R_{t_f} = \{Comparison Operator\}$
set the operational term instances $t_f \in R_{t_f}$ to list of candidate operational terms
e.g., \{OP3, OP4, OP5, OP6, OP7, OP9, OP10, OP11\} (section 1.2, chapter 5)
while there are operational term instances $t_f$ in the list of candidate operational terms
  if there is an assigned precondition $p'_t$ to $t_f$ then
    e.g., $p_{op10} = (Atomic Value \land Quantitative Value)$
e.g., $p_{op11} = \neg(Quantitative Value)$
    if $p'_t$ is satisfied due to definitions 2.5, 2.6 (section 2.2.2, chapter 5) then
      add to list of inferred (suggested) operational terms
    else
      ignore $t_f$
e.g., ignore \{OP10, OP11\}
  else
    add to list of inferred (suggested) operational terms
end of while
suggest list of inferred (suggested) operational terms
e.g., \{OP3, OP4, OP5, OP6, OP7, OP9\}
< SELECT > operational term
e.g., \{OP4\}

END Algorithm 6.0

In this case, the final query state would be e.g., \{Patient $\rightarrow \{Coronary angiography, atypical angina\}$, age $\rightarrow \{OP4(\text{equals or more than}) \rightarrow 48\}$, gender $\rightarrow$ male, stress test $\rightarrow$ positive, risk factors $\rightarrow$ less than 2\}. This is equivalent to the query "Which patients are candidates for coronary angiography with atypical angina having age which is equal or more than 48, positive stress test and less than 2 risk factors?".

Note that the algorithm for examination of satisfaction of preconditions assigned to operational terms is similar to the algorithm 3.0 described in the previous section with slight modifications:
a) the query context, against which the expressed precondition is examined, is the set \( C_{T_q} \) of classes/collections of the domain term instance(s) affected by the operation/operator - context of roles (see also definition 2.6, section 2.2.2, chapter 5), b) the literals of preconditions refer to classes/collections (roles) of the domain terms.

Recall that operational terms are information objects too, and therefore, they are suggested as objects rather than simple symbols. Hence, the intentional meaning, in addition to the linguistic elements, provides an insight into the meaning of operators or operations before final selection by the end-user. This becomes more obvious, if univariate functions such as \{OP18, OP19, OP20, ...\}, or bi-/multivariate functions are considered.

For example, the following is the form of algorithm 6.0 when a univariate function is applied or inferred for properties within a given query context:

**BEGIN Algorithm 6.0**

if \( q_e \) is the current query state  
* * e.g., \{Patient, Coronary angiography, atypical angina, age, gender, male, stress test, positive, risk factors, less than 2\}  
and \( t_s \) is the activated term by the end-user  
* * e.g., \( t_s = \{age\} \)

get the context of roles \( r_{ts} \) of \( t_s \)  
* * e.g., \( r_{ts} = \{Single Property, Numerical Variable\} \)

get the roles \( R_{t_f} = \bigcup_{t_f \in R_{t_f}}\) of operational terms associated with \( r_{ts} \)  
* * e.g., \( R_{t_f} = \{Univariate Functions\} \)

set the operational term instances \( t_f \in R_{t_f} \) to list of candidate operational terms  
* * e.g., \{OP18, OP19, OP20, OP21, OP22, OP23\} (section 1.2, chapter 5)

while there are operational term instances \( t_f \) in the list of candidate operational terms  
  if there is an assigned precondition \( p'_{t_f} \) to \( t_f \) then  
    e.g., \( p_{op18} = (Domain Value), p_{op19} = (Domain Value) \)  
    if \( p'_{t_f} \) is satisfied due to definitions 2.5, 2.6 (section 2.2.2, chapter 5) then  
      add to list of inferred (suggested) operational terms  
      else  
        ignore \( t_f \)  
        e.g., ignore \{OP18, OP19\}  
    else  
      add to list of inferred (suggested) operational terms  
      e.g., \{OP20\}

end of while  

suggest list of inferred (suggested) operational terms
e.g., \{OP20, OP21, OP22, OP23\}

\(<\textit{SELECT}>\) operational term

e.g., \{OP20\}

**END Algorithm 6.0**

In this case, the final query state would be e.g., \{Patient \rightarrow \{Coronary angiography, atypical angina\}, OP20(mean value) \rightarrow age, gender \rightarrow male, stress test \rightarrow positive, risk factors \rightarrow less than 2\} which is equivalent to the query "What is the mean value of patients' age with atypical angina, candidate for coronary angiography having a positive stress test and less than 2 risk factors?".

Assuming that \(t_s\) would have been the term \{stress test\} or \{estimated uterine weight\}, all of the operational terms \{OP20, OP21, OP22, OP23\} would have been excluded from the set of inferred (suggested) terms, since the assigned precondition \(p'_i = \neg(Categorical\ Variable)\) to each of \{OP20, OP21, OP22, OP23\} is not satisfied, since the context of roles of \{stress test\} or \{estimated uterine weight\} is \{Single Property. Categorical Variable\}.

A number of bi- or multivariate functions can also be assigned to a particular term within a given query state. If it happens that the activated term is classified as, for instance, \{Complex Property\} or \{Entity\}, there is a connecting edge between \{Complex Property\} or \{Concept\} classes and the class of \{Interrelationship Function\}. Therefore, the assigned preconditions are satisfied. However, the number of properties or variables affected by an interrelationship function needs to be taken into account. A set of preconditions (operational term circumstances) needs to be considered, each of which refers to a particular property (argument, variable).

For example, the set of preconditions (operational terms circumstances) \(P_F = \{p_1 = (Single\ Property), p_2 = (Single\ Property)\}\) might be assigned to the class/collection of \{Bivariate Functions\} as subclass of \{Interrelationship Function\}, in order to trigger them as semantically valid only if two single properties are affected by this operation in a given query state.

If we specify \(P_F\) such as \(P_F = \{p_1 = (Single\ Property \land Categorical\ Variable), p_2 = (Single\ Property \land Categorical\ Variable)\}\), then it is expected that both property terms are classified as \{Categorical Variable\}, which means that a particular bivariate function makes sense only if both arguments are categorical variables.

Recall that the satisfaction of operational terms circumstances \(P_F\) is a matter of a set of contexts of roles for the property terms (variables) currently participating in a given query state (definition 2.7, section 2.2.2, chapter 5). Therefore, algorithm 6.0 needs to be modified to have it check the semantic validity of functions with more than one argument (property term). To this end, we generalize algorithm 6.0.
The behaviour of algorithm 7.0 is such that preconditions, if there are any, are examined first at the classification layer of operational terms and subsequently at the term instances layer. This reflects the issue that classes of operations must be defined together with their expected behaviour in terms of the nature and number of arguments expected. However, certain operations within the same class might show different or exceptional behaviour in terms of more specific categories of arguments they need to operate meaningfully.

BEGIN Algorithm 7.0

$q_c$ is the current query state

\begin{itemize}
  \item \textit{e.g., \{Patient, Coronary angiography, atypical angina, age, stress test\}}
\end{itemize}

$t_s$ is the activated term by the end-user

\begin{itemize}
  \item \textit{e.g., $t'_s = \{\text{Patient}\}$ or $t''_s = \{\text{Age}\}$ or $t'''_s = \{48\}$ the activated term}
\end{itemize}

\textbf{get} the roles/classes of $t_s, r_s$

\begin{itemize}
  \item \textit{e.g., \{Entity, Concept\} when $t'_s$}
  \item \textit{\{Single Property, Categorical Variable\} when $t''_s$}
  \item \textit{\{Domain Value, Atomic Value, Quantitative Value\} when $t'''_s$}
\end{itemize}

- in the following, only $t'_s$ will be considered - \textbf{get} the roles/classes $R_{t,f} = \bigcup r_{t,f}$ of operational terms associated with $r_{t_s}$

\begin{itemize}
  \item \textit{e.g., $R_{t,f} = \{\text{Interrelationship Function, Bivariate Function, Multivariate Function}\}$}
\end{itemize}

\textit{through \{Entity\} $\rightarrow$ \{Interrelationship Function\}}

\textit{and through \{Interrelationship Function\} $\rightarrow$ \{Bivariate Function, Multivariate Function\} (section 2.1, chapter 5)}

set $R_{t,f}$ to list of candidate operational term classes

\textbf{while} there are operational term classes $r_{t,f}$ in $R_{t,f}$

\begin{itemize}
  \item if there is an assigned set of preconditions $P_T$ to $r_{t,f}$ \textbf{then}
  \begin{itemize}
    \item \textit{e.g., $P'_T = \{p_1 = \text{(Single Property)}, p_2 = \text{(Single Property)}\}$ $\rightarrow$ \{Bivariate Function\}}
    \item i.e. expecting two variables as input
    \item \textit{e.g., $P''_T = \{p_1 = \text{(Single Property)}, p_2 = \text{(Single Property)}, p_2 = \text{(Single Property)}\}$ $\rightarrow$ \{Multivariate Function\}, i.e. expecting three variables as input
  \end{itemize}
  \item if $P_T$ is satisfied due to definition 2.7 (section 2.2.2, chapter 5) \textbf{then}
  \begin{itemize}
    \item \textit{e.g., only $P'_T$ is satisfied}
    \item \textbf{add} $r_{t,f}$ to list of inferred (suggested) operational term classes
    \item \textit{e.g., Bivariate}
  \end{itemize}
  \item else
  \begin{itemize}
    \item \textit{ignore $r_{t,f}$}
    \item \textit{e.g., ignore \{Multivariate\}}
  \end{itemize}
\end{itemize}
else
    add to list of inferred (suggested) operational term classes
end of while

Start examining preconditions at the instance layer of operational terms.

for each inferred (suggested) operational term classes
    set the operational term instances \( t_i \in R_{o_j} \) to list of candidate operational terms
    e.g., \{OP100, OP101, OP102, OP103...\} (section 1.2, chapter 5)
    while there are operational term instances \( t_{j} \) in the list of candidate operational terms
        if there is an assigned set of preconditions \( P_t \) to \( t_{j} \) then
            e.g., \( P_t = \{p_1 = (\text{Single Property} \land \text{Numerical Variable}), p_2 = (\text{Single Property} \land \text{Numerical Variable})\} \rightarrow \{OP101, OP102, OP103\} \)
            if \( P_t \) is satisfied due to definition 2.7 (section 2.2.2, chapter 5) then
                add to list of inferred (suggested) operational terms
            else
                ignore \( t_{j} \)
                e.g., ignore \{OP101, OP102, OP103\}
        else
            add to list of inferred (suggested) operational terms
    end of while

suggest list of inferred (suggested) operational terms
    e.g., \{OP100\}
< SELECT > operational term
    e.g., \{OP100\}

END Algorithm 7.0

Note that the set of preconditions \( P_t = \{p_1 = (\text{Single Property} \land \text{Numerical Variable}), p_2 = (\text{Single Property} \land \text{Numerical Variable})\} \) assigned to the operational terms \{OP101, OP102, OP103\} can be conceived as more specialized preconditions which hold for members of the operational term class \{Bivariate Function\}. It extends the set of preconditions \( P'_T = \{p_1 = (\text{Single Property}), p_2 = (\text{Single Property})\} \) holding for the whole class in that the property terms (variables) need to express numerical data, i.e. data measured on a numeric scale. Since this is not the case for both \{age, stress test\}, \{OP101, OP102, OP103\} are excluded.

Satisfaction of preconditions for operational terms has been considered, up to now, as a "black box". The following algorithm incorporates the reasoning underlying the satisfaction of a set of preconditions for operational terms, whether they are posed to classes or instances of operational terms. The algorithm is based on the definitions 2.5, 2.6, 2.7 concerning constraints over operational
terms (section 2.2.2, chapter 5). Due to these definitions, the set of preconditions are examined against contexts of roles as the power set of roles (classes) within which a particular domain term is a member.

In other words, the algorithm sets the truth value of set of preconditions \( P_T \) to true only if each precondition as an assembly of classes/roles is set to true based on the assumption that all classes/roles (literals) of a precondition also appear within a particular context of roles. Since the number of preconditions indicates the number of expected arguments, \( P_T \) is also turned to false, if it happens that not enough contexts of roles \( C_{T_q} \) have been considered - less arguments than expected -, or more contexts of roles \( C_{T_q} \) than preconditions \( p_t \in P_T \) - more arguments than expected.

BEGIN Algorithm 8.0

\( q_c \) is a given query state

get activated term \( t_s \)

construct context (set) of roles \( C_{T_q} \) for \( t_s \)

if \{Entity\} or \{Complex Property\} \( \in C_{T_q} \) then

apply recursively until \{Single Property\} \( \in C_{T_q} \)

visit connected node (term) \( t_s' \) to \( t_s \) in partial graph \( q_c \)

construct context (set) of roles \( C'_{T_q} \) for \( t_s' \)

if \{Single Property\} \( C'_{T_q} \)

insert \( C'_{T_q} \) into set of contexts of roles \( \cup C_{T_q} \)

else ignore \( C'_{T_q} \)

else

insert \( C_{T_q} \) into set of contexts of roles \( \cup C_{T_q} \)

Start examining set of preconditions (circumstances) against set of contexts of roles \( \cup C_{T_q} \).

get set of preconditions \( P_T \)

while there is a precondition \( p_t \in P_T \)

while there are any unvisited \( C_{T_q} \) in \( \cup C_{T_q} \) and \( C_{T_q} \neq {} \)

get next \( p_t \)

get next unvisited \( C_{T_q} \)

if it holds that literals \( q_j \subseteq C_{T_q}, j \geq 0 \) then

mark \( C_{T_q} \) as visited

set \( p_t \rightarrow true \)

else

get next unvisited \( C_{T_q} \)

end of while

while


if it holds that \( \forall p_t, p_t \rightarrow true \) and
there are no unvisited \( C_{T_q} \) in \( \bigcup C_{T_q} \) then
(exclusion of case where more arguments than the expected ones are in a given query state)
  set \( P_T \rightarrow true \)
else
  set \( P_T \rightarrow false \)

END Algorithm 8.0
Chapter 7

EXPRESSIVENESS AND COMPUTABILITY

Haldane: *Where is the bedeutung of a proposition in your system, Turing? It is worth talking in terms of a universal grammar in mind, but it is also possible to construct meaningless propositions.*

Turing: *For example?*

Haldane: *"Red thoughts walk peacefully".*

. . . . . *This is an example of a proposition which is formulated correctly, according to the rules of the English grammar. If your theory is correct, then I was able to construct such a proposition, since I was able to activate the English version of the universal grammar in my mind, the semantic content of which equals to zero. Where can I find in your theory that this proposition makes no sense?*

Turing: *It is very simple, I do not know.*


In the two previous chapters, we discussed the nature of the query input alphabet in terms of intentional meaning and context representation (chapter 5) and the algorithms underlying the state transition functions of the automaton (chapter 6). These were necessary conditions for the specification of the abstract machine $M$ that accepts the queries, which are generated interactively during end-user/machine interaction, as statements of the query language MDDQL.

This chapter deals with the expressiveness and computability of MDDQL. For this purpose, a more formal specification of the language is presented in section 1. in terms of a set of production rules - MDDQL conceived as generative language. Even though preconditions, as defined in chapter 5, cannot all be expressed by production rules, the reader should have an impression of what kind of queries can be expressed in MDDQL. However, recall that only the syntactic part of the language can be expressed by that and not the semantic contents.

Furthermore, in order to show that, given this specification, there will always be some state transition algorithms which make acceptability of query statements computable, MDDQL is treated as a
mathematical object, in section 2., in order to prove this property. In other words, there must be some constructive or recursive characterization of statements (sentences) of MDDQL as a sublanguage of a natural language.

It is interesting to note that the expectation of useful mathematical descriptions of the data of language such as constructive or recursive characterization of sentences stems from developments in logic and the foundations of mathematics during the first half of the twentieth century. One important source was the growth of syntactic methods to analyze the structure of formulas, as in Skolem normal form and Löwenheim’s theorem, and in the Polish School of Logic (Sentential calculus, Categorical grammar), and in Quine’s Mathematical Logic. Another source is in the post-Cantor-paradoxes constructivist views of L. E. J. Brouwer and the Intuitionist mathematicians, and in the specific constructivism techniques of Emil Post and Kurt Gödel, in recursive function theory, and from a somewhat different direction in the Turing Machine and automata theory.

1. EXPRESSIVENESS

Most visual query systems or languages typically have no underlying formal syntax definitions such as BNF notation for textual query languages, since the two-dimensionality of such languages exceeds the capabilities of string-based grammars. The semantics is often operational and expressed by rewriting rules which transform the proposed language into some other well-defined target language.

Since the formulation of a query becomes a matter of system guidance on the basis of semantic constraints rather than direct usage of diagrammatic representation of conceptual schemas or other database model abstractions, MDDQL falls into the category of generative languages that are used by the Artificial Intelligence and Computational Linguistics communities. In this category of languages, the grammar rules are used for the generation of phrases rather than for parsing of already formulated phrases. It is out of the scope of this thesis to provide a thorough comparison with widely known generative grammars such as attribute grammars, augmented transition networks, etc.

Generative grammar rules determine what to say and not how to relate it to the listener or viewer as covered by the visual formalism. Moreover, they determine how to map it into a string of words covered by the chosen vocabulary as provided by the assigned words to terms. All three stages are involved in the genesis of a language. All three issues have been considered for the genesis of MDDQL through the knowledge representation issues of domain alphabet terms, where, in addition, meaning and interpretation of terms are also covered.

However, we feel the need to have a formal specification of MDDQL in terms of generative grammar rules in order to gain an insight into the difficulties of expressing meaning or semantics of a language.
as a natural sublanguage for the formulation of queries by means of BNF notation. These rules are a form of production rules, where the symbol Q on the left side must be interpreted as generate query. Furthermore, since the grammar rules describe only the syntactic part of MDDQL, expressiveness is stated in terms of syntactic constructs of a formulated query.

The following list gives an overview of these production rules to construct sequences of terms as members of the Domain of Interest. Application of these production rules leads to a general query that consists of concepts followed by properties and optionally followed by values, when a conditioned query is considered. In particular, given that \( E, R, P, CVD \) are subsets of \( T^D \) as the set of terms constituting the Domain of Interest which is, in turn, a subset of the MDDQL alphabet, with \( E \) the set of terms classified as Entities being a subset of Concepts, \( R \) the set of terms classified as Relationships being a subset of Concepts, \( P \) the set of terms classified as Properties and \( CVD \) the set of terms classified as Value Domains. Everything written in uppercase is conceived as being a non-terminal symbol, whereas \( e \in E, r \in R, p \in P, cvd \in CVD \) as well as \{and, or, not\} are conceived as terminal symbols.

\[
\begin{align*}
Q \rightarrow & \text{CONCEPT PROPERTY } \{\text{VALUEDOMAIN}\} \\
\text{CONCEPT} \rightarrow & \text{ENTITY } \{\text{RELATIONSHIP ENTITY}\} \\
\text{ENTITY} \rightarrow & e \{\text{ENTITY}\} \\
\text{RELATIONSHIP} \rightarrow & r \\
\text{PROPERTY} \rightarrow & p \{\text{JUNCTION } \{\text{not}\} \text{ PROPERTY}\} \\
\text{JUNCTION} \rightarrow & \text{and } | \text{ or} \\
\text{VALUEDOMAIN} \rightarrow & cvd \{\text{or } \text{VALUEDOMAIN}\}
\end{align*}
\]

Therefore, the definition of the production rules indicate the fact that a well-formed query has the following characteristics:

- A well-formed query starts with a term classified as an entity \( e \in E \), optionally followed by either an entity term (recursively) or a relationship and an entity term, and continues with one or more property terms.

- A term standing for a property \( p \in P \) might be followed by other properties (recursively). In case that a conditioned query is considered, the property term might be followed by terms standing for a value domain or single values.

- A value term \( cvd \in CVD \) optionally follows a property in a conditioned query where \( cvd \) might be followed, in turn, by further value terms \( cvd \in CVD \), recursively.

- The logical connector \{or\} holds among the value terms which follow from the same property terms, whereas the logical connectors \{and, or\} hold for the property terms occurring within the context of the well-formed query. The logical connector \{not\} is optionally assigned to a property term.
The default comparison operator for value terms is the equals to operator.

The recursive definitions enable the expression of complex concepts, properties and/or well-restricted value domains within a query. Conditions should be expressed, however, by using a similar technique which led to the definition of attribute grammars [Knu68]. In contrast to MDDQL, in attribute grammars only conditions operating over attribute-value records have been considered and are targeted towards the application of particular rules. In MDDQL, conditions can be expressed in terms of all alphabet terms such as concept terms and are targeted towards the consideration of particular alphabet terms. On the other side, it would be cumbersome or even impossible to express these conditions within the context of production rules.

In the following, the above set of production rules is augmented by (non) terminal symbols which refer to operations and/or operators expressed within a well-formed query. Given this enhancement and/or extension of the set of production rules, the consideration of a particular operation or operator is optional and depends on the nature of the alphabet term. Therefore, a comparison operator COP might occur immediately before a value term, a single dimensional operation SDO might occur immediately before a property term, and a multi-dimensional operation MDO might occur immediately before an entity term.

\[
Q \rightarrow \text{CONCEPT PROPERTY } \{\text{VALUEDOMAIN}\}
\]

\[
\text{CONCEPT} \rightarrow \text{ENTITY } \{\text{RELATIONSHIP ENTITY}\}
\]

\[
\text{ENTITY} \rightarrow \{\text{MDO}\} \ e \ \{\text{ENTITY}\}
\]

\[
\text{RELATIONSHIP} \rightarrow r
\]

\[
\text{PROPERTY} \rightarrow \{\text{SDO}\} \ p \ \{\text{JUNCTION } \{\text{not}\} \ \text{PROPERTY}\}
\]

\[
\text{JUNCTION} \rightarrow \text{and } \mid \text{or}
\]

\[
\text{VALUEDOMAIN} \rightarrow \{\text{COP}\} \ \text{cvd } \{\text{or} \ \text{VALUEDOMAIN}\}
\]

\[
\text{COP} \rightarrow \rangle \mid < \mid >= \mid <= \mid ....
\]

\[
\text{SDO} \rightarrow \text{maximum } \mid \text{minimum } \mid \text{average } \mid \text{frequency } \mid ....
\]

\[
\text{MDO} \rightarrow \text{distribution } \mid \text{regression analysis } \mid ...
\]

Note that the assignment of a particular operation or operator as a terminal symbol is a particular conditions satisfaction which cannot be expressed in terms of production rules. Therefore, for example, the decision whether or not to address a multi-dimensional operation makes sense is made upon the number or properties currently participating in the query context. This cannot be expressed in terms of production rules.
2. COMPUTABILITY

In order to show that finite algorithms can be created which synthesize any sentence (statement) of a language, i.e. the structure of statements is computable, we must show, first, that the properties of discreteness, linearity, contiguity and finite grammar hold [Har91].

For this reason, we must consider a language as having a mathematical structure. This is possible only if we observe the language on the basis of whatever characterizes the particular combinations of words that occur in language. The representation model of the input alphabet, as described in chapter 5, in particular that part which refers to the representation of context in terms of connecting edges and constraints, provides a partial ordering of words. Their co-occurrence is determined by the morphology of the multi-layered, directed graph $G = (V, E)$. Therefore, MDDQL can be characterized and studied as a mathematical object on the basis of combinations of words as well as by constraints on combinations.

Therefore, the whole structure of syntax is determined by a system of constraints each of which states a departure from equiprobability in word co-occurrence. Syntax and the information it expresses is characterized by a partially ordered set of departures from equiprobability, something which is usefully subject to mathematical investigation. Thus discreteness, linearity, contiguity and finite grammar are observable properties of MDDQL which makes it amenable to mathematical treatment, despite the fact that meaning is embedded within structure.

**Discreteness:** Given the nature of the terms constituting the input alphabet as discrete elements, i.e. elements resulting from a finite set of application domain terms, it turns out that the MDDQL alphabet has grammatical regularities. Thus the grammar is free of phenomena such as intonation of hesitation or of exaggerated matter-of-factness.

**Linearity:** Despite the fact that certain terms are nested within others, whenever concepts, etc., have a recursive structure, we can consider that their co-occurrence is linearly ordered. The linearity of terms is not to be confused with the linear orderings in mathematically defined sets for the description of language. In the latter case, we are describing language in terms of a set which is closed under some operation; and we ask whether the set is linearly ordered. Even doing so, MDDQL is a set $A$ of linear orderings of words which is closed under the word-concatenation operation.

**Contiguity:** Given the graph based structure of the MDDQL alphabet model, contiguity is the only elementary relation that can be expressed between two words in a word-sequence formed on the base of a contiguous space of discrete objects. Therefore, the word sequence has to be
contiguous without spaces between, since there is no way of identifying or measuring the spaces. The only property that makes a sequence a construction of the grammar is that the objects are not arbitrary words but words of particular classes. Contiguity is a useful property when a computer program is designed to analyze the structure of sentences.

**Finitary grammar:** Despite the recursivity in structure of the alphabet the growth of the vocabulary is finite - bound to the set of all terms used within a particular application domain. The number of terms and of edges are, in principle, countable.

Concluding, given these properties, MDDQL can be defined as a *sublanguage* of a natural language with one or another set of entities closed under stated relations or operations. Besides, there will always be algorithms for computing the structure of statements so formulated, although meaning has been injected into syntax. However, the algorithms are *decidable* under the precondition that no cycles are formed for the *term dependency graphs* as described in chapter 6, section 2.
Chapter 8

TOWARDS A COMPREHENSIVE QUERY ANSWERING SYSTEM

Minkisi are complex objects clearly not the product of a momentary impulse. ... To do justice to objects, a theory of them must be as complex as them.

—Wyatt MacGaffey, in Astonishment and Power

From a systemic point of view, the meaning driven querying methodology is provided by a system which is characterized as a Semantically Advanced Query Answering System - SAQAS. The system is being developed and implemented as a Web-based application relying on a three-tier architecture (section 1.). The three layers roughly correspond to the well-known concept of separating presentation, application and operational logic.

Given that the third (back-end) layer corresponds to a data repository, the front-end and middle-tier layers correspond to the visual query interfaces, as presented in chapter 4, and the automaton implementation component (representation model of input alphabet + inference engine), respectively. A more thorough description of the architecture and implementation of the middle layer is given in (section 2.).

1. A GENERAL OVERVIEW OF THE SYSTEM ARCHITECTURE

During the design and development of SAQAS emphasis has been placed on existing classification schemes (see figure 8.1) for Web based applications which vary from conventional programming languages to Internet ones, as well as communication protocols and services provided by Web servers. Given the wide variety of Web-based technologies and the requirements profile of applications, the classification scheme for Web applications has been made on the basis of tasks which can be performed at all three levels.
The chosen classification is not intended to be an exhaustive classification tree for all possible technologies and mixing approaches but rather a guide map for the development of the system. This guide map, as presented in figure 8.1, puts the emphasis either on server-sided or client-sided applications depending on the location of the execution of the tasks. For a more detailed description for the classification scheme, see also [Kap98].

Figure 8.2 gives an overview of the system components. Given this system architecture and the classification scheme as depicted in figure 8.1, SAQAS can be classified as a client-sided, database supported Web application, which clearly considers database management systems (DBMSs) as data sources from which data must be integrated into dynamically generated Web pages or Java Applets. The latter are used as the end-user querying and/or answering interfaces.

The classification as a client-sided, database supported Web application is dictated by the fact that the query construction process takes place at the client side with the inference algorithms and relevant alphabet downloaded from the middle layer and running on the client machines. Since the inference algorithms and the representation model are realized in the programming language Java, the system can be used either as a stand-alone application or as a Web application (Web client). The client should support a Java Virtual Machine (recommended version: 1.2 or higher).

At the middle layer resides the inference engine and the repository for the representation of meaning of query alphabet terms. Both realize the automaton specification as presented in chapters 6 and 5, respectively. The inference engine is provided by the libraries with the implemented inference algorithms needed for the context based guidance of the end-user. The Java based implementation of the inference algorithms provides an Application Programming Interface for the realization
Towards a comprehensive query answering system

2. IMPLEMENTATION OF THE AUTOMATON AS AN INFERENCE ENGINE

The middle layer is conceived as a knowledge repository acting as a server for the querying clients and as a client itself for the data repositories at the bottom layer. Each time a client wishes to start a querying session, she/he connects to the middle tier server which delivers the algorithm(s) for the end-user guidance to the client for local execution. Therefore, a client-sided execution of the graphical query interfaces to the specified automaton. The dashed arrows indicate the need to download the inference engine component and the relevant terminology base to the client for the interactive query construction process. Examples of both the inference engine and the representation model are given in appendix C.
query construction guiding algorithms eliminates the query construction computing overhead at the middle tier server.

This contributes to a better overall system performance by distributing the computing resources needed for the query construction session to the clients' machines and not having them run at the application server site, which will increase the demand for processing power at the middle layer, especially when large user communities are involved. Since all algorithms have been implemented in the Java programming language, execution of the algorithms at the client site on different operating system platforms requires a Java Virtual Machine, which is activated either from an Internet Browser or as stand-alone application.

In order to perform the inference tasks upon the represented meaning of terms, the running algorithms take into consideration the knowledge about the context of alphabet terms as made available by the representational model of meaning (chapter 5). The system currently used as platform of meaning representation is OMS Java [KN99], which is used as an object-oriented application framework for the Java environment.

In this sense, OMS Java provides a workspace for the implementation of the inference engine which relies on extensions of the data modelling constructs provided by the Java programming language through constructs supporting manipulation of binary associations. This relies on the

---

Figure 8.3. The multi-lingual, constraints based specification of conceptualization of the terminology and its meaning.
object model OM as suggested by [Nor93, Nor95]. The conditioned, multi-layered graph used as representational model of meaning is simulated by collections [Nor93, Nor95] standing for binary associations among collections of terms. An example of both model specification and instantiation of the representation model of meaning of application domain terms in OMS Java is given in appendix C, section 2.

Because both the implementation of algorithms and the knowledge representation is done in Java, a platform independent execution environment is provided. However, the expression of preconditions is currently done by using data modelling constructs provided by Java and OMS Java. Therefore, preconditions are conceived and expressed as objects standing in an association with particular terms (objects), i.e. assignment of preconditions to terms. As far as preconditions are expressed in terms of other objects within the representation model, the representation of preconditions does not pose any additional requirements. In the case that preconditions must be expressed in terms of mathematical expressions, another representation platform and/or model must be considered.

Expressing the knowledge for the querying terminology of a particular domain of discourse is equivalent to the instantiation of the representation model as depicted in figure 8.3. The instantiation can be a subject of requirements analysis or reverse engineering, when a legacy system is considered. Both instantiation processes can be supported by tools, which are currently designed and have been partly implemented as a semester project.

In the case that instantiation is subject to requirements analysis (specification of conceptualization), the end-user based specification of terminology could be supported by model instantiation tools which support a multi-lingual environment as well as the expression of context of terms, i.e. holding assignments and conditions. In order to avoid internal details of the underlying meaning representation language, the currently elaborated tools provide a natural language based interface.

In the case that instantiation is subject to semi-automatic extraction of data semantic dependencies provided by data mining and/or reverse engineering techniques (bottom-up approach), the instantiation of the representation model still must be made in interaction with the end-user, since particular kinds of semantic dependencies embedded in data interpretation need to be confirmed by the end-user.

Since there is a separation between linguistic elements of terms and their meanings as syntactic elements of a query, on one side, and implementation symbols used by a particular data repository, on the other side, additional components are foreseen which enable the mapping of linguistic elements to implementation symbols as well as the realization of the MDDQL query transformation logic. Currently, only a single data repository is considered for the mapping and query transformation logic.
3. THE QUERY EXECUTION AND/OR TRANSFORMATION PERSPECTIVE

MDDQL has been discussed as a meaning embedded sublanguage, and therefore, as a syntactic object. However, it could be extended by an execution engine which takes into consideration the nature of the language. For example, planning and execution of the query could follow the formulation process immediately, since in MDDQL formulated queries are already interpreted as syntactic objects. Furthermore, the planning/execution trees should be enhanced by a term rewriting process for keeping the language independent from any symbols chosen to represent the underlying data repository. This also enables the execution of queries formulated by linguistic elements in different natural languages, whereby the same query results are addressed.

An example of such a query execution perspective is given in figure 8.4 and is currently practiced and examined in the application domain of Hysterectomy (see also appendix B). The constructed query addresses all instances of patients described as Premenopausal patients with asymptomatic leiomyomata, with estimated uterine weight of less than 300 grams and a growth of 20% within 12 months. The formulated query takes the form of a tree as a subgraph which directly corresponds to the structures of the knowledge representation model for the alphabet terms. The leaf nodes are extended by the selection operation, whereas the inner nodes are augmented by the projection operation. However, a more detailed examination of the query execution engine lies outside the scope of this thesis.

Concerning the targeted query result in terms of model-theoretic semantics, the result of a query would be the relevant set of instances, e.g. patients, as entities within a particular world, e.g., a
Towards a comprehensive query answering system

particular sample in a space(hospital)/time context. A world-time pair constitutes an index which might be further refined by multi-dimensional indexes [Ben75, Rob81, GG98] which refer to attribute/value pairs for a particular set of instances. This resembles Montague's model-theoretic semantics.

![Figure 8.5. The query transformation and execution perspective](image)

Figure 8.5 depicts the execution alternative of an MDDQL query when the targeted data repository is a database which understands a database-specific query language such as SQL (Structured Query Language). MDDQL can be transformed into an SQL statement by applying a query transformation algorithm. An example is given in appendix C. The outcome of the application of the transformation algorithm, such as an SQL statement, is conceived as a syntactic object too. For this purpose, a metadata database is being used which gives an insight into the structure of the current implementation model. For example, information of what attributes belong to which relations as well as the mapping between linguistic elements and implementation symbols is included in the meta-data database.

4. THE PERSPECTIVE OF GENERATING ANSWERS

So far, the semantically advanced query answering system has been presented from the query formulation and transformation point of view. In order to complete the whole picture of the system, however, the "answer generation" component will be briefly covered in this section. This component is concerned with the system-aided "synthesizing" and/or "construction" of answers,
analogously to the system-aided "synthesizing" and/or "construction" of queries. Moreover, this component also deals with the presentation of query results according to the semantic contents of the posed query and/or result.

Figure 8.6. The query result (answer) generation perspective

Figure 8.6 depicts the major processes and the associated knowledge involved in the generation and presentation of a query result. Assuming that the raw query result is a set-oriented result (set of tuples, set of objects, set of documents) that depends on the data repository to which the query has been posed, a transformation of the implementation symbols included in the raw query result to linguistic elements might take place first. This is necessary, in particular, when data are encoded in order to symbolize a word or phrase in a given natural language.

Therefore, having a query posed, for example, in English, will generate the same results as a query posed in German, for example. This is relevant to both the database schema and the data values as well as to operational terms. In order to meet the requirement of multi-lingual environments, the mappings between linguistic elements and implementation symbols need to be considered for the transformation process. For example, the following mappings might be relevant:

\{ST\} \rightarrow \{\text{Stress test, Stress test}\}
\{0\} \rightarrow \{\text{positive, positiv}\}
\{\text{REVA-PATIENTS}\} \rightarrow \{\text{Patients for Revascularization, Patienten fuer Revaskularisation}\}

Furthermore, an assembling of semantic elements for the query answer might be necessary in order to generate a comprehensive (semantically enriched) answer. This might include elements such as the natural language based formulation of the initial query, results of (analytical) operations, and set of instances involved in the operation. The answer might take the form of a report rather than a simple presentation of query results.
Further query answer assembling issues might be addressed when query results from different data repositories must be dealt with. Typically, if different semantic contents are involved despite the fact that the same representation symbols (homonyms) have been chosen or the same semantic contents are involved despite the fact that different representation symbols (synonyms) have been chosen, then the presence of semantic clarification is crucial.

Finally, various answer presentation media might be taken into account such as the choice of Internet Browser. Therefore, enhancement of the (assembled) answer through elements of a particular presentation (generalized mark-up) language such as HTML or XML might be an auxiliary input to the presentation logic. Additionally, alternative views of a specific query result might be possible depending on the nature of the posed query. For example, summaries of data might be presented either by tabulations or graphically by charts.
Seite Leer / Blank leaf
Chapter 9

EPILOGUE

...however, the "system" is (as regards logic) a free play with symbols according to (logically) arbitrarily given rules of the game. All this applies as much (and in the same manner) to the thinking in daily life as to a more consciously and systematically constructed thinking in the sciences.


1. SUMMARY

The role of meaning and semantics in application discourses for a system-guided construction (formulation) of meaningful queries has been investigated in this thesis. In particular, a representation model for meaning and semantics as well as an implemented inference engine, which drives the query construction process by using the represented meaning of application discourses have been described.

The resulting meaning driven querying methodology meets the requirements of

- no learning of a particular query or natural language based syntax for querying a data repository,
- no understanding of the meaning of the underlying database schema in terms of adequate interpretations of data constructs such as relations, classes, collections,
- embedding natural language based interpretation of the meaning of acronyms used as attributes and values,
- awareness of the context which relates to the data as expressed in terms of measurement units, explanation or definitions, constraints, etc.

The Meaning Driven Data Query Language - MDDQL as well as the Semantically Advanced Query Answering System which resulted from the implementation of the meaning driven querying
methodology provided the basis of the querying mechanism currently applied to the querying of semantically enriched data such as clinical guidelines, and collected or evidence-based data. The "Second Opinion System" introduced in various Swiss hospitals and medical centres as a decision and medical quality support system incorporates the philosophy of a "Semantically Advanced Query Answering System". MDDQL has also been suggested and introduced as a query language for the "Information Management System of Mines Actions (IMSMA)".

2. LESSONS LEARNED

A major contributing factor to the acceptability of the system has been the involvement of the user in the design and validation of solutions as well as the anticipation and assessment of the impact of these solutions during their use. Although the latter difficulty has been repeatedly dealt with by publications, associations, meetings and methods, it is still not so obvious in research and development projects.

The difficulty might be of cultural nature, since many computer-scientists, from a research-oriented point of view, tend to underestimate the contributions that users could provide and consider themselves "smart" enough to definitely anticipate them. From an industrial perspective, companies keep on considering user involvement primarily as a definite cost to be paid in view of an uncertain future benefit. Hence, they tend to implement user-involved design methods only if obliged by the demand of the customer.

Another major contributing factor for acceptability of the system was an increase in efficiency of people performing their duties without this resulting in

- extra organizational costs,
- inconveniences,
- dangers and dissatisfaction for the user,
- undesirable impacts on the context of use and/or environment,
- long periods of learning assistance and maintenance

and not a technologically "perfect" solution or a multiple functional product without defects.

In the literature, most of the above listed requirements are synthetically associated to the qualitative software characteristic called usability. Even in the latest ISOs publications on the quality of software products, like ISO/IEC 9126, "quality" is defined as the "capacity of the product to help
certain users reach certain objectives in an effective, efficient, safe and satisfactory way, in certain contexts of use”.

However, in order to meet the goal of high usability, the driving force behind the presented querying approach, as far as query answering systems are concerned, has been intelligence in terms of query context recognition and system-aided formulation of queries and generation of answers. The key issue has been the computer understanding of a user’s world and not user’s understanding of a computer world.

Furthermore, we have shown that a key factor for increasing computer understanding of a user’s world can be computational semantics, an often neglected issue in computer science. Perhaps this is so because of the difficulties and/or complexity of injecting semantics into a world of symbols. However, without representation of meaning and semantics, no software could ever infer and anticipate or assess worlds, if these worlds are not known to it.

Moreover, without representation of meaning and semantics, knowledge exchange becomes difficult, especially, when different domain scientists are involved in a case study or a research assisting information system. Even within the same scientific domain, communication among experts depends on common understanding is enabled or supported by the system.

The representation model of meaning of a query alphabet, we have presented, resulted from the lack of adequate conceptual/data modelling tools or languages to cope with concrete value domains of attributes, multi-lingual expressions, through preconditions-relevant associations holding between entities and attributes or between attributes and value domains. Moreover, the capability of expressing preconditions, which turn any statement of value or attribute assignment to be relative and context dependent was a major contribution to the inference logic for meaningful suggestions.

On the other hand, a major difficulty has been the consideration of variables within literals of preconditions. At the present state of development, the inference engine works very well with preconditions expressed by constants. An extension towards inclusion of variables in preconditions would be a benefit, in particular, when constraints must be formulated which are not bound to the given query alphabet.

In summary, from a systemic and knowledge representation point of view, “smart” inferences are not only bound to connectionism models but also to “constraints” and context recognition. The latter adds complexity to the inference engine but is considered crucial. If we would like to talk in terms of “intelligent” inferences, from a human-oriented point of view, we must also be able to talk about “context based reasoning”. The latter might be a key issue for addressing ”meaningful information” in the “age” of massively producing and disseminating information and knowledge through the Internet.
3. FROM PROTOTYPES TO OPERATIONAL USE

The system has been experimentally used by Ospedale Civico di Lugano and Triemlispital Zuerich as a querying or evaluation platform of medical guidelines (classification rules) and evidence-based patient data (see also appendix B, section 1). The query results have been used for research studies concerning the appropriateness and necessity of medical interventions in cardiology and hysterectomy at the medical institutions mentioned above.

During the prototyping phase, the system has also been tested and approved by IMSMA (see also appendix B, section 3). Further testing beds have been provided by Glaxo Wellcome (appendix B, section 2) and the Snow and Avalanche Research Institute, ETH Zurich (appendix B, section 4) where a context based querying of observer's data concerning snow related geophysical parameters has been considered.

Currently, the system, as described in chapter 8, has been used operationally at the Cardiocentro Ticino (CCT) as a query and evaluation platform of medical decisions concerning cardiological interventions such as coronary angiography and revascularization (appendix B, section 1). The system will act as a knowledge factory where not only knowledge querying is addressed but also knowledge providing mechanisms in terms of validation of classification rules and modifications of existing ones considered to be a contribution to the medical community.

In this sense, the system has also been taken in operational use at the gynecological clinics of the University of Zurich, the Cantonal hospital of Schaffhausen and at the Bruderholz hospital, Canton Basel, where a pilot study of medical interventions concerning hysterectomy has been launched.

In particular, the querying mechanisms of classification rules and that of instances description oriented queries is used operationally. Querying of classification rules relies on transformation of MDDQL-based queries to SQL concerning decision tables managed by a relational database management system. The instances description oriented queries relies on transformation of MDDQL queries to SQL statements concerning the database schema of evidence-based data for patients in the medical domains of cardiology and hysterectomy. The query results are presented in a set-oriented manner where additional interpretations of data in the preferred natural language enhance the query result.

The part of the system, where instances description oriented queries are enabled, will be taken in operational use by the field modules of IMSMA (appendix B, section 3). This affects querying of data gathered by mines clearing agents at the place of actions within various countries (ca. 30 countries) such as Estonia, the Kosovo area, Ethiopia, etc.

In all these application domains, the operations currently supported operationally by the MDDQL visual query interface are restricted to univariate functions, since bi- or multivariate functions pose
additional challenges to naive users. Without an adequate interpretation of the query results as provided by the answer generation mechanism, queries including such functions are only expected by skilled users.

Currently, the system is also tested for the purposes of context-based querying/gathering of data related to geophysical parameters for snow and avalanche research as referred by observers (appendix B, section 4). It is expected to be used operationally the year 2001.

4. OUTLOOK

The Semantically Advanced Query Answering System with the Meaning Driven Data Query Language - MDDQL as a query language is conceived as knowledge providing and validating system, where the key issue is the role of semantics in both query formulation and answer generation. However, such a system poses additional challenges to the components of

- hypothesis (expressed by a classification rule) validation/verification based on real data,
- application discourse meaning and semantics modelling tool which meets the requirements of the meaning representation model presented in this thesis,
- enhancement of the query execution engine by data mining algorithms and/or statistical procedures,
- enhancement of answer presentation mechanism by semantically advanced visualization techniques,
- query transformation and execution engine when more than one date repositories are involved, which are currently tested.

In particular, querying in terms of hypothesis (classification rule) validation against evidence-based data would enhance the already existing querying mechanism of classification rules. This is based on time/spatial constraints as well as on probability theory. Furthermore, statistical operations and/or data mining algorithms will be integrated within the query alphabet and will be considered by the query execution engine.

The latter currently resides on an MDDQL transformation component which translates MDDQL-queries to database specific ones. At the moment, the generation of execution plans through creation of execution trees is being investigated. The execution tree will be a direct mapping of the graph-based (MDDQL) query formulation to an execution tree such that execution of a query can be done at the middleware layer of the system and not always shifting it to the high-end database engine. This will enable the consideration of additional operations which
are not directly supported by the database specific query language or addressing of aggregate results.

Furthermore, the currently available visual query interfaces are being adapted in order to provide the facility of starting the query construction process with any term out of the set of domain alphabet terms. Since the major philosophy of the query construction process with MDDQL is to incrementally refine the query context in terms of the major concepts (themes) we are focussing on, starting with an arbitrary term means that the system will try first to clarify the context of the term through the suggestions as made by the inference engine. For instance, if the entry term would have been temperature, the system would suggest the concepts of air or snow, since it must be clarified, if we are talking about air temperature or snow temperature.

Tightly coupled with this approach is the consideration of terms where more than one source are taken into account. The problem of semantic heterogeneity as stated by homonyms (same elements mean different things) and synonyms (similar things are represented by the same elements) found in different repositories is conceived as a knowledge-based suggestion of related terms according to query context as formed by the set of all terms appearing in the participating repositories.

Finally, an XML-based representation of the meaning of the query alphabet terms upon which the inference engine of MDDQL operates is tested. It is expected that an XML-based representation of the application domain alphabet terms will increase interoperability, since it is not dependent on a particular application platform. However, the lack of semantic constructs such as expression of preconditions currently restricts applicability of such a language as context representation language.
Chapter 10

BIBLIOGRAPHY

References


References 139


References


References


[MKBC96] Shen W. M., Ong K., Mithander B., and Zaniolo C. Advances in Knowledge Discovery and Data Mining, chapter Metaqueries for Data Mining, pages 375–398. AAAI, 1996.


A MEANING DRIVEN QUERYING METHODOLOGY


References 147


A MEANING DRIVEN QUERYING METHODOLOGY


150  A MEANING DRIVEN QUERYING METHODOLOGY


GLOSSARY

SYNTAX refers to the valid symbols of a language, their internal structure, and the means and methods supporting their representation.

SEMANTICS is concerned with the relationship between the linguistic symbols and their meaning.

SEMANTIC SPACE is part of a communications system. It provides a basis of common understanding with respect to the terms used to model (a subset of) the real-world.

FORMALISM (or formal language) is a means to represent a system of real world objects and their relations in an unambiguous way such as to enable their computation.

INFORMATION is the item being exchanged by communicating agents. It includes the syntax, the semantics, and a physical carrier (e.g. sound waves in the case of humans speaking with each other). Information is considered as synonymous with datum.

KNOWLEDGE is defined as the internal state of an agent following the acquisition and processing of information.

An ABSTRACT MACHINE implements an algorithm to process information. It has a set of valid inputs and, conversely, a set of well-defined outputs. It also has an internal state. Each input changes the internal state and produces an output. The term "abstract" refers to the black box concept. Thus, the machine can also be a software program instead of a set of inter-operating physical components.
STATEMENTS describe the interrelationship, i.e. the mutual assignment of the defined terms. Statements are the (not yet refuted) "laws" of the theory. They primarily aim at guiding the actual research process.

AXIOMS are independent, complete and consistent statements.

THEOREMS are derived from axioms.
Appendix B
Application paradigms

In the following, a brief overview of a series of application paradigms is given, for which the meaning driven querying methodology has been applied in order to cover the need of providing another way of data and information access. They are all application paradigms with a rather complex and/or large terminology, respectively, database schemas. Since domain sciences such as medicine, physics or geo-sciences are the source of advanced domain terminologies, the meaning driven querying methodology suited very well in such environments 1., 2., 4.

However, even if other data repositories are taken into consideration such as in case 3., where given a large database schema and complexity of terminology referring to military or humanitarian actions the need of using a semantics based querying approach seems to be indispensable.

Acknowledgements are dedicated to

- the Institute for Social and Preventive Medicine (ISPM) at the University of Zurich involved in 1., 2.,
- the pharmaceutical company Glaxo Wellcome involved in 2.,
- the Institute for Security Studies and Conflicts Research, at the Swiss Federal Institute of Technology involved in 3.,

which financially supported the development of the system behind the Meaning Driven Querying Methodology.
A MEANING DRIVEN QUERYING METHODOLOGY

1. A SECOND OPINION SYSTEM (SOS) IN MEDICINE

Research into the appropriateness and necessity of medical interventions may be the most complex field of innovation in health care and technology assessment. Nevertheless, the appropriateness of the indication of any medical intervention is probably more important than its outcome since even ideal quality-components of structures, processes and outcomes of medical treatments become irrelevant without a proper indication.

The first part of the appropriateness initiative in interventional cardiology and gynecology was to formulate guidelines. To define the appropriateness for coronary angiography, coronary revascularisation, in the field of interventional cardiology and hysterectomy, literature based consensus methods were used. The second part was to disseminate the guidelines in a user-friendly manner via the Internet.

Furthermore, research on indications with real data on clinical outcome may open new opportunities to validate indication-guidelines using Internet/database technology for data analysis concerning clinical decision-making [SFK+00, SFK+99, KNS98]. This new technology facilitates the evaluation of appropriateness and necessity criteria in combination with clinical outcomes.

In Switzerland, the discussion of the appropriateness of indications as a main factor of quality of care became an important topic in the early 1980s. In the beginning, indications were considered part of processes in health care, in particular from the viewpoint of providers of care. In the meantime, there was a tendency to move the topic from the quality of processes to a separate field of quality which stands in front of the traditional processes, structures and outcome fields. Since 1993, a modified RAND methodology has been tested in Switzerland in the areas of interventional cardiology, in gynecology, in gastroenterology and in laminectomy.

In 1996, the topic of quality in health care including quality control became even more important because of a new federal law on health insurance. In one of the paragraphs (article 58), the prescriptions (KV 77.1) say that providers and insurance carriers need to outline contracts on the topic of quality including appropriateness.

In the projects of quality assurance of indications and outcomes in interventional cardiology and gynecology, the basis for nationally recognized guidelines for coronary angiography, coronary revascularisation procedures and hysterectomy were developed using a modified Delphi method. In all projects, a combination of evidence-based medicine, expert opinions and consensus methods were used. The projects for interventional cardiology were based on previous work (literature review, basis for structures for indications). In gynecology, new indication structures and literature reviews were set up. As a novel feature in gynecology, a combination of the original approach with a real consensus method to formulate guidelines for hysterectomy was found to be suitable.
2. ANALYSIS OF HOSPITALISATION INFECTIONS

The meaning driven querying methodology is also being applied within the scope of an information system concerning the collection and analysis of data referred to infections occurred during hospitalization of patients in Swiss hospitals. Initially, the focus has been the infections referred during operations, but currently all kinds of infections are considered.

The used terminology refers, among others, to infection diseases as well as to medical treatment procedures. It complies with international standards and is subject to analytical queries. All underlying semantics refer to internationally specified definitions.

The goal of the system is to monitor medical quality services as provided by Swiss hospitals by analyzing the causes of occurrence of infections as well as the application of medical procedures. Query/analysis results will be subject of discussions for further improvement of medical services during hospitalization of patients.

3. AN INFORMATION MANAGEMENT SYSTEM FOR MINE ACTION (IMSMA)

Switzerland decided to strengthen its involvement in humanitarian demining by establishing the "Geneva International Centre for Humanitarian Demining" (GICHD). Part of this initiative is the development of an Information Management System for Mine Action (further IMSMA) for the United Nation Mine Action Service (further UN MAS).

The Geneva International Centre for Humanitarian Demining tasked the Center for Security Policy at the Swiss Federal Institute of Technology to develop this Information Management System. The Information Management System will provide the UN with improved capabilities for decision-making and information policy related to Mine Action.

The system handles data structured into three separate layers. On the lowest level we find operational information, on the medium level strategical information and on the top level the political information. Individuals and organizations can access data in the layer(s) assigned to them (according to their need). Technical information can be accessed by all users of the system at all stages.

Data is collected at the Mine Action Centers and entered into the field module. In addition to the local data managing capabilities that the module provides to the MAC, data will be transferred to the Headquarter Module where the consolidation and analysis will be performed. The results of the data managing at the headquarter level can be transferred back to the field for guidance purposes.
In addition, it can be accessed by the UN family, organizations involved in Mine Action, Donors, Governments and other interested organizations and individuals.

The Field Module has to be user friendly. As most of the Mine Action Centers use Microsoft Office as their software, the field module will also be based on this software to shorten the learning period. The field module should include a Geographical Information System. The Field Module consists in part of a predefined database as part of the "Starter Kit" which facilitates the start-up of a new mine action centers. Existing MAC's can also use it to improve the functions of their reporting system and data collection. With the use of the system, data formats can be standardized to make data consolidation and evaluation possible on a higher level.

Within the scope of IMSMA, the investigation of the meaning driven querying methodology as a data and information retrieval system resulted into the development support of the system as well as its usage as a front-end querying interface for the Field Module at a first stage.

4. A REGIONAL AVALANCHE INFORMATION AND FORECASTING SYSTEM

The Swiss avalanche warning services deal mainly with reports of snow condition in the Swiss Alps under the co-ordination of the Swiss Federal Institute for Snow and Avalanche Research. The risk factor of an expected avalanche is evaluated due to data collected by various sources such as

- 80 so-called comparison stations,
- ca. 50 automatic measurement stations (IMIS- and ENET-Stations),
- ca. 70 14-days description of snow profiles,
- Weather predictions of various meteorological services,
- questionairres filled in by observers,
- informations from local experts.

Estimation of avalanche risk follows, principally, on the basis of daily measurements of physical parameters as delivered by 80 so-called “comparison stations”. They are distributed in the Swiss Alps and positioned in a height between 1000 and 2500 meters. The data refers to the main weather related parameters such as air temperature, wind and the snow characteristics such as snow height, penetration depth, snow temperature, etc. Triggered avalanches are also being reported and stored. Meteorological data is also used by the risk estimation procedure as delivered by the Swiss Meteorological Services (SMA).
In order to cope with the large amounts of data and information, various visualization techniques are used which give an insight into the contents of data. However, within the scope of a Regional Avalanche Information and Forecasting System (RAIFoS), we also investigated semantics and constraints based data querying and collection techniques as experienced and practiced by the meaning driven querying methodology. This resulted into the usage of this technique when observer’s data are collected and/or queried as a pilot project.
Seite Leer / Blank leaf
Appendix C
Implementation issues

The following will give an overview of the implementation issues underlying the inference engine for the transitions from a current end-user/machine (query) state to a subsequent one (section 1.), as presented in chapter 6, as well as the implementation of the representation model of meaning of terms (section 2.) as presented in chapter 5. They are both implemented on a Java and an OMSJava based platform, respectively. Alternative platforms for the representation of meaning of terms could be an XML based (eXtensible Markup Language) repository, which is not addressed within this thesis.

1. JAVA BASED IMPLEMENTATION OF THE STATE TRANSITION ALGORITHMS

As already stated in chapter 6, the algorithms underlying the inference engine for the guidance of the end-user through semantically consistent query states vary in complexity according to the complexity of queries to be submitted. This also has an impact on the presentation features, and consequently, on the end-user visual query interface chosen for a query construction session.

For example, figure C.1 depicts an overview of the structure of the source code for the implementation of the inference engine to be used for the end-user guidance mechanism for simple instances description oriented queries as well as for classification rules. It is also used as an Application Programming Interface (API) for the implementation of the end-user visual query interface as referred to in chapter 4, section 1 and 3.

Figure C.2 depicts an overview of the structure of the source code for the implementation of the inference engine to be used for the end-user guidance mechanism for advanced queries such as those described in chapter 4, section 2. The consideration of not only AND-connected queries as
A MEANING DRIVEN QUERYING METHODOLOGY

Despite the differences in complexity of the inference engines due to the targeted families of queries, the implementation architecture is characterized by a common philosophy. The major

well as the embedding of operations required a slightly different end-user visual query interface with more advanced presentation features. Therefore, it also provides an Application Programming Interface (API) for the implementation of the end-user visual query interface as referred in chapter 4, section 2.

Figure C.1. The Java based implementation of the end-user guidance mechanism (state transition functions) for simple instances description oriented queries and classification rules
idea is to use a state blackboard as a volatile repository in which the terms already addressed by the end-user within a current query state temporarily reside. The inference engine is characterized either as driver or interpreter, depending on the perspective under which we conceive it; driving the end-user and/or the query construction process or acting as an interactive interpreter for the query language MDDQL. In all cases, inferences are drawn by having a look at the current contents of the state blackboard and the knowledge referring to the meaning of terms (see also next section).
Acting as an API for the implementation of the visual query interfaces, the inference engine provides methods with which all inferred terms are delivered to the graphical user (visual query) interfaces as instances, the variables of which refer to semantic information tightly coupled with the term under consideration. The graphical user interfaces decide upon the presentation of terms according to the semantic contents of terms.

2. OMSJAVA BASED IMPLEMENTATION OF THE REPRESENTATION MODEL OF MEANING

2.1 THE MODEL SPECIFICATION: AN EXAMPLE

The knowledge referring to the meaning of terms is currently represented and provided by the data definition and manipulation language supported by OMSJava, a Java based realization of the OM model. This enhances the data constructs of native Java through binary associations and/or operations which are extensively used for the representation of collections of connecting edges among terms.

In the following, an example is given from the model specification in terms of the data definition language in OMSJava:

```
SCHEMA terminologyDB;

OMCollection= -;
OMInstance= -;

string = java.lang.String;
date = oms.otnJava.OMDate;
text = java.lang.String;
integer = java.lang.Integer;
alphabet: demo.terminologyDB.Alphabet;
domainofinterest: demo.terminologyDB.DomainOfInterest;
concept: demo.terminologyDB.Concept;
property: demo.terminologyDB.Property;
domainvalues: demo.terminologyDB.DomainValues;
measurementunit: demo.terminologyDB.MeasurementUnit;
operation: demo.terminologyDB.Operation;
precondition: demo.terminologyDB.Precondition;
```
Appendix C: Implementation issues

```plaintext
type alphabet
  ( name : string;
    symbol : string;
    description : string;
    priority : string;
    appearance : string;
  );

type concept subtype of alphabet
  ( annotation : string;
  );

type property subtype of alphabet
  ( instantiationMode : string;
    measurementUnit : string;
  );

domainvalues subtype of alphabet
  ( icon : string;
  );

type operation subtype of alphabet
  ( abbreviation : string;
    explanation : text;
  );

type precondition
  ( connector : string;
    terms : set of alphabet;
  );

collection Alphabet : set of alphabet;
collection DomainOfInterest : set of domainofinterest;
collection Concepts : set of concept;
collection Entities : set of concept;
collection MainThemes : set of concept;
collection Relationships : set of concept;
collection Properties : set of property;
collection NumericalVariable : set of property;
collection ContinuousVariable : set of property;
collection DiscreteVariable : set of property;
```
collection CategoricalVariable : set of property;
collection DomainValues : set of domainvalues;
collection Intervals : set of domainvalues;
collection DiscreteIntervals : set of domainvalues;
collection ContinuousIntervals : set of domainvalues;
collection AtomicValues : set of domainvalues;
collection ArithmeticValues : set of domainvalues;
collection DescriptiveValues : set of domainvalues;
collection Operations : set of operation;
collection Operators : set of operation;
collection Functions : set of operation;
collection LogicalOperators : set of operation;
collection LogicalConnectors : set of operation;
collection ArithmeticOperators : set of operation;
collection ComparisonOperators : set of operation;
collection ExistentialOperators : set of operation;
collection IntervalOperators : set of operation;
collection MatchingOperators : set of operation;
collection UnaryOperators : set of operation;
collection BinaryOperators : set of operation;
collection ScalarFunctions : set of operation;
collection NumberFunctions : set of operation;
collection CharacterFunctions : set of operation;
collection ConversionFunctions : set of operation;
collection DateFunctions : set of operation;
collection StatisticalFunctions : set of operation;
collection UniVariateFunctions : set of operation;
collection UniVariateCategorical : set of operation;
collection BiVariateFunctions : set of operation;
collection MultiVariateFunctions : set of operation;
collection Quantifiers : set of operation;
collection Preconditions : set of precondition;
collection Context : set of (domainofinterest,domainofinterest);
collection AssertedLinks : set of (domainofinterest,domainofinterest);
collection ComposedBy : set of (domainofinterest,domainofinterest);
collection NaturalKinds : set of (domainofinterest,domainofinterest);
Appendix C: Implementation issues

collection CompositeConcepts : set of (concept,concept);
collection NaturalKindsOfConcepts : set of (concept,concept);
collection RelatedBy : set of (concept,concept);
collection Relates : set of (concept,concept);
collection MirroredBy : set of (concept,concept);
collection CharacterisedBy : set of (concept,property);
collection CompositeProperties : set of (property,property);
collection NaturalKindsOfProperties : set of (property,property);
collection ConstrainedBy : set of (property,domainvalues);
collection CompositeValues : set of (domainvalues,domainvalues);

collection TermRoles : set of (OMInstance, OMCollection);
collection AssignedOperations : set of (OMCollection, OMCollection);
collection ConditionedBy : set of (alphabet, precondition);

constraint : Context association from DomainOfInterest (0 : *) to DomainOfInterest (0 : *);
constraint : AssertedLinks association from DomainOfInterest (0 : *) to DomainOfInterest (0 : *);
constraint : NaturalKinds association from DomainOfInterest (0 : *) to DomainOfInterest (0 : *);
constraint : ComposedBy association from DomainOfInterest (0 : *) to DomainOfInterest (0 : *);
constraint : NaturalKindsOfConcepts association from Concepts (0 : *) to Concepts (0 : *);
constraint : CompositeConcepts association from Entities (0 : *) to Entities (0 : *);
constraint : RelatedBy association from Entities (0 : *) to Relationships (0 : *);
constraint : Relates association from Relationships (0 : *) to Entities (0 : *);
constraint : MirroredBy association from Relationships (0 : *) to Relationships (0 : *);
constraint : CharacterisedBy association from Concepts (0 : *) to Properties (0 : *);
constraint : CompositeProperties association from Properties (0 : *) to Properties (0 : *);
constraint : NaturalKindsOfProperties association from Properties (0 : *) to Properties (0 : *);
constraint : ConstrainedBy association from Properties (0 : *) to DomainValues (0 : *);
constraint : CompositeValues association from DomainValues (0 : *) to DomainValues (0 : *);
constraint : ExpressedIn association from DomainValues (0 : *) to MeasurementUnits (0 : *);
constraint : ConditionedBy association from Alphabet (0 : *) to Preconditions (1 : *);

constraint : CharacterisedBy subcollection of Context;
constraint : ConstrainedBy subcollection of Context;
constraint : ExpressedIn subcollection of Context;
constraint : RelatedBy subcollection of Context;
constraint : Relates subcollection of Context;
constraint : ComposedBy subcollection of Context;
constraint : NaturalKinds subcollection of Context;
constraint : AssertedLinks subcollection of Context;
constraint : MirroredBy subcollection of Context;
constraint : NaturalKindsOfConcepts subcollection of NaturalKinds;
constraint : CompositeConcepts subcollection of ComposedBy;
constraint : NaturalKindsOfProperties subcollection of NaturalKinds;
constraint : CompositeProperties subcollection of ComposedBy;
constraint : CompositeValues subcollection of ComposedBy;

constraint : DomainOfInterest subcollection of Alphabet;
constraint : Concepts subcollection of DomainOfInterest;
constraint : Properties subcollection of DomainOfInterest;
constraint : DomainValues subcollection of DomainOfInterest;
constraint : Entities subcollection of Concepts;
constraint : MainThemes subcollection of Entities;
constraint : Relationships subcollection of Concepts;
constraint : CategoricalVariable subcollection of Properties;
constraint : NumericalVariable subcollection of Properties;
constraint : ContinuousVariable subcollection of NumericalVariable;
constraint : DiscreteVariable subcollection of NumericalVariable;
constraint : Intervals subcollection of DomainValues;
constraint : ContinuousIntervals subcollection of Intervals;
constraint : DiscreteIntervals subcollection of Intervals;
constraint : AtomicValues subcollection of DomainValues;
constraint : ArithmeticValues subcollection of AtomicValues;
constraint : DescriptiveValues subcollection of AtomicValues;

constraint : Operations subcollection of Alphabet;
constraint : Operators subcollection of Operations;
constraint : Functions subcollection of Operations;
constraint : UnaryOperators subcollection of Operators;
constraint : BinaryOperators subcollection of Operators;
constraint : LogicalOperators subcollection of Operators;
constraint : LogicalConnectors subcollection of LogicalOperators;
constraint : ComparisonOperators subcollection of Operators;
constraint : ArithmeticOperators subcollection of ComparisonOperators;
constraint : ExistentialOperators subcollection of ComparisonOperators;
constraint : IntervalOperators subcollection of ComparisonOperators;
constraint : MatchingOperators subcollection of ComparisonOperators;
constraint : ScalarFunctions subcollection of Functions;
constraint : NumberFunctions subcollection of ScalarFunctions;
constraint : CharacterFunctions subcollection of ScalarFunctions;
constraint: ConversionFunctions subcollection of ScalarFunctions;
constraint: DateFunctions subcollection of ScalarFunctions;
constraint: StatisticalFunctions subcollection of Functions;
constraint: UniVariateFunctions subcollection of StatisticalFunctions;
constraint: UniVariateCategorical subcollection of UniVariateFunctions;
constraint: BiVariateFunctions subcollection of StatisticalFunctions;
constraint: MultiVariateFunctions subcollection of StatisticalFunctions;
constraint: Quantifiers subcollection of ComparisonOperators;

END terminologyDB;

2.2 THE MODEL INSTANTIATION: AN EXAMPLE

The instantiation of the model is divided into two parts:

1. the instantiation and classification of terms,
2. the instantiation of the context of terms due to
   - connecting edges (binary associations),
   - specifying preconditions

Since construction of a query can be done by using words representing the query terms in more than one natural language, each model instantiation is dedicated to a specific natural language and realized by a corresponding file.

The inference engine runs in a workspace created at the client site where the corresponding file of the natural language specific model instantiation is imported according to the selected language. Given that the translation of semantic contents in more than one natural language is required, only the part of model instantiation referring to the instantiation and classification of terms must be changed. The second part referring to the context of terms remain unchanged.

In the following, an example is given from the model instantiation in terms of the data manipulation language in OMSJava. The example refers to the application domain of Mines Cleaning Actions as supported by the IMSMA (see also appendix B, section 4):

Instantiation and classification of terms

DOMAIN TERMS

// Concepts
create object 1;
dress object 1 as concept values (name = 'Explosive Devices';
description = 'Mines, UXO, and several other explosives as provided by OC Inc.';
icon = "";
appearance = "");

create object 15; dress object 15 as concept values (name = 'Country';
description = 'Countries which manufacture explosive devices';
icon = "";
appearance = "");

// Relationships
create object 300;
dress object 300 as concept values (name = 'Manufactured in';
description = 'Relationship between explosive devices and manufacturing countries';
icon = "";
appearance = "");

create object 301;
dress object 301 as concept values (name = 'Manufacturing';
description = 'Relationship between countries and manufactured explosive devices';
icon = "";
appearance = "");

// Properties
create object 500;
dress object 500 as property values (name = 'Nomenclature';
description = 'Naming of devices';
icon = "");
create object 501;
dress object 501 as property values (name = 'Shape';
description = 'A particular form or shape for a given device';
icon = "");

create object 502;
dress object 502 as property values (name = 'Material';
description = 'Description of the construction material of devices';
icon = "");

create object 503;
dress object 503 as property values (name = 'Name';
description = 'Name of manufacturing country';
icon = "");

create object 504;
dress object 504 as property values (name = 'Purpose';
description = 'Destructive purpose of explosive device';
icon = "");

create object 505;
dress object 505 as property values (name = 'Category';
description = 'Kind of explosive device';
icon = "");

// Value domains

create object 18;
dress object 18 as domainvalues values (name = 'Anti-tank';
description = 'Explosive devices for the destruction of tanks');
icon = ";
);

create object 19;
dress object 19 as domainvalues values (
   name = 'Anti-aircraft' ;
   description = 'Explosive devices for the destruction of aircrafts';
   icon = "");

create object 20;
dress object 20 as domainvalues values (
   name = 'Anti-personnel' ;
   description = 'Explosive devices for targeted at citizens';
   icon = "");

create object 21;
dress object 21 as domainvalues values (
   name = 'Anti-helicopter' ;
   description = 'Explosive devices for the destruction of helicopters';
   icon = "");

.................................

create object 2000;
dress object 2000 as domainvalues values (
   name = 'fuse' ;
   description = 'Devices with electric circuit (explode at hit or after a programmed time)';
   icon = "");

create object 2001;
dress object 2001 as domainvalues values (
   name = 'non-fuse' ;
   description = 'The opposite of fuse (see also fuse)';
   icon = "");

create object 2040;
dress object 2040 as domainvalues values (
name = 'BF' ;
description = "" ;
icon = "" ;
);

create object 2010 ;
dress object 2010 as domainvalues values (name = 'BF1' ;
description = "" ;
icon = 'http://macnally.inf.eth.ch/minimum/icons/bf-1.gif';
);

create object 2011 ;
dress object 2011 as domainvalues values (name = 'BF2' ;
description = "" ;
icon = 'http://macnally.inf.eth.ch/minimum/icons/bf-2.gif';
);

create object 2012 ;
dress object 2012 as domainvalues values (name = 'BF3' ;
description = "" ;
icon = 'http://macnally.inf.eth.ch/minimum/icons/bf-3.gif';
);

insert into collection 'Alphabet' : [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,18,19,20,...];

insert into collection 'DomainOfInterest' : [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,18,19,20,...];

insert into collection 'Concepts' : [1,15,300,301];

insert into collection 'Entities' : [1,15];

insert into collection 'MainThemes' : [1,15];

insert into collection 'Relationships' : [300,301];

insert into collection 'Properties' : [500,501,502,503,504,505];

insert into collection 'NumericalVariable' : [];
insert into collection 'CategoricalVariable': [500, 501, 502, 503, 504, 505];

insert into collection 'DomainValues': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 18, 19, 20,...];

insert into collection 'Intervals': [];

insert into collection 'ContinuousIntervals': [];

insert into collection 'DiscreteIntervals': [];

insert into collection 'AtomicValues': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 18, 19, 20,...];

insert into collection 'DescriptiveValues': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 18, 19, 20,...];

OPERATIONAL TERMS

create object OP3;
dress object OP3 as operation values (name = 'equal to'; abbreviation = '='; explanation = "");

create object OP4;
dress object OP4 as operation values (name = 'equal to or more than'; abbreviation = '>='; explanation = "");

create object OP5;
dress object OP5 as operation values (name = 'equal to or less than'; abbreviation = '=<='; explanation = "");

create object OP6;
dress object OP6 as operation values (name = 'greater than'; abbreviation = '>'; explanation = "");
create object OP16;
dress object OP16 as operation values (  
   name = 'Minimum' ;
   abbreviation = 'MIN' ;
   explanation = "" ;
);
create object OP17;
dress object OP17 as operation values (  
   name = 'Maximum' ;
   abbreviation = 'MAX' ;
   explanation = "" ;
);
create object OP18;
dress object OP18 as operation values (  
   name = 'Frequency' ;
   abbreviation = 'FQ' ;
   explanation = "" ;
);
create object OP19;
dress object OP19 as operation values (  
   name = 'Relative Frequency' ;
   abbreviation = 'RFQ' ;
   explanation = "" ;
);
create object OP20;
dress object OP20 as operation values (  
   name = 'Mean (Average) Value' ;
   abbreviation = '' ;
   explanation = 'MV' ;
);
insert into collection 'Operations' : [OP3,OP4,OP5,OP6,OP7,OP9,OP10,OP11,OP12,...];
insert into collection 'Operators' : [OP3,OP4,OP5,OP6,OP7,OP9,OP10,OP11,OP12,OP13,OP14,OP15];
174 A MEANING DRIVEN QUERYING METHODOLOGY

insert into collection 'UnaryOperators' : [];

insert into collection 'BinaryOperators' : \{OP3,OP4,OP5,OP6,OP7,OP9,OP10,OP11,OP12,...\};

insert into collection 'LogicalOperators' : [];

insert into collection 'LogicalConnectors' : [];

insert into collection 'ComparisonOperators' : \{OP3,OP4,OP5,OP6,OP7,OP9,OP10,OP11,OP12,...\};

insert into collection 'ArithmeticOperators' : \{OP3,OP4,OP5,OP6,OP7,OP9\};

insert into collection 'ExistentialOperators' : \{OP12,OP13,OP14,OP15\};

insert into collection 'IntervalOperators' : \{OP10\};

insert into collection 'MatchingOperators' : \{OP11\};

insert into collection 'Quantifiers' : \{OP14,OP15\};

insert into collection 'Functions' : \{OP16,OP17,OP18,OP19,OP20,OP21,OP22,OP23,OP24,OP25\};

insert into collection 'StatisticalFunctions' : \{OP16,OP17,OP18,OP19,OP20,OP21,OP22,OP23,OP24,OP25\};

insert into collection 'UniVariateFunctions' : \{OP16,OP17,OP18,OP19,OP20,OP21,OP22,OP23,OP24,OP25\};

insert into collection 'UniVariateCategorical' : \{OP18,OP19\};

insert into collection 'BiVariateFunctions' : [];

Instantiation and classification of the context of terms

EDGES AMONG DOMAIN TERMS


insert into association 'ComposedBy' : [];

insert into association 'NaturalKinds' : [];

insert into association 'AssertedLinks' : [];

insert into association 'MirroredBy' : \{(300,301)\};

insert into association 'RelatedBy' : \{(1,300),(15,301)\};

insert into collection 'Relates' : \{(300,15),(301,1)\};

insert into association 'NaturalKindsOfConcepts' : [];

insert into association 'CharacterisedBy' : [(1,500),(1,501),(1,502),(1,504),(1,505),(15,503)];

insert into association 'CompositeProperties' : []; 

insert into association 'NaturalKindsOfProperties' : []; 


insert into association 'CompositeValues' : [];

PRECONDITIONS

create object PC1;
dress object PC1 as precondition values ( 
  connector = 'NOT' ;
  terms = [1,3,43] ;
);

create object PC2;
dress object PC2 as precondition values ( 
  connector = 'NOT' ;
  terms = [1,2,43] ;
);

create object PC3;
dress object PC3 as precondition values ( 
  connector = "" ;
  terms = [1,2] ;
);

create object PC4;
dress object PC4 as precondition values ( 
  connector = 'NOT' ;
  terms = [1,41] ;
);

create object PC5;
dress object PC5 as precondition values ( 
  connector = 'NOT' ;
  terms = [63,6] ;
);
create object PC6;
dress object PC6 as precondition values ( 
  connector = "" ; 
  terms = [1,3] ; 
); 

create object PC7;
dress object PC7 as precondition values ( 
  connector = "" ; 
  terms = [69,79] ; 
); 

create object PC8;
dress object PC8 as precondition values ( 
  connector = "" ; 
  terms = [69,80] ; 
); 

create object PC9;
dress object PC9 as precondition values ( 
  connector = "" ; 
  terms = [2] ; 
); 

create object PC10;
dress object PC10 as precondition values ( 
  connector = 'NOR' ; 
  terms = [189,4] ; 
); 

create object PC11;
dress object PC11 as precondition values ( 
  connector = 'NOT' ; 
  terms = [417,2033] ; 
); 

create object PC12;
dress object PC12 as precondition values ( 
  connector = 'NOT' ; 
  terms = [412,6] ; 
);
insert into collection 'Preconditions' : [PC1,PC2,PC3,PC4,PC5,PC6,PC7,PC8,PC9,PC10,PC11,PC12,...];

insert into collection 'KnockingOutPC' : [PC1,PC2,PC3,...];

insert into collection 'SerializablePC' : [PC5,...];


insert into association 'ConditionedBy' : [(11,PC12),(67,PC3),(68,PC6),(1000,PC3), (64,PC5),(175,PC3),(78,PC8),(91,PC7),(96,PC9),(418,PC11),...];

insert into association 'AssignedOperations' : [(NumericalVariable,UniVariateFunctions), (CategoricalVariable,UniVariateCategorical),(AtomicValues,Operators), (ArithmeticValues,ArithmeticOperators),(DescriptiveValues,MatchingOperators),...];
Curriculum vitae

Epaminondas Kapetanios

- **8.10.1963:** Born in Athens/Hellas

- **1969-1975:** Primary school in Athens

- **1975-1981:** Secondary school in Athens

- **1981-1983:** Teacher Training College, Athens


- **1988-1990:** Postgraduate studies (Aufbaustudium) at the Institute of Program Structures and Data Organization, Dept. of Computer Science, Technical University of Karlsruhe, Germany. **Thesis:** A Natural Language based Explanation Component for Deductive Databases.

- **1990-1996:** Scientific employee at the Research Centre Karlsruhe - Technology and Environment, Karlsruhe, Germany. **Project management** for the development of a scientific information system for data management, analysis and knowledge extraction in Atmospheric Research for the MIPAS-B2 balloon experiments.

- **Since 1.1.1997:** Research assistant at the Institute of Information Systems, Dept. of Computer Science, ETH-Zurich