PROMISE: Modeling and Predicting User Behavior for Online Analytical Processing Applications

Carsten Sapia

Institut für Informatik
der Technischen Universität München

PROMISE: Modeling and Predicting User Behavior for Online Analytical Processing Applications

Dipl.-Inform.Univ. Carsten Sapia

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. J. Schlichter
Prüfer der Dissertation:	1. Univ.-Prof. R. Bayer, Ph.D.
	2. Univ.-Prof. Dr. H. Stoyan
	Friedrich Alexander Universität Erlangen-Nürnberg

Die Dissertation wurde am 30.11.2000 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 09.04.2001 angenommen.

Title of the Thesis:

PROMISE: Modeling and Predicting User Behavior for Online Analytical Processing Applications

Author:

Dipl. Inform. Univ. Carsten Sapia

Keywords:

OLAP, multidimensional data modeling, query prediction, prefetching, speculative query execution, dynamic view materialization, predictive eviction algorithms

Abstract:

The workload that is generated by users of explorative navigational information systems typically contains characteristic patterns. If these task- respectively user-specific patterns are known to the system, they can be used to dynamically predict user interactions at runtime. These predictions enable the usage of predictive caching strategies and speculative execution strategies.

Online analytical processing (OLAP) systems are database systems that are designed to interactively explore data that is structured according to the multidimensional data model. This thesis presents an approach (called PROMISE) to represent and acquire information about navigational patterns of OLAP systems and to use them in improving dynamic materialization (caching) strategies. To this end, the thesis contains a formal model to represent behavior of OLAP users taking into account the navigational and iterative query formulation via a graphical front-end tool. An according pattern model (based on the combination of different generalized Markov models) allows for representing navigational patterns. The PROMISE prediction algorithm uses these patterns in order to dynamically predict a set of queries at any point during a session. The prediction framework is completed by an algorithm that induces and updates the pattern information for the user's actual behavior online during the system's operation. In order to demonstrate the potentials of using prediction results for dynamic system optimization, we present two approaches improving the performance of semantic query level caches in OLAP system: through improved benefit estimation functions that allow more efficient admission and eviction algorithms and by means of speculative execution techniques. An empirical analysis of the characteristics of user behavior in a real-world data warehouse environment and performance measurements using simulated traces with data from a real world application demonstrate the usefulness of the approach.

Acknowledgements

A thesis is never solely the result of the efforts of a single person but is a mirror of the scientific and social environment of the author. Therefore, I would like to take the opportunity to express my gratitude to all the people who have directly or indirectly contributed to this thesis.

First of all *Prof. R. Bayer, PhD.* deserves my gratitude for making this work possible and for his continuous trust in my scientific ideas. I also thank *Prof. Dr. H. Stoyan* for co-supervising this thesis.

Continuing chronologically, I owe thanks to my parents *Claus* and *Gisela Sapia* who can rightfully be regarded as the 'initial' contributors to this thesis by constantly supporting and encouraging my education without ever pushing me into a certain direction. I also want to thank my Grandparents Günther and Marianne Sapia who provided constant moral as well as material support during my master studies.

Of course, my special acknowledgements belong to all my colleges at the Knowledge Bases Research Group of FORWISS who shared joy and pain with me for over four years. Especially, *Dr. Markus Blaschka* deserves my admiration and gratitude for always having an open ear for my questions and (sometimes spooky) ideas and for spending his precious time on countless inspiring discussions. Without his constant support and encouragement this thesis could not have been written.

I also have to deeply thank *Wolfgang Wohner* for sharing an office with me and for all the fun we had inside and outside of the office. I will definitely miss all the interesting discussions (about life, philosophy and the rest of the universe; sometimes even computer science).

During the last (and naturally most busy) month *Karl Hahn* and *Bernd Reiner* brilliantly took over as much work as possible in the ESTEDI project thus allowing me to fully dedicate my own time to this thesis.

Dr. Gabriele Höfling deserves the merits for pointing me to the interesting research area of data warehousing. She also persuaded me to stay with FORWISS after my graduation and to undertake the endeavor of writing this PhD thesis.

Dr. Volker Markl served as my role model of a scientist. He also deserves acknowledgements for his successful efforts in restoring an inspiring and a rewarding working environment as the deputy of the FORWISS Knowledge Bases Group.

I also thank my master student *Eduard Scherer* for implementing large parts of the PROMISE evaluation framework under critical time conditions.

I would also like to thank the project partners of our industrial projects, namely *Wacker Chemie GmbH, ESG GmbH, DaimlerChrylser AG* and *Active Knowledge GmbH* for their trust in my innovative capabilities and for the valuable insights into real-world requirements I could get during these projects. Additionally, I want to express my gratitude to all the members of the GI-discussion group "*Konzepte des Data Warehouse*" for the interesting meetings, fruitful discussions, countless inspirations and the encouragement to continue my work.

However, more than anyone else, my partner *Sabine* provided me with the strength and endurance to finish this work. She did not only tolerate my long working hours but also continuously encouraged me to carry on and even took the burden of proof-reading the whole thesis.

Table of Contents

«The most exciting phrase to hear in science,
the one that heralds the most discoveries,
is not "Eureka", but "That's funny...".»
-- Isaac Asimov

Chapter 1 Introduction

This thesis describes the PROMISE approach, which stands short for '**Pr**edicting User Behavi**o**r in **M**ultidimensional **I**nformation **S**ystem **E**nvironments'. We start off by discussing the motivation of the approach (Section 1.1) and defining objectives (Section 1.2) for our work. An outline together with some advice and suggestions on how to read the thesis (Section 1.3) complete this introductory chapter.

1.1 Motivation

Information is more and more becoming the driving factor of today's economy and social life. In order to make consistent (business) decisions, analysts need fast and flexible access to complexly structured information. This demand for interactive ad-hoc analysis of both structured and unstructured data has led to the development of a large variety of interactive information systems based on different paradigms. The World Wide Web (WWW) using the hypertext paradigm is certainly the most well-known system for semi-structured (mostly qualitative) data. Additionally, all major companies are currently in the process of building large data warehouses storing large amounts of quantitative data. These warehouses are accessed using interactive analysis tools (for example, OLAP tools) that allow decision makers to navigationally explore the data relevant to the decision process. Further examples for interactive navigational analytical information systems are Digital Libraries, Tourist Information Systems and certain types of eCommerce applications.

The unifying property of these systems is that access to the data is given in an explorative and navigational fashion (the user is "surfing" the data). Despite the large volume of the data that has to be managed by these systems (currently in the range of Giga- or Terabytes) interactive response times are an essential requirement ("responses at the speed of thought"). Performance optimization in interactive explorative systems is more difficult compared to classical transaction processing systems because the queries of the user (who acts as the dynamic part of the system) are not known in advance (like in an application that accesses data according to a certain algorithm). However, analysis of web navigation paths has shown (e.g. [HPP+98], [PP99a]) that users of a web site do not behave completely arbitrary, but that their behavior exhibits certain patterns mirroring the intention of the user (her analytical task)

and/or the special peculiarities in the users analysis behavior. It has already been proven (for example, [ANZ99]) that knowledge about these patterns can be used in order to improve the performance of information systems by predicting and anticipating the user's requests. Apart from these dynamic system optimizations, knowledge about these patterns provides an essential input for the (re-)design of the applications and their underlying data model (for example, the structure of the web site or the database schema).

As already mentioned, exploratory navigational data access is not limited to WWW applications, but Online Analytical Processing (OLAP) applications also exhibit these characteristics. These systems are often (but not necessarily) used as front-ends to data warehouses. OLAP database systems have become not only a thriving market (volume in 1999: 2.5 Billion US$ [PC00]) but a large number of scientific publications[1] indicates that OLAP systems have also become an important and active area of scientific research throughout the last years. However, the impact of the navigational data access characteristics and the existence of typical navigational patterns to OLAP-system and -application design has not been addressed so far. Nevertheless, systematic research on this topic promises substantial performance gains for OLAP systems: dynamically adapting to the user's analytical workflow increases the qualitative and quantitative performance of the system. E.g., an improved schema which fits the analytical requirements better will enable analysis tasks that have not been possible beforehand or a higher productivity of current analysis tasks. Automatically adapting the user interface to the user's current tasks will certainly improve productivity of the overall system. But more important – at least for the scope our thesis – are the quantitative performance gains. E.g., anticipating the user's next query to the system can dramatically reduce the latency perceived by the user through application of speculative execution techniques.

Consequently in this thesis, we address the problem of acquiring and exploiting knowledge about the existence of navigational patterns in the behavior of OLAP system users in order to improve the OLAP system's performance. Thereby, our main contributions are to provide a comprehensive framework that allows for acquiring, representing and exploiting navigational pattern information in OLAP systems and the systematic discussion of the integration of this framework into an OLAP cache manager.

As the largest part of this thesis addresses the development and adaptation of techniques for OLAP systems, Section 1.1.1 gives a brief introduction to the concepts and architectures of OLAP systems and their terminology. In order to illustrate the typical areas of application for these systems, Section 1.1.2 introduces the example scenario that we will use throughout this thesis. Our approach aims at exploiting characteristics of the OLAP user behavior. To this end, Section 1.1.3 systematically compiles the most important characteristics of this behavior. The importance of considering user behavior both for the design of OLAP database systems, and for OLAP applications is stressed by Section 1.1.4. It discusses the role of user behavior in the different phases of the OLAP system's lifecycle.

1.1.1 Architecture of OLAP Tools

Online Analytical Processing (OLAP) tools are designed for the interactive analysis of quantitative data that is relevant for decisions. The idea is to provide the analyst with a graphical interface that allows for interactively formulating and manipulating queries against a database

[1] From 1997 on, all large database conferences (VLDB, SIGMOD, ICDE) have at least one session on OLAP systems and several sessions on data warehouses. In 1999, the DAWAK conference has been established, focussing exclusively on data warehouse and OLAP related research. The 'Gesellschaft für Informatik' (German Society for Informatics) has dedicated active working groups on the topics of datawarehousing and multidimensional databases since 1996.

without any programming knowledge. The graphical user interface is chosen because of the possibility of intuitively formulating queries without profound IT knowledge. However, this requires that the data is structured according to a schema that naturally reflects the user's perception of the domain. In order to meet this requirement, OLAP systems present data at the logical level according to a special data model (*multidimensional data model*). The reason for this is that this data model (extensively described in Section 3.1) is well suited to mirror the inherent structures and analysis of business application domains. Thus, schemata formulated with respect to this data model are both intuitive to the enduser and expressive enough to allow for complex data analysis. This claim has been confirmed by the practical success of today's OLAP products.

Nevertheless, at the physical level data can be managed using any appropriate data model. The most popular approaches that are taken by commercial systems (cf. [DSHB98]) are the usage of the relational data management systems (called ROLAP approach) and array based multidimensional mechanisms (called MOLAP approach). Research prototypes also propose the usage of object oriented (for example, [BSH98]) or object-relational (for example, [ZRT+98]) storage mechanisms. However, the data storage is an internal feature of the OLAP database that should be entirely hidden from the user (analogously to the physical organization of data in a relational database e.g., using memory pages or clustering indexes). Therefore, throughout the thesis, we do not make any assumptions about the internal organization of the data facilitating the application of our approach to any current or future OLAP architecture.

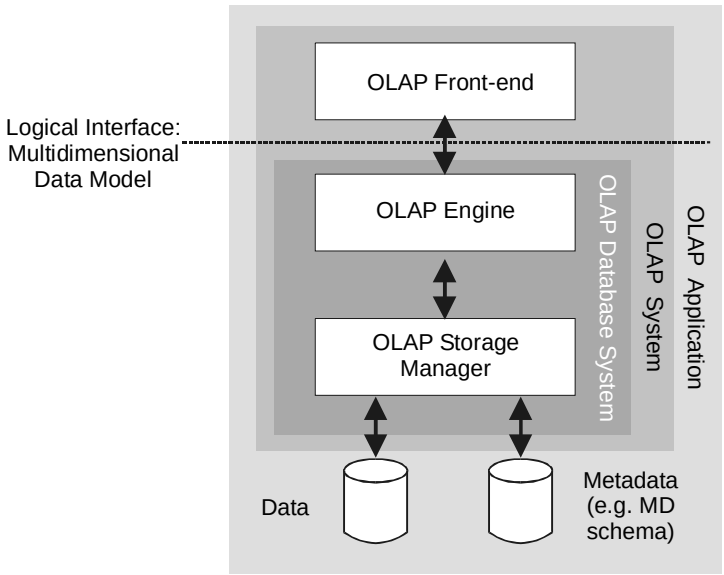

Figure 1.1: Terminology and Reference Architecture for OLAP Applications

Figure 1.1 shows the architecture of OLAP systems and clarifies our terminology. As already mentioned, the *OLAP database system* exposes a logical multidimensional data view that can be used to manipulate and query the data managed by the system. Data manipulation operations formulated with respect to this logical interface are mapped by the *OLAP Engine* to the physical data storage. In this process, it uses *metadata* (for example, the multidimensional schema and its mapping to the physical data structures). The data itself (and the metadata) is managed by the OLAP storage manager (for example, a relational database system in the case of a ROLAP architecture). However in an *OLAP system*, queries to the data are not formulated using a query language or an API but are interactively constructed using the *OLAP front-end* that also visualizes the results of the query. OLAP systems that are configured for a special

application domain (by providing a data schema, inserting data into the database, customizing the interface, predefining reports etc.) are called *OLAP Applications*.

Current product development and recent research activity in the field of OLAP systems has been mainly focused on describing the multidimensional data model (cf. Section 3.1) including a set of according query formalisms (cf. Section 3.2) on the logical and the conceptual level, enabling interactive response times for single queries by means of static (especially preaggregation) and dynamic view materialization (cf. Section 5.1.1 for a survey of dynamic approaches and Section 5.3.2 for selected static approaches), indexing (e.g., [MZB99], [WB98])) and clustering (e.g., [MRB99], [JLS99b]). However, the analysis of user behavior and the exploitation of the acquired information has so far been neglected.

1.1.2 Example Scenario: Material Management and Distribution Logistics

In order to illustrate the concepts throughout this thesis the following scenario of a material management system will be used. It is based on a combination of three real world case studies we performed at FORWISS, Knowledge Bases Research Group together with our industrial partners DaimlerChrylser AG (analysis of diagnosis data for vehicles, [HBD97]), Wacker Chemie GmbH (distribution logistics) and ESG GmbH (military logistics planning). Of course, all the company-specific details have been changed or omitted to protect the 'innocent' and to keep the complexity of the demonstration scenario manageable.

The material management planning system of our example scenario should be used in planning the supply of spare parts for different kinds of vehicles (automobiles, trucks etc.). A consolidated data warehouse contains data about which vehicles are deployed in which geographic regions, information about repairs of these vehicles (including which parts were exchanged, what type of failure occurred, the geographic location of the repair) and what kind of stock-keeping systems are available in the different regions.

A typical task of a logistic planner in this environment is to determine a distribution and stock-keeping strategy for a certain geographic region or repair location (for example, *how many parts are kept locally to satisfy anticipated demands*). To reach a decision, the human analyst uses information that is provided by the system concerning which parts fail how often under which conditions (climate, usage profile etc.). Due to the large number of influencing factors, no closed model is available to predict the failure characteristics. Therefore, the planner has to use expert knowledge together with the 'historic' data provided by the system to formulate a hypothesis about the anticipated demand for spare parts. Additionally, factors like budget, stock capacity, stock keeping strategies, priorities of demands etc. have to be taken into account to reach the final decision of determining the number of spare parts that should be kept in stock at a certain location for a certain period of time. Even from this short introduction it should be clear that the analysis process is far from being trivial and cannot be easily automated. However, the analysis process follows a methodological structure of subtasks. That means the analyst e.g., first tries to find out, if the geographic region he is planning for shows any anomalies in the failure characteristics compared to other regions (for example, the failure characteristics of a desert environment differ from those of an arctic environment). As a result, he picks one or more geographical regions that are similar to the location he is planning for. For these regions, the characteristics of previous years, quarters and month are being analyzed and so on.

This example already shows some of the characteristics of the user's interaction with the OLAP system. The next section systematically compiles the typical properties that are important to our approach.

1.1.3 Characteristics of OLAP User Behavior

The characteristic interaction between the analyst and the OLAP application has several properties that are a consequence of the OLAP user interface philosophy and the analytical environment in which they are mainly applied:

- *Session Oriented Exploratory Data Analysis*. When working with an OLAP system, the user typically has a (business) question in mind (e.g, "*How many parts for the steering assembly should we keep in stock at repair locations in Germany in order to guarantee the necessary degree of maintenance service*?"). In order to answer this question, the user formulates a query (for example, "*How many parts have been used for repairs of vehicles at the different locations in Germany during the last year*?"), analyzes the results, builds a hypothesis (for example, "*unscheduled repairs of the steering assembly mostly occur after cold periods.*") and verifies the hypothesis by executing another set of queries. All subsequent queries that are executed in order to solve the same analysis task are grouped to a *session*. The analysis of a real world decision support environment (see Section 6.1 for details) has shown that these sessions can contain up to several hundred queries depending on the complexity of the analysis task.
- *Interrupted by Cognitive Processes*. Due to the exploratory nature of the data analysis task, the dialog between the user and the system consists of two alternating phases: While the system processes the query, the user is waiting for the results. The duration of this *processing phase* is observed by the user as system latency time (which should of course be as short as possible). After having received the results, the user has to analyze the results. This is a cognitive process which takes some time and does not involve any system interaction. Therefore, from a systems point of view the cognitive process is a 'black box' that is not deterministic but predictable. The outcome of this process cannot be directly perceived by the system. Instead, the system must use the query executed at the end of the cognitive process as an indication to the outcome of the consideration process. Our empirical study in (cf. Section 6.1) has shown, that the consideration time (duration between two processing phases) is orders of magnitude larger than the execution time for the next query for a significant part of the queries (85% in the examined environment). This underlines the potential for applying speculative materialization techniques.
- *Navigational, Iterative Query Formulation.* According to the basic philosophy of OLAP systems the user interactively formulates the queries using a graphical front-end. Queries can be formulated in two different ways: The first alternative is to execute a predefined query template with a set of parameters that are determined in an ad-hoc way (for example, a report showing the total repairs in a geographical region for a year). This option is typically used to start a session or to begin a subtask (for example, *finding out if an assembly shows abnormal failure characteristics in a given region*). The second alternative is to iteratively modify the definition of the previous query by applying a set of multidimensional query transformations (e.g., slice, dice , drill-down) that are provided by the OLAP front-end tool. This process will be described in more detail in Section 3.2.2 (informally) and Section 3.2.4 (formally). The consequence of this iterative query formulation procedure is that the resulting user behavior exhibits strong locality properties.
- *Task and User Specific Patterns*. The sequence of queries for an analysis session mirrors the procedure that has been deployed by the user in order to solve her analysis task. As similar problems (for example, *determining the stock-keeping strategy for different locations*) will always be solved using the same workflow, sessions of the same user or user group solving these problems will be similar. Consequently, the query sequences exhibit patterns that are characteristic for the analysis task and/or the peculiarities of the users ap-

proach to the problem. Mathematically speaking, this means that the probability for a query to occur are mainly dependent on the current state of the analysis process and the specific user.

Having pointed out the special characteristics of the data access behavior, the next section will argue that it is beneficial to address these peculiarities in OLAP application and system design. It will show that knowledge about user behavior plays a central role throughout the lifecycle of an OLAP application, although this has so far not been fully realized by the OLAP community. This assessment is shared for example by [Vas00].

1.1.4 The Role of User Behavior in OLAP Applications

The purpose of an OLAP application is to satisfy the user's analysis requirements. Therefore, it should be obvious that knowledge about the user's query behavior is most important during the whole lifecycle of the OLAP system. Nevertheless, this aspect is underrepresented in current research work and in the state-of-the-art practice of designing OLAP applications. Therefore, this section systematically points out the role of user behavior for the different lifecycle phases of an OLAP application.

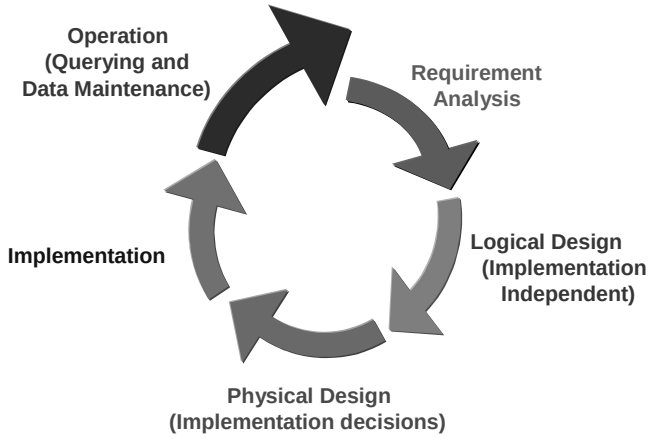

Figure 1.2: The Different Phases of the Iterating Design and Maintenance Cycle of OLAP Applications

Figure 1.2 shows the iterative macro-model for the OLAP application lifecycle ([Sap99]). The goal of the *Requirement Analysis* phase is to gather functional requirements for the system. As the high level requirement towards an OLAP system is the support of the analytical process of the user(s), the requirement analysis should contain descriptions of typical analytical workflows[2]. For this purpose a conceptual (implementation independent) description of user behavior is necessary. During the *Logical Design* phase these requirements have to be transformed into a logical schema (following the multidimensional data model) for the OLAP database. This schema must reflect the requirements as the static schema determines which queries can be executed. Therefore, the design of the logical OLAP schema should be driven by a description of the (anticipated respectively requested) user query behavior and a description of the interconnections between the dynamic model (user behavior) and the static schema. Further motivation for a user behavior centric modeling process comes from an observation the author has made in several real world projects: users usually can state very well which analytical task they want to perform, but have difficulties understanding the static data schema that is necessary to answer their queries.

[2] During later iterations of the design cycle, a description of these workflows can be derived from or compared to a description that is deduced from user behavior observed during the former *Operation* phases.

Knowledge about the anticipated user behavior is also a key input for the subsequent *physical design*. During this phase, the logical schema has to be mapped to a specific implementation (e.g., MOLAP or ROLAP). This includes decisions about physical optimization techniques (e.g., indexing schemes, physical clustering, materialization strategies, denormalization). Comprehensive research of physical optimization strategies for Data Warehouses and OLAP systems has been done for specific implementation strategies (relational e.g., [GHR+97] or multidimensional e.g., [FB99]). Irrespective of the approach, all of these solutions require the definition of a typical workload (i.e., a weighted set of queries or query patterns) as input. This workload can be obtained from the statistical analysis of query log files. However, during the system's initial design phase, this information is not available and when redesigning the system, it is unknown, if this data can be extrapolated into the future. This underlines the necessity to formally capture information about anticipated user behavior during early design phases and to formally document these requirements.

But even if static workload characteristics (for example, the relative frequency of a query in all sessions) are available, using them does only exploit a small part of the knowledge about user behavior in OLAP systems. In environments where user behavior exhibits navigational characteristics, the probability for a query to be executed next is highly dependent on the queries that were executed directly before it. Regularities of this kind are necessary for approaches that optimize the systems performance at runtime (i.e., during the *Operation* phase). We call these kinds of optimization *dynamic* optimization approaches. Most importantly, dynamic materialization techniques (e.g., caching and speculative execution techniques) can benefit from knowledge about navigational patterns. These techniques require that information about user behavior can be made known to the OLAP database system or can be acquired by the database itself during its operation (requiring a good representation of knowledge about user behavior patterns and according induction algorithms). Based on this information, the future user behavior can be predicted and the caching and prefetching strategies can be adjusted to optimally support the anticipated workload (for example, prefetching the results of the predicted query) which results in a substantially improved response time for the user.

Summing up, knowledge about anticipated user behavior can be exploited for OLAP application design and for OLAP database system design in a variety of ways. Taking the point of view of a database system's engineer, we are mostly interested in the additional opportunities that open up for dynamic system optimization techniques (caching and prefetching, cf. Chapter 5). Consequently, we will focus on these issues in this thesis. Additionally, further ideas about how to incorporate OLAP user behavior modeling into an OLAP application design methodology can be found in [Sap99], [BSH00] and in Section 7.3.1 of this thesis.

1.2 Objectives

Following the motivation discussed in the previous section, we can now define the high-level objectives of our approach thus roughly specifying the scope of this thesis. These objectives are the basis for a detailed specification of the PROMISE framework in Chapter 2 and define the evaluation criteria for the assessment of our approach in Chapter 7.

The basic assumption is that the behavior of an OLAP user (i.e., the types and sequences of queries she executes) is not deterministic but predictable. That means that for each analysis task the user is performing and for each user, specific patterns of interaction behavior exist. Therefore, the overall objective of the PROMISE approach is to *acquire*, adequately *represent* and *exploit* knowledge about these patterns in order to *dynamically improve* the response time of OLAP systems. This means that the following issues have to addressed by our approach:

- *Provide an Adequate Representation of Knowledge about Regularities in OLAP User Behavior*

  This work will present a formal representation technique for data access patterns that are caused by the characteristic query behavior of OLAP application users. A prerequisite for the pattern descriptions is a formal description of user interactions with OLAP systems (cf. Section 3.2) taking into account the peculiarities of the graphical front-end component. The pattern representation should adequately consider the inherent properties of OLAP user behavior in order to allow precise predictions. Furthermore, it must be flexible enough to represent different types of patterns on different levels of detail (cf. Section 4.2). In order to allow for an efficient prediction, the pattern information must be represented in a compact way and be accessible efficiently. This requires appropriate data structures and algorithms. An additional requirement for the pattern representation is that it must be possible to maintain the pattern information during the 'normal' operation of the OLAP system.

- *Provide an Efficient Prediction Algorithm*

  The core of the PROMISE framework is the prediction algorithm that computes a set of probable next interactions based on the pattern information expressed according to the pattern representation and the current context of the user's session. This prediction algorithm will be called during system runtime. Therefore, it must be designed in order to execute efficiently and scale well to growing pattern sets. However, the most important requirement is that the prediction algorithm provides a good accuracy i.e., that the queries predicted by it will be actually executed with a high probability by a user that behaves according to the assumed patterns.

- *Provide Efficient Algorithms for the Acquisition of Knowledge about User Behavior*

  An important factor influencing the accuracy of the prediction algorithms is the correct acquisition of the corresponding patterns. Basically, knowledge about the user behavior can be acquired using two different strategies: By analyzing the interactions the user performed with the system in the past (*ex-post* or *inductive* method). This requires techniques to analyze interaction logs that have been recorded (system centric approach). It is also possible to model the knowledge of domain experts about their analytical workflow (*ex-ante* or *deductive* method). In this thesis, we will focus on the inductive method, because it directly influences the architecture of OLAP systems, thus being interesting from a database point of view. We are aiming at learning algorithms that can be carried out online while the user queries the database.

- *Improve Dynamic System Optimization Techniques*

  Prediction in itself is not useful until its results are applied to improve the system's performance. As already pointed out, a multitude of enhancements are possible. However we already argued that we see the highest potential in the improvement of dynamic materialization techniques (i.e., caching techniques) which will consequently be the focus of this thesis. Therefore, we have to provide mechanisms to integrate the prediction algorithm with current OLAP caching algorithms and discuss extensions to the algorithms that make use of the predicted results in order to drive eviction decisions and to enable speculative execution of queries.

- *Practically Demonstrate Potentials of the Approach*

  Of course, the improvements of the PROMISE approach to the OLAP system's performance should be quantified. Being a heuristic approach, an analytical proof of the feasibility (let alone the optimality) of the approach is not possible. Therefore, we prove the feasibil-

ity of the approach by means of empirical evaluations. For every component (i.e., the prediction algorithm, the induction process and the caching algorithms), the evaluation process should measure its absolute performance and the influence of varying parameters. In order to research the impact of different kinds of user behavior on the performance of our approach, we employ a user simulation. The measurements should be carried out on real-world data.

Apart from these objectives concerning the topics to be discussed, our approach is also influenced by the following general objectives:

- *Clear Separation between Logical and Physical Aspects*. As already mentioned, OLAP systems can be implemented using different storage techniques (sometimes e.g., referred to as ROLAP, MOLAP). However, these implementation decisions are hidden from the user by the logical data model layer of the system. Therefore, any description of user behavior (and its patterns) should be independent from the orthogonal physical implementation strategies. This is especially important for OLAP systems, because many of the earlier but fundamental publications (mostly pragmatically oriented like e.g., [Inm96]) mixed up the physical and logical aspects, which has led to a lot of confusion, both in the research and the industrial user community.
- *Extensibility and Comparability of the approach.* As this is the first approach to apply prediction and anticipation techniques to OLAP systems, we see our work as the starting point for a series of research in this area. In order to be usable as a solid foundation for this kind of work, we aim at clearly structuring the problem domain, identifying and formally describing the most important design decisions. Therefore, we put great emphasis on providing formalisms to describe the fundamental issues and on explicitly mentioning and discussing our design decisions and their impacts on the overall approach.

  Additionally, we regard it as important that our approach can be compared to similar approaches in other domains. Therefore, we describe a large part of our approach independent of the application area of OLAP systems.
- *Smooth Integration with current State-of-the-Art Techniques*. Significant research work in OLAP database systems has been done in the past years on different abstraction levels: physical issues, logical data modeling issues and conceptual methodological issues. Wherever possible, we compare our approach to the work done so far and show possibilities of integrating our work with existing approaches.

1.3 Outline and Reader's Guide

This section describes the outline of the remaining thesis and can be regarded as recommendations on which parts of the thesis should be read and in what order depending on the readers interest.

The following Chapter 2 gives a comprehensive overview of the topic area of this thesis (i.e., prediction of user behavior). It contains a formal specification of the encountered challenges that have to be addressed by a prediction environment. It is therefore suited for readers who want to gain a comprehensive overview of the ideas discussed in this thesis. As the presented framework is independent of its application to OLAP tools, this chapter is also valuable for readers interested in prediction methods for other application areas (for example, predictive WWW proxy caches).

The topic of Chapter 3 is the theoretical background of OLAP database systems, namely the data model and the according query formalisms. The proposed formalisms constituting the

formal foundation for the thesis' core are derived after comparing the most important scientific approaches in this area. Therefore, this chapter is also valuable for readers looking for a comprehensive overview and assessment of research about multidimensional data models and query languages. It can be skipped by readers being familiar with current state-of-the-art OLAP systems research. However, we recommend at least to read our definition of the multidimensional model (Section 3.1.3) and canonical queries (Section 3.2.3) as these formalisms keep reappearing throughout the core of the thesis.

Chapter 4 and Chapter 5 constitute the core of this thesis. Chapter 4 describes an OLAP specific instantiation of the general prefetching framework developed in Chapter 2. This includes especially the pattern model (which is a combination of different Markov Models) and the according prediction algorithm. Therefore, it is intended for readers being interested in the details of our prediction approach and those wanting to implement it.

Chapter 5 discusses the application of the PROMISE/OLAP framework to improve the performance of OLAP caching systems. This section is also of general relevance to readers who are interested in the impacts of prediction techniques on semantic query level caching, especially in the OLAP area. It is suited for persons that are considering the benefits and efforts of integration prediction into an existing (OLAP) caching system.

As most of the algorithms proposed in this thesis are heuristic, their quality cannot be assessed using an analytical model but has to be proven by an empirical evaluation. Chapter 6 presents our evaluation framework, details the evaluation procedures and presents the results of both analyzing characteristics of real world user behavior and the performance of our prediction, caching and prefetching algorithms. The results confirm that our techniques enable substantial speed-ups for OLAP systems. The chapter is of special interest to readers looking for a practical evaluation of the PROMISE approach.

Finally, the contributions of the PROMISE approach, a direct comparison with related work, and possible extensions of the algorithm are discussed in Chapter 7. This chapter also contains further directions of research and might therefore be suited for scientists considering research in this area.

Notably, the thesis does not contain a single chapter that surveys the state-of-the-art scientific research and product development. As this thesis deploys, extends and augments techniques that have been developed in different fields of computer science research (e.g., OLAP systems, prefetching techniques, representation techniques for probabilistic patterns, caching techniques), we decided to split this survey according to the different areas. Consequently, we present the respective surveys at the appropriate point in this thesis (i.e., immediately before we use the techniques respectively before we propose techniques that are superior to current state-of-the-art practices). Therefore, the discussion of state-of-the-art practices is distributed across the following chapters:

- Multidimensional data model descriptions (Section 3.1)
- Query formalisms for multidimensional data (Section 3.2)
- Predictive prefetching techniques (Section 4.1)
- Caching algorithms for OLAP systems (Section 5.1)

In order to ease reference to formal definitions and symbols, we have included a Glossary at the end of this thesis, which lists the most important symbols, definitions and abbreviations.

«If we knew what it was we were doing,
it would not be called research, would it?»
-- Albert Einstein

«An undefined problem has an infinite
number of solutions»
-- Robert A. Humphrey

Chapter 2 The PROMISE Framework – A Bird Eye's View

The high level goal of the PROMISE approach is to enhance the performance of an OLAP system using information about patterns in the user's query behavior. In order to accomplish this, several different but interconnected issues have to be discussed. This includes e.g., how to represent the pattern information, how to obtain and maintain the patterns and how to actually exploit the pattern information for system optimization (for example, via prefetching and speculative execution strategies). The purpose of this chapter is to systematically compile and motivate these issues and their interdependencies. The chapter therefore constitutes a refinement of the problem statement given in chapter 1 and can be regarded as a 'roadmap' to the rest of this thesis. Throughout this chapter, we explicitly introduce assumptions and restrictions for the core part of the thesis to ensure a manageable complexity of the solution presented in this thesis.

The result of this chapter is a functional reference model for the pattern-driven prediction process. We break down the overall process into a set of different sub-processes that communicate via well defined interfaces. For each process we describe its functionality and motivate the main challenges involved in designing this component. Additionally, we specify a data representation for each interface. The framework assumes a general navigational information system (e.g., a hypertext application or an OLAP application), thus being independent of the actual system type (for example, an OLAP system). For each component, we additionally enumerate the steps that have to be taken to build a system specific (for example, for OLAP systems) instance of the framework. The core of this thesis (Chapter 4 and Chapter 5) then develops an OLAP specific instance of this general framework and discusses its application to predictive prefetching algorithms for OLAP systems.

The reference model does not only define the scope of this thesis but also builds a basis for discussing the application and adaptation of techniques from other research areas (for example, data mining) and for comparing different prediction approaches from different areas of application (e.g., file systems, Web-based systems, Object-oriented databases, see Section 4.1). Therefore, the PROMISE reference framework constitutes an important contribution of this thesis.

Before identifying and describing the different components of the framework, we start by presenting the most important objectives that drove the framework's design in order to moti-

vate our design decisions (Section 2.1). Subsequently, Section 2.2 gives an overview of the whole framework and describes the relationship between the components. Section 2.3 and 2.4 then describe the interfaces between processes of the framework. Each of the main processes is then discussed separately in Sections 2.5 to 2.8. The summary presented in Section 2.9 discusses how we reflected the design goals in the framework design and contains a graphical roadmap to the Chapters 4 to 6 of the thesis.

2.1 Objectives and Design Goals for the PROMISE Framework

Before presenting the framework itself, we first list the design goals that drove the development of the reference framework for the prediction process:

- *Applicability to a broad class of navigational information systems*.

  Benefiting from patterns in the navigational behavior of the user is not restricted to the area of OLAP systems. Therefore, the only assumption about the actual information system type contained in the design of the PROMISE framework should be that a human user interacts with the system in a navigational (explorative) fashion. Nevertheless, the framework must provide facilities to add the system type specific requirements when instantiating the framework for a specific system type (for example, an OLAP system). Additionally, predictive approaches have already been successfully applied to other systems that are characterized by such a navigational data accesses (e.g., branch prediction for pipelined microprocessor architectures or prefetching for file systems, see Section 4.1 for a comprehensive overview of application areas). However, a comparative discussion of these approaches is not easily possible, due to the different system types. The PROMISE reference framework therefore should be a suitable basis for the comparison of predictive approaches from different application areas regarding their functionality, algorithms and scope.

- *Facilitate the reuse of existing techniques developed in different research areas*.

  Different aspects of the prediction framework (for example, the induction of navigational patterns from observed user interactions) have been researched in various other contexts (for example, data mining algorithms for the analysis of customer behavior). The structure of the framework should facilitate the reuse, integration and adaptation of these techniques for the PROMISE approach. That means that the approach should not only be applicable to a wide range of information systems (previous objective) but benefit from different techniques which have already been developed or will be developed in future.

- *Support a wide range of applications for the prediction results*.

  Prediction techniques can be applied for different purposes (e.g., to improve cache admission and eviction algorithms, to enable predictive prefetching strategies, to build adaptive user interface etc.). Therefore, the structure of our prediction framework itself should not contain any assumptions about the deployment and further processing of the prediction results. Although obviously, the different areas of application (e.g., predictive prefetching, clustering strategies, visualization of user behavior, system redesign process) exhibit different requirements regarding the prediction results (e.g., the level of detail, the content of the prediction and the quality of the predictions). This means that the framework must offer mechanisms (for example, a generic abstraction concept) which can be used to configure an instance of the framework regarding the requirements of the application which uses the prediction results.

The above considerations about the design of the PROMISE framework lead to the following design decisions:

- *Modular design.*

  The framework is composed of independent components that communicate via well defined interfaces. This functional separation of concerns enables the modular instantiation of the framework by changing the algorithm for a single component and combining different algorithms for different components, e.g., combining different pattern induction algorithms with different prediction algorithms.

- *Facilitate the usage of patterns on different levels of generalization.*

  Having in mind a wide range of application areas (see Section 2.8), the PROMISE framework needs to support different levels of abstraction for the patterns and the prediction results. For example, the conceptual graphical modeling of anticipated user behavior during system design is necessarily done on a higher level of abstraction than for example observation of interaction patterns that are necessary to dynamically predict queries at runtime.

  Furthermore, different prediction applications require different information. For example, static tuning techniques (like preaggregation) do not make use of information about the statistical relationships between subsequent data accesses generated by the user's navigation. Instead they need information about how often an aggregation is being computed. On the other hand, dynamic tuning techniques like caching need information about the navigation characteristics.

These considerations lead to the framework architecture which is described in the following section.

2.2 A Process-Oriented Overview

The PROMISE framework can be abstractly modeled as a set of communicating processes. Targeting a generic functional framework, at this point we do not make any assumptions about the actual algorithms. Instead we formally describe the interfaces between the components by specifying data structures for the input and output data. The functional properties of the components itself are described in an informal way. This conforms to the modular architecture of the framework enabling the combination of different algorithms for different instances of the framework. The discussion of possible algorithms is the subject of chapters 4 and 5.

Naturally, the core process of the PROMISE framework is the *prediction process* (cf. Figure 2.1). This process receives information about the current context of the user's session. This may include the user's name, a session id, the last interactions performed so far during this session etc. Which information is actually passed to the prediction process is dependent on the requirements of the prediction algorithm being deployed. The algorithm generates a set of *predictions* i.e., interactions with their according probabilities. The prediction is based on a prediction model which takes into account the *patterns* of the user's behavior. These patterns are specific for individual users, user groups, user roles or application areas or any combination. When actually instantiating the framework, it is an important decision for the performance of the framework whether patterns should be recorded per user or per user group and how user groups or user roles should be defined. Although, as this decision is specific to the application domain of the information system, we do not make any assumptions about this issue right now.

It is important to note that the patterns cannot be 'hard-wired' into the prediction algorithm but are passed as a parameter to the prediction process. Section 2.5 discusses the design of the prediction process in greater detail. Of course, the patterns themselves have to be formulated according to a formalism (called pattern model) that can be understood and exploited by the prediction algorithm (see Section 2.4 for an in depth discussion of pattern models and a clarification of the notion of a pattern for the context of this work).

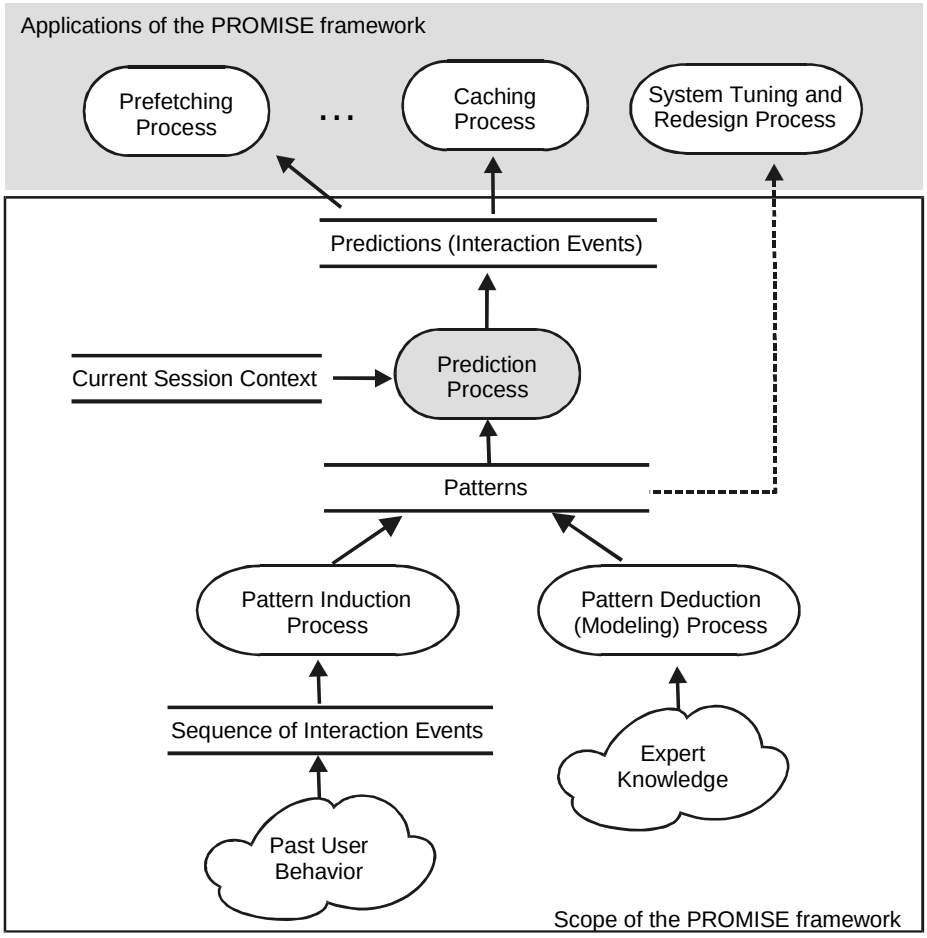

Figure 2.1: A Process Oriented View of the PROMISE Framework

The most common approach to gain information about the patterns that characterize the user's behavior with the system is an analysis of the interaction behavior of the user in the past. In the PROMISE framework, this is being done by the *pattern induction process* (see Section 2.6) which derives patterns for the interaction behavior from a representation of the user's past behavior (also called *ex post* approach). As we assume a system centric approach (in contrast to a human centered approach adopted by the user modeling community), the induction process works with the representation of the user's behavior that is being perceived by the information system (for example, a log file of the user's interactions).

As already motivated in chapter 1, the patterns that are present in the interaction behavior of the user are a mirror of the inherent structure of the analysis problem the user is working on (e.g., searching a document on the WWW or defining a new distribution strategy for spare parts). Therefore, pattern information can also be deduced from knowledge about the characteristic workflow of the analysis process. This approach is incorporated into the PROMISE framework through the *pattern deduction (modeling) approach* (see Section 2.7), where a do-

main expert specifies the structure of the analysis process in a detailed way, such that patterns can be easily derived from this description (*ex ante* approach).

We regard the ex-post and the ex-ante approach as alternative ways of generating patterns. A well implemented induction process normally derives more significant and valid patterns. Nevertheless, it assumes that a considerable amount of recorded interactions are available for induction. In early phases of the system design (for example, when the system is not even implemented) it is therefore inevitable to use the modeling approach.

On the application side, the prediction results can be used for a multitude of purposes (for example, for *predictive caching* and *prefetching algorithms*). But not only prediction results are beneficial for the information system, but also the patterns themselves. As they contain information about typical workloads, they can be interpreted in order to drive tuning decisions or can be used to redesign or evolve the information system. However a deeper coverage of this 'side-effect' is out of the scope of this thesis.

2.3 Modeling User Interactions

A formal description of the user's interaction with the information system is important, as such a description is necessary to define the following interfaces:

- The *result of the prediction process* describes interactions the user is likely to execute next according to the prediction model.
- The *description of the session context*, which is a parameter to the prediction process contains a description of the users interaction in the immediate past.
- The *input to the pattern induction process* is a description of past user interactions.

Furthermore, a formal description of the user's interactions is also a prerequisite for the formulation of patterns which are the data structure exchanged between the pattern induction and the prediction process.

Therefore, the first fundamental step towards predicting the user's behavior is to formally model the user's interactions with the system. We call this model the *user interaction model*. Generally speaking, the user's activity during working with the information system is a continuous process. However, from the system's point of view, the cognitive process of the user is perceived as a discrete sequence of atomic interactions. Taking a system oriented view, we therefore represent the user's behavior as discrete *atomic interaction events* which are perceivable by the information system (e.g., the execution of a multidimensional query or following a link within a hypertext environment).

Each *atomic interaction event e* is characterized by an *event type* t_e. Depending on the type of the event, additional information about the event is recorded. This additional information is modeled using a set of attributes for each event type t_e. The same attribute can be applicable to different event types. For example, events caused by executing of a multidimensional query belongs to the event type *query execution*. Attributes describing this event type are the time when the event was initiated, a user id of the user initiating the event, a session id and a description of the query. The execution time and user ID are attributes shared with other types of events.

For the class of information systems that we assume for PROMISE (see previous section), the user works with the system in a session oriented way. In order to mirror this central assumption in the interaction model, it is necessary to group the atomic interaction events to *sessions* or *composite interaction events*. Each session *Ses* consists of a finite, discrete, line-

arly ordered sequence of *interaction events* $Ses =< e_1,\ldots,e_n >$[3]. The linear order is defined by the point in time when the events were initiated by the user. It is legitimate to assume linear sessions as we only perceive user initiated events and even if the system processes requests in parallel, the requests of a single user starting these processes are still sequential. This is also true in multi-user environments because every session is executed by a unique single user. This means that a session constitutes a virtual single user environment. Of course, a single user can still work in different sessions simultaneously. The order of the events $e_1,...,e_n$ defines a logical timescale for the session *Ses*. Notably, this definition only specifies the syntactical structure of sessions but consciously leaves the question of how to actually group the events unanswered. For the abstract definition of the framework and for the design of the algorithms this is sufficient. However, when actually instantiating the system for a specific system type and application domain, the decision of how to group single interaction events to sessions is critical as it may have a large impact on the performance of the system. The reason for this is that in the context of PROMISE framework only patterns between interactions in the same session are being recorded and exploited. The session defines the *entity of prediction* for the PROMISE approach. In the ideal case, the definition of a session based on the semantic content of the session i.e., a session always groups interactions that are semantically linked in the application domain of the information system (for example, a session lasts as long as a user visits Web pages that are related to the same topic). However, in the general case, this prohibits the automatic recognition of sessions, because a detailed modeling of application semantics would be necessary. Therefore, in practice often an approximation of this ideal grouping has to be deployed, using conditions that can be easily evaluated by the system automatically (for example, interactions belong to the same session, if they occur within a predefined timespan, or interactions belong to the same session, if they occur between special interactions like a system login and a logoff). Additionally, some system type-specific limitations have to be overcome (for example, the statelessness of the HTTP protocol in WWW environments, solutions for this problem have been proposed e.g., in [CMS99] and [ZAN99]).

Depending on the environment, integrity constraints apply to the attributes of *atomic interaction events* (for example, forbidding certain combinations of attribute values) and to sessions forbidding certain sequences of event types (for example, demanding that the *start session* event always occurs before the *terminate session* event). These global integrity constraints mirror either the special semantics of the information system type (for example, the WWW domain) or the limitations of the user interface (which e.g., only allows certain interaction sequences). We decided to include these restrictions as an integral part of the user interaction model. This helps to build a better prediction model because an integrity constraint must be fulfilled for all possible events and sessions. Therefore, these integrity constraints themselves can be interpreted as patterns which apply to all sessions and are independent of the application domain (although they are specific for the system type). Consequently, they can be used by the prediction algorithm to narrow down the set of candidates for subsequent interactions during the prediction process by sorting out candidates which do not fulfill the global integrity constraints.

The above considerations about the modeling of the user's interaction with the information system are summarized by the following set of definitions. The schema for events in an application domain is defined by a *user interaction model*.

[3] we use pointy brackets to denote finite sequences, e.g. Ses:=<s_1,s_2,s_3>. To reference the i^{th} element s_i of a sequence S, we use notational abbreviation S[i]

Definition 2.1 (User Interaction Model)

A user interaction model *IM* is defined as a tuple $IM := (T, A, attr, I)$ where

- T is a finite set of *event types*
- A is a finite set of attribute names describing events. Each attribute $a \in A$ has a domain $dom(a)$ attached.
- $attr : T \rightarrow 2^A$ is a function mapping each event type to the set of attributes which can be used to describe events of this type.
- I is a set of integrity constraints defined over the set of events and the set of event sequences. ◆

While the *user interaction model* defines the structure of the user's interactions, the interactions itself are described by 'instances' of the user interaction model. The simplest form of an interaction is an *atomic interaction event.*

Definition 2.2 (Atomic Interaction Event, Atomic Event Space)

Let $IM := (T, A, attr, I)$ denote an user interaction model. An *atomic interaction event e* (short *event*) for is defined as a tuple $e := (t_e, v_e)$, where $t_e \in T$ is called the *type* of *e.* If $attr(t_e)=\{a_1,\ldots,a_k\}$ denotes the attributes for this type t_e , then $v_e \in \bigtimes_{i=1,\ldots,k} dom(a_i)$ is a tuple containing the attribute values.

The *atomic event space* for *IM* is defined as the set of all possible atomic events fulfilling the integrity constraints and is denoted as E_{IM}. ◆

For notational convenience, we reference the attribute values of an event $e := (t_e, v_e)$ as e.<attribute name>. E.g., if the attribute name is 'time', the corresponding value for the event *e* is denoted as *e*.time. Additionally, we use the function *type*: $E_{IM} \rightarrow T$, $type(e)=t_e$ to denote the type of an event.

An atomic interaction event represents the finest granularity of events that are perceivable by the system. These atomic interaction events can be combined to represent sequences of interactions called *sessions* or *composite interaction events*.

Definition 2.3 (Composite Interaction Event, Composite Event Space)

A valid session (or composite interaction event) *Ses* in $IM := (T, A, attr, I)$ is defined as a finite sequence $Ses \in E_{IM}^*$.

We denote the set of all composite interaction events that fulfill the integrity constraints defined in the user interaction model as the *composite event space* E_{IM} of *IM*. Thus,

$$E_{IM}^C := \left\{ Ses \in E_{IM}^* \mid \forall i \in I : Ses \text{ fulfills } i \right\}$$

◆

As we will see later, for the prediction process, the events occurring directly before an event *e* are particularly interesting as they can be used to predict the next events. The sequence of the *m* events just before an event *e* are referred to as the *session context* (of order *m*) of this event *e*. If the session contains less than *m* events before *e*, the session context consists of the sequence of all events that occurred so far in this session.

Definition 2.4 (Session Context of an Event)

For a session $Ses := <e_1, e_2, .., e_n>$ the *session context* of order m for an event e is defined by the function $cont_m^{Ses} : E_{IM} \rightarrow E_{IM}^*$ with

$$cont_m^{Ses}(e) := \begin{cases} < e_{i-m}, e_{i-m+1}, \ldots, e_{i-1} > & \exists i \in [1;n] : Ses[i] = e \wedge i > m \\ < e_1, \ldots, e_{i-1} > & \text{if} \quad \exists i \in [1;n] : Ses[i] = e \wedge 1 \leq i \leq m \\ \perp & else \end{cases}$$

Notably, we assume that each event e is unique within a session (i.e., that at least the time-stamp attribute has a different value). Therefore, the session context for an event e is uniquely defined . ◆

The following example illustrates these concepts by presenting a very simple interaction model for the World Wide Web.

Example 2.1: (A Simple Interaction Model for the WWW)

A very simple interaction model $IM_{www} := (T_{www}, A_{www}, attr_{www}, I_{www})$ describing a World Wide Web user model might contain two types of interaction events: the user following a link from a page or a user choosing an URL from her bookmarks.

- T_{WWW}={*follow_link, follow_bookmark*}

For both event types it is sensible to record the URL of the destination and the time when the event was executed. For the *follow_link* event type also the starting URL can be recorded.

- A = {*time, destination_URL, start_URL*}
- $attr_{WWW}$(*follow_link*)= { *time, start_URL, destination_URL*}
  $attr_{WWW}$(*follow_bookmark*)= { *time, destination_URL* }

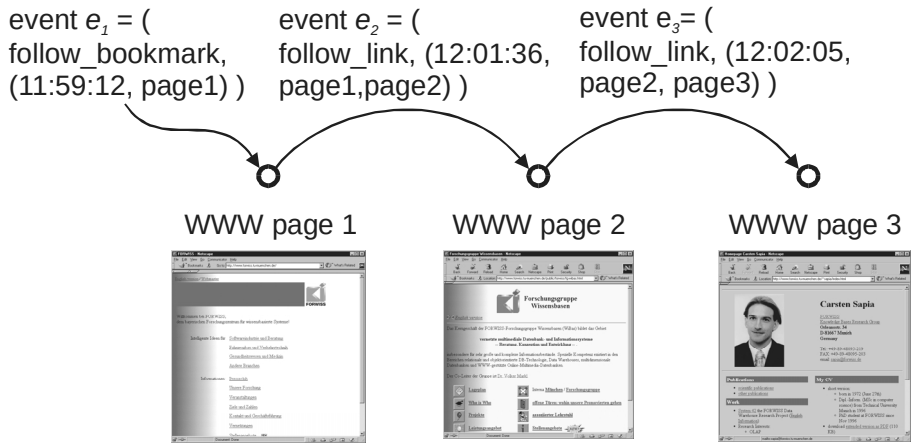

Figure 2.2: Visualization of a Sample Session Conforming to the User Interaction Model IM_{WWW}

Figure 2.2 illustrates a sample navigational session. The WWW user starts the session by choosing a link from the bookmarks which takes him to page 1. From there, he navigates to page 2 and subsequently to page 3 following links contained in these page. According to the above user interaction model IM_{WWW}, this session is modeled as the following sequence of events $Ses_{example} = <e_1, e_2, e_3>$. The attribute values for the events are shown in Figure 2.2.

An obvious integrity constraint for valid sessions is that for every event of the type *follow_link,* the start URL must be equal to the destination URL of the previous event. This con-

straint is a consequence of the WWW user interface design and is formalized by the following integrity constraint, which is an element of I_{WWW}.

- $\text{integrity}_1 := \left(\begin{array}{c} \forall S \in E^*_{IM_{WWW}} : \forall < e_1 e_2 > \in E^2_{IM_{WWW}} : \\ t_{e_2} = \text{follow\_link} \Rightarrow e_2.\text{start URL} = e_1.\text{destination URL} \end{array} \right)$

Another integrity constraint for event of the type *follow_link* is that the destination URL must be contained as a link in the referring page (i.e., the start URL). This is a consequence of the hypertext paradigm. In contrast to the previous integrity constraint, this constraint cannot be checked using only the information provided with the events. Therefore, the following formalization uses an external predicate *contains_link*(*a*,*b*) which indicates that page *a* contains a link to page *b*:

- $\text{integrity}_2 := \left(\begin{array}{l} \forall e = (t_e, v_e) \in E_{IM_{WWW}} : t_e = \text{follow\_link} \Rightarrow \\ \text{contains\_link}(e.\text{start URL}, e.\text{destination URL}) \end{array} \right)$

Of course, the complete set of integrity constraints contains more integrity constraints regarding the attribute values of atomic events or the events of composite interaction events.

- $I_{WWW} := \{\text{integrity}_1, \text{integrity}_2, ...\}$

◆

Summing up, the user interaction model provides a formal description of the interaction the user can perform with the systems. This means that the first step of a system specific instance of the framework is the design of a user interaction model. The design of the OLAP specific interaction model IM_{OLAP} will be discussed in Section 3.3.

2.4 Modeling Patterns in User Interactions

The *Pattern Model* representing the patterns in the user's interactions is the central communication point of the PROMISE reference framework. The pattern induction process as well as the modeling process both produce results which must conform to the pattern model and the patterns constitute the primary input for the prediction process. The pattern model formalizes statistical properties of interaction events and event sequences and is therefore based on the interaction model described in the previous section.

The pattern model defines how knowledge about regularities in the user's behavior is being represented. Therefore, its design directly determines the quality of the prediction results and the improvements of the system's performance through the application of the prediction results. On the other hand, the information about patterns must be stored, maintained and evaluated at runtime. This calls for compact representation techniques and data structures that allow for an efficient prediction.

The concept of a pattern is of central importance for the rest of this work, and so far our understanding of this term for the context of PROMISE has not been properly defined. Therefore, the following Section 2.4.1 discusses a possible general definition of patterns and pattern schemata independent of the PROMISE framework and the prediction of user interactions, before we present the two dominant types of patterns relevant to the PROMISE environment together with a formal definition of a pattern model in Section 2.4.2.

2.4.1 What is a Pattern in General?

The purpose of this section is to clarify our understanding of the term *pattern* and the associated terms: *pattern representation language* and *pattern schema* independently of the PROMISE framework in a general and abstract way. The recognition and formulation of patterns plays a role in different research areas (image analysis, knowledge discovery in databases, machine learning etc.). Consequently, a variety of different definitions for the term pattern has been introduced. However, it is commonly agreed that a general and precise definition of this term (like information, knowledge etc.) is not sensible. For the PROMISE approach, we follow the understanding of a pattern that is common in the area of Knowledge Discovery in Databases (KDD) and formulated e.g., in [FPS+96b], [KZ96].

Let us assume a set of abstract objects C (called facts or cases in the KDD terminology; e.g., a set of multidimensional query sessions or insurance contracts) for which we want to formulate patterns. Each object (or case) is described by a certain set of attributes (for example, the time when the session started, the user executing the session, the queries executed during this session or the income of the contractor). A *pattern* describes a regularity regarding a subset of objects in C (e.g., sessions containing a certain query q also contain the queries q_1 and q_2 after the query q or insurance contracts that were profitable are owned by persons with an income of over $105.000).

A pattern should compactly describe objects in an interesting subset of C (for example, all sessions that contain the query q or all insurance contracts that were profitable). We call a formal expression p that intensionally describes such a subset $C_p \subseteq C$ a *pattern*. Mathematically speaking, a pattern p is a formal predicate over the set of objects C that identifies a subset $C_P \subseteq C$ of the cases C. The formal language L that is used to represent the patterns is called the *pattern representation language* (or pattern representation for short). Notably, the pattern representation must not only contain a definition of the syntax of the patterns but also somehow define the semantics (i.e., the interpretation in a certain application domain) of the pattern expressions that are expressible using the syntactic definitions. Naturally, the choice of representation language is a very fundamental design decision as it restricts the set of patterns that can be formulated. This problem will be addressed in the following Section 2.4.2 for the abstract framework and will be refined in Chapter 4 specifically for OLAP systems.

A single fixed pattern representation still allows for formulating a large set of different patterns all conforming to the language description. When designing algorithms that require patterns as an input (which we have to do for the PROMISE framework), it is desirable to further specify restrictions for the set of patterns in order to describe the type of patterns which can be processed by the algorithm. Therefore, we introduce the notion of a pattern schema which (analogously to a database schema) defines a common structure for a set of patterns. Formally, a *pattern schema* is a pattern which contains free variables. A pattern is an *instance of a pattern schema*, if it can be obtained from the pattern schema by instantiating the free variables or by binding the free variables using quantifiers.

In order to illustrate these concepts, we can loosely compare a pattern model (pattern representation, pattern schema and patterns) to the model of a database. The pattern representation language of the pattern model corresponds to the data model (for example, the definition of a relation in the relational model) of the database in the sense that it fundamentally determines what is expressible in the context of the database/pattern model. A pattern schema corresponds to a database schema (for example, the schema of a relation) as it specifies the common structure for the data/patterns that are considered valid within the context of the database/pattern model. Algorithms are designed with respect to certain pattern schemata. That

means that they can work with any set of patterns that contains only instances of the appropriate pattern schema. This analogy also shows that the choice of a pattern representation is a very global decision while the choice of pattern schemata is algorithm specific. Therefore, the choice of appropriate pattern schemata for the prediction of OLAP user behavior is subject to discussion in Chapter 4 where the prediction algorithm for OLAP systems is discussed.

The following definition summarizes this understanding of patterns as instances of pattern schemata with respect to a certain pattern representation:

Definition 2.5 (Pattern Representation, Pattern Schema, Pattern)

Let L be a formal language which contains predicates to describe subsets of a set of objects C. An expression p of the language L is called a *pattern* characterizing a subset C_p of C if it describes the objects in the subset C_p in an intensional way[4].

If a pattern expression contains free variables or parameters, the expression is called a *Pattern Schema*. The language L is called the *Pattern Representation Language*. ◆

Example 2.2 (Representing Patterns and Pattern Schemata Using Rules)

Rules are a popular form of representing patterns. A rule has the form LS⇒RS, where LS and RS are predicates of an appropriate formal language. The rule LS⇒RS has the semantic meaning that if the interpretation of LS is true, the interpretation of LS also holds. Let us consider the example of insurance contracts. Each contract is described by a set of attributes (e.g., age of the contractor, income of the contractor, a flag indicating if the contract is profitable). An interesting pattern characterizing the profitable contracts is:

income > **$105.000⇒ profitable** = **'yes**'

A pattern schema defines the structure for a set of patterns. e.g., the following relatively restrictive pattern schema fixes the structure for both sides of the rule and only allows the variation of the parameter x which defines the limit for the income:

income > x ⇒ **profitable='yes'**

Patterns conforming to this schema are *income>5000⇒ profitable='yes'* or *income>40.000⇒ profitable='yes'*

The following pattern schema allows more freedom for instances, as the attributes for the left and right hand side of the rule are also parameters of the schema.

[attribute-name1] > x ⇒ [attribute-name2] = y

Patterns conforming to this schema are *income>5000⇒ profitable='yes'* or age>*60⇒ profitable='no' and* income>*100.000⇒age=60.* Still the schema demands that the left and right hand side only contain one attribute-name. A pattern schema that allows two attributes to influence the target attribute this is the following:

[attribute1]>x ∧ [attribute2]>y ⇒[attribute3]=z

◆

The goal of defining patterns in the PROMISE context is to build a model which describes the navigational interaction of the user with the system. Generally, two fundamental approaches can be distinguished (see e.g., [FPS96b]): the probabilistic approach and the deterministic approach. A deterministic approach is based on the assumption that the modeled domain (in

[4] of course, the term 'intensional' is subject to interpretation (reflecting different views of the term pattern). For example [FPS96] demands that the description "is simpler (in some sense) than the enumeration of all objects of the subset"

our case the interaction of a user) can be described in a deterministic way. In the context of our pattern model this means that a pattern is only considered valid (and should therefore be part of the model), if it holds for all possible cases for all time (this corresponds to the definition of a pattern in Definition 2.5) .

In contrast to this, a probabilistic model allows for uncertainty in the modeling process. That means, the model contains means to express the uncertainty (usually probabilities). As the domain we are modeling involves the cognitive process of interactive users (which cannot be modeled in a deterministic way), we use the probabilistic approach for PROMISE. This means that all patterns are probabilistic in the sense that they are regarded valid even if they do not hold for all possible cases in the application domain (for example, even if a person that has an income of over $105.000 may have an insurance contract which is not profitable). The pattern described in Definition 2.5 is a deterministic pattern. In order to express a probabilistic model, every pattern is extended with a probability value, which denotes the certainty that a new case satisfies the pattern.

Definition 2.6 (Probabilistic Pattern)

Assuming a (deterministic) pattern *p*, the tuple (*p*, *prob*) is called a *probabilistic pattern*, if $prob \in [0;1]$ is the probability that *p* holds in an application domain. ◆

Example 2.3 (Probabilistic Rules)

An example for a probabilistic pattern type are probabilistic rules. A probabilistic rule is based on the definition of a rule and is a tuple (LS⇒RS, *prob*), where *prob* denotes the conditional probability of predicate RS if LS holds. Thus, $prob(LS \Rightarrow RS) = prob(RS|LS)$. ◆

2.4.2 Patterns in the PROMISE framework

The previous section gave a very general definition of a pattern as a predicate describing a significant subset of a set of objects C. This section takes a closer look at the regularities that should be described by patterns in the PROMISE framework. The PROMISE interaction model (see Definition 2.1) defined two types of "objects": *atomic interaction events* (described by a type and certain attributes) and *composite events (Sessions)* composed from atomic interaction events. Therefore, we can basically distinguish two different types of patterns (see also Figure 2.3 for a visualization):

- *Single-Interaction Patterns* describing interdependencies between the attribute values of a single interaction event. I.e., the pattern describes how the value of certain attributes influences the values for other attributes for a single interaction event irrespective of the session context. E.g., Multidimensional queries containing restrictions for vehicles of brand 'BMW' often also contain a restriction for 'Bavaria'.
- *navigational (or sequential) interaction patterns* capture regularities between subsequent interactions during a session. E.g., query *q* is often contained in sessions that previously contained queries q_1, q_2 and q_3 in this order. A sequential interaction pattern models the fact that the probabilities for an interaction event to have certain attribute values is dependent on the session context of the event. The session context of an event are the last events that occurred directly before the event itself (see Definition 2.4). Notably, the predicate describing the sequential pattern may refer to single attribute values of the events involved, e.g., describing the influence of an attribute value av_1^1 in event e^1 on another attribute value av_3^4 in event e^4.

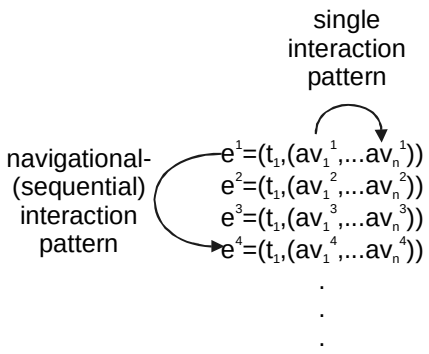

Figure 2.3: Different Types of Interaction Patterns

The patterns contained in the sequence of interaction events are a representation of different constraints from the information system's domain:

- *Integrity constraints of the interaction model* (cf. Definition 2.1): The interaction model defines a set of integrity constraints (for example, that for a follow link event, a link must exist between the start and the destination page). This includes restrictions of the user interface design. E.g., if it is known that the system only supports links with a single target, the interaction model is simpler. These patterns are independent of the application domain and are known at the design time of the prediction algorithm and the pattern schemata.
- *Structure of the analysis process and constrains of the application domain:* Constraints of the application's domain (such that it is not sensible to analyze a certain combination of geographic regions and vehicle although it would be allowed by the data model) and the inherent structure of the analysis process are mirrored as characteristic patterns in user behavior. These patterns are not known when designing the prediction algorithm as they are application domain specific (for example, logistics planning).

Independent of the two basic types (single vs. navigational), PROMISE uses generalized patterns i.e., patterns formulated on a higher level of abstraction. The definition of event types and their corresponding attributes for an interaction model implicitly defines the finest granularity on which patterns can be defined, because the expression describing the pattern is formulated with respect to the attributes. On the other hand, application domain specific classifications (see Section 3.1) can be defined for events e.g., by using classifications defined for single attributes of the event. For example, if the time of occurrence is an attribute of the event and is being measured in seconds in the interaction model, a classification hierarchy over the events can be defined using the hour when the event took place. If the formal language that is used to define the patterns allows for using classification respectively generalization functions as part of a predicate, it is possible to formulate so-called generalized or abstract patterns. E.g., *income_class*='medium' $\Rightarrow$ *profitable*='yes'.

Generalization leads to patterns with a higher significance especially in application areas where the number of possible events is large (or even infinite). It is important to note though that generalized patterns naturally carry less information than detailed patterns. Therefore, the prediction process can only make generalized predictions from generalized patterns. That means, the prediction algorithm must use additional models in order to provide the missing information.

In order to formulate generalized patterns for the PROMISE framework, we have to define generalizations over atomic and composite events. Mathematically speaking, a generalization is the definition of a complete partitioning of the set of all events (resp. sessions). Such a partitioning defines an equivalency relation $\alpha \subseteq E_{IM} \times E_{IM}$ on the set of all atomic (resp. complex)

events. Depending on the requirements of the prediction application and the prediction algorithm used to implement the prediction process, different generalization strategies are feasible. Therefore, a domain specific instance of the pattern model defines generalizations for the interaction events. Generalizations can be defined for simple events or for composite events (sessions). For the scope of the PROMISE framework, we assume that the set of generalizations is designed in advance and thus is static for the system. A possible extension of this concept by inducing possible generalizations from the interaction data is discussed in Chapter 7.

The following definition summarizes the above considerations about the components of an interaction pattern model, forming the core of the PROMISE framework:

Definition 2.7 (Interaction Pattern Model)

An Interaction Pattern Model (or short pattern model) for an interaction model IM=(T, A, *attr*, I) is defined as a tuple PM_{IM} =(L, G, S), where

- L defines a first order logical language which contains typed variables for all the event attributes $a \in A$. L is called the Pattern Representation Language.
- $G=\{g_1,..., g_k\}$ is a finite set of generalization (or grouping) functions for events. Each generalization function g_i is defined by an according equivalency relation $\alpha_i \subseteq E_{IM} \times E_{IM.}$
- $S=\{s_1,..., s_m\}$ is a set of pattern schemata formulated using L possibly containing generalization functions of G

The set of patterns which are formulated using L and are instances of a pattern schema $s \in S$ is denoted as P_{PM}. ◆

Example 2.4 (A Simple Pattern Model for WWW Interactions)

In this example, we present a pattern model for the interaction model IM_{WWW} introduced in Example 2.2 that can be used to model regularities between accesses to web pages that often occur directly after one another in user sessions. Such a pattern model can be used for predicting the page that is likely to be visited directly after a given page. The according pattern model $PM_{IM\,WWW} = (L_{WWW}, G_{WWW}, S_{WWW})$ contains an appropriate pattern representation language L_{WWW} which contains

- a set of typed variable symbols for URLs (x and y), a set of variable symbols for simple interaction events (a and b) and for sessions (Ses) and
- symbols to formulate first order logical expressions (e.g., '$\Rightarrow$', '$\wedge$'),
- a special relation symbol $follows_n(a,b)$ which indicates that event b follows event a within a look-ahead window of size n in a session S. For the rest of this example let $Ses \in E^*_{IM_{WWW}}$ denote a session and $a,b \in E_{IM_{WWW}}$ denote (atomic) interaction events. That means, for all $Ses \in E^*_{IM_{WWW}}$ and $a,b \in E_{IM_{WWW}}$ the relation

  $follows_n : E^*_{IM_{www}} \times E_{IM_{www}} \times E_{IM_{www}}$ with

  $follows_n(Ses,a,b) :\Leftrightarrow \exists U,V,W \in E^*_{IM_{WWW}} : Ses = U \circ a \circ V \circ b \circ W \wedge |V| < m$.

  For example $follows_2$ (<*abcde*>, *'b'*, *'d'*) evaluates to *true* while $follows_1$(<*a,b,c,d,e*>, *b, d*) evaluates to *false*.

A simple pattern schema (from S_{WWW}) that characterizes patterns where two pages are visited directly one after the other is described as follows:

$$\forall Ses \in E^*_{IM_{WWW}}\ a,b \in E_{IM_{WWW}}:$$
$$a.\text{destination\_URL} = x \wedge \text{follows}_1(Ses,a,b) \Rightarrow b.\text{destination\_URL} = y$$

The pattern schema contains two free variables *x* and *y* which can be instantiated by URLs. For notational convenience we denote instances of this pattern schema as $x \rightarrow_1 y$. An example pattern for this schema would be "http://www.forwiss.de"$\rightarrow_1$"http://www.forwiss.de/wibas/".

Another schema that captures patterns where a page is visited within three navigation steps looks like this:

$$\forall Ses \in E^*_{IM_{WWW}}\ a,b \in E_{IM_{WWW}}:$$
$$a.\text{destination\_URL} = x \wedge \text{follows}_3(Ses,a,b) \Rightarrow b.\text{destination\_URL} = y$$

Assuming a classification of web pages e.g., into personal homepages, publication pages, project pages etc., interaction events can be classified according to the type of their destination page. This classification of events (which is an element of *G*) is formally defined by an equivalency relation $class_{URL} \subset E_{IM_{WWW}} \times E_{IM_{WWW}}$ which is defined by the following expression:

$$(a,b) \in class_{URL} \Leftrightarrow \text{type}(a.\text{destination\_URL}) = \text{type}(b.\text{destination\_URL}) \text{ for } a,b \in E_{IM_{WWW}}$$

Using this classification it is possible to define generalized pattern schemata, like the following, which captures interdependencies of the type of two subsequently chosen pages (for example, that requests for publication pages are followed by requests for personal homepages):

$$\forall Ses \in E^*_{IM_{WWW}}\ a,b \in E_{IM_{WWW}}:$$
$$type(b.\text{destination\_URL}) = \text{publication page} \wedge \text{follows}_1(Ses,a,b) \Rightarrow$$
$$type(a.\text{destination\_URL}) = \text{homepage}$$

◆

The appropriate design of the pattern model (choosing a pattern representation and designing pattern schemata and generalizations) is the most critical issue when building a domain specific instance of the PROMISE framework. If the pattern model is too simplistic it may not allow for capturing the characteristic patterns of the user behavior, no matter how much effort is put into designing the prediction and pattern induction process. On the other hand, a highly complex model (which ensures a good expressiveness) allows for formulating, detecting and exploiting more sophisticated patterns. Nevertheless, a more complex model also means a more complex induction and maintenance process and causes higher storage costs. Another well-known problem that can occur with models that are too complex is overfitting (for example, [FPS96b]): If a model allows for representing the patterns on a very fine level, this leads to models that very accurately represent past data, but loose predictive accuracy as they do not generalize at all.

The essential design of the pattern model for OLAP applications is detailed in Section 4.3. There, we discuss an OLAP specific interaction pattern model based on Markov models. We develop a formalism to represent the patterns and discuss different generalization techniques and their impact on the prediction process and its applications.

So far, we have described the data models which are necessary to describe the communication between the different processes of the framework. The next step is to take a closer look at

the dynamic processes themselves and to identify the challenges involved in designing these components.

2.5 The Prediction Process – The Core of the PROMISE Framework

The *prediction process* forms the core of the PROMISE framework. It is initiated and controlled by the process which applies the prediction results in order to improve the information system performance (called *application process* for the rest of this section). A predictive caching process is an example for such an application process. Whenever the application process needs predictions of future user interactions (for example, when determining if an object should be evicted from the cache or kept because it is likely to be accessed in the near future), it starts the prediction process. The prediction process can then access the current *session context* (or session history) which contains administrative information about the current session (for example, user name etc.) together with a log of the last *m* events that occurred during the session. The session context is described according to the interaction model *IM*. The number *m* of events known to the prediction algorithm is called the look-back window of the prediction process. Additionally, the prediction process needs access to the patterns which are expressed with respect to an according interaction pattern model PM_{IM}.

Notably, the prediction process only evaluates the patterns in order to generate predictions. The conclusions for the application system are drawn solely by the application process (for example, whether or not to evict an object from the cache). This separation of concerns allows a separate discussion of different prediction algorithms without knowledge about the internals of the application process (for example, a predictive caching process) and vice versa.

The design of prediction algorithms for OLAP interactions based on different pattern interaction models is thoroughly discussed in Section 4.3. However the most interesting general issues when designing the prediction process include:

- *Choice of prediction results.* The application process requires that the prediction process returns the predictions that are most beneficial for the task of the application process. However, the definition of "benefit" varies for different applications. The definition of benefit may be based on the probability of the event, e.g., the prediction process can predict the most likely event or all events which fulfill a minimum probability threshold. Another alternative would be that a fixed number of most probable events are being returned. A more complex alternative is the usage of an application-specific cost/benefit-function. This function, which contains a parameter for the predicted probability is passed to the prediction algorithm by the application process. The prediction algorithm then returns predictions with the highest cost/benefit ratio according to this function. E.g., if the application process maintains a query level cache, some queries can be re-computed cheaply while others are very costly to compute. Clearly, it is a good strategy to cache queries that are costly to re-compute and have a high probability of being executed next. Therefore, the cost/benefit ratio formula for this application process would contain the re-computation effort as costs and the probability of the query being executed in the near future as benefit. For PROMISE, we discuss the different alternatives and their implication to our algorithm in Section 4.3.
- *Generalization of prediction results.* Normally, the prediction process predicts events on the granularity of the interaction model, but for some applications it might be sufficient respectively advantageous to return generalized predictions. E.g., if a proxy cache in a WWW environment maintains documents of different types (e.g., images, dynamically generated pages, static pages), for an eviction decision it might be enough to receive a

prediction about what type of document is likely to be accessed next instead of the actual document.

- *Time Awareness*. A time-unaware prediction process assumes a logical timescale where each unit of time corresponds to the occurrence of an execution event. In contrast to this, a time-aware prediction also takes the timing of the anticipated requests into account. This means that the prediction process does not only produce a probability for a predicted event, but a probability distribution over time (for example, stating that the event will be initiated within the next 2 minutes with 40% probability and within 3 minutes with 60% probability). This information can be provided to the application process as additional information e.g., to drive a cost based decision. A time aware prediction process implies that the patterns must contain information about the typical timing of events. Our prediction algorithm presented in Section 4.3 uses a discrete time model. An extension of the algorithm towards a time aware model is discussed in chapter 7.

The prediction algorithm together with the choice of an accurate interaction pattern model are the key parameters for the performance of a prediction framework. Being a heuristic approach, its suitability has to be evaluated empirically by comparing the performance (for example, accuracy of the prediction) of the framework for different kinds of workloads. For our OLAP specific instance of the PROMISE framework, this evaluation is performed using a trace-driven simulation environment which is described in greater detail in Chapter 6, where we also present and interpret the simulation results. In order to study the influence of different parameters (e.g., the size of the look-ahead window, the composition of the workload), we deploy artificially generated interaction traces for the simulation.

2.6 The Induction Process – From Interactions to Patterns

The pattern induction process generates the patterns which are essential to the prediction process. Its input is a list of sessions (composite interaction events) that were executed by different users or user groups in the past. The sessions are described according to an Interaction Model *IM*. The process produces a set of patterns which are described according to the interaction pattern model PM_{IM} (which has to be defined in advance). In other words, the patterns are instances of the pattern schemata defined in the pattern model and are formulated using the pattern representation language of PM_{IM}. This ensures that the produced patterns can be directly used as input to the *prediction process*.

Formally, the pattern induction process is a mapping I from a sequence of composite interaction events formulated according to an interaction model *IM* (which are in turn sequences of atomic interaction events) to a set of patterns conforming to a interaction pattern model PM_{IM}.

$$I_{PM_{IM}} : \left(E^{C}_{IM}\right)^{*} \rightarrow P_{PM}$$

When designing a system type specific instance of the pattern induction process (for example, for OLAP systems), the following issues have to be discussed to formulate requirements regarding the algorithms.

- *Degree of human interaction*. The induction can be carried out automatically or semi-automatically with the involvement of an administrator or domain expert who checks the plausibility and validity of the patterns. A goal of the PROMISE framework is to keep the additional effort for administrating the predictive information system as low as possible. Furthermore, in our framework, the fully interactive pattern deduction process already facilitates the integration of expert knowledge into the overall process. Therefore, an impor-

tant requirement for the induction process is that it is able to operate without human intervention.

- *Incremental vs. Full update techniques.* Some induction algorithms need all the interactions perceived so far in order to update the model (*full update*). In contrast to this, *incremental induction algorithms* updates the patterns based only on a small set of recent interactions. This enables the algorithm to work online, that means that the pattern information can be adjusted online every time a user interaction occurs. Considering the large amount of user interactions that can occur and the fact that maintenance time for information systems often is a critical factor, for PROMISE, we prefer incremental techniques that maintain the patterns online during the user interactions take place.
- *Changing user behavior.* The patterns in the user's behavior change over time. E.g., when the user is still inexperienced with the analytical capabilities of the system, other navigation paths occur. Therefore, the pattern induction mechanism must incorporate mechanisms to adjust the set of patterns to this changing behavior. This includes for example the ability to assign a higher weight to patterns that occurred in recent interactions compared to patterns present in interactions that occurred less recently.

After having discussed the basic requirements, we now take a look at the impact that these requirements have on the reusability of existing algorithms. The induction of different kinds of patterns from a set of given events is being extensively researched in the *Knowledge Discovery in Databases* area in the context of *Data Mining Algorithms*. The PROMISE pattern induction process constitutes a special case of KDD. Therefore, this section describes the special requirements and peculiarities of the PROMISE framework that have to be taken into account when adapting general data mining and pattern induction algorithms.

A multitude of data mining algorithms have been proposed, but to point out the special requirements of the PROMISE framework it is sufficient to look at the abstract structure of these algorithms. In general, the core of the pattern induction algorithm consists of two phases: parameter search and model search. The *parameter search* assumes a fixed schema for the patterns and finds instances for the parameters of the pattern schema such that the input data optimally fits the resulting expression. E.g., for the following pattern schema

$$income > X \Rightarrow profitable = \text{'yes'}$$

the parameter search phase finds values for X such that the input data is well described by the resulting rule. Of course, the result is dependent on the method that the algorithm uses to measure how good a given dataset fits a given pattern (called evaluation strategy in [FPS96b]). Depending on the definition of the evaluation strategy and the search space for the parameters, a solution for the parameters can either be obtained in closed form or by a heuristic search algorithm (for example, greedy algorithms). A second phase (called *model search*) of the abstract general induction algorithms varies the pattern schemata itself. As the set of pattern schemata is infinitely large for most of the interesting pattern models, this variation is usually performed in a heuristic way. After varying the pattern schemata, a new parameter search is started.

As already pointed out in the previous sections, for the PROMISE framework we assume that a fixed (system type specific, e.g., OLAP specific) set of pattern schemata is defined at the design time of the system. These predefined schemata correspond exactly to the set of pattern schemata which can be processed by the prediction algorithm. Therefore, for our framework a pattern induction process does not contain a model search phase but only a parameter search phase. Additionally, we target a solution, where the optimal parameterization for a pattern schema regarding a set of past interactions can be obtained in closed form such that no

heuristic search algorithm is needed. This greatly simplifies the structure of the induction process and enables us to fulfill the requirements of an automatic and incremental online algorithm. Section 4.3 presents an incremental weighted relative frequency counting technique. Possibilities to relax the assumption that pattern schemata are known in advance are discussed in Section 7.3.3.2.

2.7 The Modeling Process – Deduction of Patterns

Apart from inducting patterns from user interactions observed in the past, it is also feasible to model anticipated user behavior in advance. This is typically done by domain experts who know the structure of the domain-specific analysis process (for example, the methodology used to plan distribution strategies for spare parts). That means that in contrast to the pattern induction process, the modeling process cannot be fully automated. Therefore, we cannot define an algorithm for this process. Instead a methodology for this modeling process and its integration into a conceptual information system modeling methodology has to be discussed.

It is not feasible to demand that the domain expert specifies the analysis process using a mathematical model (for example, using a rule based mechanism). Instead, an intuitive graphical notation must be provided for the domain expert to specify the dynamic work characteristics. Therefore, the PROMISE framework needs a modeling formalism that is intuitively understandable by the experts and can easily be transformed into a representation that can be interpreted by the prediction algorithm. This means, it is necessary to define a graphical representation of the pattern model and to integrate this graphical model into a design methodology for the targeted type of information systems (e.g., a hypertext design method like HMT [SZ00], or a data warehouse design methodology, e.g. [BSH00], [HH99], [GR99]).

Another positive aspect – besides enabling prediction – of systematically modeling the user's behavior during early phases of the system design process is that these models document the user's requirements in a natural way. Our experience from specifying several real world data warehouses with our project partners show that end users can usually describe the structure of their work very well (for example, as a detailed workflow), but most often have difficulties defining and understanding static data models that mirror the static nature of their business. Therefore, instead of first designing the static part of the system (the data model for the application, e.g., the link structure of a Web site or the multidimensional schema of an OLAP application), it is possible to design the dynamic model of the information system (i.e., the typical user interaction) and later derive the static structures that are necessary to enable the dynamic usage. In other words, the specification of characteristic navigation patterns can be used to drive the design process resulting in a design that mirrors the user requirements in a more adequate way than the result of a process centered around the static system specification. This corresponds to the idea of a use-case driven application design ([Jac92]).

The FORWISS BabelFish Design Methodology allows for specifying, generating and maintaining OLAP applications based on graphical views of an extensible meta-model ([BSH00]). The easy extensibility of the meta model makes BabelFish ideally suited for coupling with the PROMISE/OLAP framework (see Section 7.3.1 and [BSH00]). The extensible meta-model of BabelFish describes all the objects that are necessary to specify and maintain an OLAP solution. Views of this meta-model are presented to the designers in a graphical way. Through these views, the designers interactively manipulate the specification of the OLAP application. [BSH00] describes this mechanism in a more detailed way and an integration of the PROMISE approach with the BabelFish design methodology.

2.8 Application of Prediction Results – Materializing the Benefits

As already motivated in chapter 1, the results of the prediction process can be deployed in a multitude of ways to build adaptable navigational systems. For this thesis, we focus on the implications of a prediction algorithm for the dynamic caching behavior of a navigational information system.

In general, caching techniques rely on the assumption that the data access workload (which is indirectly being generated by the interactive user in our environment) shows simple repetitive patterns. Therefore, results of previous data accesses (or query processing in a database centered environment) can be stored in a relatively small cache for future references. What kinds of objects are managed by the cache is strongly dependent on the semantic level on which caching is performed in the information system architecture. E.g., a low-level cache which resides close the persistent data storage typically manages objects with little semantic meaning (for example, pages), while caches which are located closer to the end user (with respect to the layered system architecture) typically manage semantically richer objects (for example, query results). These caches are usually referred to as semantic caches ([DFJ+96]). An integral part of the PROMISE idea is to use semantics of the application domain (for example, mirrored in the data model and typical navigation patterns) for the decisions of the cache manager. Consequently, we assume a semantic cache that manages objects on a granularity similar to the access granularity of the user (for example, a query level cache). All of the proposed OLAP specific caches (see Section 5.1) fulfill this property as do e.g., all the WWW proxy caches.

The improved performance of a system with an integrated cache compared to a system without caching is mainly a consequence of the fact that cache memory is cheaper to access than the original data (mostly because it resides on a higher level in the storage hierarchy than the original data, e.g., in the server's main memory instead of the server's secondary storage). In systems where the result of a user interaction does not only have to be retrieved but also has to be computed first (like in any database system), the calculation time can be additionally shortened by (partly) caching computed results.

The performance of the caching algorithm critically relies on the ability of the cache manager to keep the set of cached data objects (cache content) in such a way that the content enables optimal cost savings regarding data access costs. As cache space is limited, for every new data object, the cache manager has to decide if it should be placed in the cache (*admission algorithm*) and if so, which other object(s) should be removed from the cache in case that there is not enough room for the new object (*eviction algorithm*). This decision is driven by an estimation of the future benefits of a cached object (for example, the cost savings that can be achieved using this object to answer future data accesses) compared to the cost for maintaining the object in the cache (for example, the space which the object occupies).

The idea of the PROMISE approach is to provide the cache manager with the capability to use prediction information which is provided by the prediction process in order to improve the cache benefit. A caching algorithm can principally make use of prediction information for the following two tasks:

- enhancing the design of admission/eviction algorithms and
- enabling speculative execution (prefetching) strategies.

The following two sections highlight the most important issues of those two application areas which will be discussed in depth in Chapter 5.

2.8.1 Predictions Improving Cache Management

When deciding, if a data object (the content of an object is dependent on the granularity of the caching algorithm) should be stored in the cache, normally a cost/benefit function is being evaluated which leverages the incurred costs (cache storage space of the object) and the benefits (potential to speed up future data accesses). If the cost/benefit function is above a certain threshold, the object is admitted to the cache. When the cache reaches a certain size, new objects can only be admitted to the cache if existing objects are replaced. The task of the eviction algorithm is to identify the objects in the cache that are least likely to be beneficial for reacting to future interactions. Both, the admission and the eviction algorithm need functions to estimate the benefits of a cached object for future data accesses.

The admission and eviction algorithms use assumptions about the locality of future data accesses to decide which previously referenced items should be kept in the cache. Different forms of locality can be distinguished (actual algorithms can combine different types of locality measures):

- *Locality of the access regarding the current cache content*. I.e., the cache manager assumes that objects currently being in the cache because they were accessed recently will be accessed again in the near future. This kind of locality assumption is classically used for database and processor caches as it optimally models loop like structures (for example, repeated execution of similar queries) of programs (a prominent example is the Least Recently Used strategy).
- *Locality regarding the data space*. If the application domain of the data contains a natural definition of locality (as for example spatio-temporal data), this locality can be used to drive the admission and eviction decisions. This means e.g., that objects which are located 'near' the last object being accessed are less likely to be evicted.
- *Semantic locality*. If it is possible define a semantic distance between objects (for example, web pages with similar content), this measure can also be used to drive caching decisions. The locality used by the PROMISE approach can be subsumed under this category as objects which are often accessed consecutively during a session can be defined to have a large locality.

It is obvious that both the eviction and the admission algorithms can benefit from a prediction algorithm that can enumerate likely future interactions (interactions that are semantically near to the previous interaction) as both need to calculate the potential benefit for future interactions. Let us assume that the prediction algorithm computes the set of the n most likely future interaction events $e_1,...,e_n$ together with their respective probabilities $p_1,..., p_n$. The benefit of a cached object o for the processing of interaction event e (for example, a query in an OLAP system) is given by the formula $b_e(o)$. The actual formula depends on the granularity of the caching strategy. The overall potential benefit B(o) of a cached object o can then be computed as

$$B(o) = \sum_{1 \le i \le n} b_{e_i}(o) \cdot p_i$$

Of course, the benefit function b(o) is strongly dependent on the type of the information system and the caching strategy. A discussion of OLAP system specific predictive admission and eviction caching algorithms is included in Chapter 5.

2.8.2 Improving Cache Performance by Enabling Speculative Execution

Prefetching or speculative execution[5] strategies can complement the demand fetching strategies of classical caching algorithms. Classical caching strategies only consider objects that have been produced by past queries. But under certain circumstances, it is possible to reduce the latency time perceived by the user if the cache manager can initiate prefetching (respectively speculative execution) requests. In such an environment, the cache manager makes use of the prediction results to speculatively fetch objects into the cache (during idle times) that are likely to be beneficial for future data accesses (e.g precomputing the most likely multidimensional query). Figure 2.4 shows the communication between the user and the information system and illustrates the effect of an improved response time.

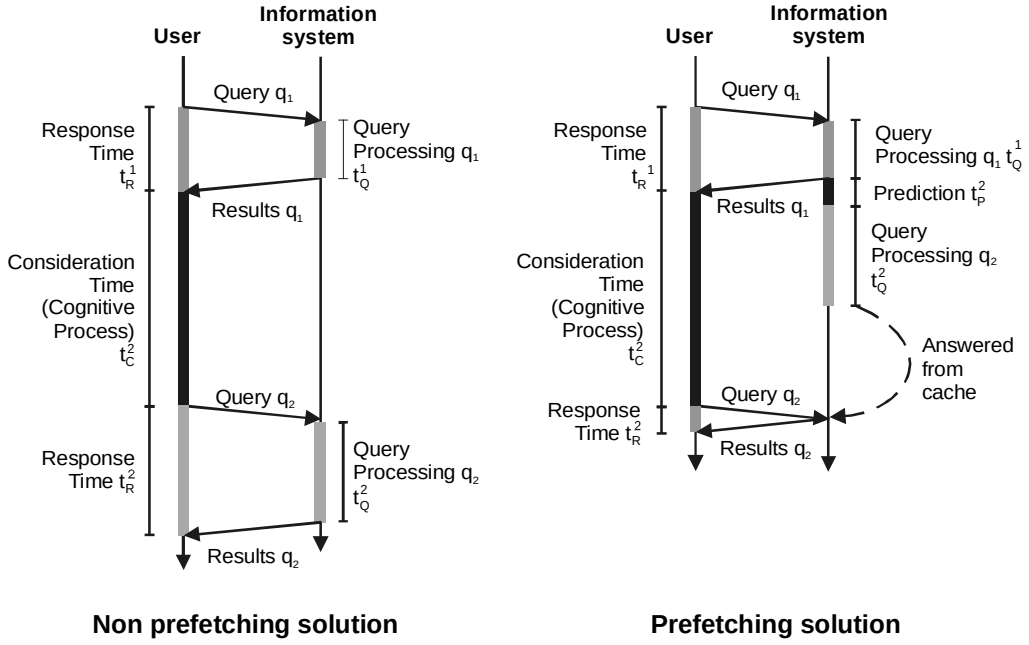

Figure 2.4: The Principle of Reducing Response Times by Predictive Prefetching and Caching

A session starts with the user executing an interaction that leads to a query[6] q_1 against the informaton system. While the query is being transmitted to the information system, executed, and the results are sent back to the user, the user waits (this is the latency time perceived by the user). Afterwards, a cognitive process takes place when the user evaluates the result of the last query and builds a new hypothesis. For our illustration, we assumed that the system is idle during this *consideration time* (after which the user executes query q_2)[7]. If q_2 can be predicted using information about the session so far (for example, query q_1) and information about typical patterns, the query can be executed during the cognitive process. If the prediction was correct, the next query can be directly answered from the cache, reducing the response time perceived by the user. If the user executes the query while the speculative execution of the query

[5] We use the terms 'prefetching' and 'speculative execution' synonymously throughout this thesis. For systems that retrieve data in the form it is stored (for example WWW servers serving static pages), we prefer the term prefetching and for systems that perform more or less complex computations with the raw data before presenting it to the user, we use the term speculative execution.

[6] The term query is used here in a broad meaning also subsuming e.g. a simple HTTP request against a WWW server.

[7] This may sound like an oversimplification, as of course the system might be busy executing queries from other users during this time. Albeit, the empirical analysis in section 6.1 shows that this assumption is not unrealistic in real life environments. As long as the system has some idle times, the approach of prefetching can be beneficial.

is still in process (that means $t_P^2 < t_C^2 < t_P^2 + t_Q^2$), the user's latency time is still shorter compared to the non-prefetching solution by $t_C^2 - t_P^2$ as the query is already running when the user executes it. If the consideration time is even shorter than the prediction time $t_C^2 < t_P^2$ no speed-up is achieved. However, our analysis of real world user behavior (see Section 6.1) shows that for a percentage of over 80% of the queries the true prefetching assumption (i.e., $t_C^2 >> t_Q^2$) is fulfilled (cf. also [Sap00]).

Usually, a system does not operate in a single user mode (as assumed in our illustration) but has concurrent accesses of different users. Obviously in such an environment, it has to be ensured that speculative queries are executed in a way that they do not disturb regular queries that are executed by other users. This can be done by assigning lower priorities to speculative queries than to regular queries.

In order to achieve a reduction of the latency time, the result of the prediction has to be correct, that means the actual interaction must be equivalent to the predicted interaction[8]. Ideally, the design of the predictive prefetching system should ensure that the latency time is not increased, if case the prediction is wrong. In order to illustrate this problem, let us consider that the last prediction was wrong i.e., the actual interaction event (with its associated query q_2) is not equivalent to the predicted interaction event (with the associated query $\bar{q}_2$). If the execution of the speculative query $\bar{q}_2$ is already finished ($t_C^2 > t_P^2 + t_Q^2$) the latency time for query q_2 is not affected at all. In this case, the system has to decide if the speculative query results should be kept in the cache. Figure 2.5 illustrates the extended query execution process of the predictive information system.

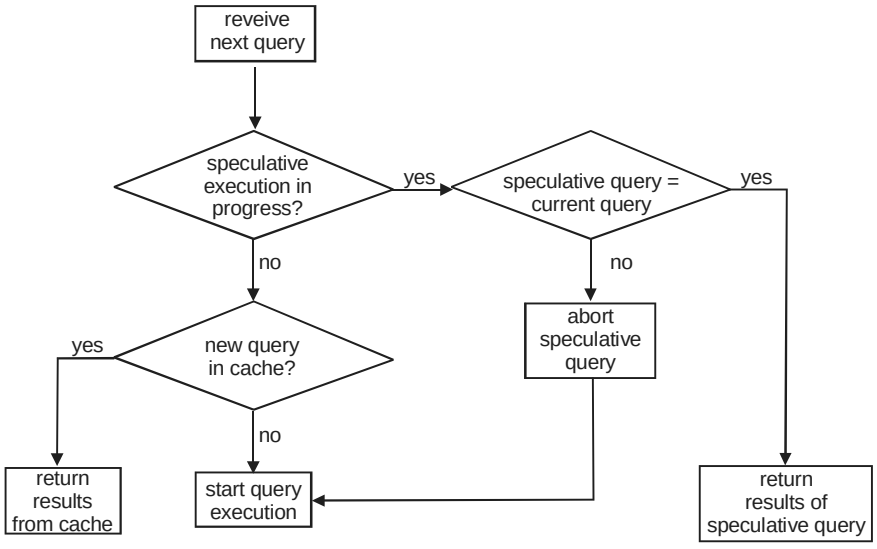

Figure 2.5: Control Flow for the Extended Query Processing Including Speculative Execution

The most critical case is that the speculative data access for query q_2 is still in progress when the next query arrives. In this case, the execution has to be aborted before the execution of the query q_2 can begin. Depending on the system type and the nature of the query, this might incur additional costs which in turn increases the latency time of the user. This effect should be

[8] In systems, where the prefetched objects are not disjoint (for example in a database system, where a query can be computed from the results of a different query), a speed-up is also possible, if parts of the predicted query can be used in the computation of the actual query.

avoided as far as possible on the information system side, by exploiting mechanisms to decrease the cost of aborting an operation and on the prediction side by avoiding the speculative execution of unlikely queries. The avoidance of unnecessary queries is also sensible, because wrong speculative executions also increase the usage of system resources (e.g., network bandwidth, processor and I/O time, cache space).

Summarizing, a speculatively executed query incurs costs (execution time, space for result storage, usage of system resources) and can result in certain benefits (shorter latency time) if the prediction is successful. Therefore, it is essential for the performance of the overall system, to decide whether a predicted interaction event is to be speculatively executed or not based on a comparison of potential costs incurred by a speculative execution (both in execution time and storage costs of the result) and the potential savings by speculatively executing the interaction. The following considerations about the design of such a cost/benefit model should make clear that such a model is far from being trivial due to the large number of influencing parameters:

- *finding an appropriate measure for benefit.* The evaluation of the potential benefit of a query result for future queries is a difficult problem, as this requires assumptions about the nature of the future queries. Another problem complicating the design of a benefit measure is that a precomputed query result cannot be only used to answer the query itself, but also queries that (partly) contain the precomputed query (see Section 3.2.3 for a more detailed discussion of the derivability problem). Furthermore, the benefit is also dependent on the execution time of the query compared to the estimated consideration time of the user. This measure is in turn requires an accurate cost estimation.

- *finding an appropriate cost model.* This includes the estimation of the computational costs of a query using a cost model for the query execution. It is also important to note that a speculative execution also incurs storage costs for the intermediate result. This is especially important, as cache space is usually limited and other cached results have to be evicted from the cache in order to make room for the execution results. Therefore, the benefits of the evicted results also have to be accounted for in the cost function.

Summarizing, the influencing parameters for a prefetching decision model include:

- the current cache content
- a cost/benefit estimation model
- the current load of the system resources

Figure 2.6 summarizes the coarse structure of the prefetching process and its connection to the prediction process. The prediction process provides potential queries and predicts their probability. A cost/benefit estimation algorithm estimates the costs and the benefits of the queries taking into account the cache content, system load, caching parameters etc. The results are used by a decision process to decide according to the prefetching strategy if a prefetching request is being initiated.

The cost/benefit function used by the estimation process is specific for the type of information system and the caching algorithm deployed. In Chapter 5, we discuss appropriate cost/benefit estimations for an OLAP specific caching algorithm and their implications.

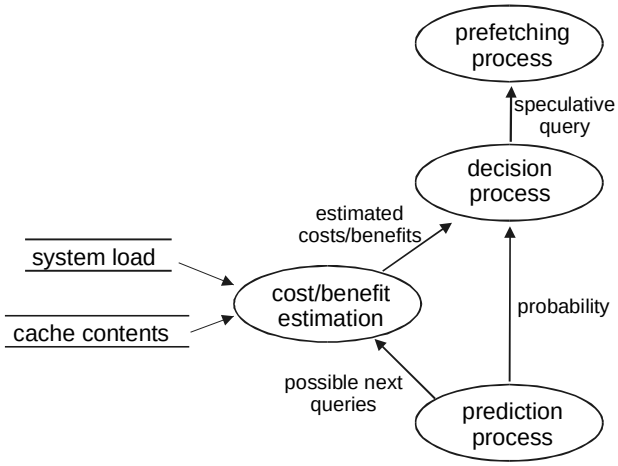

Figure 2.6: A Schematic Overview of the Predictive Prefetching Process

2.9 Summary and Conclusions

This chapter refined the specification of this thesis's topic by describing a comprehensive framework of communicating processes which are necessary to integrate prediction capabilities into a generic navigational information system. It additionally presented a (semi-)formal functional definition of the processes and discussed the most important issues that have to be taken into account when designing a system type specific instance of the framework.

One of the objectives was to design a framework which is generic in the sense that it only assumes an information system with navigational access capabilities. In particular, it should be totally independent of OLAP systems and can therefore be applied as a reference model to compare approaches from different application areas and to discuss the applicability of different techniques (for example, Data Mining algorithms). This objective is mirrored in the definition of an abstract interaction model and an associated abstract pattern model. Another objective was to support a wide range of applications for the prediction results. This was achieved by containing abstract generalization functions in the definition of the pattern interaction model. These functions hide the details of the application specific functions which have to be provided when defining a instance of the framework. The possibility to reuse existing techniques was the last objective in the framework's design. This requirement was addressed by strictly modularizing the framework, such that e.g., existing pattern induction algorithms can be easily inserted into the framework.

The rest of this thesis presents PROMISE/OLAP, an OLAP specific instance of the PROMISE framework. Chapter 3 first discusses a selection of the most important issues in OLAP system design and implementation, containing the basis for the definition and discussion throughout the core of this thesis (Chapter 4 to Chapter 6).

To instantiate the PROMISE framework for OLAP systems (cf. Figure 2.7), it is first necessary to define a user interaction model (see Definition 2.1). The development of this OLAP specific interaction model is the topic of Chapter 3. As the interaction model describes the interactive operations a user can perform on data that adhere to a multidimensional schema, it is based on the formal description of a multidimensional schema (Section 3.1) and the corresponding description of the query formalism (Section 3.2).

The next step is the definition of an Interaction Pattern Model (see Definition 2.5), which defines a formal pattern representation and a set of pattern schemata and generalization func-

tions for events (Chapter 4). The OLAP specific pattern model is developed in Section 4.2. It discusses the representation of patterns in OLAP user behavior as Markov-Models. It also presents different generalization mechanisms for OLAP queries. This model is the basis for the discussion of an OLAP specific prediction algorithms in Section 4.3 and a corresponding pattern induction algorithm in Section 4.4.

In order to materialize the benefits of the prediction, it has to be integrated e.g., into OLAP caching algorithms. This idea is the topic of Chapter 5. The prediction results can be used to improve the admission and eviction strategies and enable introducing active caching strategies facilitating speculative execution of OLAP queries.

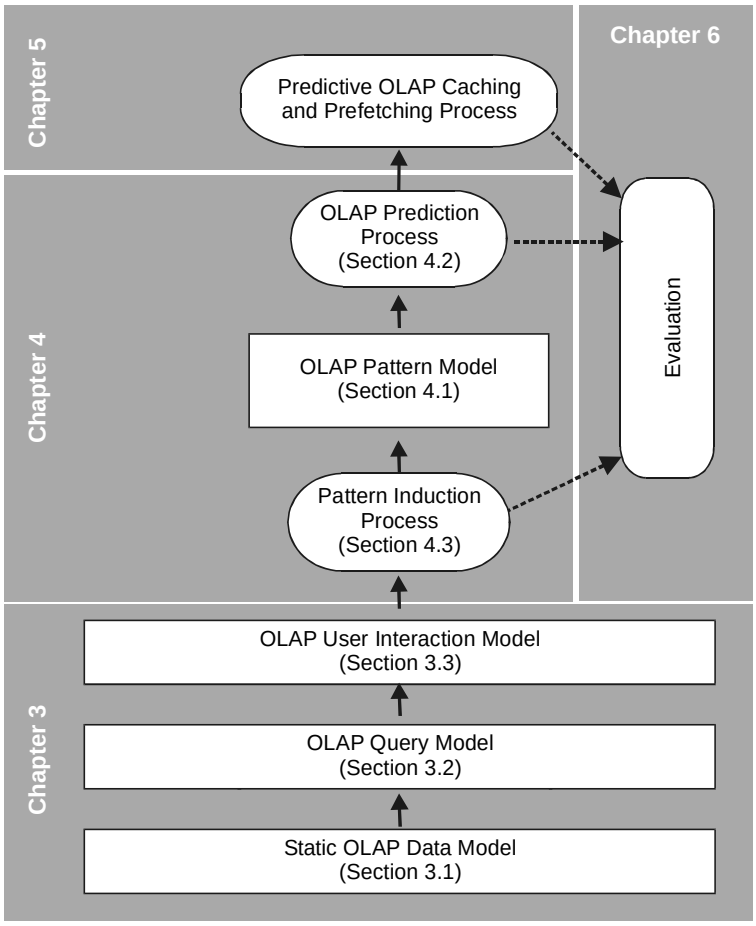

Figure 2.7: Detailed Overview of the Different Models and Processes of the PROMISE Approach

Of course it is important to evaluate the performance (e.g., predictive accuracy, improvements of caching strategies) of the OLAP framework. Being a heuristic approach, we empirically evaluate the feasibility of the framework in Chapter 6. There, we present an empirical analysis of real world user behavior and analyze the performance of our algorithms (using different parameter setups) by means of a user simulation.

The above definition of the PROMISE framework contained several explicitly mentioned assumptions and restrictions which were necessary design decisions to ensure a manageable complexity of the solution presented in this thesis. An extension of the framework by relaxing some of the constraints is the topic of Chapter 7.

«Man's mind stretched to a new idea
never goes back to its original dimension.»
-- Oliver Wendell Holmes.

Chapter 3 Modeling and Querying OLAP Databases

The general prediction framework presented in the previous chapter is a solid conceptual and formal basis to develop instantiations that are specific for certain types of navigational information systems. The rest of this thesis will be devoted to the development and application of such an instantiation that is specific to Online Analytical Processing (short: OLAP) environments.

The first step in the development of OLAP systems was taken by the industry (e.g., Kenan IRI[9]). Nevertheless, soon after the first OLAP products began to be commercially successful in combination with data warehouse solutions (around 1995), the research community has begun to adopt this field and discuss the main issues from a scientific point of view. From a database system engineer's point of view, the most important distinctive feature of an OLAP system compared to other database systems (for example, relational database systems) is that it supports the multidimensional data model on a logical level. Consequently, all the design decisions discussed in the context of database management systems (e.g., physical data storage and access structures, query processing and optimization techniques, caching techniques) should be re-evaluated regarding the special semantics of the multidimensional data model. As a comprehensive overview of this active field of research is certainly beyond the scope of this thesis, this chapter especially focuses on the topics which are of direct importance for defining the OLAP specific instance of the PROMISE framework.

For our work, a formal definition of the multidimensional data model is fundamental, as this in turn is the basis for formalizing queries. Therefore, we define our understanding of the multidimensional concepts based on a discussion of the main differences between existing approaches in Section 3.1. The result is the PROMISE/OLAP MD-data model which serves as a formal foundation throughout this thesis. Section 3.2 deals with the question of how to formulate queries in the presence of a multidimensional data model. A survey of scientific approaches (Section 3.2.1) and a comparison with real-world OLAP interfaces (Section 3.2.2) provides the motivation for defining the comprehensive PROMISE/OLAP query model in Sections 3.2.3 and 3.2.4. The most important distinctive features of the PROMISE query model are that it takes into account the typical OLAP user interface and that it contains means

[9] This product is now owned by Oracle Corp. and marketed under the name Oracle Express Server.

to model iterative query formulation. Therefore, this model constitutes an important contribution of this thesis.

3.1 The Multidimensional OLAP Data Model

The central characteristic of an OLAP system is the support of the multidimensional data model on a logical level. The OLEDB for OLAP standard ([Mic98]) that is currently emerging as the de-facto standard for commercial systems suffers from the lack of formal foundation (like the OLAP council proposal [OLA98]). Neither the semantics of the data model nor the query language (MDX) are formally described. Albeit, a formal definition of the concepts that form the data model is essential to our work, as the user formulates queries with respect to this data model and terminology. Thus, a formal interaction model (see Chapter 4) must be based on a formal understanding of the multidimensional data model. At the time of writing, the discussion how such a model should look like is still in progress in the scientific community. This is indicated by the large number of proposals about this topic that have been published during the last two years (see 3.1.1.5 for a comprehensive overview). Our paper [SBH99] which has been frequently cited certainly played a role in sparking and structuring this discussion.

In this section, we identify the most important design decisions of the formalization process and discuss how existing approaches address these issues. We conclude by defining the PROMISE/OLAP data model by adopting the concepts from different approaches which are best suited for our approach.

When using the term multidimensional data model in the context of data analysis applications (OLAP, scientific and statistical databases etc.), typically two important orthogonal concepts are subsumed:

- *Organization of base data according to a multidimensional space:* The subject of analysis (called fact) is typically an event type (for example, a vehicle repair). Each occurrence of the event is being interpreted as a point in a multidimensional space. The coordinates of the point are defined by discrete attributes characterizing the event (for example, the time of the repair).
- *Classifications:* Classifications (for example, in a taxonomy) are used by domain experts to structure the application domain. The multidimensional data model contains information about these classification schemes. This allows for an elegant definition of operations which leads to an intuitive query formalism that can be understood by the domain experts.

Therefore, in our opinion the multidimensional data model is not very accurately named and should be better referred to as the multidimensional data model with classifications ([SBH99]). This is mirrored in the structure of our data model and in the discussion of the structure of this section. The description of the PROMISE/OLAP data model consists of two parts: the definition of classifications (Section 3.1.2) and the definition of the multidimensional space (Section 3.1.3). Section 3.1.1 discusses the objective and alternatives of the data model design. A relational implementation of the data model is the subject of Section 3.1.4.

3.1.1 Considerations about the Design

This section systematically compiles the most important design decisions of the data model design process and classifies the existing approaches according to these decisions. Additionally, we discuss the implications of the different approaches, choose an alternative for PROMISE/OLAP and argue why this alternative is best suited.

3.1.1.1 General Considerations

The basic objectives of logical data model design are not specific to the multidimensional data model. This includes the proper separation of schema and instances for all elements of the data model (data cubes and classifications) and the independence of data model description from physical implementation.

A fundamental decision when designing a formal data model is whether to understand the multidimensional data model as autonomous from the ubiquitous relational model. If this view is adopted, it means that the data model introduces cubes, dimensions and classifications as the main entities ([VS99] calls this approach *cube oriented*) and describes it using common mathematical concepts. This approach is followed by the vast majority of OLAP data models (e.g., [CT98a], [AGS97], [Vas98], [Leh98a], [DT97]). Of course, this does not mean that the concepts are far from the relational model. Actually, most of these approaches also present a relational mapping of the model for implementation purposes. In contrast to this, the relationally oriented approaches (e.g., [LW96], [GL97]) define the multidimensional model by specializing and re-interpreting selected elements of the relational model (for example, interpreting a foreign key relationship as a dimension of the cube).

Conclusion: We choose a cube oriented approach as in our opinion, it is better suited to describe the user's perception of the data model on a logical level. This approach allows us to represent the peculiarities of the multidimensional model without re-interpreting common relational concepts.

3.1.1.2 Describing the multidimensional space

The multidimensional organization of data is very close to the inherent structure of many problem domains. Events concerning the subject of analysis (for example, a vehicle repair) are interpreted as points in a multidimensional space. The event is being characterized by a set of attributes (called *event attibutes*, e.g., time of the repair, duration of the repair). A subset of these event attributes are used to determine the coordinate of the corresponding point, e.g., the time of the repair. As these attributes span the multidimensional space, they are called *dimensions*[10]. In practical applications, dimensions possess discrete values, this means the multidimensional space is partitioned into so-called cells[11]. These cells contain values for the remaining event attributes (called measures or measure attributes, e.g., number of repairs). This idea is often visualized using a cube metaphor (Figure 3.1).

Thus, the multidimensional model clearly contains a separation of the event attributes into coordinates (*dimensions*, also called qualifying data) and cell content (*measures*, also called quantifying data). From the application's point of view, these two types of event attributes represent different semantics which is mirrored in the type of operations that are executed on the different attribute types. Restrictions on dimension attributes are typically used to restrict which cells are contained in the query result, while measures are used to compute aggregated values. Albeit, this separation is not necessarily obvious nor static as measures can be converted to dimensions (increasing the number of dimensions) and vice versa.

Thus, the interesting issue of the data model design is how to reflect the separation of measures and dimensions and the dualism of the concepts. Most of the proposed data models (see 3.1.1.5) follow the approach first introduced in [AGS97]. The definition of the cube schema contains a standard partitioning of the attributes into measures and dimensions. To

[10] Some approaches call these attributes *dimension attributes*.
[11] However, the approach can also be applied to continuos dimension domains.

reflect the dualism between measures and dimensions, the corresponding algebra contains conversion operations that restructure the cube by converting measures to dimensions and vice versa (for example, called fold/unfold in [AGS97]). An alternative approach is taken by strongly relationally oriented approaches (e.g., [GL97], [MRB99]): here, no separation between measures and dimensions is defined, the partitioning is determined for each query instead (cf. [Mar99]). Another interesting approach can be found in [PJ99], who do not model the concept of measures but only allow dimensions thus modeling all measures as dimensions.

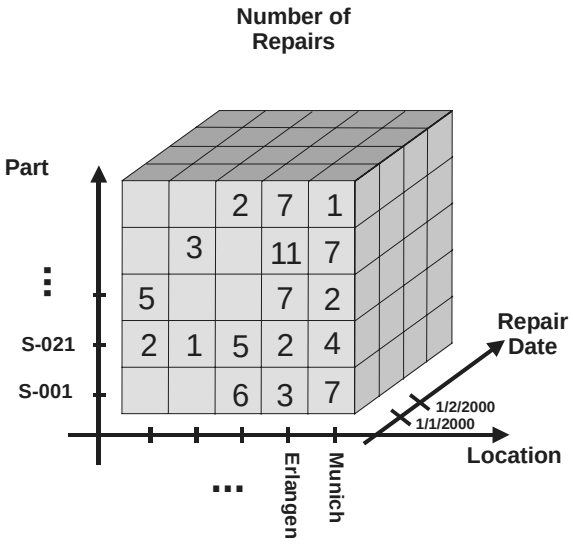

Figure 3.1: Visualizing the Multidimensional Data Space Using the Cube Metaphor

Conclusion: The user's different semantic perception of measures and dimensions is important for the PROMISE/OLAP approach as this difference is being reflected by the query behavior of the user. For most cases a standard partitioning of event attributes into dimensions (qualifying data) and measures (quantifying data) can be given at the system's design time. Therefore, we include the information which attributes are mostly treated as measures and which are mostly deployed as dimensions in the schema of the database.

3.1.1.3 Representing classifications

Independent from any IT support, domain experts structure their application domain using classifications (e.g., grouping single products to product groups or classifying birds according to families). This is a proven mechanism to reduce the complexity of the domain and to model domain knowledge. It has been recognized from the beginning [CCS93] that classifications should be incorporated into the OLAP data model due to their central importance to the analyst. The user navigates along the classifications when analyzing the data. The system then automatically computes the data according to the classification level used in the user's query.[12]

So far, no consensus has been reached in the scientific literature how to incorporate the important concept of classification hierarchies into the formal data model. Early approaches ([AGS97], [Bau99], [BPT97]) do not explicitly incorporate classifications in their data model. Instead, [AGS97] provides a combined grouping and aggregation operation in the multidimensional cube algebra. This function expects a grouping function (i.e., the classification of dimension elements) as a parameter. That function assigns each element of the corresponding dimension (for example, 'Munich') of the data cube to a parent value (for example, 'Bava-

[12] The strategy whether this computation is being done at query time or in advance (preaggregation) is a physical tuning parameter of the OLAP database system and should not be part of the data model definition.

ria'). The set of all parent values forms the domain for the dimension of the result cube. Another proposal in this direction is contained in [BPT97] where classifications are modeled as functional dependencies between relational attributes.

It has been widely recognized ([SBH99], [PJ99], [VS99], [JLV+99]) that this 'second class citizen'-approach – while being very flexible – is not fully suitable for OLAP databases. The main reason is that the classifications are normally already known at schema design time (in contrast to query time) and thus should be part of the multidimensional schema. This captures additional application semantics which is beneficial for a multitude of purposes (e.g., automatic generation of front-end tools, adaptation of clustering strategies ([MRB99], [JLS99a]), making query processing more efficient [ABD+99]). Therefore, more recent approaches incorporate classifications as 'first class citizens' into their static data model ([CT97], [DKP+99], [DT97], [Leh98a], [PJ99], [Vas98]).

Following the basic considerations about the data model design, each concept of the data model should have schema (intension) and instances (extension). This means that classifications should be described on two levels of abstraction: the classification schema (containing the classification levels) and the classification instance (containing the actual grouping). Surprisingly enough, this issue is only explicitly addressed by a small number of existing data models, namely [Alb01] and [Leh98a].

The next design issue deals with the expressiveness of a classification. A popular concept is a classification hierarchy, where objects are structured using a tree. Each level of the tree corresponds to a classification criterion (called *classification level*). Nevertheless, often a single hierarchy is not enough to describe a domain. Instead different alternative classification hierarchies exist, therefore a more expressive classification model allowing alternative classifications with shared classification criteria is introduced by [CT97] and has later been adopted by [Vas98], [DKP+99] and [PJ99].

What remains is the issue of how to describe the relationship between a classification and a dimension. The large majority of the approaches (e.g., [CT98a], [Vas98], [Leh98a], [DKP+99]) model the classification schema as part of the dimension schema. I.e., the 'first-class' element is a dimension which contains the classification as a part of the dimension definition. Some of those approaches include the classification instance into the data model, some do not. The view of a dimension as a structured concept (while being useful for OLAP systems) somewhat clashes with the mathematical understanding of the term 'dimension' as a flat, ordered list of elements (mostly an interval of discrete numeric values). Therefore, an alternative way of incorporating the classification hierarchies into a multidimensional model is to define the classifications independently from the term dimension ([MRB99]).

Conclusion: For our PROMISE/OLAP approach, it is beneficial to represent classifications in the database schema as the classifications carry a lot of application semantics that can be used in order to predict user behavior. Therefore, we introduce an explicit construct for classifications in the data model. We support multiple hierarchies with shared levels in order to allow for expressing complex classification relationships. Additionally, we explicitly distinguish between the classification schema describing the classification criteria with their relationships (classification intension) and the classification instance (classification extension). As we want to preserve the mathematical meaning of a dimension as a flat collection of values, we introduce the concept of classifications independently from the concept of a dimension and link the two parts of the data model by requiring that a dimension is also a classification level (cf. 3.1.3).

3.1.1.4 Extended concepts

The multidimensional OLAP data model is a compromise between expressiveness and simplicity. On the one hand, the data structures and operations should be restricted enough such that a graphical front-end can be used to intuitively formulate queries without profound knowledge of the data model. On the other hand, the static model must be expressive enough to model a large part of the application domain and the operations should be powerful enough to formulate different sorts of queries. Of course this gives room to discussing different extensions of the data model.

In [LRT96] the authors show that description and classification attributes constitute two orthogonal concepts and that it therefore is not sufficient to represent descriptive attributes together with classification levels (as being done in all the other approaches). Consequently, [Leh98a] introduces a feature extended model, where each node of the classification possesses a distinct set of descriptive attributes that can be used to formulate restrictions. E.g., a *video device* is described by the feature '*video system*' while a *washing machine* is characterized by the feature '*tumbling frequency'* although both nodes belong to the classification level *product group*.

[DKP+99] proposes an extension of the cell structures. All of the other data models assume that the cells of the OLAP data cube only contain atomic data types or record structures built from these types. In [DKP+99], the authors describe a data model that incorporates nested data cubes, that means that each cell of the cube can itself be a cube structure (analogously to the concept of nested relations). While being valuable for design and query purposes, this concept does not increase the expressiveness of the model as every nested data cube can be mapped to an equivalent non-nested cube with more dimensions than the nested cube.

The authors of [PJ99] study the requirements of a medical data warehouse project and suggest the following extensions to the multidimensional data model:

- *non strict hierarchies*, allowing objects to be classified by more than one parent breaking up the tree structure of the hierarchy. Although similar, this concept is different from the concept of multiple hierarchies with shared levels, where alternative paths exist on the schema level.
- *many to many relationships* between fact and dimension, allowing that a fact (for example, patient treatment) may be associated with more than one element of a dimension (for example, several diagnoses).
- *different granularity for base data.* The characterization of a fact (for example, *patient treatment*) may not always be recorded on the base granularity level (for example, *explicit diagnosis*), but on a higher level (for example, *diagnosis group*) for some occurrences of a fact.

While these extensions are useful in modeling the semantics of real world applications (most of these restrictions were also encountered during our industrial projects), it remains unclear how the extended expressiveness can be incorporated into an OLAP front-end. Summarizability is a very useful property for OLAP models. [LS97] discusses the conditions that have to be fulfilled in order to ensure summarizability. Non strict hierarchies, many to many relationships and different base granularities violate these conditions as they may lead to double counting. In [PJ99] it remains open how these problems can be solved.

Conclusion: We consciously do not incorporate these extended concepts into our PROMISE/OLAP data model as they unnecessarily complicate the data model, introducing problems which are irrelevant to the focus of our work.

3.1.1.5 Overview

Approach	Relational/ Cube Oriented	Schema/ Instances (Data Space)	Classifications	Multiple hierarchies (shared levels)	Schema/ Instances (Classifications)	Classification/ Dimension	Qualifying/ Quantifying Data
Albrecht [Alb01]	Cube	Yes	first-class	Yes (No)	Yes (Yes)	combined	static
Agrawal Gupta, Sarawagi [AGS97]	Cube	No	second-class	--	--	--	static
Baumann [Bau99]	Cube	No	second-class	--	--	--	static
Blaschka [Bla00]	Cube	Yes	first-class	Yes (Yes)	Yes	combined	static
Cabibbo/Torlone [CT97], [CT98a]	Cube	Yes	first-class	Yes (Yes)	implicit	combined	static
Datta/ Thomas [DT97]	Cube	No	first-class	Yes (No)	implicit	combined	static
Gyssens/ Lakshmanan [GL97]	Relational	Yes	first-class	Yes (No)	implicit	--	static
Lehner [Leh98a]	Cube	Yes	first-class	No	explicit	combined	static
Li/Wang [LW96]	Relational	Yes	first-class	Yes (No)	implicit	--	dynamic
Markl, Ramsak, Bayer [MRB99]	Relational	Yes	first-class	No	implicit	separated	dynamic
Pedersen/ Jensen [PJ99]	Relational	Yes	first-class	Yes (Yes)	implicit	combined	only dimensions
Vassiliadis [Vas98]	Cube	No	first-class	Yes (Yes)	implicit	combined	static
PROMISE/OLAP	Cube	Yes	first-class	Yes (Yes)	explicit	separated	static

Table 3.1: A comparison of existing OLAP data models with the PROMISE/OLAP data model

Table 3.1 summarizes the comparison of the most important data models with the PROMISE/OLAP data models with respect to the criteria discussed in the previous sections. It lists the following criteria:

- The column *Relational/Cube oriented* states whether the model relies on the relational model or not (see 3.1.1.1).
- The column *Schema/Instances (Data Space)* lists if the approach explicitly distinguishes between schema and instances when describing the data space (see 3.1.1.1).
- The column *Classifications* indicates the treatment of classifications by the respective approach, distinguishing first-class and second-class strategies (see 3.1.1.3).
- The column *Multiple Hierarchies (Shared Levels)* investigates if an approach supports multiple hierarchies and if these multiple hierarchies can share levels. (see 3.1.1.3).
- The column *Schema/Instances (Classifications)* indicates if the approach explicitly distinguishes between schema and instances when describing the classifications (see 3.1.1.3).
- The column *Classification/Dimension* indicates the relationship between a dimension and the classification (see 3.1.1.3). The value '*combined*' shows that the definition of a classification is incorporated into the dimension definition thus violating the mathematical notion of a dimension.

- The column *Quantifying/Qualifying data* states how the approach handles the partitioning of attributes into dimensions and measure attributes. The value *'static'* means that a distinction is fixed at schema design time. While *'dynamic'* indicates that the distinction is being made at query time (see 3.1.1.2).

3.1.2 Modeling Classifications

Having motivated the design decisions, the following two sections develop the formal definition of the PROMISE/OLAP data model. According to the two aspects of the model (multidimensional data space and classifications), we first formalize classification structures in this section and then define the multidimensional structures in the next section (3.1.3).

The multidimensional OLAP data model is primarily targeted towards statistical data analysis. As already motivated, the concept of classifications plays an important role for this application area. Basically, a classification on a set of objects defines a grouping of objects with the same characteristics (building equivalency classes). The grouping is done according to a classification criterion (for example, the geographic location of cities). A complex classification typically contains different levels of classification (grouping cities by region, regions by country and countries by continents). Thus, a *classification level* represents a classification criterion.

Definition 3.1 (Classification Level, Classification Node)

A classification level l is a finite set of objects from a domain $dom(l)$. An object $x \in l$ is called *classification node* of level l. ◆

Each node x of a classification level is additionally described by a set of descriptive attributes. Classical multidimensional approaches require that all nodes of a classification level possess the same description attributes. [LRT96] argues that this in not expressive enough for some real world applications and consequently [Leh98a] introduces an extended multidimensional data model which offers a special 'feature' element to overcome this restriction. Nevertheless, for the scope of this thesis, we assume that the set of descriptive attributes is the same for all nodes of a classification level. Therefore, we denote the set of descriptive attributes of a level l as $A(l)$. Furthermore, we assume that each node implicitly possesses a unique object identifier (OID or label) which is used to refer to the node instead of using the descriptive attributes for reference purposes.

Example 3.1 (Classification Level)

In our material management example (see Section 1.1.2), the analyst wants to analyze data according to geographic regions. Therefore, we introduce a classification level *geogr. region* with the domain *dom*(*geogr. region*) = {Bavaria, 'USA East', 'USA West',.....}. Another classification level may be the part of a vehicle. Such a classification level *part* could include the descriptive attributes $A(part)$={part#, price, weight, description}. ◆

These classification levels are used to build the more complex *classification schema*. A classification schema represents the information, which classification levels are related and which are not. This information is modeled as a classification relationship between the levels of the classification schema. A classification relationship between *level a* and *level b* has the semantic meaning that the criterion being represented by *level b* (for example, geographic region) constitutes a classification of the criterion corresponding to *level a* (for example, city).

Thus, a classification schema is a graph structure with levels as nodes and classification relationships as edges. Albeit, some structural restrictions to the general graph structure have to be fulfilled in order to preserve the semantics of a classification:

- The classification relationship must be reflexive as each level can be (trivially) classified according to itself.
- The classification relationship is transitive as the concatenation of two classification groupings (for example, *city* by *geogr. region* and *geogr. region* by *country*) defines a new classification (‘*city*’ by ‘*country*’).
- The classification relationship is antisymmetric as a level *a* cannot be a classification for *b* if *b* is a classification for *a*.

These considerations are summarized by the following formal definition:

Definition 3.2 (Classification Schema)

A classification schema Ψ is defined as tuple $\Psi = (L_\Psi, class_\Psi)$ with

- $L_\Psi = \{l_1,\ldots l_k\}$ is a set containing the *k* distinct classification levels of the classification.
- $class_\Psi \subseteq L_\Psi \times L_\Psi$ defines the classification relationship between different levels. The relation *class* must fulfill the following properties:

$$(l,l) \in class_\Psi \quad \forall l \in L_\Psi \tag{1}$$

$$(l_1,l_2) \in class_\Psi \wedge (l_2,l_3) \in class_\Psi \Rightarrow (l_1,l_3) \in class_\Psi \quad \forall l_1,l_2,l_3 \in L_\Psi \tag{2}$$

$$(l_1,l_2) \in class_\Psi \Rightarrow (l_2,l_1) \notin class_\Psi \quad \forall l_1,l_2 \in L_\Psi \tag{3}$$

◆

For notational convenience, we introduce the function $L(\Psi)$ that maps a classification schema to the set of levels contained in the classification schema i.e., $L(\Psi) := L_\Psi$.

It is obvious that the relation $class_\Psi$ defines a partial order on the set of classification levels L_Ψ as it is reflexive (1), transitive (2) and antisymmetric (3). Therefore, instead of $(l_1,l_2) \in class_\Psi$ we use the notational abbreviation $l_1 \leq_\Psi l_2$ (reading: “l_1 can be classified according to l_2.”). Also as convenient shorthand, we introduce the relations $<_\Psi$ and $>_\Psi$ as follows[13]:

$$(l_1 >_\Psi l_2) :\Leftrightarrow (l_2 \leq_\Psi l_1) \wedge l_1 \neq l_2$$

$$(l_1 <_\Psi l_2) :\Leftrightarrow (l_1 \leq_\Psi l_2) \wedge l_1 \neq l_2$$

For specifying (and visualizing) a classification schema, it is sufficient to specify the base relation $class_\Psi^B$ of $class_\Psi$. The base relation is the minimal relation $class_\Psi^B$ that produces *class* when the reflexive transitive closure operation is applied. I.e.,

$$(class_\Psi^B)^* = class_\Psi \ \wedge \ class_\Psi^B \subseteq x \quad \forall x \text{ with } \left(x \subseteq L_\Psi \times L_\Psi\right) \ \wedge \ \left(x^* = class_\Psi\right)$$

As a classification schema is a special case of a graph structure with classification levels as nodes and classification relationships as edges, it can be easily visualized. Figure 3.2 shows an extract of the classification schema for our running scenario. It uses a rectangle to represent each level and edges to depict the classification relationship. With a view to greater clarity, we omit the reflexive and transitive edges thus only visualizing the base relation $class_\Psi^B$.

Example 3.2 (Classification Schema)

[13] Notably, the greater than (>) relationship is not the complement of the less or equal than (≤) relationship, because ≤ is not total.

The sample classification schema $\Psi_{Ex} = (L_{\Psi_{Ex}}, class_{\Psi_{E_x}})$ depicted in Figure 3.2 looks as follows:

$L_{\Psi_{Ex}}$ = { *customer, geogr. region, country, location, climatic region, type of repair unit, part, part group, assembly, day, month, week, year*}

$class^{B}_{\Psi_{Ex}}$ = { (*customer, geogr. region*), (*geogr. region, country*), (*location, geogr. region*), (*location, climatic region*), (*location, type of repair unit*), (*day, month*), (*day, week*), (*month, year*), (*week, year*), (*part, part group*), (*part, assembly*) } ◆

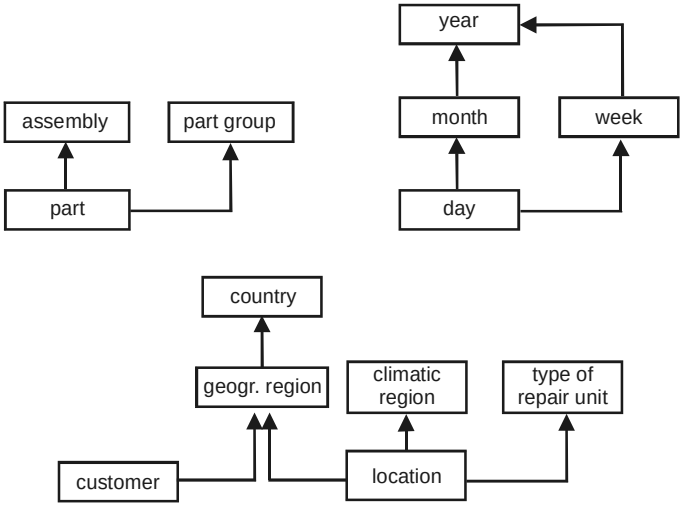

Figure 3.2: A Sample Classification Schema

Notably, as we do not demand that $class_{\Psi}$ has to be a total order, we allow the classification schema not to be fully connected (see example) and to contain alternative paths. These paths are deployed by the user for interactive navigation purposes and for the system to dynamically aggregate data to the level being requested.

Definition 3.3 (Classification Schema Path)

Assuming a classification schema $\Psi = (L_{\Psi}, class_{\Psi})$, each sequence $P = l_1, \ldots, l_z$ of levels with $l_i \in L_{\Psi}; 1 \le i \le z$ is called a *classification schema path* (or short classification path) if and only if the following condition is fulfilled: $(l_i, l_{i+1}) \in class^{B}_{\Psi} \quad \forall 1 \le i < z$. ◆

Example 3.3 (Classification Schema Path)

The following sequences are valid classification paths for our above example (cf. Figure 3.2):

P_1 = day, month, year

P_2 = location, geogr. region, country

P_3 = geogr. region, country

while the following sequence is not a valid classification path:

P_4 = location, month ◆

Each classification schema path *P* defines a total order on the levels in *P* thus defining a special case of a classification schema: *a classification hierarchy*. A useful property of a classification hierarchy is that its instance (see below) has the form of a balanced tree[14]. The left hand side of Figure 3.3 shows a sample for such a classification hierarchy schema. Every classification schema defines a set of classification hierarchies (all valid classification paths).

Sometimes, it is convenient not to refer to the domain *dom*(*l*) of a single classification level *l*, but to the union of the domains of the levels of a whole classification schema. Therefore, for notational convenience, we extend the definition of the *dom*-function to a classification schema.

Definition 3.4 (Domain of a Classification schema)

For a classification schema $\Psi = (L_\Psi, class_\Psi)$, we define the domain *dom*(Ψ) as follows: $dom(\Psi) = \bigcup_{l \in L_\Psi} dom(l)$. ◆

Example 3.4 (Domain of a Classification schema)

The domain of the example schema $dom(\Psi_{Ex})$ contains the elements of all levels in $L_{\Psi_{Ex}}$ i.e., $dom(\Psi_{Ex})$={'1/1/2000', '1/2/2000',…, 'Jan 2000', 'Feb 2000',…, '2000',…,'Munich' ,…)

◆

The classification schema defines constraints for the classification instance. The instance of a classification level is defined by the set of nodes of that level. In order to describe the instance of the classification relationship between two levels *a* and *b*, we use a grouping function $group_{a,b}$ that maps each element of level *a* to the parent element in level *b*.

Definition 3.5 (Classification Instance)

The instances I_Ψ of a classification schema $\Psi = (L_\Psi, class_\Psi)$ are described by a set of base grouping functions:

$$I_\Psi = \left\{ \text{group}_{l_1,l_2} : dom(l_1) \rightarrow dom(l_2) \,\middle|\, \forall l_1, l_2 \in L_\Psi \wedge (l_1, l_2) \in class_\Psi^B \right\}$$

◆

Example 3.5: (Classification Instance)

A possible instance for the running example contains a grouping function mapping locations to geographic regions, as a direct classification relationship exists between those levels (cf. Figure 3.2). This function $\text{group}_{\text{location, geogr. region}}$ maps each single location to the parent region. Some example mappings include:

$\text{group}_{\text{location, geogr. region}}$('Munich') = 'Bavaria'

$\text{group}_{\text{location, geogr. region}}$('Frankfurt') = 'Hessen' ◆

A visualization of the principle of the grouping function can be seen in Figure 3.3. For the special case of classification hierarchies, the instances defined by the grouping functions form the structure of a balanced tree (see Figure 3.3, right hand side).

[14] this does not constitute a restriction for real world applications, as unbalanced hierarchies can be balanced by introducing additional virtual classification nodes.

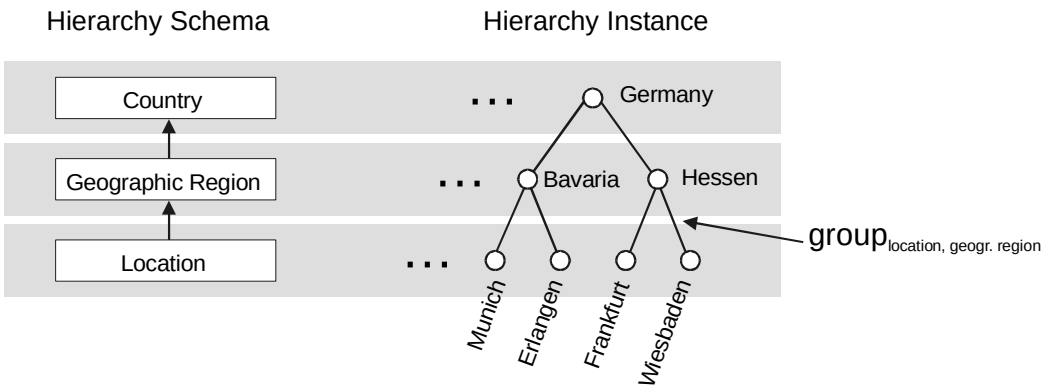

Figure 3.3: Schema and Instance of a Classification Hierarchy Defined by a Classification Schema Path

A more general grouping function $g_{s,t}$ for arbitrary levels $s,t \in L_\Psi$ can be derived from the set of base grouping functions I_Ψ if a dimension schema path $s,l_1,\ldots,l_z,t$ exists that starts at level *s* and ends at level *t* (that means if s can be classified according to *t* or formally if $s \leq_\Psi t$). The function $\text{group}_{s,t}$ is then computed as the concatenation of the base functions along the classification path: $\text{group}_{s,t} = \text{group}_{s,l_1} \circ \text{group}_{l_1,l_2} \circ \ldots \circ \text{group}_{l_{z-1},l_z} \circ \text{group}_{l_z,t}$.

Example 3.6 (Grouping Function)

For example the grouping function for classifying locations according to countries is computed by the concatenation of the grouping from location to geographic region and from geographic region to country. Thus,

$$\text{group}_{location,\ country} = \text{group}_{location,\ geogr.\ region} \circ \text{group}_{geogr.\ region,\ country}$$

◆

In order to ensure the summarizability property ([LS97]), we demand that the following condition must be fulfilled by the instance: If multiple paths p_1,p_2 from level *s* to level *t* exist in a classification schema, the grouping functions must be defined such that the grouping function group_{p_1} using the levels of path p_1 and the grouping function group_{p_2} using the levels of path p_2 produce the same results for all nodes of *s*.

The concept of a classification schema that was introduced so far is independent of the data actually stored in a multidimensional database. It constitutes an instrument to model semantic relationships in the analyst's application domain (material management in our example). Nevertheless, the actual data in an OLAP database is stored on a certain base level of granularity. This level is defined by the granularity with which the events of interest (for example, repairs) are recorded (for example, daily). It is not possible to classify the data on this base level according to levels which cannot be reached by a classification path. For example, data cannot be analyzed on a level that is smaller than the base granularity. If repairs are recorded on the level of geographic regions, it is impossible to analyze repairs according to the location as the information which location performed the repair is not available.

To reflect these considerations in our data model, we introduce a specialization of a classification schema which is called *classification lattice* (we will later show that this structure really possesses the properties of a lattice). It is defined by choosing a base level (the level on which the data is being recorded, e.g., location). The classification lattice contains the subgraph that is defined by the base level (cf. Figure 3.4). The intention behind this definition is to identify all the levels that can be used to classify data on the base level. Additionally, we

add a special classification level <*base_level*>.*all* (for example, geogr. region.all) to the classification lattice. This special element is larger (according to $\leq_\Psi$) than any other level. It contains only one node labeled *'all <base_level>'* (for example, all geogr. regions). The purpose of this level is to enable the grouping of all nodes of the base level to one single node.

The following definition formally defines the construction of the classification lattice for using an arbitrary classification level from a given classification schema as its base level.

Definition 3.6 (Classification Lattice)

For a classification schema $\Psi = (L_\Psi, class_\Psi)$ and a level $l \in L_\Psi$, we define the classification lattice $\Psi|_l = (L'_\Psi, class'_\Psi)$ as follows (l is called base level of the lattice).

- $L'_\Psi = \{a \mid a \in L_\Psi \wedge l \leq_\psi a\} \cup \{'< l > .all'\}$
- $class'_\Psi \subseteq L'_\Psi \times L'_\Psi$ with: $class'_\Psi = (class_\Psi \cap (L'_\Psi \times L'_\Psi)) \cup \{(a, '< l > .all') \mid a \in L'_\Psi\}$ ◆

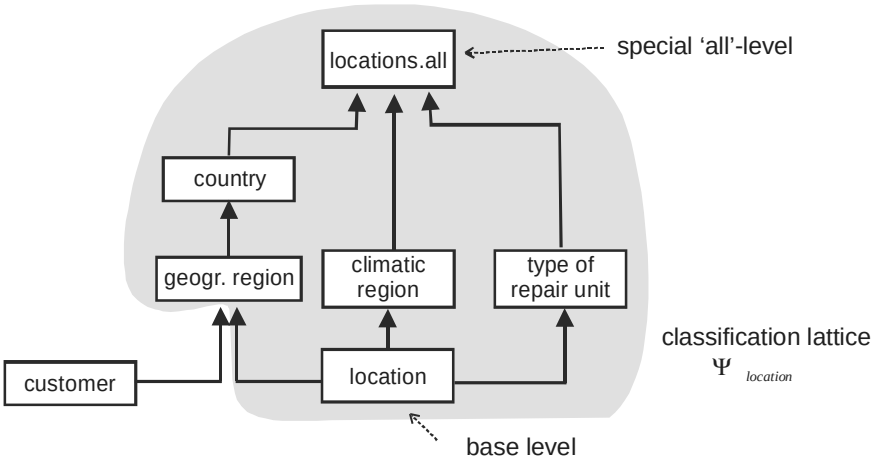

Figure 3.4: A Classification Lattice Constructed from the Sample Classification Schema Shown in Figure 3.2

Example 3.7 (Classification Lattice)

For the classification schema $\Psi = (L_\Psi, class_\Psi)$ defined in the above example (see Figure 3.2), the base level *'location'* defines the following classification lattice: $\Psi|_l = (L'_\Psi, class'_\Psi)$. A visualization of the lattice is shown in Figure 3.4 shaded in gray.

- L'_Ψ = { location, geogr. region, climatic region, type of repair unit, country, all locations }
- $class'^{B}_\Psi$ = { (location, geogr. region), (location, climatic region),
  (location, type of repair unit), (geogr. region, country),
  (country, all locations), (climatic region, all locations),
  (type of repair unit, all locations) } ◆

So far, we have defined the classification lattice and shown how to construct it from a classification schema but not yet proven that the structure really possesses the formal properties of a lattice.

Theorem 3.1 (Validity of Classification Lattice)

Every classification lattice $\Psi|_{base} = (L'_\Psi, class'_\Psi)$ defined on the classification schema $\Psi = (L_\Psi, class_\Psi)$ has the following properties:

- $(l,l) \in class'_\Psi \quad \forall l \in L'_\Psi$ (1)
- $(l_1,l_2) \in class'_\Psi \wedge (l_2,l_3) \in class'_\Psi \Rightarrow (l_1,l_3) \in class'_\Psi \quad \forall l_1,l_2,l_3 \in L'_\Psi$ (2)
- $(l_1,l_2) \in class'_\Psi \Rightarrow (l_2,l_1) \notin class'_\Psi \quad \forall l_1,l_2 \in L'_\Psi$ with $l_1 \neq l_2$ (3)
- Two levels $l_1 \in L'_\Psi$ and $l_2 \in L'_\Psi$ have a least upper bound (LUB) (4)
- Two levels $l_1 \in L'_\Psi$ and $l_2 \in L'_\Psi$ have a greatest lower bound (GLB) (5)

◆

Proof 3.1 (Validity of Classification Lattice)

The full formal proof is omitted here and can be found in Appendix A. Basically, the proof for properties (1)-(3) consists of two parts. First we prove that the conditions hold for all levels $l \in L'_\Psi \setminus \{'l.all'\} = L'_\Psi \cap L_\Psi$ because the properties are fulfilled by the original classification schema Ψ. The second part of the proof shows that the construction of the lattice ensures that the properties are preserved by adding the special level *l.all*.

The proof for properties (4) and (5) makes use of the fact that by definition *l.all* is an upper bound as all other levels are smaller and that the base level *l* defines a lower bound as is by definition smaller than all other levels. As the set of levels is finite, this implies the existence of a LUB, respectively a GLB. ◆

Conditions (1)-(3) of Theorem 3.1 are just the conditions demanded for a classification schema showing that a classification lattice is a special case of a classification schema. Therefore, the following corrolar allows us to transfer all the definitions (schema path, instances, domain) for classification schemata to classification lattices.

Corrolar 3.2 (Classification Lattice is a Special Case of a Classification Schema)

Every classification lattice is a classification schema. ◆

Proof 3.2 (Classification Lattice is a Special Case of a Classification Schema)

Because of Theorem 3.1, the conditions (1),(2) and (3) are fulfilled for a classification lattice. Therefore, it is a valid classification schema according to Definition 3.2. ◆

One of the noteworthy consequences from the above corrolar is that we can also use the definition of instances (Definition 3.5) for a classification lattice. Notably, all grouping functions $g_{s,l.all}$ from an arbitrary level *s* in the lattice to the special level *l.all* produce the value "all" for every node of *s*. Thus, $g_{s,l.all}(x)=all$ for all $x \in s$.

3.1.3 Putting Things Together: The Multidimensional Database Schema

A multidimensional cube is the logical unit of data storage in an OLAP system (corresponding to a relation in a relational database). A cube corresponds to the subject of analysis – called *fact* (for example, a repair of a vehicle). According to the separation of qualifying and quantifying data, the schema of an n-dimensional cube consists of two components:

- a set of *n* dimensions that span the multidimensional space (the edges according to the cube metaphor) and
- a set of measures that describe the event of interest (the structure of the content of the cube cells according to the cube metaphor).

Therefore, a multidimensional cube schema is defined as follows:

Definition 3.7 (Multidimensional Cube Schema)

An n-dimensional cube schema C conforming to a classification schema $\Psi = (L_\Psi, class_\Psi)$ is defined as a tuple $C_\Psi = (D_\Psi, M)$, where

- $D_\Psi = (d_1, \ldots, d_n) \in (L_\Psi)^n$ is a list of classification levels[15] called the dimensions of C. The nodes of d_i are called the *dimension members* of d_i.
- $M = (m_1, \ldots, m_k)$ is a set of names called the measure names. Each m_i has a domain $dom(m_i)$ attached to it. ◆

For notational convenience, the set of dimensions and measures for a cube schema are ordered. Without restricting the expressiveness of our approach, we assume that the dimensions are ordered in an arbitrary but unique way (for example, using the lexicographic order of the dimension names). We only use the position of a dimension or measure for identification purposes. Nevertheless, in order to make our examples more readable, we additionally assign a name to each dimension by which it is referenced in the text (instead of using the ordinal number). This name is put in front of the dimension, when defining the dimension, like *repair time* in the following example:

Example 3.8 (Multidimensional Cube Schema)

Two example cubes conforming to the classification schema Ψ_{Ex} defined in the previous section are:

C_{Rep}= ((repair time: day, repair location: location, repaired part: part),
{ # of repairs, # of persons })

C_{Ser}= ((service time: month, service location: location),
{duration}) ◆

Each dimension *d* has a dual function. On the one hand, a dimension spans the multidimensional space that is used to organize the data points and on the other hand, each dimension corresponds to exactly one classification level in the classification schema. Therefore, each dimension d defines a classification lattice $\Psi|_d$. This classification lattice contains all the levels that can be used to classify the corresponding dimension *d* during the analysis. It also defines all possible navigation paths that the user can follow during an interactive analysis.

Notably, we explicitly allow two or more dimensions to share the same classification path i.e., $d_i \leq d_j$ for $i \neq j$ resp. the same classification level. This is semantically meaningful, as different dimensions may share the same domain. E.g. one dimension modeling the order entrance (level day) and the other modeling the order delivery (also level day).

Analogously to the definition of instances for the classification, we also give a definition of a cube instance as a function that maps the cube coordinates to measure tuples.

Definition 3.8 (Instance of a Cube)

The instance of a Cube with schema $C_\Psi = (\{d_1, \ldots, d_n\}, \{m_1, \ldots, m_k\})$ is defined by a function I_C with $I_C : dom(d_1) \times \ldots \times dom(d_n) \rightarrow (dom(m_1) \times \ldots \times dom(m_k))$.

The set of all possible cube instances (i.e., different mapping functions) for a given cube schema is denoted as Ξ_C. ◆

[15] For a set S, we use the notation S^n to denote n-tuples build from elements of the set.

Notably, we do not demand the instance function to be total, that means that for some tuples $c \in dom(d_1) \times \ldots \times dom(d_n)$, the value of I_C is undefined[16] (for example, because a vehicle of a certain type was not repaired at a certain location using a certain part). Using the cube metaphor, this corresponds to an empty cell. If we look at a single dimension, it is interesting to note, which of the dimension members reference only empty cells (for example, a day on which repairs took place irrespective of the location and the part). We call the set of dimension members which at least reference one filled cell as the active domain of a dimension with respect to a cube instance.

Definition 3.9 (Active Dimension Domain)

For the instance of a Cube with schema $C_\Psi = (\{d_1, \ldots, d_n\}, M)$, the active domain **actdom**(d_i) of dimension d_i ($1 \le i \le n$) is defined as follows:

$$\text{actdom}(d_i) := \{x \in dom(d_i) \mid \exists c \in dom(d_1) \times \ldots \times \{x\} \times \ldots \times dom(d_n) : I_C(c) \neq \perp\}$$

◆

Example 3.9 (Instance of a Cube, Active Dimension Domain)

For the cube C_{Rep} defined in Example 3.8**,** the function I_C has the following signature:

$$I_C : dom(\text{day}) \times dom(\text{location}) \times dom(\text{part}) \rightarrow \big(dom(\#\text{ of repairs}) \times dom(\#\text{ of persons})\big) \cup \{\perp\}.$$

A sample value for the function I_C is:

I_C('1/5/2000', 'Munich', 'S-013') = (2, 5) indicating that in 'Munich' on the first of May 2000 2 repairs were performed exchanging part 'S-013'. 5 persons were involved in the repairs.

This means that 'Munich' is part of the active domain **actdom**($d_{\text{repair\_location}}$) of dimension *repair location.* ◆

Of course, a multidimensional database may contain more than one cube and several cubes may share dimensions or classification paths (like the cubes in our example). To this end, a multidimensional database schema contains a common classification schema and a finite set of cube schemata that conform to this classification schema. The classification schema defines the inter-cube relationships in the database. Thus, formally an MD database schema is defined as follows:

Definition 3.10 (MD database schema)

A multidimensional database schema Ω is defined as a tuple $\Omega = (\Psi, \chi)$ where

- $\Psi = \{L_\Psi, class_\Psi\}$ is a classification schema and
- $\chi = (C_1, \ldots, C_p)$ is finite set of cube schemata, each conforming to Ψ *i.e.*, the dimensions of C_i are taken from L_Ψ. ◆

This algebraic description of a multidimensional schema is useful for defining queries and query patterns but is not very helpful to visualize the schema. Therefore, graphical notations for the visualization and interactive design of multidimensional database schemata have been proposed. Throughout this thesis, we use a slight variation of the ME/R Notation ([SBH+99]) which has been co-developed by the author and which is the basis of the BabelFish design

[16] This phenomenon is referred to as sparsity of the data cube and is very important for the efficient storage of the data cube, as most practically occurring data cubes are extremely sparsely populated (under 1% of the possible cells actually contain a value).

framework ([BSH00]). Figure 3.5 shows the elements used by the ME/R notation to visualize the different concepts.

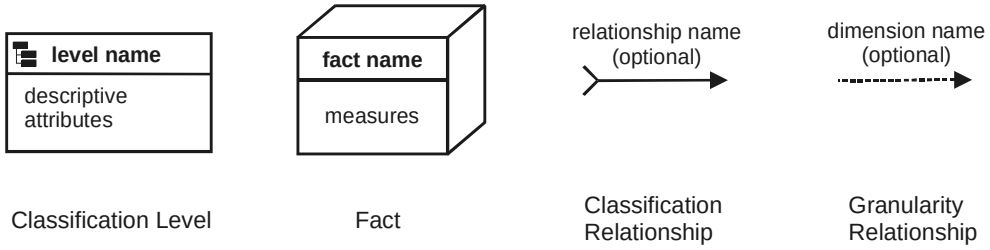

Figure 3.5: The Graphical Elements of the ME/R Notation

Example 3.10 (MD Database Schema)

A multidimensional database schema can be constructed from the sample classification (see Example 3.2) and the sample cubes defined in Example 3.8. The result is the MD database schema $\Omega_{Ex}=(\Psi_{Ex}, \{C_{rep}, C_{ser}\})$ which is visualized in Figure 3.6. ◆

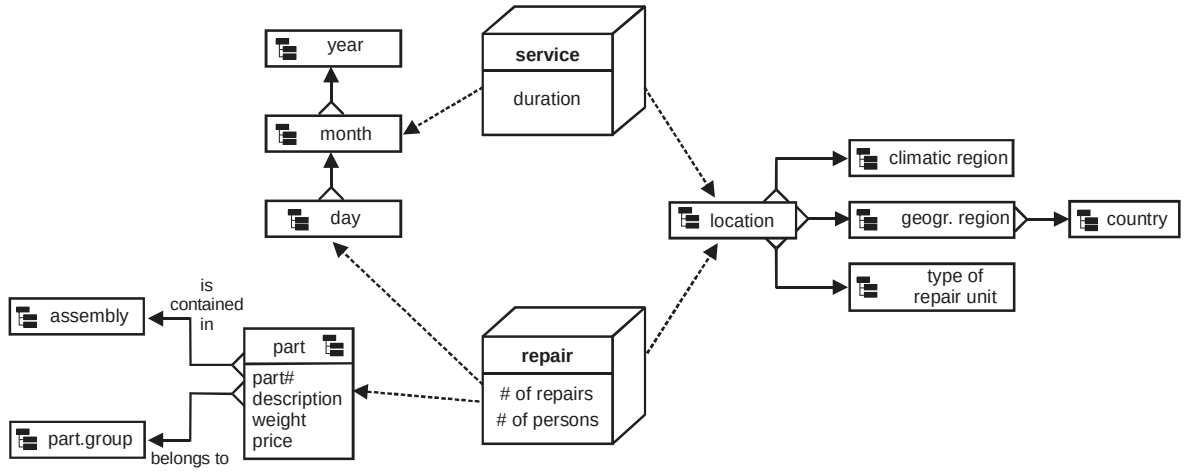

Figure 3.6: A Visualization of the Sample Multidimensional Database Schema (ME/R Notation)

3.1.4 A Relational View

Considering the vast amount of research effort, product and application development that has been spent on relational database systems, it is no surprise that many OLAP systems are being implemented as additional layers on top of relational database technology (the popular term for these systems is ROLAP systems). Therefore, it is necessary to define a relational view of the multidimensional data structures.[17] We will use this mapping throughout the thesis to clarify our concepts defined on a multidimensional level by showing the consequences for a possible relational implementation.

The basic concept of a multidimensional data cube can be mapped in a very natural way, using the dualism between a relation and the multidimensional space [Mar99]. By far more interesting is the mapping of classification schemata to a relational schema. The most widely used pragmatic approach to map multidimensional schemata is the *star schema approach*. It is mainly used for its structural simplicity. The base cube is being mapped to a fact table which contains the measures of the cube as attributes and an attribute for each dimension. The classi-

[17] Besides the mapping of the data models on a logical level, it is of course important to provide an efficient physical data organization for the relational view. Considerations about this can e.g. be found in [MRB99].

fication lattice for each dimension is being mapped to a table (a dimension table) which contains an attribute for each classification level contained in the lattice and the attributes of these levels. A foreign key constraint relates the fact table and the different dimension tables[18]. Naturally, this view violates the normalization constraints, as the attributes of the dimension table that correspond to classification levels being related by a classification relationship are by definition functionally dependent (semantics of the classification relation).

Definition 3.11 (Relational view of a Cube schema)

For the n-dimensional cube schema $C_\Psi = (D_\Psi = \{d_1,\ldots,d_n\}, M = \{m_1,\ldots,m_k\})$ conforming to the classification schema Ψ, the relational view is defined as a set of table schemata $R(C_\Psi)=\{dt_1,\ldots,dt_n, f\}$ where[19]

- $dt_i \subseteq \left(\prod_{l_p \in \left(L(\Psi|_{d_i}) - \{d_i.all\}\right)} \left(dom(l_p) \times \prod_{a \in A(l_p)} dom(a) \right) \right)$

  defines the schema for the *i*-th dimension table which contains the levels of the classification lattice $\Psi|_{d_i}$ (except the special attribute *all*) and their descriptive attributes.

- $f \subseteq dom(d_1)\times\ldots\times dom(d_n)\times dom(m_1)\times\ldots\times dom(m_k)$ defining the schema for the fact table which contains the measures as attributes and a foreign key to each of the dimension tables. ◆

Example 3.11 (Relational View of a Cube Schema)

For the cube C_{rep} defined in Example 3.8**,** the relational view $R(C_{rep})$ results in the following relational schema (the attribute names can be chosen arbitrarily, the underlined attributes build the primary key of the relation):

dt_1= time (day: *dom*(*day*), month: *dom*(*month*), week: *dom*(*week*), year: *dom*(*year*))

dt_2=repair_location (location: *dom*(*location*), geogr. region: *dom*(*geogr. region*),
climatic region: *dom*(*climatic region*), country: *dom*(country),
type of repair unit: *dom*(*type of repair unit*))

dt_3= part (part: *dom*(*part*), part#: *string*, weight: *number*, price: *currency*, description: *string*)

f= (repair date: *dom*(*time*), repair location: *dom*(*location*), repaired part: *dom*(*part*),
#repairs: *dom*(*#repairs*), #persons: *dom*(*#persons*))

The star schema is depicted in Figure 3.7 ◆

Of course, other relational views for the cube schema exist, e.g., the snowflake schema being a normalization of the star schema but we do not cover these in detail here as the star schema suffices to demonstrate the relational implementation of our concepts.

Being a view (i.e., a mapping), the star schema looses semantic information contained in the multidimensional model. To clarify this claim, we exemplarily discuss the representation of classification information. The classification relationships of the classification schemata are not explicitly contained in the star schema definition. As they are one to many relationships between the attributes of the corresponding levels (for example, between city and geogr. re-

[18] When actually implementing the relational view, surrogate keys are being used for the foreign key relationships for reasons of efficiency.

[19] The symbol Π is used to denote the cross-product of several factors.

gion), they are transformed to functional dependencies (FD) of attributes in the dimension tables. This is problematic because:

- The FDs cannot be derived from the relational schema nor is it sensible to induce the FDs from the data contained in the database. Therefore, this application semantic is lost unless recorded in further tables.[20]
- As functional dependencies are a more general concept than classification, other functional dependencies may exist that do not necessarily imply a classification relationship. E.g., between price and turnover.

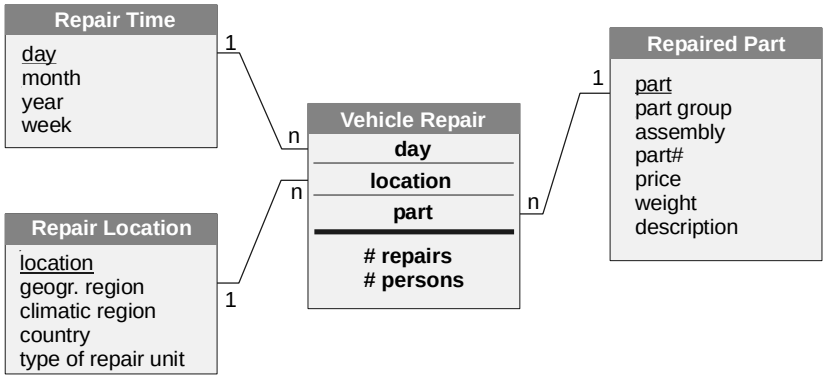

Figure 3.7: The Relational View (Starschema) of the Sample Cube C_{rep}

As a consequence, the mapping from a multidimensional schema to the star schema cannot be bijective. This is a further motivation to base our user model on the conceptual multidimensional level. Nevertheless, we use the relational view throughout the thesis to illustrate a possible realization of our concepts using a relational database system.

3.2 Querying OLAP Databases

The most important interactions of users in the context of the PROMISE/OLAP framework are those that produce a query that has to be computed by the OLAP database system. Such a query corresponds to an atomic interaction event described in Section 2.3. Therefore, a formal specification of a single OLAP query against a given multidimensional schema is needed for our approach. The design of such a formalism taking into account the current state of product development and research is the topic of this section.

The emerging industry standard for communication between OLAP front-end tools and OLAP database systems is the OLEDB for OLAP standard which has been introduced by Microsoft ([Mic98]). It includes the specification of a query language especially tailored to the multidimensional data model, called MDX (multidimensional expression language). Being an emerging industry standard with a large impact on the commercial OLAP community, MDX was of course considered as a basis for PROMISE/OLAP. However, it turned out to be unsuited for PROMISE/OLAP because of the following reasons:

- *lack of formalization.* MDX lacks a formalism (for example, an algebra) resolving ambiguities in the current 'prose' definition of the of the query language semantics.
- *weaknesses in the design*. Being a pragmatic approach, the MDX language mixes aspects of data manipulation with data presentation elements. In our opinion, these two aspects are

[20] Commercial OLAP products overcome this problem by keeping special tables as an extension of the system catalog, where the additional semantic information is stored.

orthogonal and therefore, the result presentation of the data (for example, which data is displayed as columns) should not be part of the query language.

- *no GUI.* Until now, no intuitive user interface exploiting anything near the expressiveness of MDX has been introduced.

Although not being suited as a basis for our approach, we recognize the great practical relevance of MDX. Therefore, we will make use of MDX to illustrate our concepts in examples wherever appropriate.

Most of the multidimensional data model specifications surveyed in the previous section also propose according query formalisms. Consequently, this section starts by providing a classification of these scientific approaches and discussing their suitability for PROMISE/OLAP (Section 3.2.1). As already pointed out, OLAP queries in real world environments are formulated using a graphical front end which naturally restricts the classes of queries that can be executed and has a large impact on the types of patterns (the pattern schemata) which occur in user sessions. As OLAP interfaces, their structure, and their impact on the class of formulated queries have not yet been systematically treated in the research community, we compare the interactive query interfaces of different leading OLAP products and derive an abstract description of an OLAP user interface (Section 3.2.2).

By presenting a definition of the class of queries considered for PROMISE/OLAP together with a canonical description formalism, Section 3.2.3 constitutes our synthesis of the theoretically oriented approaches and the practically oriented approaches. A discussion of the assumed restrictions and their impact on the expressiveness of the query model is the topic of Section 3.2.5 which also gives instructions on how to implement the canonical OLAP queries using SQL and MDX language.

Before looking at the approaches, the following list summarizes the most important requirements of the PROMISE/OLAP framework regarding the OLAP query formalism. It therefore defines the criteria we use for evaluating the suitability of techniques proposed by product vendors and the scientific community throughout this section.

- *Formal Semantics.* The semantics of the query specification must be formally described. Most importantly, the result of the query regarding a given schema and its extensions (instances) must be specified.
- *Descriptive Formalism.* Although having well defined operational semantics, the description of a query itself should not contain details about the operations which are necessary by the database system to compute the query. The aspect of efficiently evaluating a query is orthogonal to the goal of predicting the structure of the next query (see next item). Therefore, we are looking for a high level descriptive formalism.
- *Canonical Form.* The PROMISE/OLAP predictions are based on the assumption that queries with similar semantics occur at characteristic phases of the analysis process. The semantics of a query description are defined by the result it produces on an arbitrary extension of the cube schema. If a formalism allows more than one (especially infinitely many) ways of formulating semantically equivalent queries, it is important for our approach that all these queries can be easily normalized to a canonical form which is used to represent the class of semantically equivalent queries during the prediction and prefetching process.
- *Navigational User Behavior.* The formalism must not only contain means to specify single queries but also to formulate sequences of queries that are the result of applying multidimensional query transformation (for example, ‘rotating’ the results of a previous query).

- *Nature of the User Interface.* It is essential that the PROMISE/OLAP query formalism takes the structure of the user interface into account. One reason for this requirement is that the class of possible queries is being restricted by the structure of the user interface which should be mirrored in the formalism. Another reason is that the user interface defines the possible navigational query transformations that are used to formulate sequences of interdependent queries. Therefore, it heavily influences the type of patterns occurring in query behavior.

We evaluate each class of approaches according to these criteria. The evaluation can be found at the end of the corresponding section. Table 3.2 shows the qualitative scoring model used in evaluating the approaches according to the criteria.

++	criterion excellently fulfilled according to our requirements
+	criterion is fulfilled but not to the extent envisioned in our set of requirements
-	criterion is only partially fulfilled
--	fulfillment of this criterion is not achieved or can only be fulfilled using complex workarounds

Table 3.2: Scoring Model to Evaluate Query Approaches according to their Suitability for PROMISE/OLAP

3.2.1 Formal Approaches

As described in the previous section, the discussion about the static part (defining the data structure) for the multidimensional data model is still in progress. Consequently, no commonly agreed description of queries in the presence of a multidimensional schema is available. Although the details of the different proposals naturally differ because of the different descriptions of the underlying data structures, they can be classified according to the following categories:

- *algebraic approaches* that specify single queries as a sequence of algebraic operations (cf. 3.2.1.1).
- *logic based approaches* that use description logics to specify a query as a logical predicate which characterizes the properties of the query result (cf. 3.2.1.2).
- *graphical approaches* that derive a query specification from a graphical formalism (cf. 3.2.1.3).

A detailed discussion of these different classes of approaches can be found in the following three sections. We restrain from covering each individual approach in detail (please refer to the corresponding survey papers [SBH99] or [VS99] or to the appropriate original papers) as this is not necessary to evaluate their suitability for this work.

3.2.1.1 Algebraic approach

The basic idea of the algebraic approaches is to define a set of operators working on the static data structures defined by the data model. A query is described by a sequence of operations together with the according parameters thus providing a quite procedural description of a query.

In the multidimensional case these operations manipulate the multidimensional data structures defined by the data model i.e., instances of cubes (e.g., adding or removing dimensions or aggregating base data to a higher granularity). Each operator takes one or more instances of a multidimensional cube together with further parameters (for example, aggregation functions, predicates) and produces another cube instance. Notably, these algebraic operations must not be confused with the interactions the user performs (for example, drilling down) between two queries using the OLAP interface. While an algebraic operation specifies how the result of a single query can be derived from the set of basic structures stored in a database, an interaction defines how the next query is derived from the previous one[21].

An interpreter for the execution of queries represented in this algebraic way can be straightforwardly implemented by providing an implementation of each of the basic operations. Albeit, the semantically same query can be expressed using a large number of different (possibly infinitely many) algebraic expressions. This allows for using algebraic transformations rules in order to optimize the execution of the query. Generally, the algebraic representation of a query is best suited for processing the query internally in the database (in our case OLAP) system and for discussing query optimization issues.

Analogously to the static data model part, an OLAP algebra can be based on an existing algebra (for example, the relational algebra) by adding new operators and extending existing operators or can be designed from scratch. The answer to the question which of these approaches should be taken is linked with the basic decision if the multidimensional data model is regarded as independent from the relational model. The most prominent and one of the earliest approaches to formulate queries referring to multidimensional structures in the relational data model is the Cube operator proposed by Gray et. al. [GCB+97]. This operator is a generalization of the relational group-by operator and has been integrated into the SQL Server 7.0 and IMB DB/2 products. Other extensions of the relational algebra have been proposed by [GL97] and [LW96]. As argued before, the independence of the multidimensional data model on a logical level is a fundamental assumption of the PROMISE/OLAP approach. Therefore, for the rest of this evaluation, we only take native multidimensional query formalisms into account.

The query language (or algebra) is specified by giving a (minimal) set of operations that manipulate the static data structures provided by the data model. The approaches ([AGS97], [DT97], [LW97], [GL97], [Vas98]) mainly differ in the set of operations being defined. The proposed operations can be broadly classified according to the following categories:

- *Restructuring operations* encompass all operations that change the structure of the cube. These operations can be further categorized by their functionality:
  - *resolve dualism between dimensions and measures.* When introducing the multidimensional data model in 3.1, we already remarked that dimensions and measures are dual concepts in the sense that the same data entity can be regarded as a dimension or a measure depending on the query context. Data models that statically partition data into dimensions and measures therefore provide operations to override this decision for the scope of a query, thus allowing dimensions to be used as measures in a query and vice versa.
  - *change cube granularity.* This class of operations changes the cube granularity according to the classification schema. Notably, only the approach of Vassiliadis ([Vas98])

[21] However, the result of an interaction can well be a query specification that is being extended with an additional algebraic operator.

contains an operation to increase the granularity of a cube (corresponding to a drill-down operation), while all other approaches only allow for grouping elements to a higher classification level. Some approaches (for example, [AGS97]) implicitly assume the execution of an aggregation operation (manipulating the cell values) together with the granularity change while other approaches model this as an autonomous operation (see *cell manipulations* below) thus assuming bag semantics for cell values in intermediate query results.

- *Dimensional restrictions* are operations that remove dimension members (and the corresponding cells) from the cube thus not changing the cube schema but reducing the active domain of a cube instance. The condition when members are to be removed from the cube can either be formulated according to the classification structure (for example, only keeping the days of the year *1998*) or more generally by any other predicate defined on the domain of the dimension (for example, removing dates larger than 1/6/1998).
- *Join operations* take two cubes as parameters and produce a single cube. The cubes must share a set of common dimensions. Different semantics for the join can be defined analogously to the inner- and outer-join semantics defined for the relational model. [BG00] and [Alb01] contains a more detailed discussion of different multidimensional join semantics.
- *Cell manipulations* do neither change the dimensionality of the cube nor the active domain of its dimensions but change the cell values. These functions include simple scalar operations (for example, multiplying every cell value to a constant) but also aggregations that map a set of values to a single value.
- *Data model specific operations* subsume all the operations that are necessary to reflect the extended concepts of the data model (see Section 3.1.1.4). This includes e.g., operations to navigate nested cubes in a nested data model ([DKP+99]) or operations to manipulate the result with respect to features ([Leh98a]).

The approaches of [AGS97], [GL97], [DT97] and [Vas98] additionally describe a mapping of their operations to the relational algebra with aggregation functions ([OOM87]). This implies that the relational algebra with aggregations is at least as expressive as these approaches. Although, to the best of our knowledge, no substantial research has been done so far on the exact expressiveness of OLAP query formalisms.

To give an impression of how a query is formulated using an algebraic approach, we use the algebraic approach defined by [AGS97] as an illustrative example:

Example 3.12 (The Algebraic Approach of Agrawal et. al. as a Case Study)

The basic data structure (called MD Cube) of the approach is an n-dimensional array. Each of the dimensions is a flat collection of values that are being used to address the cells of the cube. Each cube cell has the structure of a k-tuple ($k \geq 0$) which contains the cell values[22] (we call this tuple the *measure tuple*). Thus, a cube instance (the approach does not distinguish between schema and instances) can be defined as a 3 tuple $C=(D,E_C,N)$, where

- D is a set of dimension names (each dimension has a corresponding domain).
- E_C is a total function mapping the coordinates (cross product of the dimensions domains) to a measure tuple or to the special value '⊥' denoting an empty cell.
- N is a list of measure names that are used to label the elements of the measure tuples.

[22] For the special case k=0 each cell is a boolean value.

Notably, the model does not contain any information about the classification schema. The repair data cube introduced in Example 3.8 could be expressed using this model in the following way:

C=(*repair date, repair location, repaired part,* E_C, (*# of repairs, # of persons*))

The following operations are proposed to manipulate MD cubes:

- ***pull, push*:** These operations are used to achieve the symmetrical treatment of cell values (measures) and dimension members. *Push* inserts the values of a dimension as a new component into the measure tuple thus resulting in a cube with (*k*+1)-tuples as cells. *Pull*(*i*) adds an additional dimension to the cube which contains the values of the *i*-th element of the measure tuple as dimension members.
- ***destroy dimension*:** This operation reduces the dimensionality of the cube by one. It removes a dimension from the cube. The operation requires that the dimension only contains one member.
- ***restriction*:** This operator evaluates a predicate P for a given dimension D_i and removes the elements of D_i which do not satisfy the predicate from the dimension, thus reducing the size of the cube but not its dimensionality.
- ***join*:** The join operation combines two cubes (of dimensionality m and n) which share k dimensions. It produces an n+m-k dimensional cube. The "join-partners" in each of the shared dimensions are determined by user-defined functions. Additionally, an element combining function has to be given that is applied to all measure values that get mapped to the same cell of the resulting cube. The combining function can e.g., be used to aggregate the cell values of the cube.

The authors also define the operator merge, which is not atomic in the sense that it can be expressed by using the self-join operator with a special function. This operator can be used to express aggregation operations.

A possible specification for the query which is described by the following "prose" text is given below: "Give me the cumulated number of repairs split up by month for locations in Bavaria during the year 1999."

```
destroy_dim (
  merge(
    restrict(
      restrict( C, f_r-up( day→year )(day) = 1999),
      f_r-up(location→geogr.region)(location)=Bavaria ),
  {[repair time,f_r-up(day→month)], [repaired part, f_const]}, f_sum),
repaired part)
```

Figure 3.8: Sample Query Using the Algebraic Approach of [AGS97]

The functions $f_{r\text{-}up}$(<level1>→<level2>) are the mapping functions for the grouping of dimension elements and must be accordingly defined and provided at query time. They are not part of the database schema. The function f_{sum} is a user defined function calculating the sum and f_{const} returns a constant value irrespective of the input value.

The *restrict* operation removes all cells from the cube that do not belong to the year 1999 and to the geographic region Bavaria. The *merge* operation aggregates the data to the requested result granularity (using the summation function provided as a parameter) for those dimensions that are not queried at the lowest granularity level (dimension repair time to *month* and dimension repaired part to *all parts*). Because all values of the repaired part dimension are grouped to one value, a constant can be used as the grouping function. The destroy dimension

function removes the repaired part dimension from the query result. This can be done because the dimension only contains one element after the previous merge operation. ◆

Evaluation for PROMISE/OLAP:

Generally, algebraic approaches allow for a very precise definition of the semantics of a query, by formally describing the operational semantics of each basic operation. However, the rather procedural style of the algebraic query specification conflicts with our requirement of a descriptive formalism. The most important advantage of algebraic approaches when designing query optimizers for database systems, namely the existence of transformations that preserve the semantics of a query turns out to be the largest disadvantage for our purpose. Designing a canonical form from an algebraic expression is complicated and the transformation of a query to the canonical form can become time consuming. Being targeted at a database-internal representation of queries, algebraic approaches do not (and should not) take the user interface structure and the behavior of the user into account. The only multidimensional approach that offers a concept for dealing with navigational user behavior is the approach of P. Vassiliadis [Vas98], where the author proposes to store the base granularity of the cube for all intermediate results of an OLAP session such that drill-down operations (that request data on a finer granularity than the current intermediate results) can be efficiently formulated and evaluated by the system. Table 3.3 summarizes the evaluation results.

Suitability for PROMISE/OLAP: Algebraic approaches				
semantics	**descriptiveness**	**canonical form**	**navigation**	**user interface**
++	-	--	-	--

Table 3.3: Schematic Evaluation of Algebraic Approaches according to their Suitability for PROMISE/OLAP

3.2.1.2 Logic Based approach

The logic based approaches use descriptive logic formalisms to specify queries. That means that a query is specified as a predicate over a set of data structures. The set of syntactically correct (well-formed) formulas is usually described by a logical calculus. This approach is more descriptive than the algebraic approach because properties of the result are described by the query expression rather than giving a procedural description of how to compute the results. Therefore, a logic based approach is suited as a basis for defining descriptive query languages and discussing the expressiveness of such languages.

Regarding the large set of multidimensional algebras described in the last section, it is surprising that only a very small number of approaches tackle the problem of designing a query language for OLAP database systems using description logics ([CT98a],[HMR97]).

The formalism presented in [HMR97] is an extension of the well established DATALOG language. Each cell of the data cube is regarded as a DATALOG fact and typical structural manipulations and aggregation operations are modeled as logical rules. The paper defines model-theoretic semantics and an equivalent fixpoint semantics for the extensions. The authors also derive an intuitive evaluation schema for multidimensional query expressions.

The approach of Cabbibo and Torlone [CT97], [CT98a] is based on a calculus for f-tables. An f-table is the data structure representing a data cube. The calculus contains scalar functions and aggregate functions as well as the grouping functions of the classification schema (called roll-up functions in the terminology of the approach).

Example 3.13 (The Logical Approach of Cabbibo and Torlone as a Case Study)

The central data structure of the approach ([CT97]) is an f-table which can be compared to a relational table with the difference that an f-table possesses two distinct types of columns: dimension columns and measure columns. The domain for the dimension columns is defined by a dimension which is a structured entity consisting of the classification schema and instance. The f-table schema for our example would look like the following:

```
REPAIR[repair time: day, repair location: location, repaired part: part]
      →[ # of repairs: numeric , # of persons: numeric]
```

Figure 3.9 shows a possible representation of a query using the REPAIR f-table: "Give me the cumulated number of repairs split up by month for locations in Bavaria during the year 1999."

```
query = (geogr.region, month: r | r =                                          1
      sum(day, location: num | ∃num,p (                                        2
            [num,p] = REPAIR[day, location, part]                              3
            ∧ month=R-UP_day^month(day)                                        4
            ∧ geogr.region =R-UP_location^geogr. region (location) ) )          5,6
      ∧ R-UP_month^year(month)=1999                                            7
      ∧ geogr.region = 'Bavaria')                                              8
```

Figure 3.9: Sample Query Using the Logic-Based Approach of [CT97]

The query specifies the result as an *f*-table with two dimensions named *geogr. region* and *month* and a single result measure *r* (line *1*). The aggregation of the base data to the required granularity is specified by the aggregation function sum (line *2-6*). It specifies an aggregation of the base data from *days* to *month* (line *5*) and from *location* to *geogr. region* (line *6*). As no condition is given for part, all data in this dimension is aggregated. Notably, the variables month and geogr. region are bound by the definition of the result f-table (line *1*). The restrictions of the repair time dimension and the geographic region are specified by lines *7* and *8*. ◆

Evaluation for PROMISE/OLAP

The above mentioned logical approaches both offer well defined semantics of the query results with respect to the underlying data model. The main advantage of a logic based approach is that a logical predicate provides a very descriptive way of formulating queries. This is the reason why these approaches are very suited to serve as a foundation for declarative query languages (like e.g CQL [BL97]). However, the problem of defining a canonical form for logical expressions it is at least as difficult as the design of an algorithm that determines if two distinct logical expressions have the same semantics. Although none of the approaches addresses this problem, it is likely that a solution is NP-hard (the query equivalence problem has been proven to be NP-complete for SQL queries [SKN89]).

The most important drawback of the logic based approaches in the context of the PROMISE/OLAP framework is that they do not take the user interface structure into account. Modeling iterative query specification (due to navigational user behavior) could be done by formulating the next query using the expression of the previous query as a sub-expression. However, none of the approaches has tackled this problem so far.

Suitability for PROMISE/OLAP: Logical approaches				
semantics	**descriptiveness**	**canonical form**	**navigation**	**user interface**
++	++	--	-	--

Table 3.4: Schematic Evaluation of Logic-Based Approaches according to their Suitability for PROMISE/OLAP

3.2.1.3 Graphical approaches

The driving idea behind the graphical approaches to query language design is to bridge the gap between the domain expert's requirement for an intuitive way of querying a database and the database system needing a precisely defined query specification. These approaches argue that a graphical representation of the multidimensional database schema can be intuitively understood by the end user and be used in formulating queries. Therefore, both graphical solutions for OLAP systems that have been proposed ([GMR98],[CT98b]) use a graph-based visualization of the multidimensional database schema (comparable to our graphical ME/R notation).

Golfarelli et. al. introduce a visualization of a multidimensional schema for conceptual design purposes [GMR98]. The proposed model is called DF-model and represents a cube schema as a labeled graph. The interesting part of the approach regarding query specification is the possibility to mark nodes of the graph that represent classification levels. A mark indicates that this level is often used by analysts to formulate queries. However, this approach is mainly targeted at defining typical query workloads (that can be regarded as patterns) during system design, rather than the specification of individual queries or query sessions.

The graphical approach of Cabbibo and Torlone [CT98b] is especially designed to enable the user to incrementally formulate a query starting form a visualization of the logical multidimensional schema. A so called f-table constitutes the basic data structure of the underlying data model. Comparing the data model to the PROMISE/OLAP data model, it is noteworthy that the definition of an f-table schema combines the notions of a cube schema definition and the corresponding classification schema in the PROMISE/OLAP approach. An f-table instance however corresponds to the understanding of a cube instance in PROMISE/OLAP. The graphical query language relies on the visualization of an f-table schema as a labeled graph with different node types representing f-table names, dimensions[23], classification levels, and descriptions for classification levels. A query is specified as a sequence of f-graphs $G_0,...,G_m$, where G_0 is a representation of the database schema and G_m is a representation of the query result. The sequence is called s-graph. The intermediate steps are generated by a user that interactively marks nodes (indicating the result granularity respectively the result measures), labels edges (indicating the aggregation functions to be used) and nodes (specifying restriction predicates) or inserts edges (to combine data from different f-tables). The semantics of the query are formally defined by an algorithm that derives a mapping function from the source f-table instances to the resulting f-table. This mapping can be computed stepwise from consecutive steps of the s-graph.

Evaluation for PROMISE/OLAP

Deriving an algebraic description of a query from the s-graph specification allows for a precise definition of the semantics of a query. However, the approach of [CT98b] only defines

[23] Notably, in [CT97] the understanding of a dimension comprises the classification structure consisting of the classification levels. Therefore, a dimension is represented as a hyper-node in the graph which contains the classification levels as sub-nodes.

the manipulation operations to the graphical structure and the mapping process to the algebraic representation in an informal way.

The main idea of iteratively transforming and manipulating the source schema until the schema of the query is obtained (like proposed in [CT98b]), comes very close to our understanding of the navigational query formulation being exploited for the predictive PROMISE/OLAP approach. However, it is problematic to provide the user with a representation of the (rather abstract) database schema, as the user often does not understand the static structure of the schema. Due to this observation, real-world OLAP tools (see next section) do not present the schema but the result of the last query to the user who then manipulates the representation of the results rather than the schema.

Another, more serious problem for our approach is that a formulation of a canonical query form is very difficult given the fact that a query is specified as a sequence of labeled graphs.

Suitability for PROMISE/OLAP: Graphical approaches				
semantics	**descriptiveness**	**canonical form**	**navigation**	**user interface**
+	+	--	++	+

Table 3.5: Schematic Evaluation of Graphical Approaches According to their Suitability for PROMISE/OLAP

3.2.2 The Structure of Interactive OLAP Query Interfaces

The issue of how to formulate a query has not only been addressed by researchers but also by product vendors who implement graphical interfaces to their products. One of the basic philosophies of an OLAP tool is that the user can formulate queries without any programming skills. Therefore, the user interface component of the OLAP tool provides the user with a graphical representation of the multidimensional database schema which is used to interactively formulate queries. No standard exists for the layout and functionality of the user interface such that the operations and interactions necessary to formulate a given query are naturally very tool specific. This is problematic for our approach as we need to base the PROMISE/OLAP approach on a general description in order not to restrict its applicability. However, a deeper analysis of the different products shows that the underlying functionality which is relevant for the OLAP database is similar for all these tools despite of their different front-end layouts.

Consequently, in this section we develop a model for an OLAP user interface abstracting from the peculiarities of different products and thus hiding them to our approach. The resulting abstract user interface definition is based on an analysis of the most important OLAP systems that were compared in [DSV+97] and were used by the author for the implementation of several real-world projects. As illustrative examples for this work we use the systems Oracle Express, Cognos PowerPlay and Informix MetaCube[24]. Not all of the described elements can be found in every product nor are all special features of products covered. The purpose of this description is to provide the reader with an impression of the typical operations that can be executed in an OLAP system and their implications for the design of a query representation formalism.

[24] These tools were chosen to represent the widest possible variety of different architectural philosophies, including a client based OLAP product (PowerPlay), a full-fledged multidimensional database server (Express) and an OLAP implementation based on a relational database system (MetaCube). For a more detailed comparison see also [DSHB99].

In OLAP interfaces, the result visualization is closely linked to the formulation of queries as the user directly manipulates the result visualization in order to retrieve new data. Therefore, before describing how interactive query formulation takes place using the abstract OLAP interface, it is necessary to briefly describe the visualization of multidimensional query results i.e., multidimensional cube instances. Each cube instance is visualized by the tools using a so-called *MD presentation*. Depending on the needs of the user and the characteristics of the data, this presentation can be performed in a multitude of ways (e.g., as cross-tables, 2D/3D bar charts, pie charts or even as 3D renderings). We are not interested in the actual type of the visualization, as it does not influence the query that has to be executed in order to retrieve the data from the database. Therefore, we use the term *multidimensional presentation* (short *MD presentation*) to subsume all the different forms of visualizing multidimensional data.

Each type of MD presentation has a fixed maximum number of data dimensions that can be visualized (e.g., 1 dimension for 2D bar charts[25], 2 dimensions for a cross table or a 3D bar chart or 3 dimensions for a 3D rendering)[26]. If the data stored in the database is of a higher dimensionality than the number of dimension allowed by the presentation, the result of a query must be reduced in dimensionality.

One way to decrease the dimensionality of a cube stored in a database during a query is to restrict a dimension (for example, repair location) to a single element (for example, Germany) and then to remove the dimension from the cube (the corresponding algebraic operations have been described in the previous section). Of course, the user needs to know the restriction element in order to correctly interpret the results (for example, to know that the presented figures only apply to Germany). These dimensions (called *selection dimensions* of the query) are not directly contained in the MD presentation (for example, the cross table) but are represented elsewhere in the interface. E.g., Oracle Express and Informix MetaCube visualize the corresponding elements next to the table in the upper left corner (see Figure 3.14 and Figure 3.15) while Cognos PowerPlay uses a dimension bar on top of the MD presentation to show the current restriction elements for the different dimensions (see e.g., Figure 3.11).

The remaining dimensions (called *result dimensions* of the query) span the multidimensional space which is being visualized by the MD presentation. If there are still more result dimensions than the maximum number of dimensions allowed for an MD presentation type, the dimensionality has to be artificially reduced e.g., by building the cross product of the elements of two or more dimensions and visualizing the result along one of the dimensions of the MD view (for example, along the axis of a table). All the parameters that are necessary to display the MD presentation apart from the actual raw data (e.g., the mapping of result dimensions to display axes, color schemata, diagram type, diagram specific information) are called the *configuration of the MD presentation.*

Example 3.14 (Multidimensional Presentation)

In order to demonstrate the concepts of OLAP user interfaces and queries, we use an extended version of the repair cube schema example which was first introduced in Example 3.8. As shown in Figure 3.10 it contains five dimensions (vehicle, day, location, part and type of repair) and three measures (# of repairs, duration of repairs, # of persons). The cube schema for the example looks as follows:

[25] Notably, one display dimension is used to visualize the measure values.

[26] Of course, more than one dimension of the query result can be mapped to a single dimension e.g., by building the cross product of the elements of two or more dimensions and visualizing the result along one of the dimensions of the MD view.

```
C_Rep= (  ( repaired vehicle: vehicle,
            repair time: day,
            repair location: location,
            repaired part: part
            type of repair:: type of repair ),
          (#repairs, duration, #persons) )
```

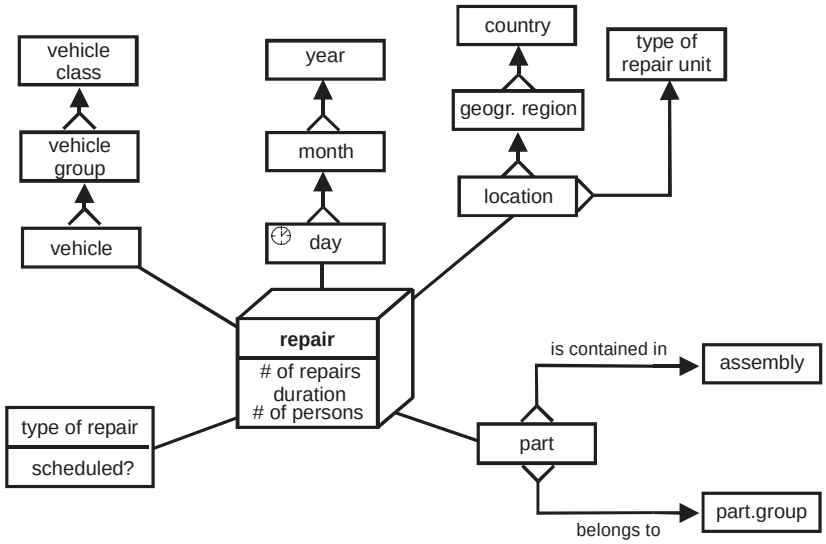

Figure 3.10: An Extended Version of the Repair Cube Schema

For this example query, the user restricts the *repair time* to the year '1998' and the repaired vehicle to the vehicle class 'Cars' and chooses to see the results split up according to the countries (dimension repair location) and the vehicle group (dimension repaired vehicle). Figure 3.11 shows how Cognos PowerPlay 6.5 visualizes the results of such a query. The dimension bar above the result visualization shows the different dimensions of the multidimensional database schema and restrictions that have been made to reduce the set of classification nodes being used to calculate the query result.

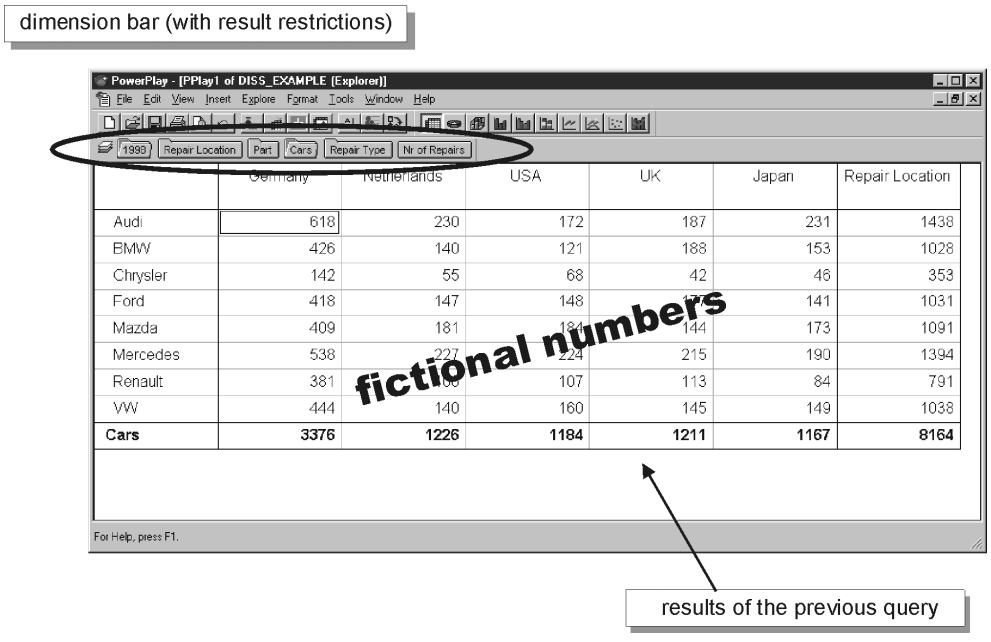

Figure 3.11: The Structure of the Cognos PowerPlay 6.5 Interface

In our example, the dimension repair time has been restricted to the year '1998' (leftmost entry on dimension bar), while e.g., the repair location is not restricted (second entry from the left). The rightmost entry on the dimension bar indicates which measure is being displayed by the visualization (in this case the number of repairs). Figure 3.12 shows a different visualization of the same query results using a 3D bar chart. ◆

A comparison of the MD presentation used by PowerPlay, Oracle Express (see Figure 3.13), and Informix MetaCube (see Figure 3.14) shows that the main differences between the user interfaces merely concern the layout but not the underlying functionality. It also shows that the distinction between selection dimensions and result dimensions and the concept of restricting the base data using a single node of the classification is inherent to all the products.

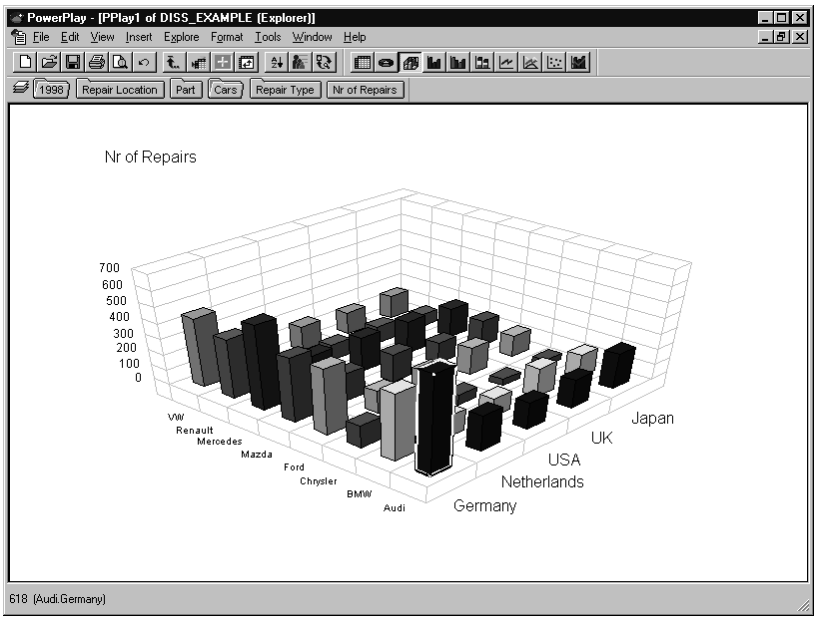

Figure 3.12: A 3D Bar Chart as an Example for an MD Presentation

An important feature of common OLAP interfaces is that the iterative specification of queries is done directly using the result visualization. The philosophy behind this interface design is that the user should not be aware of the database in the background and should concentrate on the data exploration task without having to change the interface for analyzing data or formulating/refining the queries. After starting a session, the system displays the results of a predefined query (or a default query). The user can then interactively manipulate the MD presentation by applying certain transformations to the presentation. The transformations that can be directly handled by the front-end component (for example, swapping the axes of a diagram) are carried out immediately, while changes that require new data from the database trigger a new query which is automatically generated by the front-end tool and passed to the OLAP processor.

Typical transformations are the exchange of result and selection dimensions using e.g., a 'drag and drop' mechanism (called *rotate* or *pivot* transformation) or changing the granularity of the displayed result e.g., by double-clicking a classification node which should be split up using the next lower level (called *drill-down* resp. *roll-up* operation). The type of transformation being available is again tool specific. Some of the tools offer very 'fine-grained' transformations thus requiring the user to perform several operations to achieve the desired effect while some tools offer more complex operations (for example, the simultaneous drill down in two dimensions by double clicking a table cell in the MD presentation) that can be regarded as

a grouping of several simple interactions. Some tools automatically execute a new query after each transformation (*single transformation execution model*) while other tools offer the user the possibility to explicitly execute the next query after having performed all the necessary transformations (*multiple transformation execution model*). Typically, tools that can offer better response time due to their storage architecture (for example, because they use a native multidimensional database engine) prefer the single transformation execution model, while those tools with longer response times (for example, OLAP tools based on relational storage) tend to prefer the multiple transformation execution model.

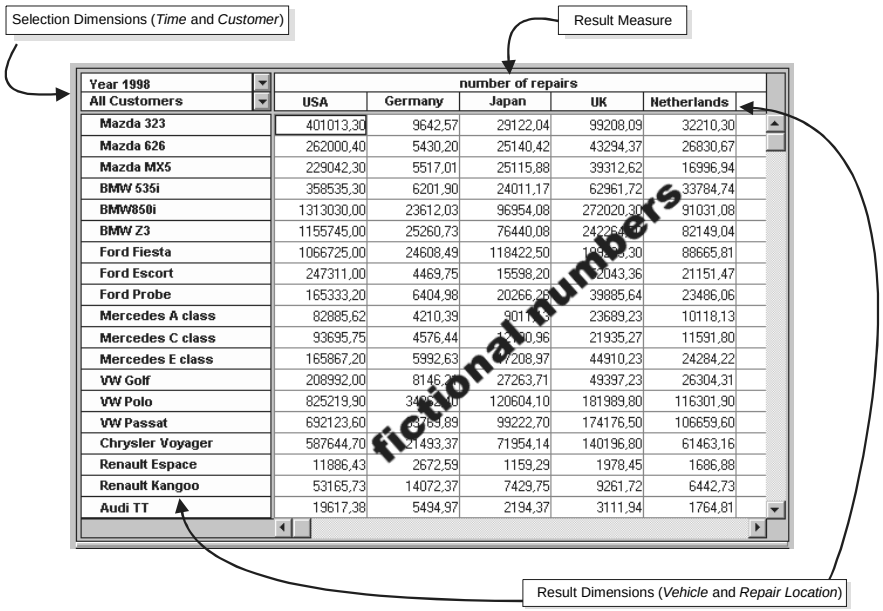

Figure 3.13: The User Interface of Oracle Express 6.2

Conclusions for PROMISE/OLAP: The relevant part of the actual user interface for PROMISE/OLAP is the MD presentation as it is deployed by the user to indirectly interact with the database. The actual queries are being generated from the configuration of the MD presentation. This means that each configuration of the MD presentation uniquely corresponds to a (canonical) database query. However, the configuration of an MD presentation contains more details (e.g., type, layout information) that are not relevant for the generation of the database query. Consequently, instead of describing the user's interactions in terms of the user interface, for PROMISE/OLAP we represent the user's interactions as canonical OLAP queries (corresponding to a class of MD presentation configurations). Section 3.2.3 formally defines the PROMISE/OLAP canonical query based on the understanding of an MD presentation presented here. On the user interface level, the interactive part of the user's behavior is represented as changes to the configuration of the MD presentation. Albeit, for our approach we are only interested in changes that transform the corresponding (canonical) query. Therefore, in Section 3.2.4, we define a set of transformations that modify canonical OLAP queries and use these transformations throughout the thesis to abstractly represent the user's interactive behavior.

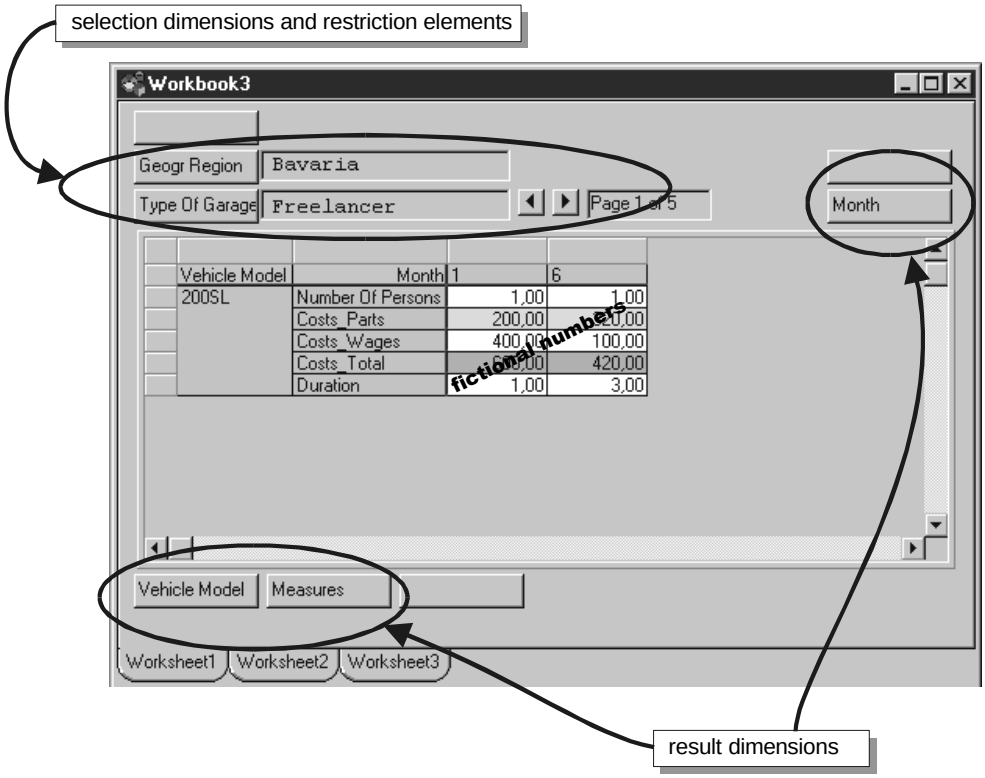

Figure 3.14: The User Interface of Informix MetaCube 4.2

3.2.3 The PROMISE/OLAP Model for Canonical Queries

As argued in the previous sections, for the PROMISE/OLAP approach, a canonical descriptive representation of queries is necessary that takes into account the restrictions of the typical user interface (MD presentation). This section defines a canonical query form which we use throughout the rest of this thesis to formulate single queries against the OLAP database system. The design of the canonical query is guided by the structure of the abstract OLAP interface described in the previous section.

By means of this interface, the user interactively defines a canonical query by selecting a set of interesting measures, called the result measures (for example, *number of repairs*). The user further specifies a specific area of interest by restricting the set of dimension members for some dimensions of the cube schema C_Ψ and by determining the granularity (i.e., the classification level) on which he wants to have the result data displayed. The restriction predicate for the base data (i.e., the data stored in the granularity given in the schema definition) is implicitly formulated using the classification lattice of the dimension. The user picks a *restriction element* from any level of this classification lattice for each dimension that should be restricted. E.g., the user can restrict the *repair time* dimension to days in the year *1999*, by choosing the restriction element *'1999'* which belongs to the classification level *year*. For the dimensions which are not restricted by the user, we assume the special *<dimension_name>.all* element as the restriction element. The level to which the restriction element belongs is called the *restriction level* of the query for the corresponding dimension. The set of restriction elements uniquely identifies a sub-cube of the original data cube. The necessary aggregations of base data are specified by giving a result granularity for each dimension. The result granularity is defined by picking a level (called *result level* or *result granularity*) from the classification lattice of each dimension of the cube. In order to ensure a meaningful result, the result granularity must be finer than the restriction level with respect to the classification

lattice. This especially implies that a classification path exists in the classification for each dimension lattice from the result granularity to the restriction level. For example it is not meaningful to select all years containing the month 'January'.

Example 3.15 (Defining a Canonical Query)

Figure 3.15 visualizes the selections and aggregations defined by a canonical query for an abstract two dimensional data space. Each of the dimensions (dimension one left hand side, dimension two at the bottom) has a simple hierarchy defined over their members, which consists of three levels: $L_{1,1}$,$L_{1,2}$ and $L_{1,3}$ for dimension one respectively $L_{2,1}$,$L_{2,2}$,$L_{2,3}$ for dimension two. The restriction element r_1='I' for dimension 1 (left hand side) is specified on level $L_{1,1}$ while the restriction for dimension two is formulated with respect to level $L_{2,1}$ (r_2='A'). This defines a rectangular area of the data space which is used to calculate the result[27]. The structure of the result is defined by the result granularity for each dimension, which is depicted by a dotted line in Figure 3.15. For dimension 1 (left) the result granularity is $L_{1,2}$ while $L_{2,1}$ is the result granularity for dimension 2. The structure of the query result is shown in the right bottom corner.

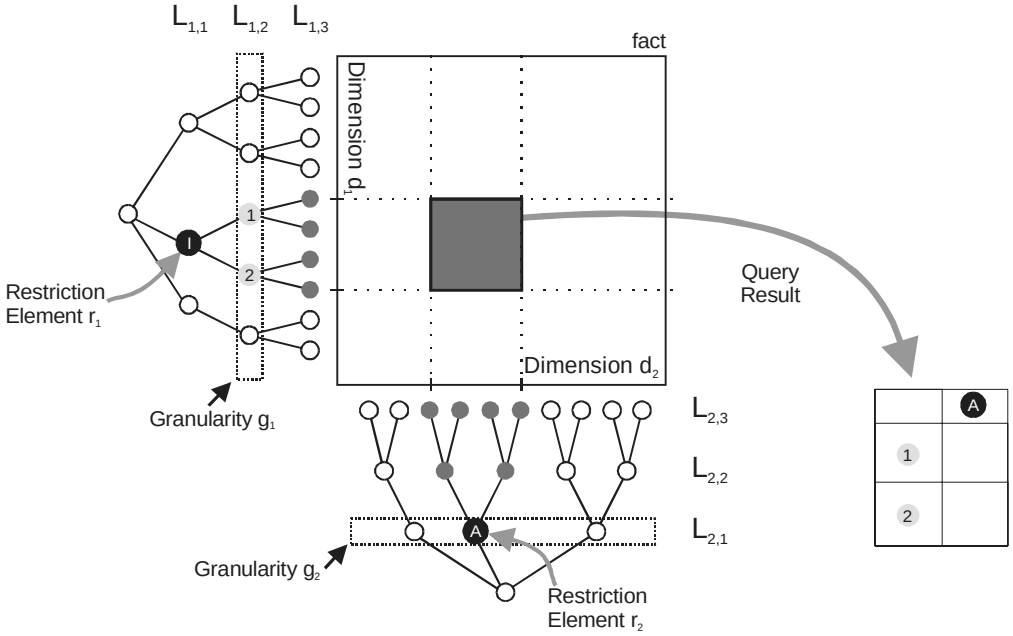

Figure 3.15: Visualization the Effects of a Canonical OLAP Query

Notably, for dimension 2 in our example, the restriction level equals the result level. Thus this is a *selection dimension*, while dimension 1 is a result dimension of the query. ◆

The following definition formalizes our informal definition of a canonical OLAP query. The semantics (i.e., the results of the query) are described in Definition 3.13.

[27] Of course, in a working OLAP system, the result can also be derived from precomputed aggregates. At this point, we do not take preaggregation into account as this is a form of internal system tuning and should not be part of the descriptive query formulation.

Definition 3.12 (Canonical OLAP Query):

For an n-dimensional cube schema $C_\Psi = (\{d_1,\ldots,d_n\}, M_C)$ with the corresponding classification schema $\Psi = (L_\Psi, class_\Psi)$, a canonical query q_C is defined as a tuple $q_C = (M_q, R_q, G_q)$ where

- $M_q \subseteq M_C$ is the *result measure set*.
- $R_q = (r_1, r_2, \ldots, r_n)$ with $r_i \in dom(d_i), 1 \le i \le n$. R_q is called the *restriction vector* of q_C. The elements r_i of R_q are called *restriction level* of the respective dimension d_i.
- $G_q = (g_1, g_2, \ldots, g_n)$ with $g_i \in L(\Psi|_{d_i})$. G is called the *granularity vector* of q_C. The elements g_i of G_q are called *result level* or *result granularity* of the respective dimension d_i.

In order for q_C to be well-formed, the result granularity has to be smaller or equal than the restriction level, formally:

$$\forall i \in \{1,\ldots,n\} : g_i \le_\Psi r_i \qquad \text{(WF)}$$

We call the set of all well-formed queries against a given cube schema C_Ψ the *canonical query space* of this cube schema and denote it as Θ_{C_Ψ}. ◆

Notably, the condition for well-formed canonical queries also implies that the classification schema contains a path that contains both, the level of the restriction element, and the result granularity. For any given cube schema, the set of well-formed canonical queries is finite as the number of restriction elements for each dimension, the number of classifications levels in the classification schema and the number of measures is finite. The cardinality of this set has the following upper bound (which assumes that all possible queries are well-formed):

$$\left|\Theta_{C_\Psi}\right| \le \underbrace{\sum_{i=1}^{|M_C|} \binom{|M_C|}{i}}_{\substack{\text{number of distinct result} \\ \text{measure sets}}} \cdot \underbrace{\prod_{i=1}^{n} \left|dom(\Psi|_{d_i})\right|}_{\substack{\text{number of distinct} \\ \text{restriction vectors}}} \cdot \underbrace{\prod_{i=1}^{n} \left|L\left(\Psi|_{d_i}\right)\right|}_{\substack{\text{number of distinct} \\ \text{result granularity vectors}}}$$

However, the exact number of well-formed queries is smaller than this upper bound because of the condition for well-formed queries (WF). For the storage estimation of the prediction models in the following chapter, we need a more precise approximation of the size of this set. The exact number depends on the distribution of the classification nodes across the levels of the classification lattice in each dimension and on the number of classification schema paths in the classification lattices of the different dimensions. Let *cpaths*($\Psi|_{lev}$) denote the set of all maximal classification schema paths for the classification lattice $\Psi|_{lev}$ i.e., all paths that connect the LUB (the cube dimension) with the GLB (the level <*dimension*>.*all*). Each path consists of a linearly ordered list of classification levels $p_1,\ldots,p_n$. Then the number of well-formed canonical queries can be better approximated by the formula shown in Figure 3.16 that takes the structure of the dimension lattices into account (see Example 3.16 and Figure 3.17 for an application of this formula).

$$\left|\Theta_{C_\Psi}\right| \leq \underbrace{\sum_{i=1}^{|M_C|}\binom{|M_C|}{i}}_{\text{number of distinct result measure sets}} \cdot \prod_{i=1}^{n}\left(\underbrace{|dom(d_i)|}_{\substack{\text{restriction on the lowest level} \\ \Rightarrow g_i = d_i}} + \sum_{p \in cpaths(\Psi|_{d_i})} \underbrace{\left(\sum_{j=2}^{len(p)-1} \left(|dom(p_j)| \cdot j \right) \right)}_{\substack{\text{restriction on intermediate level } (p_j) \\ \Rightarrow \text{granularity can be on same level or lower (j possibilities)}}} + \underbrace{\left| L\left(\Psi|_{d_i}\right) \right|}_{\substack{\text{restriction on the all level} \\ \Rightarrow \text{all result granularities possible}}} \right)$$

Figure 3.16: Estimating the Number of Well-formed Queries for a Cube Schema

Example 3.16 (Canonical OLAP Query):

An example query against the extended vehicle repair cube introduced in Example 3.14 might be "Give me the number of repairs for the year '1999' and the 'steering' assembly split up by the geographic regions of 'Germany'". The corresponding canonical OLAP query has the following form:

```
q_example := ( {#repairs},
              ( all vehicles , 1999, Germany      , steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

Figure 3.17 contains a calculation of the number of possible restriction/granularity combinations that fulfill the condition (WF) for the example schema (according to the formula shown in Figure 3.16). For this 5-dimensional schema, the number of possible combinations is $3{,}9 \cdot 10^{12}$. Consequently, the number of possible well-formed canonical OLAP queries is 7 times higher (number of possible measure combinations that can be selected), thus $2{,}7 * 10^{13}$. ◆

So far, we have only formally described the syntax of a canonical OLAP query and have stated the semantics of the query only informally. The following definition formalizes the result of a canonical OLAP query.

Definition 3.13 (Semantics of a Canonical OLAP Query)

Let us assume an n-dimensional cube schema $C_\Psi = (\{d_1,\ldots,d_n\}, M_C)$ conforming to the classification schema $\Psi = (L_\Psi, class_\Psi)$ and an instance I_C of the cube schema C_Ψ. The result of a canonical query $q_C(I_C)$ is again a multidimensional cube instance I'_C. However, the result instance is defined over a transformed cube schema C'_Ψ. The transformed cube schema C'_Ψ and instance $I'_{C'}$ are defined as follows:

- $C'_\Psi = (\{g_1,\ldots,g_n\}, M_C)$
- $I'_{C'}(x_1,\ldots,x_n) = \text{aggr}\left(\left\{(y=(y_1,\ldots,y_n), I_C(y)) \mid \text{group}_{d_i,\text{level}(r_i)}(y_i) = r_i \wedge \text{group}_{d_i,g_i}(y_i) = x_i\right\}\right)$

where **aggr** is the *n*-dimensional aggregation function which computes the aggregation value using the default aggregation functions for the respective measure/dimension combinations. ◆

dimension/classification level *l*	cardinality of level domains	restriction/granularity combinations
repaired vehicle		
vehicle	500	
vehicle group	50	
vehicle class	4	
vehicle.all	1	
		1*4+4*3+50*2+500=616
type of repair		
type of repair	2	
type of repair.all	1	
		1*2+2*1=4
repair time		
day	1461	
month	48	
year	4	
day.all	1	
		1*4+4*3+48*2+1461=1573
repair location		
location	500	
geogr. region	50	
country	30	
type of repair unit	3	
location.all	1	
		1*3+500+50*2+3*2+1*1=610
repaired part		
part	1000	1000
assembly	300	300
part.group	30	30
part.all	1	1
		1000+300*2+30*2+1*4=1664
all dimensions		$616*4*1573*610*1664=3.9*10^{12}$

Figure 3.17: Calculating an Upper Bound for the Number of Well-Formed Canonical Queries

3.2.4 Modeling Iterative Query Specification

As demonstrated in 3.2.2, the user works with the OLAP tool in an interactive way. An MD presentation visualizes the results of the last query and the user can formulate the next query by performing transformations to the last query definition. Nevertheless, the user does not directly manipulate the query definition (which is hidden by the front-end component) but he manipulates the configuration of the MD presentation which is then translated to a new query definition. The purpose of this section is to develop a formal model that is capable of describing the user's interaction with the OLAP front-end tool on the appropriate level of abstraction. Following the above argumentation, we describe the transformation on the level of canonical queries rather than on the MD presentation level. To this end, we introduce a set of query transformations that modify the canonical OLAP query corresponding to the current MD presentation configuration. Each query transformation operation thus describes a whole class of MD presentation manipulations (the actual presentation manipulations which belong to this class, e.g., clicking and dragging are of course dependent on the actual tool implementation). Thus, the transformation operations can be used to abstractly describe the users interaction independent of the actual implementation details of the front-end.

Formally, an OLAP query transformation operation (short query transformation) is a function $\tau : \Theta_{C_\Psi} \to \Theta_{C_\Psi}$ which transforms a canonical query q to a modified canonical query $\tau(q)$. Notably, these query transformations must not be confused with the operations of the algebraic approaches covered in Section 3.2.1.1. While the algebraic operations are used to specify the semantics of a single query (they operate on cube instances), query transformations describe the correspondence between two subsequent queries (operating on canonical queries). The query transformations are not designed with the goal of elegant formalization or implementation but should describe intuitive user interactions mirrored in the structure of the common user interfaces of current OLAP tools. Therefore, the set of transformation operations together with the definition of a canonical OLAP query constitutes the definition of the abstract OLAP user interface.

When introducing the interactive query interface of different OLAP tools in Section 3.2.1, we pointed out that the distinction between *result dimensions* (that are used to build the MD representation) and *selection dimensions* (that are restricted to a single element and thus not part of the MD presentation) is central to the user interface. This is mirrored in different types of transformations that can be performed for the different types of dimensions, resp. in different semantics of the same transformation depending on the type of dimension. Formally, the selection and result dimensions of a query are defined as follows:

Definition 3.14 (Selection Dimension, Result Dimension)

For a canonical OLAP query $q_C = \left(M, (r_1, \ldots, r_n), (g_1, \ldots, g_n)\right)$ over a cube schema $C_\Psi = \left(\{d_1, \ldots, d_n\}, M_C\right)$, we define the set of *selection dimensions* $\sigma(q)$ as follows:

$$\sigma(q) := \left\{ d_i \,\middle|\, level(r_i) = g_i \right\}.$$

A dimension which is not a selection dimension is called *result dimension*. ◆

Example 3.17 (Selection Dimension, Result Dimension)

Regarding the query q_{example} introduced in Example 3.16, the dimensions *repaired vehicle*(1), *repair time*(2), *repaired part*(4) and *repair type*(5) are selection dimensions, as the restriction element resides on the level of the result granularity. The dimension repair *location*(3) is a result dimension of q_{example}, as the restriction element *Germany* belongs to the classification level *country*.

```
q_example := ( {#repairs},
              ( all vehicles , 1999, Germany       , steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

$\sigma(q_{\text{example}})$ = {*repaired vehicle*, *repair time*, *repaired part*, *repair type*}

◆

The actual layout of the MD presentation is irrelevant for our database centered approach as it has no impact on the query that is necessary to retrieve the data from the database. E.g., for a table, it is irrelevant if the members of a result dimensions are used to label the rows or columns of the table. Also the type of the MD presentation (e.g., table, bar chart) is not important. Therefore, we refer to the abstract structure and granularity of the MD presentation for describing the possible operations (thus omitting user interactions that only change the layout of the presentation). The structure of the MD presentation is defined by the partitioning of the dimensions of the cube into result and selection dimensions.

Definition 3.15 (MD Presentation Structure and MD presentation Granularity)

The presentation structure *presstruct*(q_C) of a multidimensional query q_C against an *n*-dimensional cube schema $C_\Psi = (\{d_1,\ldots,d_n\}, M_C)$ is defined as the n-tuple *presstruct*(q_C):= ($sel_1,\ldots,sel_n$) with

$$\text{sel}_i := \begin{cases} 1 & \text{for } d_i \in \sigma(q) \\ 0 & \text{for } else \end{cases} \quad \text{for } i \in [1;n]$$

The presentation granularity $presgran_q$ of q_C is defined as the vector of granularities of the result dimensions *presgran*(q_C):=($gran_1,\ldots,gran_n$) with

$$\text{gran}_i := \begin{cases} g_i & \text{for } d_i \notin \sigma(q) \\ 0 & \text{for } else \end{cases} \quad \text{for } i \in [1;n]$$

◆

Example 3.18 (MD Presentation Structure and Granularity)

For the query $q_{example}$ (Example 3.16), the presentation structure and granularities are defined as follows:

presstruct($q_{example}$)=(1,1,0,1,1)

presgran($q_{example}$)=(0,0,*geogr. region*,0,0) ◆

For transformations that change the restriction elements of a query or the result granularity of a query, the user interactively has to choose a new restriction element respectively granularity. Following the navigational paradigm of OLAP user interface, the new restriction element or granularity is not arbitrarily chosen from the set of all possible elements or levels, but '*relatively*' to the currently selected element (the relation is defined by the classification instance respectively classification lattice). Depending on the actual transformation, the user can pick the *ancestors* of a classification element, the *descendants* or the *siblings*. The following definitions define these terms for classification instances (see Figure 3.18 for a visualization in the case of a simple hierarchy).

Definition 3.16 (Ancestor and Descendants of Classification Nodes)

The set of *ancestors* for the classification node *m* according to a classification instance I_Ψ of a classification lattice $\Psi|_d = (L_\Psi, class_\Psi)$ is defined as follows:

$$\text{ancestors}_{\Psi|d}(m) := \left\{ a \in dom(l) \;\middle|\; l \in L_\Psi \wedge \text{level}(m) <_\Psi l \wedge \text{group}_{\text{level}(m),l}(m) = a \right\}$$

Analogously, the set of descendants for a classification node *m* is defined as follows:

$$\text{descendants}_{\Psi|d}(m) := \left\{ x \in dom(l) \;\middle|\; l \in L_\Psi \wedge l <_\Psi \text{level}(m) \wedge \text{group}_{l,\text{level}(m)}(x) = m \right\}$$

◆

Some of the transformations require the user to pick a classification element as an argument to the transformation operation. Again, due to the navigational paradigm, not any arbitrary classification node can be picked for such a transformation, but only those classification nodes which are currently being displayed by the MD presentation. This set of nodes is called the set of *active dimension nodes* for each dimension and is defined by the last canonical query as follows (see Figure 3.19 for a visualization):

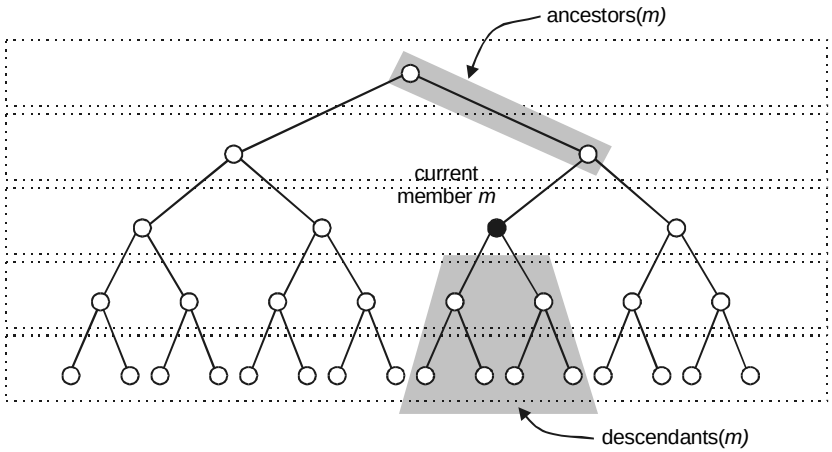

Figure 3.18: The set of Ancestors and Descendants in a Simple Hierarchy

Definition 3.17 (Active Classification Nodes)

For a canonical OLAP query $q_C = \left(M,(r_1,\ldots,r_n),(g_1,\ldots,g_n)\right)$, we define the set of *active classification nodes* $\text{active}_{d_k}(q_C)$ of dimension d_k with respect to query q_C as follows:

$$\text{active}_{d_k}(q_C) := \left\{ m \mid m \in dom(g_k) \wedge group_{g_k,level(r_k)}(m) = r_k \right\}$$

◆

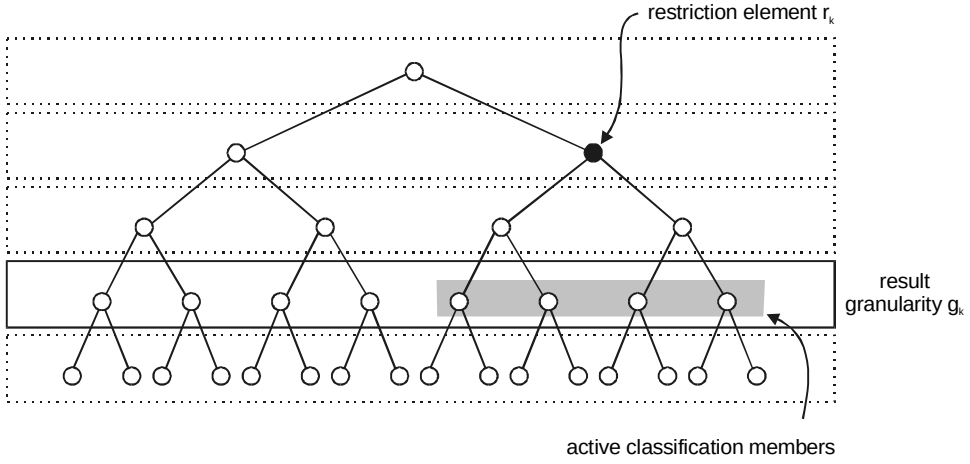

Figure 3.19: Visualizing the Set of Active Classification Nodes Regarding a Query

The following subsections contain the definition of four atomic (*slice*, *rotate-selection*, *rotate-result*, *change-focus)* and two composite query transformations (*drill-down* and *roll-up*). Additionally, we show that the set of atomic transformations is complete in the sense that it can be used to interactively specify every canonical query definition starting from an arbitrary canonical query. We motivate the criteria for our choice of transformations and briefly sketch how the actual user interface operation for each transformation could look like in an actual GUI implementation.

3.2.4.1 Changing the Restriction Criterion: Slice Transformation

The *slice transformation* changes the restriction value for a dimension d_k of a query q. The slice transformation is defined such that the application of this transformation does not change the presentation granularity and structure of q. Following the navigational paradigm for query

formulation, the new restriction element cannot be arbitrarily chosen from the domain of the classification lattice. Instead, the choice (the set of candidate restriction elements) is restricted to the siblings of the current restriction element r_k, the descendants[28] of r_k and the ancestors of r_k.

If the dimension d_k to which a slice transformation is applied is a result dimension (i.e., the restriction level is greater than the result level), the choice of candidate restriction elements must be further restricted to classification nodes which reside on a level of granularity that is larger than the result granularity defined for this dimension. Otherwise, the dimension would either become a selection dimension (in case that the new restriction element is from the result granularity level) thus changing the presentation structure, or violate the condition for well-formed canonical query if the new restriction element would be of a lower granularity than the result granularity.

If dimension d_k is a selection dimension of the query q, the result granularity of this dimension has to be automatically adjusted to the granularity of the new restriction element if this element is from a level which is not equal to the current restriction level. Otherwise, the query would either not be well-formed anymore or become a result dimension, thus violating the prerequisite that the slice operation does not change the presentation structure.

These considerations lead to the following definition of a slice transformation.

Definition 3.18 (Slice Transformation)

Let $q_C = \left(M, (r_1,\ldots,r_n), (g_1,\ldots,g_n)\right)$ be a query over an n-dimensional cube schema $C_\Psi = \left(\{d_1,\ldots,d_n\}, M_C\right)$. Let $r_{new} \in dom(\Psi|_{d_k})$ be a classification node from the classification lattice defined by dimension d_k $(1 \le k \le n)$. The *slice transformation* $\text{slice}_{d_k, r_{new}}(q_C,)$ with respect to dimension d_k and classification node r_{new} is defined as follows:

$$\text{slice}_{d_k, r_{new}}(q_C) := \left(M, (r_1',\ldots,r_n'), (g_1',\ldots,g_n')\right)$$

$$r_i' := \begin{cases} r_{new} & \text{for } i = k \\ r_i & i \neq k \end{cases} \text{ and } g_i' := \begin{cases} level(r_{new}) & for\ i = k \wedge d_k \in \sigma(q_C) \\ g_i & else \end{cases}$$

The new restriction element r_{new} must be a sibling, an ancestor or a descendant of the current restriction element according to one of the a hierarchies defined by the classification lattice of the dimension i.e.,

- $r_{new} \in dom(\text{level}(r_k)) \cup \text{ancestors}(r_k) \cup \text{descendants}(r_k)$ (S1)

Additionally, the following condition has to be fulfilled

- $d_k \notin \sigma(q_C) \Rightarrow level(r_{new}) >_{\Psi|_{d_k}} level(g_k)$ (S2)

◆

[28] Most of the tools (for example Cognos PowerPlay) even restrict the choice further by demanding that the new restriction element is an immediate descendent of the current restriction element, i.e. it is a descendent from the next lower level of granularity (for example the 12 month of 1999 if 1999 is the current restriction element).

Example 3.19 (Slice Transformation)

Let us assume that the last query was the query q_{example} introduced in Example 3.16.

```
q_example := ( {#repairs},
        ( all vehicles, 1999, Germany      , steering, all types),
        ( vehicle.all , year, geogr. region, assembly, type.all) )
```

If the user is interested to investigate the figures of the previous year, he executes a slice transformation on the time dimension. As the new restriction element 1998 is from the same level as the current restriction element 1999, the granularity has not to be adjusted.

```
slice_repair time, 1998(q_example) = ( {#repairs},
        ( all vehicles, 1998, Germany      , steering, all types),
        ( vehicle.all , year, geogr. region, assembly, type.all ) )
```

If the user chooses a new restriction element e.g., May 1999 for the selection dimension time, the result granularity is automatically adjusted to the level month.

```
slice_repair time, may 99(q_example) = ( {#repairs},
        ( all vehicles, may 99, Germany      , steering, all types),
        ( vehicle.all , month , geogr. region, assembly, type.all ) )
```

A slice in a result dimension occurs for example if the user chooses to analyze the geographic regions of the USA.

```
slice_repair location, USA(q_example) = ( {#repairs},
        ( all vehicles, 1999, USA          , steering, all types),
        ( vehicle.all , year, geogr. region, assembly, type.all ) )
```

Albeit, e.g., the classification element ‘Bavaria’ belonging to the classification level geogr. region cannot be used as a parameter to the slice transformation together with query q_{example} because it violates condition (S2).

```
slice_repair location, Bavaria(q_example) = undefined
```

◆

3.2.4.2 Changing the Presentation Structure: Rotate Transformation

The slice transformation presented in the previous section does not change the presentation structure of the query i.e., the partitioning of dimensions into selection and result dimensions. The *rotate transformation*[29] is designed in order to perform this change. For a given dimension d_k, it changes the status from result dimension to selection dimension and vice versa.

If a selection dimension is transformed into a result dimension, the restriction element remains unchanged, but the user has to give a new result granularity[30], which is a level smaller than the restriction level. The opposite rotation (from result dimension to selection dimension) does not need this additional parameter, as the result granularity is automatically set to the restriction level in order to make the dimension a selection dimension. Due to the different signature, the following definition formalizes two transformations *rotate-result* which rotates a result dimension to a selection dimension and the inverse operation *rotate-selection*:

Definition 3.19 (Rotate Query Transformations)

For a canonical query $q_C = (M, R, (g_1, \ldots, g_n))$ against an n-dimensional cube schema $C_\Psi = (\{d_1, \ldots, d_n\}, M_C)$, the rotate query transformations $\text{rotate-result}_{d_k}$ and

[29] This transformation is named rotate, as the hyper plane in the multidimensional data space which is defined by the query, is being rotated.

[30] Commercial tools (e.g.. Cognos PowerPlay, Oracle Express) often automatically use the next lower level as the new result granularity.

$\text{rotate-selection}_{d_k,l}$ for dimension d_k ($k \in [1;n]$) and classification level $l \in \Psi$ are defined as follows:

$$\text{rotate-result}_{d_k}(q_C) := (M, R, (g_1', \ldots, g_n')) \text{ with } g_i' := \begin{cases} \text{level}(r_k) & \text{for } i = k \\ g_i & i \neq k \end{cases}$$

In order to apply the *rotate result* operation, the dimension d_k has to be a result dimension with respect to query q_C i.e., $d_k \notin \sigma(q_C)$.

$$\text{rotate-selection}_{d_k,l}(q_C) := (M, R, (g_1', \ldots, g_n')) \text{ with } g_i' := \begin{cases} l & \text{for } i = k \\ g_i & i \neq k \end{cases}$$

In order to apply the *rotate-selection* operation, the following conditions have to be fulfilled:

- the dimension d_k to which the transformation is applied has to be a selection dimension with respect to query q_C i.e., $d_k \in \sigma(q_C)$. (R1)
- a path from the target level l_t to the current level of restriction g_k must exist in the classification lattice of dimension d_k i.e., $l_t <_{\Psi|d_k} g_k$. (R2)

◆

Example 3.20 (Rotate Query Transformation)

Let us assume that the last query was the query $q_{example}$ introduced given in Example 3.16.

```
q_example := ( {#repairs},
        ( all vehicles, 1999, Germany      , steering, all types),
        ( vehicle.all , year, geogr. region, assembly, type.all) )
```

If the user wants to analyze the development of repair figures regarding the parameters repair location and time of the repair (on the level of month), she might perform the following transformation that makes the time dimension a result dimension.

```
q_rotate :=
rotate-selection_repair time, month(q_example) = ( {#repairs},
        ( all vehicles, 1999 , Germany      , steering, all types),
        ( vehicle.all , month, geogr. region, assembly, type.all ) )
```

If the analysis should be restricted to the variation of repairs over time irrespective of the geographic region, a rotate-result transformation can be applied to the result of the previous transformation:

```
rotate-result_repair location(q_rotate) = ( {#repairs},
        ( all vehicles, 1999 , Germany , steering, all types),
        ( vehicle.all , month, country , assembly, type.all ) )
```

◆

3.2.4.3 Analyzing Different Measures: Focus Change Transformation

All transformations presented so far were designed to manipulate the result granularity and the restriction elements leaving the set of result measures unchanged. The focus change operation completes our set of transformations by changing the set of measures being analyzed in the result. Each application of the *change-focus* transformation adds or removes a measure from the set of result measures. More than one measure can be added/removed by applying a sequence of *change-focus* transformations.

Definition 3.20 (Change Focus Transformation)

For a canonical OLAP query $q_C = (M, R, G)$ against an n-dimensional cube schema $C_\Psi = (D_C, M_C)$, the query transformation $\text{change-focus}_{m_{change}}$ for measure $m_{change} \in M_C$ is defined as follows:

$$\text{change-focus}_{m_{change}}(q_C) := (M', R, G) \text{ with } M' := \begin{cases} M \cup m_{\text{change}} & \text{for } m_{\text{change}} \notin M \\ M - m_{\text{change}} & \text{else} \end{cases}$$

◆

Example 3.21 (Change Focus Transformation)

Let us once again assume that the last query was the query q_{example} introduced given in Example 3.16.

```
q_example = ( {#repairs},
        ( all vehicles, 1999, Germany       , steering, all types),
        ( vehicle.all , year, geogr. region , assembly, type.all ) )
```

If the user decides to additionally analyze the cumulated duration of the repairs, she transforms the query by applying a *change focus* operation to the measure *duration*.

```
change-focus_duration(q_example) = ( {#repairs, duration},
        ( all vehicles, 1999 , Germany , steering, all types),
        ( vehicle.all , month, country , assembly, type.all ) )
```

◆

3.2.4.4 Changing the presentation granularity: Drill-down/Roll-up Transformation

The transformations presented so far are atomic in the sense that none of the transformations can be expressed as a combination of the other transformations. In contrast to that, the two transformations that will be presented in the following section are not atomic as they can be expressed as a combination of rotate and slice transformations using a special set of parameters. However, these transformations are of great importance to OLAP front-ends and are therefore included in our set of transformations. The purpose of these operations is to directly change the presentation granularity of the query (i.e., the granularity of the result dimensions) without changing the presentation structure.

Both the *drill-down transformation* and the *roll up-transformation* can only be applied to result dimensions of the cube. We define two distinct transformations instead of one combined transformation for drilling and rolling in order to reflect the different parameters of these operations (see below).

A *drill down transformation* increases the level of detail for a dimension *k* in a query q. The drill-down is specified by choosing a drill-down element r_{new} (the drilling element) and a target level g_{new}. The drill-down element is picked from the set of active classification nodes of a result dimension[31] (for example, the element '1999'). The target level determines the new granularity for the dimension *k*. The new granularity must be smaller than the current granularity according to the classification schema. This means that a path from g_{new} to g_{k} must exist in the classification lattice of the dimension *k*.

[31] The restriction of the choice of elements only to active elements is a consequence of the interface design decision, that the user picks the next restriction element from the axis labels being currently displayed.

Definition 3.21 (Drill-down Transformation)

For a canonical OLAP query $q_C = \left(M, (r_1,\ldots,r_n), (g_1,\ldots,g_n)\right)$ against an n-dimensional cube schema $C_\Psi = \left(\{d_1,\ldots,d_n\}, M_C\right)$, the drill-down transformation $\text{drill}_{d_k, r_{drill}, g_{new}}$ using the classification lattice $\Psi|_{d_k} = (L_k, class_k)$ of dimension d_k ($k \in [1;n]$) is defined as follows:

$$\text{drill}_{d_k, r_{drill}, g_{new}}(q_C) := \left(M, (r_1',\ldots,r_n'), (g_1',\ldots,g_n')\right) \text{ with } r_{\text{drill}} \in dom(\Psi|_{d_k}) \text{ and } g_{\text{new}} \in L_k$$

$$r_i' := \begin{cases} r_{\text{drill}} & \text{for } i = k \\ r_i & i \neq k \end{cases} \text{ and } g_i' := \begin{cases} g_{\text{new}} & \text{for } i = k \\ g_i & i \neq k \end{cases}, k \in [1;n]$$

The following conditions have to be met:

- $d_k \notin \sigma(q_C)$, that means d_k is a result dimension. (D1)
- $r_{\text{drill}} \in \text{active}_{d_k}(q_C)$ i.e., the drill-down element r_{drill} is from the currently active set of classification nodes (see Definition 3.17). (D2)
- $g_{\text{new}} <_{\Psi|_{d_k}} g_k$ i.e., the target level must be smaller than the current granularity and thus, a path (called drill-down path) must exist in the classification lattice $\Psi|_{d_k}$ from the target level g_{new} to the current granularity g_k. (D3)

◆

Example 3.22 (Drill-down Transformation)

The query introduced in Example 3.16 lists all the geographic regions of Germany (e.g., Bavaria, Hamburg) on the result dimension *repair location*. If the user is interested in the distribution of repairs across the locations in Bavaria, she can apply the following drill-down query transformation to q_{example}:

```
drill repair location, Bavaria location (q_example) = ( {#repairs},
        ( all vehicles, 1999, Bavaria , steering, all types),
        ( vehicle.all , year, location, assembly, type.all ) )
```

◆

The *roll-up transformation* can be seen as the complementary transformation of the drill-down operation. Here, the user decreases the level of detail for a result dimension along a specific classification path. The roll-up transformation is specified by giving a new granularity for a result dimension. The classification lattice of the respective dimension must contain a path from the current granularity to the target level. As long as the new target level is smaller than the level on which the restriction is formulated, the restriction element does not change.

Albeit, if the new result granularity is not smaller than the current restriction level, a new restriction element has to be determined automatically in order to preserve the presentation structure. In this case, the new restriction element must be chosen in a way such that it is a parent of all currently active classification nodes[32]. The rationale behind this requirement is, that if the user performs a roll-up transformation, he expects to see an abstraction of all the elements he currently sees (active classification nodes of the current query). This also ensures that the result of a subsequent drill-down transformation does at least contain the currently visible classification nodes.

[32] Notably such an element always exists, as the 'all' element fulfills this condition for any set of active classification nodes.

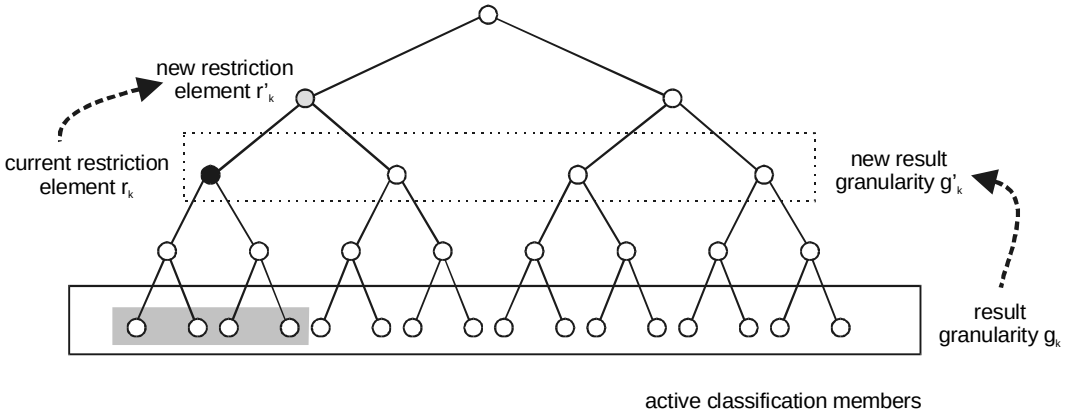

Figure 3.20: Re-computing the Restriction Element after Roll-up Transformation

Additionally, the size of the set of active classification nodes should be kept as minimal as possible in order to be easily handled by the user. Therefore, the new restriction level should be as low as possible with respect to the classification lattice. If the current restriction level, the new result granularity and the old result granularity are all from the same classification schema path *p*, the new restriction element is computed as the parent of the current restriction element on the level which is 'directly above' ($\text{father}_p(g_{\text{new}})$) the new result granularity (see Figure 3.20). As the current restriction element r_k is an ancestor of the currently active classification nodes $m \in \text{active}_{d_k}(q)$ by definition, the new restriction element r'_k being an ancestor of the current restriction element also fulfills this condition.

A special case occurs if the current restriction level and the new granularity level are from different dimension paths. This means that the data in the new query is being classified according to a criterion which is incompatible with the current criteria. In this case, the new restriction element is set to 'all'.

Following these considerations it becomes clear, that the special level <dimension name>.all should not be allowed as a parameter to the roll-up transformation. For a granularity of '*all*' no new restriction element from a higher level can be found and therefore, the respective dimension automatically becomes a selection dimension of the transformed query. As the roll-up transformation should not change the presentation structure, we do not allow the all-level as an argument to the roll-up transformation. However, an aggregation of the data to the all level is easily possible using the rotate operation (see Section 3.2.4.2).

Definition 3.22 (Roll-up Query Transformation)

For a canonical OLAP query $q_C = \left(M, (r_1, \ldots, r_n), (g_1, \ldots, g_n)\right)$ against an n-dimensional cube schema $C_\Psi = \left(\{d_1, \ldots, d_n\}, M_C\right)$, the roll-up query transformation $\text{roll}-\text{up}_{d_k, g_{new}}(q_C)$ using the classification lattice $\Psi|_{d_k} = (L_k, class_k)$ of dimension d_k is defined as follows:

$$\text{roll}-\text{up}_{d_k, g_{new}}(q_C) := \left(M, (r_1', \ldots, r_n'), (g_1', \ldots, g_n')\right)$$

$$g_i' := \begin{cases} g_{new} & \text{for } i = k \\ g_i & i \neq k \end{cases} \text{ and}$$

$$r_i' := \begin{cases} r_i & i \neq k \vee g_{new} <_{\Psi|_{d_k}} level(r_k) \\ \text{group}_{\text{level}(r_k),\text{father}_p(g_{new})}(r_i) & \text{for } i = k \wedge \text{level}(r_k) \leq_{\Psi|_{d_k}} g_{new} \\ d_k.all & \text{else} \end{cases}$$

where $\text{father}_p(g_{new})$ denotes the father of the new granularity level g_{new} according to the classification schema path through the lattice $\Psi|_{d_i}$ that is defined by the restriction level $level(r_i)$ and the result granularity g_i.

The following conditions have to be fulfilled:

- $d_k \notin \sigma(q_C)$ i.e., d_k is a result dimension. (RU1)
- $g_{new} \in L\left(\Psi|_{g_k}\right) - \{d_k.all, g_k\}$ i.e., the new result granularity is a level which is larger than the current granularity according to the classification lattice defined by d_k and the special level d_k.all is not allowed as an argument. (RU2)

◆

Example 3.23 (Roll-up Query Transformation)

Let us assume that the last query executed was the query q_{drill} introduced in Example 3.22.

```
q_drill := ( {#repairs},
        ( all vehicles, 1999, Bavaria , steering, all types),
        ( vehicle.all , year, location, assembly, type.all ) )
```

If the user chooses to change the granularity of the results in the repair location from *location* to *geogr. region*, she performs the following query transformation:

```
roll-up_(repair location, geogr. region)(q_drill) = ( {#repairs},
        ( all vehicles, 1999, Germany      , steering, all types),
        ( vehicle.all , year, geogr. region, assembly, type.all ) )
```

The restriction element of the resulting query is automatically adjusted to the element ‘Germany’ because country is the level directly above geogr. region and ‘Germany’ is the parent of ‘Bavaria’.

If the user chooses instead to perform a roll-up transformation to q_{drill} that changes the granularity in the repair location dimension from *location* to *type of repair unit*, the result looks as follows:

```
roll-up_(repair location, geogr. region)(q_drill) = ( {#repairs},
        ( all vehicles, 1999, locations.all      , steering, all types),
        ( vehicle.all , year, type of repair unit, assembly, type.all ) )
```

The restriction element for the resulting query is being set to *locations.all*, because the levels *location* (old result granularity), *type of repair unit* (new result granularity) and *geogr. region* (old restriction level) are not from the same classification path. ◆

3.2.4.5 Completeness of the Transformations

In order to show that the definition of transformation operations is justified, we show that the set of query transformations is complete w.r.t. the definition of a canonical OLAP query. That means for each pair of canonical OLAP queries (q_1, q_2), it is possible to find a finite sequence of transformation operations $\tau_1,\ldots, \tau_k$ that transforms query q_1 to q_2 i.e., $\tau_1 \circ ,\ldots \circ \tau_k(q_1) = q_2$.

We split the proof into two parts: first we show, that starting from the special query $q_0 = (\varnothing, (\text{all},\ldots,\text{all}), (d_1.\text{all},\ldots,d_n.\text{all}))$ every well-formed query q can be produced by a se-

quence of transformations T_q (Lemma 3.3). The second part shows that starting from any arbitrary query *q*, the default query q_0 can be generated (Lemma 3.4) by a transformation sequence $\overline{T}_q$. This means that for any pair of arbitrary queries q_1 and q_2, the transformation sequence $T_{q_1,q_2} = \overline{T}_{q_1} \circ T_{q_2}$ transforms q_1 to q_2 (Theorem 3.5)[33].

Lemma 3.3 (Generation of Canonical Queries)

For every (well-formed) canonical query $q = ((m_1,\ldots,m_z),(r_1,\ldots,r_n),(g_1,\ldots,g_n))$ there exists a finite sequence of transformation operations T_q such that the result of applying T_q to the query $q_0 = (\varnothing,(\text{all},\ldots,\text{all}),(d_1.\text{all},\ldots,d_n.\text{all}))$ is equal to *q* i.e., $T_q(q_0) = q$. ◆

Proof 3.3 (Generation of Canonical Queries)

We start from the query $q_0 = (\varnothing,(\text{all},\ldots,\text{all}),(d_1.\text{all},\ldots,d_n.\text{all}))$ and want to obtain the well-formed query $q := ((m_1,\ldots,m_z),(r_1,\ldots,r_n),(g_1,\ldots,g_n))$ as the result of a sequence T_q of transformation operations i.e., $T_q(q_0) = q$. In order to prove the existence of T_q, we give an algorithm that constructs such a transformation sequence and prove its correctness.

Let us assume that the set $X_q := \{\chi_1,\ldots,\chi_p\}$ contains the indexes of the result dimensions i.e., $\chi \in X_q :\Leftrightarrow d_\chi \notin \sigma(q)$. The constructed transformation sequence $T_g(q)$ has the following form:

$$T_q := \underbrace{tm_1 \circ \ldots \circ tm_z}_{=TMq} \circ \underbrace{tg_1 \circ \ldots \circ tg_p}_{=TGq} \circ \underbrace{tr_1 \circ \ldots \circ tr_n}_{=TRq}$$

$=TMq$: transformations to adjust M

$=TGq$: transformations to adjust G for result dimensions of q

$=TRq$: transformations to adjust R

with

$$tm_i = \text{change-focus}_{m_i} \text{ for } 1 \le i \le z = |M|,$$

$$tg_j = \text{rotate-selection}_{d_{\chi_j}, g_{\chi_j}} \text{ for } 1 \le j \le p = n - |\sigma(q)|,$$

$$tr_k = \text{slice}_{d_k}(r_k) \text{ for } 1 \le k \le n$$

Simple application of Definition 3.20 shows that the *first part* of the transformation sequence TM_q produces a query q' which contains the required result measures i.e., $TM_q(q_0) = q' = (M,(\text{all},\ldots,\text{all}),(d_1.\text{all},\ldots,d_n.\text{all}))$.

The application of the *second part* of the transformation sequence TG_q to q' produces a query $q'' := TG_q(q')$. TG_q changes the result granularity of query q' such that for all result dimensions of *q*, the result granularity of q'' is equal to the result granularity of *q*. The preconditions to apply the rotate-selection operation (cf. Definition 3.19) are trivially fulfilled, as all the dimensions $d_1,\ldots,d_n$ are all selection dimensions of query q', i.e $d_i \in \sigma(q')$. Applying Definition 3.19, to the query q' results in the following query:

[33] Usually, T is not the shortest sequence satisfying the demanded property.

$$TG_q(q') = q'' = (M, (\text{all},\ldots,\text{all}), (g''_1,\ldots,g''_n)) \text{ with } g''_i = \begin{cases} d_i.\text{all} & \text{for } d_i \in \sigma(q) \\ g_i & d_i \notin \sigma(q) \end{cases},\ 1 \le i \le n \quad (1)$$

From (1), it is obvious that q'' has the same selection (and result dimensions as q) i.e., $\sigma(q) = \sigma(q'')$. (2)

The *third part* of the transformation sequence TR_q adjusts the restriction elements for all dimensions by applying according slice operations to all dimensions. Due to the definition of the slice operation (cf. Definition 3.18), it automatically adjusts the result granularity for selection dimensions of q. First, we have to check if the preconditions for the application of the *slice* operations (S1 and S2 in Definition 3.18) are fulfilled. (S1), demanding that the new restriction element is an ancestor, descendant or sibling of the current restriction element is fulfilled because every node of the classification lattice is by definition an ancestor of the *all* element (which is the current restriction element). (S2) demands that for all result dimensions of the query to which transformations are applied (q'' in this case), the level of the new restriction element must be greater than the result granularity for this dimension. Thus, we have to prove that $d_i \notin \sigma(q'') \Rightarrow level(r_i) < g''_i$.

Because of (1) and (2) the following implication holds for every $i \in [1,n]$:

$$d_i \notin \sigma(q'') \overset{(2)}{\Rightarrow} d_i \notin \sigma(q) \overset{(1)}{\Rightarrow} g_i = g''_i \quad (3)$$

Additionally, because q is well-formed (cf. Definition 3.12) and $d_i \notin \sigma(q'')$, the following condition holds: $level(r_i) < g_i$. Therefore $d_i \notin \sigma(q'') \Rightarrow level(r_i) < g''_i$ (P2) is fulfilled.

The application of Definition 3.18 to the query q'' yields the following result (query q'''):

$$TR_q(q'') = q''' = (M, (r_1,\ldots,r_i), (g'''_1,\ldots,g'''_n)) \text{ with } g'''_i = \begin{cases} level(r_i) & \text{for } d_i \in \sigma(q'') \\ g''_i & d_i \notin \sigma(q'') \end{cases} \quad (4)$$

To prove the correctness of T_q we have to show that $q''' = T_q(q_0)$ is equal to q. From (4) we can directly see that the set of result measures and the restriction elements of q and q''' are equal. The equality of the result granularity vector is a direct result of applying (2) and (3) to (4):

$g'''_i = \begin{cases} level(r_i) & \text{for } d_i \in \sigma(q) \\ g_i & d_i \notin \sigma(q) \end{cases}$. Therefore, $T_q(q_0) = q$ q.e.d

◆

Lemma 3.4 (Reduction of any Canonical Query)

For every (well-formed) canonical query $q = ((m_1,\ldots,m_z),(r_1,\ldots,r_n),(g_1,\ldots,g_n))$ there exists a finite sequence of transformation operations $\overline{T}_q$ such that the application of $\overline{T}_q$ to q produces the query $q_0 = (\varnothing,(\text{all},\ldots,\text{all}),(d_1.\text{all},\ldots,d_n.\text{all}))$ i.e., $\overline{T}_q(q) = q_0$. ◆

Proof 3.4 (Reduction of any Canonical Query)

The proof of this Lemma follows the same structure like Proof 3.3 i.e., in order to prove the existence of $\overline{T}_q$, we give an algorithm that constructs such a transformation sequence and prove its correctness.

Let us assume that the list $X_q=(\chi_1,\ldots,\chi_p)$ contains the indexes of the result dimensions i.e., $\chi \in X_q :\Leftrightarrow d_\chi \notin \sigma(q)$. The constructed transformation sequence $T_g(q)$ has the following form:

$$\overline{T}_q := \underbrace{tm_1 \circ \ldots \circ tm_z}_{=TMq} \circ \underbrace{tr_1 \circ \ldots \circ tr_n}_{=TRq} \circ \underbrace{tg_1 \circ \ldots \circ tg_p}_{=TGq}$$

TMq: transformations to adjust M

TRq: transformations to adjust R

TGq: transformations to adjust G for the selection dimensions of q

with

$$tm_i = \text{change\_focus}_{m_i} \text{ for } 1 \le i \le z = |M|,$$

$$tr_k = \text{slice}_{d_k, d_k.\text{all}} \text{ for } 1 \le k \le n,$$

$$tg_j = \text{rotate\_result}_{\text{d}_{\chi_j}} \text{ for } 1 \le j \le p = n - |\sigma(q)|$$

The proof follows the same schema as the Proof 3.3. For each of the three parts of the transformation sequence, it has to be shown that the preconditions for applying the transformations are met and then the result of the transformation can be easily obtained from the definition of the appropriate transformations. For reasons of brevity, we omit the full technical proof. The first part of the transformation sequence TM_q removes all the result measures from q. The purpose of the second part of the transformation sequence TR_q is to set the selection element to the *all* element for all dimensions. Due to the semantics of the rotation transformation, the result granularity for all selection levels is automatically adjusted to the *all*-level. The remaining adjustment of the result granularity for the result dimensions of q is performed by TG_q, the last part of the sequence. ◆

Theorem 3.5 (Completeness of Query Transformations)

For every pair (q_1,q_2) of arbitrary well-formed canonical queries exists a finite sequence of transformation operations T_{q_1,q_2} such that $T_{q_1,q_2}(q_1) = q_2$

Proof 3.5 (Completeness of Query Transformations)

A transformation sequence $\overline{T}_{q_1}$ with $\overline{T}_{q_1}(q_1) = q_0$ exists according to Lemma 3.3. Analogously a transformation sequence T_{q_2} with $T_{q_2}(q_0) = q_2$ exists according to Lemma 3.4. This implies that the transformation sequence $T_{q_1,q_2} := \overline{T}q_1 \circ Tq_2$ exists and has the following property $T_{q_1,q_2}(q_1) = T_{q_2}\left(\overline{T}q_1(q_1)\right) = T_{q_2}(q_0) = q_2$ q.e.d ◆

Notably, the above proofs did not require the *drill-down* and *roll-up* transformations to construct the transformation sequence. This shows that these transformations are not necessary to ensure the completeness of the set of transformations. This is a consequence of these transformations are not being atomic in the sense that they can both be expressed as a combination of the rotate and slice operations:

$$\text{drill}_{d,r,g} = \text{rotate}-\text{result}_d \circ \text{slice}_{d,r} \circ \text{rotate}-\text{selection}_{d,g}$$

$$\text{roll}_{d,g} = \text{rotate}-\text{result}_d \circ \text{slice}_{d,r} \circ \text{rotate}-\text{selection}_{d,g}$$

(r is automatically computed, cf. Definition 3.22)

However, these transformations have stricter preconditions than the combination of the preconditions of the rotate and slice transformations (for example, the new restriction element for the drill-down operation must be from the active set of classification nodes, cf. Definition 3.21). We chose to include these transformations in our set of basic transformations, as these operations play a dominant role in present user interfaces and in the discussion of OLAP functionality throughout the research literature.

3.2.5 Considerations about the Expressiveness

When designing the query model we had the goal to be expressive enough to capture all the queries that can be formulated using the abstract graphical user interface (MD presentation) described in Section 3.2.2. On the other hand, the model should be as compact as possible. Therefore, our definition of a canonical OLAP query (Definition 3.12) contains the following implicit assumptions, which will be discussed according to their impact on the expressiveness of our formalism:

- *Single Cube Instance*. We assume that a query is always formulated with respect to one multidimensional cube instance. Especially, a query does not refer to several cube instances with different schemata. However, keeping in mind that our goal is not an easily executable definition but a conceptual description of the user's interests, this is not a critical restriction: queries that require the combination (for example, join) of different cube instances can be modeled by assuming a single cube which contains the requested combination of dimensions and measures (for example, being implemented using a view mechanism). The complex query can then be rewritten to a canonical OLAP query referring to this cube.
- *Restriction Exclusively via Classification*. All restriction predicates are formulated with respect to the classification schema. I.e., we assume that the predicates used to restrict the different dimensions only contain conditions on classification levels (for example, year = '1998'). However, this does not really restrict the expressiveness of our approach, as we can assume that all the attributes used for restriction are modeled as classification levels in alternative hierarchies. E.g., if the user wants to restrict the time dimension to all 'mondays', she can define a hierarchy on days which contains the necessary groupings of days according to weekdays.
- *Default Aggregation Functions.* The aggregation of a cube to higher granularity according to one dimension involves two steps: the grouping of elements of this dimension according to the target granularity (which is implicitly defined by the classification instance) and the computation of the aggregated value using the multi-set of values produced by the grouping step. The grouping step is defined by the grouping functions contained in the classification schema instance.
- *No cross measure calculations.* The only calculation concerning measures that is supported by our query model is the aggregation (see previous remark) of measures to a higher level of classification. More complex calculations that e.g., use different measure values to derive a new measure value (for example, multiplying *repair duration* with *Nr of persons* to get the total amount of work for a repair) can be modeled as canonical queries against a cube schema which contains the calculated value as a measure. Therefore, this restriction does not critically influence the expressiveness of our approach.

In summary, none of the assumed simplifications constitutes any severe restrictions concerning the expressiveness of our approach. The class of canonical queries modeled by the PROMISE/OLAP approach is at least as expressive as the class of queries considered by the OLAP caching approaches ([ABD+99] and [GCB+97]; see Section 5.1 for more details about

the caching approaches). It also corresponds to the class of queries that can be generated by commercially available tools (e.g., Cognos PowerPlay [Cog99], Oracle Express [Ora98] and Informix MetaCube [Inf98]).

Having discussed the assumptions and restrictions of the design of canonical queries, the following sections practically illustrates the expressiveness of the resulting class of canonical queries. To this end, we give SQL and MDX templates that can be used to implement canonical queries in OLAP databases. Assuming a star-schema representation of the multidimensional model using the relational view proposed in 3.1.4, a canonical OLAP query corresponds the SQL-Query template (called "star-join template") shown in Figure 3.21.

```
SELECT g1,...,gn, aggr(m1),..., aggr(mk)
FROM FactName, Dim1,..., Dimn
WHERE    level(r1) = r1 AND ... AND level(rn) = rn
 AND Dim1.d1=FactName.d1 AND ... AND Dimn.dn=FactName.dn
GROUP BY g1,...,gn
```

Figure 3.21: The SQL Template Corresponding to a Canonical OLAP Query

Obviously, this query template is not optimized as some of the grouping attributes $g_1,...,g_n$ are already being restricted to a single element by the conditions of the WHERE clause (these are the selection dimensions of the query).

The design of the MDX query language([Mic98]) does not distinguish between the specification of the data that is to be retrieved and the presentation of the data. Instead, an MDX query also contains a specification for the layout of the result data (for example, if a result dimension is used to label columns or rows). Our definition of a canonical OLAP query consciously does not contain any information about the layout of the data as this should not be relevant to the query processing in the database. Consequently, for defining the MDX template (see Figure 3.22), we have to assume a standard layout which contains the requested measures as columns and the cross join of all result dimension members as rows. Additionally, we assume that the different measures are modeled as members of a special dimension with the named *Measures* (this conforms to the recommendations in [Mic98]).

```
SELECT
{[Measures].[m1], [Measures].[m2],..., [Measures].[mk]} on columns
    { CROSSJOIN(
            descendants([d1].[r1],[d1].[g1])},
            descendants([d2].[r2],[d2].[g2])},
            ...
            descendants([dn].[rn],[dn].[gn])},
      ) }
      on rows,
from [Cube Name]
```

Figure 3.22: A Possible Translation of a Canonical OLAP Query to MDX

Having defined and discussed the class of queries and the transformation operations that can be performed by the user, we can now define the OLAP specific user interaction model for the PROMISE/OLAP framework.

3.3 The PROMISE/OLAP Interaction Model for OLAP Applications

This section constitutes a condensed summary of the observations presented throughout this chapter by defining a user interaction model for PROMISE/OLAP. As described in Definition 2.1, the User Interaction Model describes the simple and composite interaction event types and a formal representation of the information characterizing these types on the detailed level.

The user's interactions with the OLAP front-end tool can be modeled using two dual approaches. The first view is to describe the interaction as a sequence of canonical queries which comes very close to the perception of a session by the OLAP database system (*query based view*). In this case, an atomic interaction event is the execution of a canonical OLAP query. A session is represented as a sequence of canonical OLAP queries. The second orthogonal possibility is to describe the atomic interactions as query transformations as defined in Section 3.2.4 (*transformation based view*). A composite interaction event (session) would then be described by an event that causes the execution of a canonical query and a sequence of transformation operations transforming this query. Depending on the type of transformation execution model (single vs. multiple), the model also contains atomic events that indicate the execution of the current query by the database system (event type *update data*). Figure 3.23 visualizes the two dual ways of representing information about OLAP sessions.[34]

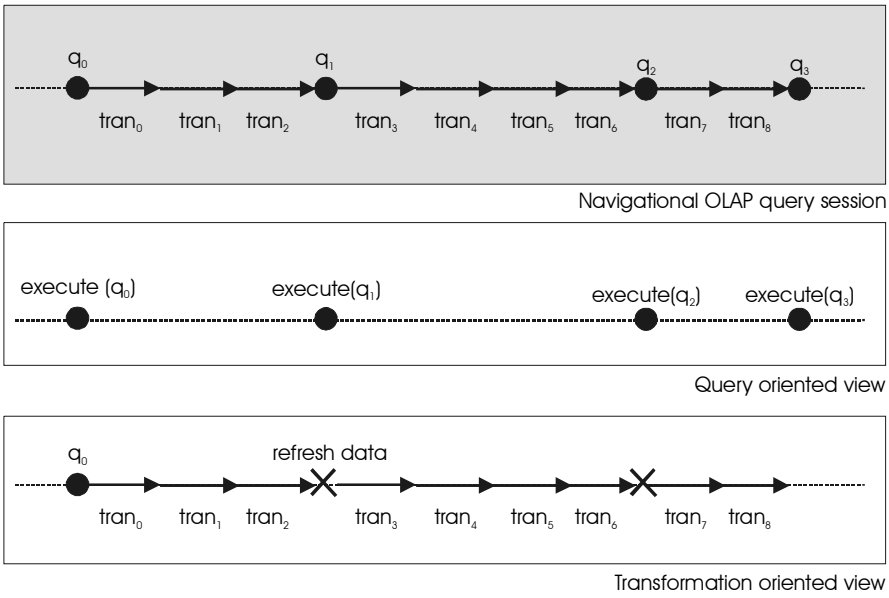

Figure 3.23: Two Dual Ways to Represent Information about User Interactions with an OLAP System

The transformation based view is a specialization of the query based view in the sense that the query based representation can be uniquely derived from the transformation based representation. For a sequence of transformations between two *refresh data* events, it is possible to compute the canonical query after each transformation using the definition of the transformation operations given in Section 3.2.4. The other way around, it is not possible to reconstruct the transformation based sequence from the query based sequence, as generally an infinite number of transformation sequences exist that transform a query q_1 to a query q_2. However, it is always possible to find a semantically equivalent transformation based sequence for each query based sequence. Starting from a sequence of canonical queries, it is possible to find a

[34] Notably, this dual view is not a unique characteristic of the OLAP domain, but is common to all application domains, where the result of a navigation step can be uniquely identified. E.g., taking the WWW interaction presented in Chapter 2, it would be possible to either record the *follow_link* events (like described in Example 2.1) or the pages visited (identified by the respective URLs).

sequence of transformations for each pair of consecutive queries (see Theorem 3.5). The concatenation of these sequences (with the corresponding refresh data events) yields a corresponding transformation-based session.

Generally speaking, the query based view on sessions is more suited to model the behavior of systems following a multi transformation execution model while the transformation based view is better suited to a single transformation execution model. However, a deeper discussion of the usability of the approaches can be found in the next chapter.

Both models are suitable as a basis for defining an interaction pattern model for OLAP systems. Moreover, as different types of generalization (abstraction) are possible for the different models, both models lead to patterns that are not only syntactically different but also semantically different. For example, a transformation based model can yield patterns containing information about which types of transformations are often executed close to each other in OLAP sessions, like that a *roll-up* transformation is very likely to occur after three consecutive *drill-down* transformations. Such a pattern cannot be directly induced from a description of the interactions formulated according to the query based view. A more detailed discussion under which circumstance the dual models should be used will be given in Chapter 4, which develops the pattern model for PROMISE/OLAP. The following two sections formally define both views of OLAP interactions starting with the query based view in Section 3.3.1.

3.3.1 The Query Based OLAP Interaction Model

As elaborated in the previous section, the central entity of description for the query based OLAP interaction model (QBIM) is a canonical OLAP query as described in Definition 3.12. The canonical OLAP query is an attribute of events of the type *execute-query*.

In order to explicitly represent the information when a session was started (for example by logging into the system or by opening a briefing book), the model also contains the event types start-session and end-session. As already discussed in Section 3.2.5, we assume that all queries of a session are executed using the same data cube instance. Therefore, it is sufficient to record the *cube instance name* and the *user_id* information as an attribute of the *start session* event. For every event irrespective of the type, we additionally record the attribute *execution time*. Further attributes could be added to this list, but these attributes are sufficient for the purpose of improving the caching strategies of OLAP systems that we have in mind. These considerations lead to the following definition of the Query Based OLAP User Interaction Model:

Definition 3.23 (Query Based OLAP User Interaction Model)

In accordance with the general definition of a user interaction model (see Definition 2.1), the query based OLAP User Interaction Model IM_{OLAP} is defined as follows:

$IM_{\mathrm{OLAP}} := (T_{\mathrm{OLAP}}, A_{\mathrm{OLAP}}, attr_{\mathrm{OLAP}}, I_{\mathrm{OLAP}})$ with

- T_{OLAP} = {*start-session, end-session, execute-query*}
- A_{OLAP} = {*user-ID, timestamp, query, cube instance*}, where
  - *user-ID* is a system specific way of identifying the user (for example a login name) which is unique for every user.
  - *timestamp* is a time value that indicates the time when the event was initiated. It can be expressed with respect to any appropriate unique timescale (e.g system time, time elapsed since system startup etc.). This attribute can be used to uniquely identify an event.

- *cube instance* is the name of the cube instance that is queried during a session.
- *query* is an attribute that contains a specification of a canonical OLAP query as presented in Definition 3.12.

- $attr_{OLAP} : T_{OLAP} \rightarrow 2^{A_{OLAP}}$ is a function mapping event types to their attributes:

  $attr_{OLAP}$(*start-session*)={*user_ID, timestamp, cube instance*}

  $attr_{OLAP}$(*end-session*)={*timestamp*}

  $attr_{OLAP}$(*execute query*)={*timestamp, query*}

- I_{OLAP} contains the integrity constraints that have to be fulfilled by any event and session conforming to the interaction model[35]. This includes the following conditions:
  - the first event of a session is the *start-session* event, and no other *start-session* event occurs in a session, i.e
    - $\forall S =< s_1,\ldots,s_m >, i \in [2;m] : type(s_1) = \text{start} - \text{session} \wedge type(s_i) \neq \text{start} - \text{session}$
  - the last event of a session is the *end-session* event and no other *end-session* event occurs in a session
    - $\forall S =< s_1,\ldots,s_m >, i \in [1;m-1] : type(s_m) = end - \text{session} \wedge type(s_i) \neq end - \text{session}$
  - each canonical query is well-formed (cf. Definition 3.12) with respect to the schema C_Ψ of the cube instance and the classification lattice Ψ of the multidimensional database. ◆

The following example demonstrates how a sample session is modeled using this interaction model:

Example 3.24 (Events in the Query Based OLAP User Interaction Model)

Let us assume, that *Repairs* is a cube instance of the *n*-dimensional cube schema C_{repair} (introduced in Example 3.14). The following session $<e_1,e_2,e_3,e_4,e_5>$ consisting of three queries against the OLAP server is an example for a composite event conforming to the query based interaction model IM_{OLAP}. The events e_1 through e_5 are examples for atomic event in this model.

```
e1 =  (Start_Session, 6/27/2000 10:39.21, (JohnDoe, Repairs) )
e2 =  (execute query, (6/27/2000 10:41.25,
        ( {# of repairs},
      ( all vehicles, all dates, Germany      , steering, all types ),
      ( vehicle.all , year     , geogr. region, assembly, type.all  ))))
e3 =  (execute query, (6/27/2000 10:44.41,
        ( {# of repairs},
      ( all vehicles , 1999 , Bavaria      , all parts, all types ),
      ( vehicle.all  , month, geogr. region, assembly , type.all  ))))
e4 =  (execute query, (6/27/2000 10:52.03,
        ( {# of repairs},
      ( all vehicles , 1998, Bavaria      , all parts, all types ),
      ( vehicle.all  , month, geogr. region, assembly , type.all  ) ) )
e5 =  (End_Session, (6/27/2000 10:52.03) )
```

◆

3.3.2 The Transformation Based OLAP Interaction Model

The transformation based interaction model (*TBIM*) can be formulated as an extension of the query based model described in the previous section, as the *TBIM* requires the same event

[35] See Section 2.3 for details about the notation of sessions.

types, attributes and integrity constraints like the *QBIM*. The *TBIM* contains additional information about the transformation operations, that we performed in order to produce the next query, this makes additional event types (transform query and update data) necessary. The *transform query* event additionally needs an attribute that describes the transformation type and its parameters. This leads to the following definition:

Definition 3.24 (Transformation Based OLAP User Interaction Model)

Extending the query based OLAP user interaction model (Definition 3.23), the transformation based OLAP User Interaction Model $IM_{\text{OLAP/T}}$ is defined as follows:

$IM_{\text{OLAP/T}} := (T_{\text{OLAP/T}}, A_{\text{OLAP/T}}, attr_{\text{OLAP/T}}, I_{\text{OLAP/T}})$ where

- $T_{\text{OLAP/T}} = T_{\text{OLAP}} \cup \{$*transform query*, *update data*$\}$
- $A_{\text{OLAP/T}} = A_{\text{OLAP}} \cup \{$ *transformation*$\}$, where
  - *transformation* is an attribute that contains the type of the query transformation and the parameters of a query transformation.
- $attr_{\text{OLAP/T}} : T_{\text{OLAP/T}} \rightarrow 2^{A_{\text{OLAP/T}}}$ with $attr_{\text{OLAP/T}}(t) = attr_{\text{OLAP}}(t) \quad \forall t \in T_{\text{OLAP}}$ and:

  $attr_{\text{OLAP/T}}$(*update data*)={*timestamp*}

  $attr_{\text{OLAP/T}}$(*query transformation*)={*timestamp*, *query transformation*}
- The set of integrity constraints $I_{\text{OLAP/T}}$ for the transformation based model is a superset of the set of integrity constraints for the query based model $I_{\text{OLAP.}}$ Thus $I_{\text{OLAP}} \subset I_{\text{OLAP/T}}$. Additionally, the following integrity constraints have to be fulfilled:

For every query transformation event *e*, the preconditions detailed in Definition 3.18-Definition 3.22 have to fulfilled by the *input query* of this event i.e., the query to which the transformation has to be applied. The *input query* is determined as follows: Let us regard a session $<e_1,\ldots,e_m>$ and an event e_k ($k \in \{1,\ldots,m\}$) of type *transform query*. The input query of event e_k is defined as the application of all query transformation events that occurred since the last *execute query* event. Formally, the last execute query event before e_k is $e_{i,}$ where

$$i \in \{1,\ldots,k\} \wedge \text{type}(e_i) = \text{execute query} \wedge \neg\exists l \in \{i+1,\ldots,k-1\} : \text{type}(e_l) = \text{execute query}$$

Let $T=<t_1,\ldots t_o>$ be the sequence of indexes of events that lie between e_i and e_k and are of type *transform query* i.e., $T=\{t \mid t \in \{i+1,\ldots,k-1\} \wedge \text{type}(e_t) =$ *transform query*$\}$. The input query *q* for the event e_k is then defined as

$$q = e_{t_1}.\text{transformation} \circ \ldots \circ e_{t_o}.\text{transformation}(e_i.\text{query})$$

- For single transformation execution systems, each *transform query* event must be immediately followed by an event of type *update data*.

◆

Example 3.25 (Events in the Transformation Based OLAP User Interaction Model)

A possible description of the example session presented in Example 3.24 using the transformation based model looks like this:

```
e1 =  (Start_Session, 6/27/2000 10:39.21, (JohnDoe, Repairs) )
e2 =  (execute query, (6/27/2000 10:41.25,
        ( {# of repairs},
      ( all vehicles, all dates, Germany      , steering, all types ),
      ( vehicle.all , year     , geogr. region, assembly, type.all  ))))
e3 =  (transform query, (6/27/2000 10:44.41,
      (drill, repair time, 1999, month) )
e4 =  (update data, (6/27/2000 10:44.41)
e5 =  (transform query, (6/27/2000 10:44.41,
      (slice , repair time, 1998) )
e6 =  (update data, (6/27/2000 10:44.41)
e7 =  (End_Session, (6/27/2000 10:52.03) )
```

◆

3.4 Summary and Conclusions

This chapter established a solid formal basis for the discussion of OLAP systems throughout this thesis. This was achieved by formally describing

- the multidimensional data model which is central to OLAP systems (Section 3.1),
- a query formalism for multidimensional data (canonical queries, Section 3.2.3) and
- the user's interactions with an abstract OLAP front-end tool (query transformations, Section 3.2.4).

These results were prerequisites for the definition of a comprehensive user interaction model (Section 3.3) which constitutes the fundamental formalism for the thesis's core.

The design of the PROMISE/OLAP MD data model was the result of carefully discussing and evaluating recent research efforts in this dynamic area of research. Due to the large diversity of proposed multidimensional data models, we chose to follow a 'best-of-breed' approach and defined a comprehensive data model adapting the concepts from different proposed data models that are most suited for PROMISE/OLAP. The resulting data model serves as a formal basis for the rest of this thesis. Nevertheless, this contribution is also valuable beyond the scope of this thesis as it defines a comprehensive native multidimensional data model, which incorporates a powerful classification model while preserving the mathematical definition of a dimension. Another distinctive feature of our model is the explicit modeling of interrelationships between different data cubes in a database schema.

A survey of the most important MD query specification techniques from science and practice showed that the scientific approaches so far are mainly oriented towards defining a basis for query optimization at a system level. In particular, the approaches lack to address that queries are formulated using a graphical user interface in an interactive way. Therefore, none of the formalisms turned out to be well suited for PROMISE/OLAP. Consequently, Section 3.2.3 presented the descriptive definition of a canonical OLAP query. An analysis of different user interfaces of commercial OLAP products revealed that the structure of the user interface and the MD presentation vary between different tools and vendors. However, the basic underlying functionality for interactive query definitions are equivalent. This leads to the definition of a set of query transformations (Section 3.2.4) which are suited to describe sessions of iteratively specified OLAP queries (composite interaction events according to the abstract framework presented in Section 2.3). This model hides the actual user interface implementation, and therefore constitutes a high level description of the interactive functionality of an OLAP tool that is relevant to the underlying OLAP database. As such, this model is an important contribution not only in the context of this thesis.

Finally, the formulation of two OLAP user interaction models in Section 3.3 forms the quintessence of this chapter. Both OLAP user interaction models have been formalized in conformance with the abstract user interaction event model of the PROMISE/OLAP framework specified in Section 2.3. One model is designed based on the canonical query description and models a session as the sequence of queries executed against the database system (*query based interaction model* - QBIM). A refinement of this model, the transaction based interaction (TBIM) model describes sessions as a sequence of query transformations. Therefore, the TBIM is characterized by a greater level of detail. Both models will be used in this thesis to describe user interactions (as a foundation for the pattern model in chapter 4 and in order to describe a metric space for queries in chapter 5). However, the description of OLAP interactions as sequences of events enables the application of a large set of sequence analysis techniques to OLAP sessions.

The next step according to the PROMISE/OLAP roadmap (cf. Figure 2.7) is the definition of the pattern model (pattern representation language, generalization functions and pattern schemata). The next chapter will be devoted to the development of this model and the interrelated issue of how to design the prediction process. In this context it will rely heavily on the user interaction model that was developed in this chapter.

«It is the theory that decides
what can be observed. »
-- Albert Einstein

Chapter 4 Pattern-based Prediction of OLAP Query Behavior

The previous chapter prepared the foundations for the PROMISE/OLAP framework instantiation by defining the user interaction model for OLAP systems i.e., formally describing the user's interactions on the most detailed level. This description is suited to describe recent user interactions (interaction logs), the current session context (interactions the user performed during the current session so far) and the results of the prediction algorithms (which interactions are expected next). What is still missing in order to describe the prediction algorithm is an appropriate formalism to model patterns in OLAP user behavior. Therefore, this chapter is devoted to the presentation of the core of the PROMISE/OLAP framework: the pattern model and the prediction algorithm.

Predicting user behavior in OLAP environments is a unique approach. Therefore in order to motivate our approach, we first briefly survey the most important approaches to modeling and predicting navigational user behavior that have been developed in other application domains (Section 4.1). Subsequently, we will present our solution for the OLAP specific pattern interaction model in Section 4.2. Being a central component of our pattern model, this also includes the discussion of feasible event generalizations. Based on this model, Section 4.3 presents a corresponding prediction algorithm and discusses issues of implementing this algorithm. Section 4.4 discusses how patterns conforming to the defined pattern schemata can be automatically induced from user interactions observed in the past, thus completing the PROMISE/OLAP framework. An exemplary application of the prediction framework to OLAP caching strategies can be found in Chapter 5 and a practical evaluation of the algorithm and its applications will be presented in Chapter 6.

4.1 Approaches to Model and Predict Navigational Access

The idea of applying prediction techniques to the query behavior of OLAP system users is a novel contribution of this thesis. However, prediction of future data accesses has already been proposed and studied in a variety of different environments that all share the characteristic of navigational data access patterns. The following list gives an overview of the most important domains where prediction has been successfully applied so far.

- *Pipelined Microprocessors*. The performance of microprocessors with deep processing pipelines can be increased by speculative execution. This technique requires a prediction

of branch targets based on the typical behavior of a program that was observed in the past (for example caused by loop-like control structures).

- *Operating Systems*. The file access of programs (for example compilers) exhibits certain regularities leading to a navigational data access. Knowledge about these patterns can be exploited by the operation system e.g., in order to do prefetching of files into the operating system buffer ([GA94], [GA96], [KL96]).
- *Database Buffer Management*. Database management systems transport data from secondary storage to the main memory using a certain granularity (for example a disk page). As data accesses caused by single queries, or by a set of queries typically exhibit certain localities (for example with respect to the current buffer content), the system performance can be increased by caching pages that have a high probability of being referenced in the near future. The probability is typically heuristically approximated based on measures collected in the recent past (for example reference frequency counts) like in virtual memory systems. However, it has been shown (for example [CKV93]) that the cache miss rate can be reduced by predicting page accesses and prefetching these pages into the cache.
- *Object Database Caches*. The object oriented data model explicitly models relationships between objects. These relationships are naturally deployed to access the objects in a navigational fashion. Anticipation of navigation steps based on typical properties of these accesses can enable prefetching techniques for object caches ([PZ91]).
- *Distributed Hypertext Applications (for example WWW)*. Hypertexts are inherently designed to be accessed using navigational techniques (following hyperlinks). Being interactive applications they have a potential of benefiting from anticipating user behavior. Anticipation of user behavior can be used to improve proxy cache design and to presend documents to the client ([Bes96],[BC96], [PM96], [FCL+99], [ZAN99]).
- *Digital Libraries*. Large collections of digital documents typically make use of tertiary storage systems for cost reasons. These media are characterized by a high latency which can be masked from the user by using predictive migration of documents to secondary storage based on probability models for co-accessing certain documents ([KW98]).

Although these approaches are from different domains, they all have in common that they dynamically build a prediction model from the perceived sequences of data accesses which are then used to predict future accesses. The main difference between the different approaches are the entities of prediction (i.e., which events are observed to induce patterns), the underlying formalism (pattern model) used to represent the observed patterns, and the prediction technique used.

Table 4.1 contains a comparison of the most prominent approaches according to these criteria:

- *Entity of Prediction*: Depending on the area of application, the approaches observe data access events at different levels of granularity (e.g., web documents, files, database pages). The common characteristic of these entities is that their number is limited within the application domain of the prediction algorithm (e.g., the number of pages in a database or the number of web pages maintained by a web server) and that the content of the objects are disjoint in the sense that typically an access to a specific entity (for example a web page) cannot be satisfied using a distinct object (for example another web page).
- *Pattern Representation*: Almost all of the approaches are based on a variation of Markov models. This basic formalism (described in greater detail in Section 4.2.2.1) builds a data structure that can be used to determine the probability of the next data access based on a lookback window of the last $m>0$ accesses observed.

The different approaches vary the size of the lookback window, some deploy a combination of several models with different lookback window sizes (also called *prefix depth*). A very interesting approach ([CKV93]) is to adapt techniques from the area of data compression (which are also based on Markov model formalisms) for prediction. The rationale behind this approach is that data compressors typically operate by postulating (either implicitly or explicitly) a dynamic probability distribution on the data to be compressed. Data expected with high probability are encoded with few bits, and unexpected data with many bits. Thus, if a data compressor successfully compresses the data, then its probability distribution on the data must be realistic and can be used for effective prediction.

Other approaches adapt the Markov model formalism by changing the look-ahead window size (also called *search depth*) to a number larger than one (for example [Bes96]) or by introducing timing information ([KW98]).

[PZ91] is the only approach that does not use Markov models as the underlying formalism, but instead deploys neural net techniques (nearest neighbor associative memory) in order to build a prediction model. However, this approach suffers from the common drawbacks of neural net approaches (for example that it is not possible to assess what characteristics have been learned by the algorithm).

- *Prediction Technique.* All algorithms deploy a data structure that stores a probability for possible predictions based on the last observed data accesses. However, the approaches vary in how they pick the candidates that should be considered by the prediction application (for example the prefetching process). The simplest technique is to choose the $k>0$ most probable candidates. However, some techniques use a threshold technique, considering all candidates that have a probability above a certain threshold as valid predictions. Consequently, the result of the prediction is a set of variable size (possibly empty). Approaches that apply different separate data structures (for example Markov models of different order) for predictions apply decision mechanisms choosing results from possibly contradicting candidates produced by the different models.

Approach	Application Domain	Entity of Prediction	Pattern Representation	Prediction Technique
[Bes96], [BC96]	WWW	Web Document access	Dependency graph (1-st order Markov model)	Threshold
[CKV93]	Buffer Management	Database Page access	Level-Ziv encoder (LZ) set of Markov models with different order (PPM), 1-st order Markov model (FOM)	Top k probabilies (giving preference to results of higher order predictions)
[FCL+99]	WWW	Web Document access	set of Markov models with different order (PPM)	Threshold
[GA94], [GA96]	Operating Systems	File access	Dependency graph (1-st order Markov model)	All (ordered by average link distance)
[KL96]	Operating Systems	File access	Set of Markov models with different order (PPM)	Top k Probabilities
[KW98]	Digital Libraries	Digital Document	Time aware Markov Model	Dynamic Threshold
[PM96]	WWW	Web document access	Dependency graph (1-st order Markov model)	Threshold
[PZ91]	OODB	Object access	Neural Net (Associative Memory)	Top probability
[ZAN99]	WWW	web document access	Different Markov Models	Top Probability

Table 4.1: An overview of different applications of prediction techniques

Comparing these application areas to OLAP systems, it is obvious that both share the characteristic of navigational data access patterns. Nevertheless, in contrast to file systems the access patterns of OLAP systems are mainly generated by users rather than programs. OODBMs, Hypertext applications and OLAP systems all posses a data model which is directly being used for navigational access (relationships, links and dimension hierarchies). If these approaches would be transferred to the OLAP domain, the entity of prediction would be a user interaction which corresponds to a multidimensional query. This shows two fundamental differences of OLAP systems compared with the other application areas:

- The number of prediction entities (queries) in any given context is considerably larger than in other domains (for example the number of files in a system).
- Different predicted objects (i.e., queries) are not disjoint in the sense, that results of a new query can be computed using several of the cached objects or parts thereof. The decision if a query can be derived from another query is called the *containment* or *subsumption problem.* Recently, this problem has been generalized to the *set derivability problem* ([AGL+99]) for algorithms that are using parts of the different objects to answer a query. However, the derivability property does not influence the prediction process described in this chapter, but will play an important role in Chapter 5.

Summarizing this survey of prediction approaches, it can be concluded that a consensus exists that Markov models are suitable as the base formalism for modeling navigational accesses. The main reasons for this are the simplicity of the model, the possibility to effectively compute predictions and the proven accuracy in many application areas. The characteristics of the application domains that have so far been target for prediction, mainly the navigational access along structures contained in the respective metamodels (e.g., links, jump instructions) are similar to the structures found with OLAP applications. However, OLAP applications differ from these in that the number of events that can be predicted is larger and that the predicted objects are not independent. Therefore, the PROMISE pattern model described in the next section will use generalizations to deal with these peculiarities of the OLAP environment.

4.2 The PROMISE Pattern Model for Interacting with OLAP Systems

The pattern model is of central importance to the prediction process as it determines which kinds of patterns can be represented and thus can be exploited to improve the system's performance. In order to reflect the importance of the design decisions involved in developing the pattern model, we start by discussing these principal design decision in Section 4.2.1. As our pattern representation language is based on Markov models, this formalism will be introduced in Section 4.2.2. We show that it is well suited to represent navigational behavior but that the naive modeling of OLAP interactions using Markov models does not produce satisfactory results. These findings provide the motivation for discussing different generalization techniques for OLAP queries in Section 4.2.3 as a solution to this problem.

4.2.1 General Considerations about the OLAP Pattern Model Design

The pattern model refers to the information about interactions that the user can perform described according to the user interaction model. However, in the previous chapter, we introduced two different interaction models mirroring different levels of detail in the observation of user behavior (query based model and the extended transformation based model). Both models share the following assumption concerning atomic events (queries):

- the class of possible queries is restricted to canonical queries as the user queries the database using a graphical formalism instead of a declarative query language.

The transaction based model additionally assumes the following integrity constraint concerning composite events (sessions):

- the navigational facilities provided by the OLAP user interface (i.e., the set of supported transformations and the query execution model) restrict the set of successor interactions.

At first, it might seem desirable to base the pattern model on the semantically richer model which includes additional information about the transformations (Transaction Based Interaction Model - TBIM). Such a detailed model would allow predictions on a finer level of granularity (on the granularity of transformations). However, in the context of the PROMISE/OLAP framework, we are only interested in the prediction of interactions that lead to a query against the database system. That means, we could only make use of these fine-grained predictions in the case that we assume a single query execution model (i.e., that a transformation always corresponds to a new query execution).

In order not to restrict the applicability of the PROMISE/OLAP pattern model to certain classes of OLAP front-end tools and applications that possess specific navigational capabilities, we decided not to make any assumptions about the OLAP user interface except that the class of queries supported by the interface are exactly the class of canonical queries[36]. Thus, the definition of the pattern schemata is based on the query based interaction model (cf. Section 3.3.1). The consequence of this design decision is that the approach can be applied to all systems generating sequences of multidimensional OLAP queries, irrespective of the front-end design. In particular, this makes the approach independent of the set of transformations and the query execution model that are actually provided by the OLAP front-end tool.

However, an important requirement for our pattern model is that the actual limitations of the deployed OLAP tool can be expressed as patterns (instead of being 'hard wired' into the pattern schemata). This enables the prediction system to 'learn' these peculiarities during the training phase using induction techniques. For example, if an OLAP tool is employed that imposes additional integrity constraints, like that the new restriction element must be an ancestor or a descendant of the current element, this should be learned by the model and no query sequences must be predicted that violate this integrity constraint.

Although the information about transformations does not influence the design of the pattern model, it is not useless,. It will play an important part, when designing heuristics for the eviction and admission algorithms of caches discussed in Section 5.2.2. We will then exploit the strong locality of query behavior according to a distance metric defined on the query space by the transformations (queries that can be generated from the last query by applying a small number of transformation have a high probability of being referenced soon).

Thus, the definition of the pattern model and the according prediction algorithm use the query based interaction model as a foundation. This means that the patterns will describe regularities between the attribute values of events defined in this model. However, we will only use the query attribute to formulate the patterns. The other attributes are handled as follows:

- *Information about the user* (User_ID): We assume that a distinct set of patterns is maintained for every user or user group, respectively. This means that the User_ID attribute is

[36] The application of our approach to front-ends allowing more expressive query formalisms is possible. In this case, for each query q, the system has to compute the minimal canonical query q' that subsumes the original query q before passing it to the prediction or induction process.

used to identify the relevant set of patterns (called prediction profile) before starting the prediction or induction process.

- *Information about the accessed cube* (Cube Instance): Each cube is characteristic for a specific task the user is performing. Therefore, we maintain distinct prediction profiles for every cube. Thus, analogously to the user information, the value of this attribute is used to determine the relevant prediction profile prior to the induction or prediction process. As all queries in a profile access the same cube it is not sensible to use this attribute in pattern schemata.
- *Timing information* (Timestamp): Assuming that events are generated strictly sequential during a session, this means two events in a session never have the same timestamp. This implies that all events in a session are distinct. The reason for including the timing information in the interaction model is that the user interaction model should not impose any restrictions on the types of patterns by omitting information that is easily available for e.g., interaction log files of all commercial systems. However, the prediction process presented here will not use the timing information (it assumes a logical timescale; cf. Section 2.5). Therefore, for this chapter we ignore the timing information and only use the sequence information (i.e., the order of events in a session).

Consequently, the next section is devoted to developing a pattern model for user sessions expressed according to the query based interaction model.

4.2.2 Modeling Patterns in OLAP User Behavior using Markov-Models

OLAP user behavior is strongly navigational. Consequently, the emphasis of PROMISE/OLAP is on navigational interaction patterns (cf. 2.4.2). This means that we are exploiting statistical dependencies of the next event (query) on the previous events (queries). Thus, assuming a look-back window size of m, we regard a sequence of m successive events in a session: $e_{t-m},\ldots,e_{t-1}$. The pattern should capture the dependency of the event e_t following this sequence in the session depending on the events in the sequence. Patterns of this type are called (probabilistic) sequential rules. Often, the following notation is used to refer to probabilistic sequential rules:

$$(e_1,\ldots,e_m \rightarrow e_{m+1}, \text{p}), \text{ where}$$

$e_i \in E_{IM}$ are events according to the interaction model IM ($i \in \{1,\ldots,m+1\}$) and p denotes the probability of this rule.

For the PROMISE/OLAP approach, we represent a set of such rules describing statistical interrelationships between subsequent events by means of Markov models. Therefore, Section 4.2.2.1 briefly introduces the formalism of discrete time Markov models and shows that it is equivalent to a set of probabilistic sequential rules. Based on this general description, Section 4.2.2.2 discusses the issues involved in representing OLAP user behavior by means of Markov models.

4.2.2.1 Discrete Time Markov Models

This subsection introduces the formalism of Discrete Time Markov Models (DTMMs) (for example [How60]). This presentation is independent of the OLAP domain or the PROMISE approach in general and forms the basis for discussing its application to OLAP user interaction patterns in Section 4.2.2.2.

DTMMs have been deployed in different areas of application in order to model probabilistic navigational user behavior and to build prediction algorithms (cf. Section 4.1). DTMMs

represent a dynamic system (in our case the user's analysis process) as a finite set of states $S = \{s_1,...,s_k\}$. At each discrete point *t* of the logical timescale (in our case defined by the discrete points in time at which the user initiates the interactions), the systems possesses exactly one active state that is denoted as $s^t \in S$. Between two discrete points in time, e.g., (*t-1*) and *t* the system autonomously changes its active state from $s^{t-1} \in S$ to $s^t \in S$. In contrast to finite state automata that use a transition function that deterministically depends on an input character, the transition in Markov models is probabilistic. This means that the probability that a specific state *s* becomes the next active state is determined by a transition probability function *p* that maps a potential successor state $s \in S$ to a probability value. This value is interpreted as the probability that the state *s* will be the next active state i.e., $s^t = s$. The characteristic and central assumption of Markov models is that this probability is only dependent on a fixed number of states *m* (called the order of the DTMM) that have been visited directly before the point *t* (i.e., the states s^{t-1}, s^{t-2}, s^{t-m}). Thus, the probabilistic transition function $p: S \times S^m \rightarrow [0,1]$ is denoted as follows:

$$p\left(s^t, (s^{t-1}, s^{t-2}, \ldots, s^{t-m})\right)$$

These basic considerations are mirrored in the following definition of a DTMM:

Definition 4.1 (Discrete Time Markov Model)

A *Discrete Time Markov Model* (DTMM) is defined as a tuple *DTMM*=(*m, S, P*) where

- $m \in \mathbb{N}$ is a number indicating the size of the lookback window (called the order of the Markov Model)
- *S* denotes a finite set of states and
- *P*: $S \times S^m \rightarrow [0;1]$ denotes the probability function for state transitions. $P_m(s, s_1,...,s_m)$ gives the conditional probability for a state transition into state *s* under the condition that the last states visited were $s_1,...,s_m$. ◆

Example 4.1 (Discrete Time Markov Model)

A simple order 1-Markov model with two states is defined by the tuple: *M*=(1,{A,B}, *P*), with the following values for *P*:S×S: *P*(A,A)=0.4; *P*(A,B)=0.7; *P*(B,A)=0.6 and *P*(B,B)=0.3 ◆

The value of the parameter *m* (order of the DTMM) defines the size of the "memory" of the model i.e., how many previous states are used to determine the probability for the next state. Finding the right value for the parameter is an optimization problem that has to be solved separately for every domain and application specifically considering the trade-off between the costs of maintaining a larger model and the benefits of increased accuracy of the predictions. Furthermore, it has to be considered that higher order Markov models need a longer session prefix to operate properly. That means that the sessions must have a minimal length and prediction is not possible during the first *m* queries of a session. The value reflects the maximum (average) number of previous results that a user takes into account when deciding about the next interaction. E.g., in the web domain, this corresponds to the number of last visited pages used to choose the next link. Chapter 6 contains results of experiments with different values for Markov models in the area of OLAP predictions. They confirm that for the OLAP domain, order values 1 or 2 already deliver good predictions. This corresponds to findings in the area of web access analysis (for example [NZA99]), where extensive analysis of logfiles has shown, that a context of maximally 2 is sufficient.

Order-1 Markov models have the advantage that they are isomorphic to labeled graph structures and can thus be elegantly visualized using Markov Diagrams (see Figure 4.1).

These diagrams represent states as nodes and possible transitions as edges labeled with the corresponding probability value of the transition function *p*. Thus, for presentation reasons, all the examples throughout this thesis will use Markov models of order 1. However, our algorithms will be able to deal with higher level models as well. The development of algorithms for higher order Markov models is eased by the fact that any order-*m* model can be represented by an order-1 model which has more states. The following theorem proofs this convenient property that allows us to consider only order-1 Markov models for the design of the base algorithms.

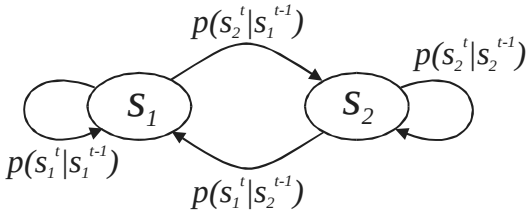

Figure 4.1: A Simple Markov Diagram

Theorem 4.1 (Equivalence of Order-m and Order-1 Markov Models)

Each order-*m* Markov model $DTMM=(m, S, P_m)$ can be simulated by an equivalent order 1 Markov model $DTMM' = (1, S', P_1')$. ◆

Proof 4.1 (Equivalence of Order-m and Order-1 Markov Models)

The idea of the proof is to give a construction method for an oder-1 Markov model $DTMM' = (1, S', P_1')$ that uses a larger state space.

The basic idea of constructing the order-1 model from the order-*m* model is that the new order-1 Markov model contains all *m*-tuples over the original set of states S as states:

$$S' = S^m$$

The probabilities for the transition function from state $s=(s_1,...,s_m)$ to state $t=(t_1,...,t_m)$ is defined as follows:

$$p(s,t) = \begin{cases} p(t_m,(s_1,\ldots,s_m)) & \text{for } s_i = t_{i-1} \text{with } i \in [2;m] \\ 0 & \text{else} \end{cases}$$

◆

In Chapter 2 we used rules as an example to represent patterns and pattern schemata. The following theorem shows that Markov models exactly represent a set of probabilistic rules that follow a fixed schema. This allows us to use both formalizations interchangeably.

Theorem 4.2 (Equivalence of Markov Models and Sequential Rules)

Any Markov model can alternatively be represented as a set of probabilistic sequential rules over sequences of states S^* that have the following schema[37]:

$$(s_1,\ldots,s_m \rightarrow_1 s, p) \text{ with } s_i, s \in S \text{ and } p \in [0;1]$$

◆

[37] The *follows* symbol has been defined in Example 2.2.

Proof 4.2 (Equivalence of Order-m and Sequential Rules)

Let R be the set of rules, then the corresponding Markov model $DTMM_R=(m, S, p)$ is defined as follows:

$$r = (s_1,\ldots,s_m \rightarrow_1 s, p_r) \in R \Leftrightarrow p(s,(s_1,\ldots,s_m)) = p_r$$

◆

Consequently, we can use the sequential rule representation interchangeably with the Markov model representation. For visualization and implementation purposes, it is more convenient to use the Markov model representation, while we will use the rule representation to formally define the pattern schemata. The following section discusses how Markov models can be applied to the PROMISE/OLAP approach.

4.2.2.2 Modeling Patterns in OLAP User Behavior Using Markov-Models

When constructing a Markov model to base the prediction process on, it is necessary to give a construction method for the state space (i.e., defining the set of states and their semantic interpretation according to the interaction model) and a definition of the transition semantics. The construction of the state space for the Markov model *M* involves the definition of a mapping function *state* from events according to the Interaction Model *IM* to the state space *State* of the Markov model. In order to be useful for prediction purposes, this mapping should be designed such that it is bijective. This becomes clear when looking at the prediction procedure: the transition function of the Markov model is used to predict a set of probable next states (see 4.3.1 for more details). Then a prediction of interaction events is derived by applying the inverse of the state mapping $state^{-1}$.

The standard way (cf. e.g., [ZAN99]) of designing the mapping for navigational user behavior to Markov models is to represent every possible interaction event as a state in the Markov model. E.g., for a world wide web navigation, this would correspond to mapping every page p (identified by its URL) to a state of the model $state(p)$ and to represent each navigation from page p_1 to page p_2 as a transition between $state(p_1)$ and $state(p_2)$. Figure 4.2 visualizes this standard mapping approach.

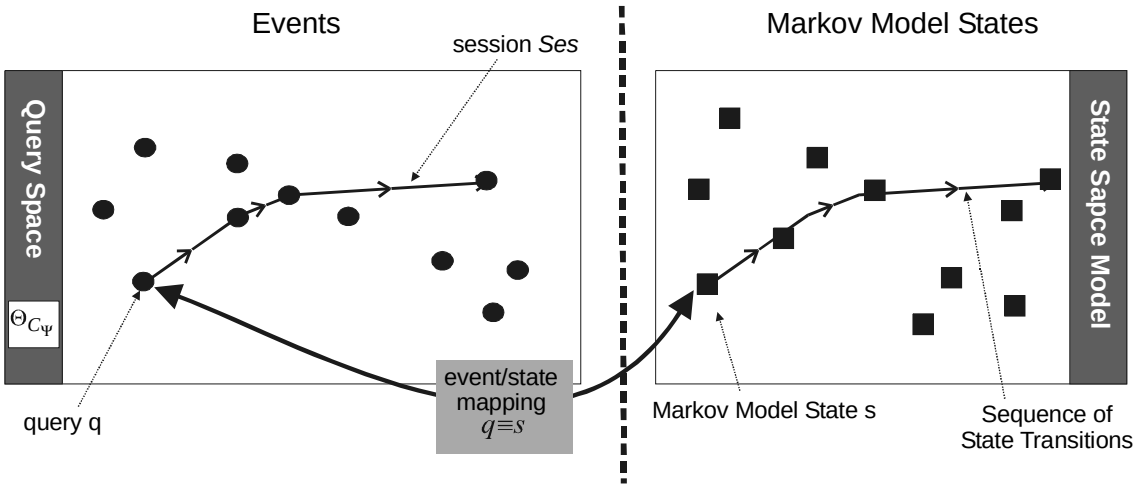

Figure 4.2: Standard Approach of Representing Atomic and Composite Events in Markov Models

The following definition summarizes the formalism that is needed to build the mapping between an interaction model and a Markov model.

Definition 4.2 (Markov Models Representing an Interaction model)

Let $M = (m, S, P)$ be a Markov model (according to Definition 4.1) and *IM* be a user interaction model (according to Definition 2.1) with the according set of events E_{IM}. *M* is called a representation of *IM* if a mapping function $state_{IM,M}$ can be defined with the following properties:

- $state_{IM,M}: E_{IM} \rightarrow S$ and
- $state_{IM,M}$ is bijective

The mapping function $state_{IM,M}$ is called the event/state mapping function. It defines an equivalency relation $\equiv_M$ in the following way:

$$e \equiv_M s :\Leftrightarrow state(e) = s$$

◆

Using the mapping function *state* it is possible to find the corresponding Markov model state for every event of a session *Ses*. This means that a session (which is a sequence of events) can be represented as a sequence of state transitions of the Markov model (respectively a sequence of Markov model states) by mapping each event of the sequence to the corresponding state. The resulting sequence is called the state sequence of a session and is formally described by the following definition:

Definition 4.3 (State Sequence of a Session)

Let $M = (m, S, p)$ be a Markov model representing a user interaction model *IM*. Additionally, let $state_{IM,M}: E_{IM} \rightarrow S$ denote the corresponding event/state mapping function. The state sequence $stateseq_M(Ses)$ for a session $Ses=<e_1,\ldots,e_k>$ is defined as follows:

$$\text{stateseq}_M(Ses) := < s_1,\ldots,s_k > \text{ with } s_i = state_{IM,M}(e_i) \text{ for } i \in [1;k]$$

◆

Applying the standard mapping procedure described above to the query based OLAP interaction model would result in a Markov model with a distinct state *state*(*q*) for each canonical query *q* representing an *execute query* event (additionally, two special states represent the *start session* and *end session* event). A transition between two states $state(q_1)$ and $state(q_2)$ representing events of type *execute query* corresponds to the execution of query q_2 directly after q_1 in a session. A transition from the special *start session* state to state $state(q_1)$ means that q_1 is the first query of a session. Analogously, a transition from $state(q_1)$ to the *end session* event represents the fact that q_1 is the last query of the session. However, this simple approach has severe technical and conceptual drawbacks:

- *Generalized patterns cannot be exploited*: The fine grained approach only predicts mere repetitive patterns that consist of exactly the same queries. In other words, the system is unable to 'learn' generalized patterns. An example for a generalized pattern is that the user typically analyzes the previous year after analyzing the current year irrespective of the rest of the query. Using the fine grained approach, this high-level pattern is represented as relatively high probabilities on a large number of transitions in the Markov model (transitions between all states that correspond to queries which contain subsequent years as restrictions). If a query only differs in some detail (for example the restriction element in one dimension) from the queries which have been used to train the Markov model, no predic-

tion is possible. This is a clear drawback in OLAP applications where e.g., the same sequence of query templates are being executed with different parameters (cf. Section 4.2.3).

- *Large State Space*: Although the set of all possible canonical queries against a given multidimensional schema is finite, it is still large enough to be prohibitive. In Example 3.16, we computed the number of well-formed OLAP queries to be in the magnitude of 10^{13}. That means that the cardinality of the query space is still orders of magnitude larger than e.g., the number of documents of a WWW site. An extremely large state space implies two kinds of problems:
  - *Storage and maintenance costs* for such a large model are prohibitively high. In order to enable an efficient prediction process, the Markov model should target a size such that it fits into main memory. Additionally, when regarding caching as an application for prefetching, the space occupied by the prediction model takes away valuable cache space. This makes a representation desirable that is as compact as possible.
  - A large number of states leads to a large set of potential successor states for a single state which in turn decreases the probability of the transitions. Therefore, this model is not only very costly to maintain but also *not very predictive*.

Our solution to these problems is using Markov models which are based on generalizations of the interactions (canonical OLAP queries). That means instead of using the individual queries to build the state space of the Markov model, we use a generalization to build a generalized Markov model. Generalization reduces the number of states in the model thus increasing its maintainability and its predictiveness.

However, a Markov model with a generalized search space can only be used to make generalized predictions. I.e., the result is not a query, but a set of queries (all members of the respective equivalency class). Therefore, it is necessary to combine a set of prediction models that can be used to predict all the details of a query. In order for the combined prediction to be accurate, the different generalization criteria which are used to build the different models have to be independent of each other.

Following this approach, the central issue throughout Section 4.2.3 will be the identification of independent generalizations for OLAP queries which are suited to be combined in a prediction of future query behavior.

4.2.3 Generalizing OLAP Queries

The solution sketched in the previous section relies on a generalization of events. In our interaction model, we distinguish between three types of events (start session, end session, execute query) with different attributes. A definition of a generalization function is based on the attributes of the events and thus has to distinguish between the different types of events. However, as we decided to disregard the timing and the user information (see Section 4.2.1), events of type *start session* and *end session* do not possess any attributes. Therefore, the generalizations of these events are not very interesting, as all the events of the same type are generalized to a single generalized event. Therefore, in this section, we only discuss the generalization of events of the type execute query which are characterized by their canonical query. The defined classifications regarding canonical queries can be easily generalized to events (see Section 4.2.4).

To this end, this section discusses different possibilities for generalizing canonical OLAP queries and argues why they are likely to produce sensible (and significant) generalized pat-

terns. Thus, we have to find criteria according to which canonical OLAP queries can be generalized[38].

A generalization of an OLAP query *q* according to a generalization criterion *g* is defined by an equivalency relation on the set of canonical OLAP queries $\alpha_g \subseteq \Theta_{C_\Psi} \times \Theta_{C_\Psi}$ (see Figure 4.3). The classes of α_g are called *query prototypes* and the set of all query prototypes according to *g* is denoted as the *generalized query space* $\wp_g(\Theta_{C_\Psi})$. α_g defines a valid generalization if $\wp_g(\Theta_{C_\Psi})$ totally partitions the canonical query space Θ_{C_Ψ} i.e., every canonical query belongs to exactly one equivalence class (prototype). This uniquely determined equivalence class for a query *q* is denoted as the prototype of the query $\wp_g(q)$ .This means that a generalization can alternatively be defined by giving a total function. $\wp_g : \Theta_{C_\Psi} \rightarrow \wp_g(\Theta_{C_\Psi})$ that maps each query to its prototype. Figure 4.3 visualizes the principle of generalizing canonical queries and clarifies the terminology. We will give examples for two different generalizations in Section 4.2.3.1 and 4.2.3.1.

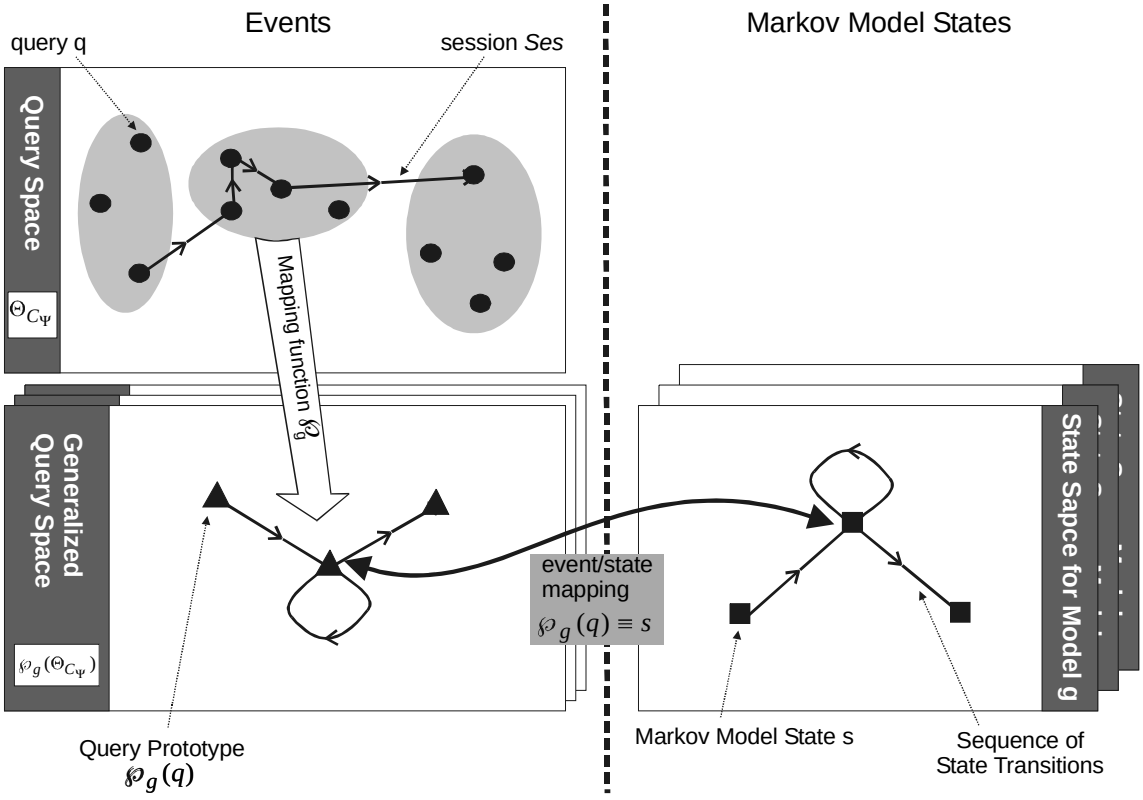

Figure 4.3: Overview of the PROMISE Pattern Model Approach

Of course, different ways of partitioning the canonical query space are possible (depending on what detail information is omitted). PROMISE/OLAP uses two heuristic partitioning methods, which are the result of an analysis of typical user behavior in OLAP environments. In order to understand why we chose exactly the generalizations for OLAP queries which will be described in the following sections, we first have to describe the typical workflow of an analytical OLAP session from the user's point of view (deploying the user's terminology):

[38] Notably, the generalization is a special case of classification as described in 3.1.2.

OLAP sessions usually start with a predefined report (from a database point of view, this corresponds to a template for a canonical query). It specifies the set of result measures, the MD presentation structure of the canonical query (i.e., the partitioning of dimensions into result dimensions and selection dimensions) and the result granularity for all dimensions. For example the definition of the report named 'Repairs by geographic Regions' could specify that repair location is the result dimension (one-dimensional result), that results are classified according to geographic regions, years and assemblies – not restricting the repair type and the vehicles. Report templates do not have to be predefined but can be obtained from other report templates by applying transformation operations. The restriction elements can be regarded as parameters to the report definition. The report definition usually contains default values for the parameters, but when opening the report, the user can change (some of) these values. A report layout is characteristic for the subtask a user is currently solving as part of the analytical workflow. The structure of the query contains information about which correlation a user is investigating at the moment he executes the query. For example, during the first phase of the analysis the logistics expert is looking for irregularities of failures within different geographical regions. This means she compares the failure rate of different assemblies in the different regions. During this phase she will not restrict e.g., the type of failure. The restrictions will always be on the level 'year' etc. Depending on the actual analysis task, the user executes these structurally equivalent queries with different parameters (for example for the current year and the geographic region for which an operation is being planned). Thus, patterns in the analytical workflow of the user are mirrored in the structure (i.e., which measures are displayed, which are the result dimensions and what is the result granularity) of the queries. This is the reason why we introduce a generalization in Section 4.2.3.1 which is based on the MD presentation structure.

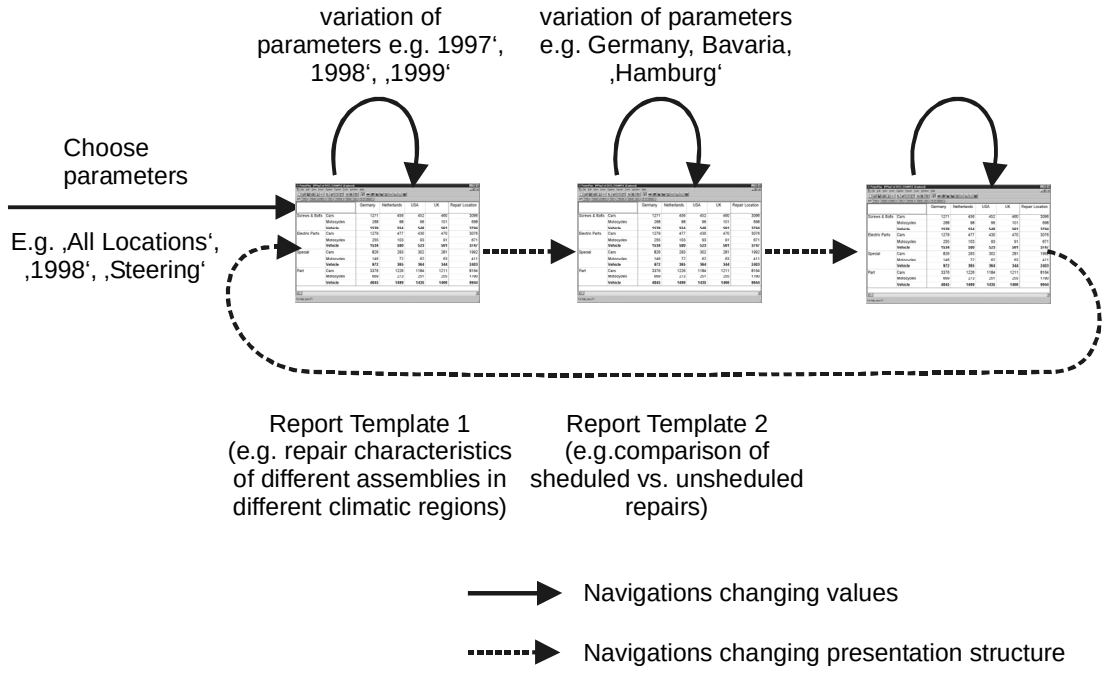

Figure 4.4: Macro-Structure of the Analysis Workflow

A second type of patterns concerns the change of report parameters (restriction elements of the canonical queries) which are not captured by the structural changes. Typically, the analyst chooses a set of parameters (e.g., the year *1999* and the *steering assembly*) and looks at a series of reports (often called a briefing book) before changing/revising the set of parameters

and executing the series of reports again. Additionally, it is common to change single parameters for one report, thus executing the same business report with different sets of parameters. Figure 4.4 visualizes the change of parameters in relation to changes of the structure.

The parameter changes (restriction element changes) exhibit regularities that are independent of the structure of the reports. These value-based patterns mirror the fact that the analyst often accesses values of a dimension in a certain order which may be problem specific. E.g., the analyst may start with restricting values to the current year and the successively analyze the years in descending order (last year, the year before last year etc.). Such an order may not always be linear. For example, in a geographic dimension, the user may start the analysis with her own region and then analyze the adjoining regions. In order to describe these value-based patterns between successive queries, we will define a value-based generalization in Section 4.2.3.2.

4.2.3.1 Generalization based on the MD Presentation Structure

As argued in the previous section, generalizations based on the MD presentation structure and granularity (called structural patterns) mirror the fact that each stage of the analysis process has a set of characteristic views on the data. The user's interest is mirrored in the result measures, the result granularities, as well as in the MD presentation structure. This leads to the following definition of the structural generalization.

Definition 4.4 (Structural Generalization, Structural Query Prototype):

Assuming a cube schema $C_\Psi = (\{d_1,\ldots,d_m), M_C)$, the set of structural query prototypes $\wp_{Struct}(\Theta_{C_\Psi})$ is defined as follows:

$$\wp_{\text{Struct}}(\Theta_{C_\Psi}) := \underbrace{2^{M_C}}_{\substack{\text{sets of}\\ \text{measures}}} \times \underbrace{\prod_{i=1}^{n} L(\Psi|d_i)}_{\substack{\text{granularity}\\ \text{vectors}}} \times \underbrace{\{0;1\}^n}_{\substack{\text{presentation}\\ \text{structure}}}$$

The structural generalization of a canonical query $q_C = (M_q, R_q, G_q)$ is defined by the function $\wp_{Struct} : \Theta_{C_\Psi} \rightarrow \wp_{Struct}(\Theta_{C_\Psi})$ mapping the canonical query q_c to its structural prototype $\wp_{Struct}(q_C)$:

$$\wp_{\text{Struct}}(q) := (M_q, G_q, \text{presstruct}(q))$$

The corresponding equivalency relation $\alpha_{\text{Struct}} \subseteq \Theta_{C_\Psi} \times \Theta_{C_\Psi}$ is defined as:

$$\alpha_{\text{Struct}}(q_1, q_2) :\Leftrightarrow \wp_{\text{Struct}}(q_1) = \wp_{\text{Struct}}(q_2)$$

◆

The number of structural query prototypes (i.e., the number of equivalency classes defined and thus the maximum number of states in the Markov model) can be computed by the following formula:

$$\left|\wp_{\text{Struct}}(C_\Psi)\right| = \underbrace{\sum_{i=1}^{|M_C|} \binom{|M_C|}{i}}_{\substack{\text{number of distinct result}\\ \text{measure sets}}} \cdot \underbrace{\prod_{i=1}^{n} \left|L(\Psi|_{d_i})\right|}_{\substack{\text{number of distinct}\\ \text{granularity vectors}}} \cdot \underbrace{2^n}_{\substack{\text{number of distinct}\\ \text{selection/result dimension}\\ \text{combinations}}}$$

Example 4.2 (Structural Query Prototype)

The query $q_{example}$ introduced in Example 3.16 and the query q_2 have the same structural prototype. Thus, they belong to the same equivalency class with respect to the structural generalization. Notably, $q_{example}$ and q_2 do not only differ in that they deploy different restriction elements, but for the dimension repair location(3), the restriction elements are also from different classification levels.

```
q_example := ( {#repairs},
              ( all vehicles , 1999, Germany      , steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type.all  ) )

q_2 :=       ( {#repairs},
              ( all vehicles , 2000, all locations, steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

For both queries, dimensions 1, 2, 4, and 5 are selection dimensions (restriction on the same level of granularity as the result). Therefore, the structural query prototype contains the presentation structure (1,1,0,1,1).

```
℘_Struct(q_example) = ℘_Struct(q_2)
              ( { #repairs},
                ( vehicle.all , year, geogr. region, assembly, type.all ),
                ( 1,1,0,1,1) )
```

The total number of possible structural query prototypes for our example schema (Example 3.14) computes to:

$$\left|\wp_S(C_\Psi)\right| = (3+3+1)\cdot(4\cdot2\cdot4\cdot5\cdot4)\cdot32 = 143360 = 1{,}4*10^5$$

◆

The figures computed in the previous example show that when constructing a Markov model using the structural generalization, the cardinality of the state space is reduced to an order that can be easiliy managed. This is especially true, as in practice, the number of states that are actually visited is a lot smaller than this theoretical maximum.

In order to illustrate Markov models using structural prototypes as states, we introduce a graphical representation for structural query prototypes. The graphical notation is shown in Figure 4.5. The top part of the oval contains the result measures of the prototype, while the left part contains the chosen result granularity for dimensions that are selection dimensions (selection levels). The granularities for the dimensions acting as result dimensions in q are shown on the right side of the oval. The icons are introduced to improve the readability of the notation. They abstractly depict a table presenting the results of the query. The parts where the respective query results (selection levels, result levels, result measures) are located are shaded in gray (for example the icon for measures has a gray area where the measures are visualized in a table). Dimensions which have a result granularity of 'all' must be selection dimensions (because no higher level of restriction exists). Therefore, in order to further increase the readability of the graphical notation, we omit selection dimensions that are restricted on the 'all' level in the visualization.

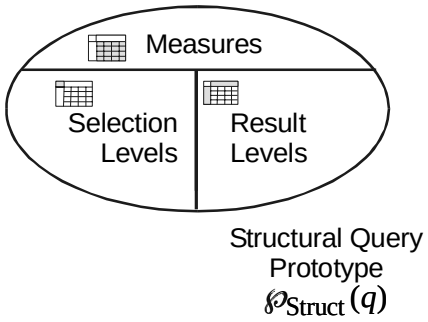

Figure 4.5: Graphical Representation of a Structural Query Prototype

4.2.3.2 Value-based Generalization

As argued before, changes in the restriction values of queries are considered largely independent from the structure of the report as they are mirroring a certain order in the user's perception of the concepts in the dimension. Furthermore, we assume that the navigational patterns occurring among the members of the different dimensions are independent (this simplification does not restrict the generality of our approach, as dependent dimensions can be combined for pattern recognition purposes). Therefore, we define a distinct value-based generalization for each dimension, resulting in *n* different value-based generalizations of a canonical query for an *n*-dimensional data model. The classes of the generalized query space are defined by the set of possible restriction elements for the dimension. Consequently, the prototype is represented by the restriction value itself. The following definition summarizes these considerations:

Definition 4.5 (Value-based Query Prototype):

The generalized query space for the value-based generalization regarding dimension d_i is defined as follows:

$$\wp_{\text{Val},i}(\Theta_{C_\Psi}) := dom(\Psi|d_i)$$

The value-based generalization of a canonical query $q = (M, (r_1, \ldots, r_n), G)$ is given by the function $\wp_{\text{Val},i} : \Theta_{C_\Psi} \rightarrow \wp_{\text{Val},i}(\Theta_{C_\Psi})$ defined as follows

$$\wp_{\text{Val},i}(q) := r_i$$

◆

The number of possible value-based prototypes can be easily obtained by the number of classification nodes in the domain of the classification lattice of the different dimensions. Thus,

$$\left| \wp_{\text{Val},i}(\Theta_{C_\Psi}) \right| = \left| dom(\Psi|_{d_i}) \right|$$

The actual order of magnitude for the domain is largely dependent on the application domain and will in most cases be largely dominated by the number of classification nodes on the lowest level of the hierarchy. The number of generalized states (restriction values) that are actually visited by a typical user are usually considerably smaller than the number of all restriction elements (for example restrictions on the lowest level of granularity are seldom for large dimensions).

Example 4.3 (Value-Based Query Prototype)

Let us once again consider the example query shown below:

```
q_example := ( {#repairs},
              ( all vehicles , 1999, Germany       , steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

As the value-based prototype (the equivalence class) is simply represented by the restriction value, the value-based prototypes according to the *repair time* and *repair location* dimension are defined as follows:

$$\wp_{\text{Val,repair time}}(q_{\text{example}}) = 1999$$

$$\wp_{\text{Val,repair location}}(q_{\text{example}}) = \text{Germany}$$ ◆

Having defined the two types of generalization (structural vs. value-based) in this section, the next section can give the full pattern interaction model (pattern representation, generalization functions and pattern schemata) and present the Markov models which are based on these generalizations and which will be evaluated by the prediction process.

4.2.4 Wrapping Up: The PROMISE/OLAP Interaction Pattern Model

The previous sections presented the formalisms we use for representing navigational patterns (Markov models or probabilistic sequential rules) and the generalizations. This section completes these considerations by formulating the pattern interaction model conforming to Definition 2.7 and summarizing the basic assumptions for the model design. We will also present an abstract data structure that is being used to represent instances of the pattern model during the prediction process. We call it a *prediction profile* as it characterizes the individual user, a user group, a certain task or the combination (cf. [Sap99]).

So far, the generalizations only considered canonical queries (cf. Section 4.2.3) and are therefore only suited to process event of type '*execute query*'. However, the definitions of query generalizations according to a criterion *g* (in our case structural or value-based) can be easily extended to all events of the interaction model by adding two special elements '*start session*' and '*end session*' to the according prototype space $\wp_{\text{g}}(\Theta_{C_\Psi})$ and by 'overloading' the generalization functions in the following way:

$$\wp_{\text{g}}(e) = \begin{cases} \wp_g(e.query) & e.type = \text{execute-query} \\ start\ session & \text{for } e.type = \text{start-session} \\ end\ session & e.type = \text{end-session} \end{cases} \quad \text{with } e \in E_{IM\,/\,OLAP}$$

As argued in Section 4.2.3, the most important assumption of our approach is the following:

- (*Assumption 1*) Patterns concerning the MD presentation structure are independent of patterns concerning the restriction values for OLAP queries (i.e., the probability that the next query has a certain structural prototype, is not dependent on the current restriction elements).

This behavior can be captured by patterns that determine the structural prototype of the next event depending on the last *m* (being the size of the look-back window) structural event prototypes. These patterns expose the following schema where $\wp_1,\ldots,\wp_{m+1} \in \wp_{\text{Struct}}(\Theta_{C_\Psi})$ are the free variables of the pattern schema:

$$\forall S \in E^{C}_{IM/OLAP} :\prec e_1,\ldots,e_{m+1} \succ \subset S \wedge$$
$$\wp_{\text{Struct}}(e_1) = \wp_{\mathbf{1}} \wedge \ldots \wedge \wp_{\text{Struct}}(e_m) = \wp_{\mathbf{m}} \Rightarrow \wp_{\text{Struct}}(e_{m+1}) = \wp_{\mathbf{m+1}} \quad \text{(PS1)}^{39}$$

In order to represent the instances of this pattern schema, we use a Markov model (called *Structural Prediction Model)*. The state space of this *Structural Prediction Model (spm)* contains one state for each element of the generalized event space according to the structural generalization (see Definition 4.4 i.e., a separate state for each structural event prototype). A transition from *state*($\wp_1$) to *state*($\wp_2$) is labeled with the probability that an event e_1 with prototype $\wp_1 = \wp_{\text{Struct}}(e_1)$ is followed by an event e_2 with prototype $\wp_2 = \wp_{\text{Struct}}(e_2)$. Figure 4.6 shows an extract of a structural prediction model for the example scenario.

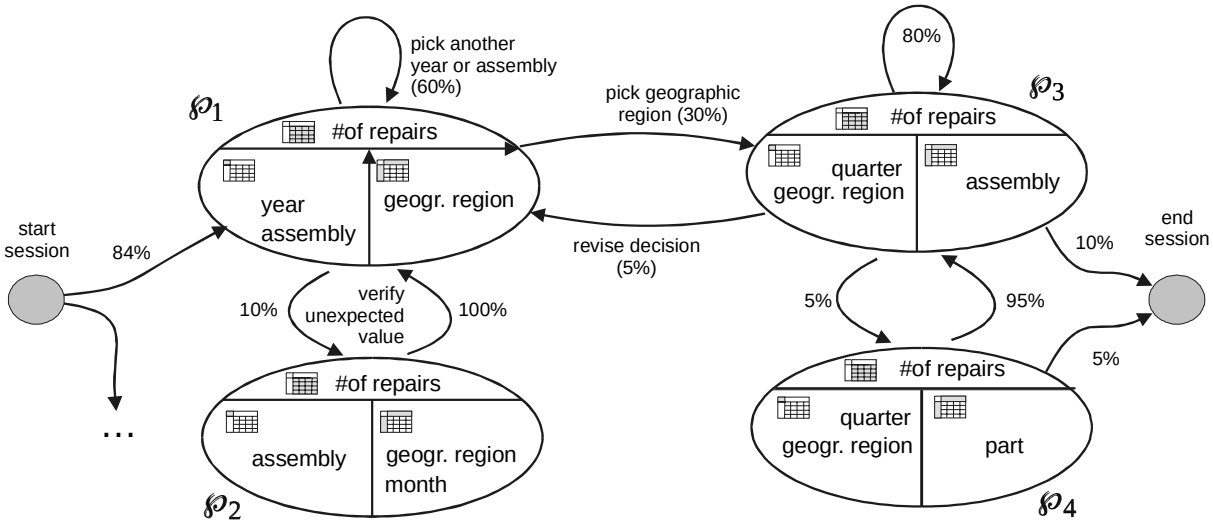

Figure 4.6: A Structural Prediction Model

As argued in 4.2.3.2, for our model of patterns concerning the restriction elements of subsequent queries in the user's sessions, we make the following additional assumption:

- (*Assumption 2*) Patterns for changes in the restriction values in the different dimensions are independent of each other (i.e., the probability of selecting e.g., '1999' after '1998' is not dependent on the current restriction e.g., in the location dimension).

Consequently, we model separate patterns for each dimension. All of these patterns are instances of the following pattern schema ($v_1,\ldots,v_{m+1} \in dom(\Psi|_{d_i})$ and $i \in [1;n]$ are the variables of the pattern schema):

$$\forall S \in E^{C}_{IM/OLAP} :\prec e_1,\ldots,e_{m+1} \succ \subset S \wedge$$
$$\wp_{Val,\mathbf{i}}(e_1) = \mathbf{v_1} \wedge \ldots \wedge \wp_{Val,\mathbf{i}}(e_m) = \mathbf{v_m} \wedge \mathbf{v_{m+1}} \neq \mathbf{v_m} \Rightarrow \wp_{Val,\mathbf{i}}(e_{m+1}) = \mathbf{v_{m+1}} \quad \text{(PS2)}$$

To represent the instances of these pattern schemata during the prediction process, we use a separate Markov Model (called *Value-Based Prediction Model; vpm*$_i$) for each dimension d_i. This model contains information about typical navigation paths in a dimension. It contains a state for each member of the domain of the dimension's classification lattice and a transition from state *a* to state *b* represents a sequence of queries <*q*,*p*> where *q* was restricted by *a* and *p* was restricted by *b*. Figure 4.7 shows two sample Value-Based Prediction Model for the

[39] Notably, we use the universal quantor ∀ in the formula. However, as we assume a probabilistic domain model (see section 2.4.2), we do not demand that all sessions fulfill the pattern but attach a probability to each pattern that the patterns holds for a session satisfying the conditions stated in the pattern.

repair location dimension. Of course, it is possible to use different look-back window sizes (Markov Models) orders for the different value-based prediction models as well as for the structural prediction model.

Notably, the navigational paradigm of the front-end tool typically restricts the possible sequences of restriction values. E.g., some operations only allow ancestors or descendents to be chosen as the new restriction element. These restrictions influence the value-based prediction model, as certain state transitions are not possible and their probability will therefore always be zero. However, according to our basic design decisions (cf. Section 4.2.1), we do not incorporate these restrictions into our design of the value-based prediction model as this would mean that the formalism only works for front-ends enforcing these restrictions. Instead, these peculiarities are learned during the induction process (for example because certain sequences never occur). If an OLAP tool follows the navigational paradigm described in 3.2.4, the classification schema structure will be mirrored in the value-based model, because navigation among classification nodes on the same level or between classification nodes and their parents will have a significantly higher transition probability.

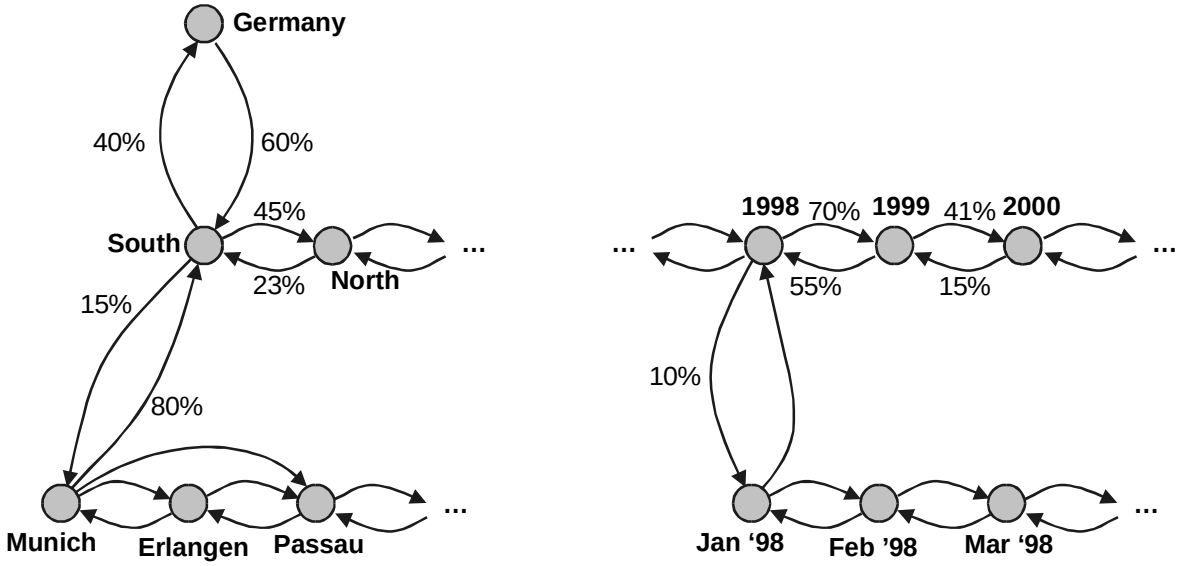

Figure 4.7: Two Value Prediction Models (VPMs) for Repair Location (left) and Repair Time(right)

The structural prediction model and the value-based prediction model contain enough information to predict a complete canonical query. However, it is important to note that the structural prediction model and the value-based prediction model are not completely orthogonal. The first restriction that links both models is the condition that all queries have to be well formed (i.e., the restriction level must be greater or equal than the result granularity according to the classification schema). The second integrity constraint concerning both models is that a structural prototype contains information about which dimension is a selection dimension and which dimension is a result dimension (we chose to include this information in the structural generalization as it is an important indicator of the user's current interest and thus characterizes the current phase of the analytical workflow.). This implies that for the selection dimensions, the restriction element must be from the level determined by the result granularity. This means that if a dimension is predicted to be a selection dimension according to the structural prediction model, the selection model must only predict classification nodes from the same level. These interdependencies can be exploited during the prediction process to limit the set of candidate restriction elements.

In order to increase the accuracy of the prediction and to better capture the typical nature of OLAP sessions, we introduce an additional assumption. Assumption 2 states that the value of a changed restriction element is independent of the structure of the last queries. However, the analysis of typical OLAP sessions (cf. Section 4.2.3) suggests that the probability for a dimension to change its restriction value (at all) between two subsequent queries is highly dependent on the structural context (i.e., the structural prototype of the last *m* queries):

- (*Assumption 3*) The probability which of the dimensions changes the restriction element between two queries is dependent on the structural session context.

The argument in favor of this assumption is that the user typically does not vary all the parameters (restriction elements) of a query at the same time, but uses a structured methodology varying only a small number of influencing parameters at time. The set of varied parameters is dependent on the phase of the analytical workflow (which is being represented in PROMISE by the structural query prototypes). E.g., when analyzing, if a geographic region exhibits time-independent characteristic repair and failure patterns (characterized by a report showing the number of repairs for different geographic regions), the analyst typically varies the restriction in the time dimension. However, during a later phase of the analysis process, when she is trying to find out, if the repair characteristics of an assembly differ from other assemblies, the varying parameter will be the geographic region.

Thus, the schema for patterns that capture regularities according to assumption 3 looks like this ($\wp_1,\ldots,\wp_m \in \wp_{\text{Struct}}(\Theta_{C_\Psi})$ and $i \in [1;n]$ are the free variables of the schema):

$$\forall S \in E^{C}_{IM\,/\,OLAP} : e_1,\ldots,e_{m+1} \subset S \wedge \\ \wp_{Struct}(e_1) = \wp_{\boldsymbol{1}} \wedge \ldots \wedge \wp_{Struct}(e_{m+1}) = \wp_{\boldsymbol{m+1}} \Rightarrow \wp_{Val,\boldsymbol{i}}(e_m) \neq \wp_{Val,\boldsymbol{i}}(e_{m+1}) \qquad \text{(PS3)}$$

Pattern instances of this schema cannot be directly represented as Markov models. However, if the 'look-back window' for parameter changes is chosen such that it is as long as the look-back window of the structural prediction, it is possible to extend the Markov model representing the structural prediction model: For every transition of the Markov model, we record the probability of a restriction change for every dimension. The *probabilistic change vector* $<p_1,\ldots,p_n>$ contains the probabilities p_i that dimension d_i changes the restriction element ($i \in \{1,\ldots,n\}$). Figure 4.8 visualizes an example of such a probabilistic change vector for a look-back windows size of 1. The vector (shaded gray) is attached to a transition of the Markov Model (from *state*($\wp_1$) to *state*($\wp_1$)). The probability that the restriction element for dimension 2 (*repair time*) changes between two subsequent queries that both have a structure corresponding to *state*($\wp_1$) is 98%. This information is used to further restrict the set of candidate restriction values for the dimension during the prediction process (see next section).

Notably, some of the change probabilities can be directly derived from the last structural prototype $\wp_{\text{Struct}}(e_m)$ of the look-back window and the predicted prototype $\wp_{\text{Struct}}(e_{m+1})$ if e_m and e_{m+1} are of type *execute-query*. Dimensions which are selection dimensions of $q_m := e_m.query$ and of $q_{m+1} := e_{m+1}.query$ and which are restricted on the 'all' level (these are the dimensions which are omitted in the graphical representation) automatically have a change probability of *0* because this level has only one element by definition. If a dimension is a selection dimension and the granularity (the selection level) changes between two queries, the restriction element must have changed too. Therefore, the change probability is automatically 100%.

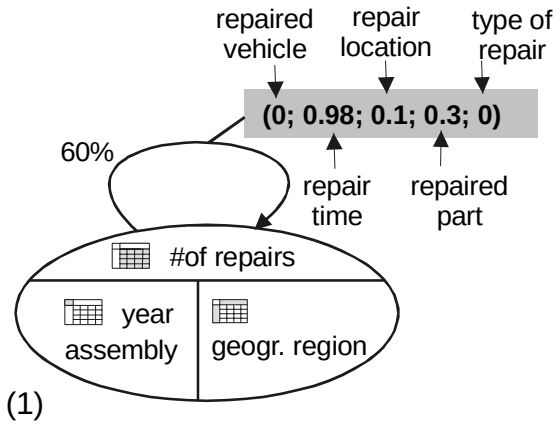

Figure 4.8: A Probabilistic Change Vector

The following definition summarizes the design of the pattern interaction model, which we use to formally describe regularities in user behavior throughout this thesis. According to Definition 2.7 this includes the definition of a pattern representation language (we use the rule based formalism, see above), the generalization expressions used in the definition of patterns (i.e., the function *Q* that computes the query sequence of a session, the function computing the structural query prototype $\wp_{\text{Struct}}$ and the functions computing the value-based prototype $\wp_{\text{Val,i}}$ for the different dimensions) and the pattern schemata (PS1-3, see above). All pattern schemata contain the maximum length of the look-back window as a parameter. As already mentioned, these are tuning parameters that have to be determined specifically for each domain. In order to allow for determining the window sizes at runtime, we formulate the interaction model such that the look-back window sizes are parameters to the definition. As it is possible to use different look-back window sizes for the different pattern types (structural and each value-based), this results in n+1 parameters ($m_{Struct}, m_{Val_1}, \ldots, m_{Val_n}$).

Definition 4.6 (PROMISE/OLAP User Interaction Model)

The PROMISE/OLAP Interaction Pattern Model based on the interaction model IM_{OLAP} is defined as $PM_{\text{IM/OLAP}} = (L_{\text{OLAP}}, G_{\text{OLAP}}, PS_{\text{OLAP}})$, where

- the pattern representation language L_{OLAP} contains first-order logical expressions with different sorts of typed variables for canonical queries, events and sessions.
- The set of generalization functions G_{OLAP} contains the functions computing the *structural* and the *value-based prototype* $\wp_{\text{Struct}}$ (cf. Definition 4.4)and $\wp_{\text{Val,i}}$ (Definition 4.5) generalizing atomic events. Thus $G_{\text{OLAP}} = \{ \wp_{\text{Struct}}, \wp_{\text{Val,1}}, \ldots, \wp_{\text{Val,n}} \}$.
- The set of pattern schemata contains the three pattern schemata introduced above (PS 1-3), Thus, $PS_{\text{OLAP}} = \{$

$$\left(\begin{array}{l} \forall S \in E^{C}_{IM\,/\,OLAP} : \prec e_1, \ldots, e_{m_{\text{Struct}}+1} \succ \subset S \wedge \\ \wp_{\text{Struct}}(e_1) = \wp_{\mathbf{1}} \wedge \ldots \wedge \wp_{\text{Struct}}(e_{m_{\text{Struct}}}) = \wp_{\mathbf{m}_{\text{Struct}}} \Rightarrow \wp_{\text{Struct}}(e_{m_{\text{Struct}}+1}) = \wp_{\mathbf{m}_{\text{Struct}}+\mathbf{1}} \end{array} \right),$$

$$\left(\begin{array}{l}\forall S \in E^{C}_{IM/OLAP} :\prec e_1,\ldots,e_{m_{Val,i}+1} \succ \subset S \wedge \boldsymbol{v}_{\boldsymbol{m}_{Val,i}+\boldsymbol{1}} \neq \boldsymbol{v}_{\boldsymbol{m}_{Val,i}} \wedge \\ \wp_{Val,\boldsymbol{i}}(e_1) = \boldsymbol{v}_{\boldsymbol{1}} \wedge \ldots \wedge \wp_{Val,\boldsymbol{i}}(e_{m_{Val,i}}) = \boldsymbol{v}_{\boldsymbol{m}} \Rightarrow \wp_{Val,\boldsymbol{i}}(e_{m_{Val,i}+1}) = \boldsymbol{v}_{\boldsymbol{m}_{Val,i}+\boldsymbol{1}}\end{array}\right),$$

$$\left(\begin{array}{l}\forall S \in E^{C}_{IM/OLAP} : e_1,\ldots,e_{m_{\text{Struct}}+1} \subset S \wedge \\ \wp_{Struct}(e_1) = \wp_{\boldsymbol{1}} \wedge \ldots \wedge \wp_{Struct}(e_{m_{\text{Struct}}+1}) = \wp_{\boldsymbol{m}_{\text{Struct}}+1} \\ \Rightarrow \wp_{Val,\boldsymbol{i}}(e_{m_{\text{Struct}}}) \neq \wp_{Val,\boldsymbol{i}}(e_{m_{Struct}+1})\end{array}\right)$$

}

The parameter m_{Struct} denotes the size of the look-back window for the structural prediction, and the parameter m_{Val_i} ($i \in [1;n]$) denote the size of the look-back window for the value-based prediction for dimension d_i. ◆

As argued before, we represent the pattern instances of the above model as Markov models which are passed as arguments to the prediction process. The following definition defines the data structure we use to represent the prediction information according to the above pattern model (called Prediction Profile).

Definition 4.7 (Prediction Profile):

A Prediction Profile for an *n*-dimensional cube schema C_Ψ is defined as a 3-tuple $\Phi_C(m_{\text{Struct}},(m_{Val_1},\ldots,m_{Val_n})) = (spm, VPM, \omega)$ where

- $spm = (m_{\text{Struct}}, state(\wp_{Struct}(\Theta_{C_\Psi})), P_{spm})$ is a Markov model of order m_{Struct}. The state space is constructed from the structural query prototype space $\wp_{Struct}(\Theta_{C_\Psi})$. P_{spm} is the transition probability function that represents the probabilistic pattern information (i.e., the conditional possibility of a transition). The model is called the *structural prediction model*.
- $VPM = (vpm_1,\ldots,vpm_n)$ is a set of *n* Markov models (one for each dimension d_i of the cube schema). Each $vpm_i = (m_{\text{Val}_i}, state(\wp_{\text{Val},i}(\Theta_{C_\Psi})), P_{vpm_i})$ with $i \in [1;n]$ is a Markov model of order m_{Val_i}. The state space is constructed from the value-based prototype space $\wp_{Val,i}(\Theta_{C_\Psi})$ of dimension d_i. It is called the *value change prediction model* for dimension d_i. P_{vpm_i} is the transition probability function computing the conditional transition probabilities.
- $\omega : \left(state(\wp_{\text{Struct}}(\Theta_{C_\Psi}))\right)^{m_{\text{Struct}}+1} \rightarrow [0;1]^n$ is a function that assigns a *probabilistic change vector* to every transition of the structural prediction model *spm*.

◆

Example 4.4 (Prediction Profile)

Let us assume that all the look-back window sizes of the structural and the value change prediction models are chosen to be 1 i.e.,

$$m_{\text{Struct}} = m_{Val_1} = \ldots = m_{Val_5} = 1$$

This means, we can use Markov diagrams to visualize the different prediction models. In our example, the *structural prediction model* contains the structural prototypes of queries against the example cube C_{repair}:

$$spm = (1, state(\wp_{Struct}(\Theta_{C_{repair}})), P_{spm})$$

Figure 4.6 shows an extract from a possible *structural prediction model*. The transition probability from *state*($\wp_1$) to *state*($\wp_3$) is 0.3. Therefore,

$$P_{spm}(state(\wp_3), state(\wp_1))=0.3$$

This transition corresponds to the following pattern that is an instance of the pattern schema (PS1):

$$\forall S \in E^{C}_{IM/OLAP} :\prec q_1, q_2 \succ \subset Q(S) \wedge \wp_{\text{Struct}}(q_1) = \wp_1 \Rightarrow \wp_{\text{Struct}}(q_2) = \wp_3$$

Value change prediction models for the *repair time* dimension (d_2) and the *repair location* dimension (d_3) are shown in Figure 4.7. They contain states that represent classification nodes of the classification lattice for the corresponding dimensions.

$$vpm_2 = (1, state(dom(\Psi|_{\text{location}})), P_{vpm_2})$$

The transition probability from *state*(1999) to *state*(2000) is 0.41. Therefore,

$$P_{vpm_2}(state(2000), state(1999)) = 0.41$$

In contrast to the *spm*, the value change prediction models do not contain transitions from a state to itself, as they are only determined to predict changes of values (see definition of according pattern schema PS2).

The probabilistic change vectors are stored with the transitions of the *structural prediction model*. Figure 4.8 depicts a sample probabilistic change vector ϖ for the transition from *state*($\wp_1$) to *state*($\wp_1$) of the structural prediction model shown in Figure 4.6. The probability that the restriction element for dimension 2 (*repair time*) changes between two subsequent queries that both have a structure corresponding to *state*($\wp_1$) is 98%. ◆

Prediction Profiles can either be built by domain experts during a conceptual modeling process or can be induced from query log files that contain the queries that were executed in the past (cf. Section 4.4). The next section discusses the design of a prediction algorithm that is based on the pattern model introduced in this section.

4.3 The Prediction Algorithm

Having defined our pattern interaction model using Markov models as the pattern representation language with the according generalization functions, this section can now present the prediction algorithm based on this representation of patterns. First, Section 4.3.1 presents a general prediction algorithm predicting state transitions in Markov models. Based on this general algorithm, Section 4.3.2 introduces the PROMISE/OLAP prediction algorithm for predicting OLAP queries using a set of interconnected Markov models for the prediction.

4.3.1 A General Prediction Algorithm Using Markov Models

Designing a prediction algorithm based on a discrete time Markov model of order-*m* *DTMM*=(*m*, *S*, *P*) is rather straightforward. At any point during a session, the context vector

$C=(s^{t-m}, ...,s^{t-1})$ containing the last m states that have been visited has to be recorded. This is being passed to the prediction process together with the Markov model M. Using the probabilistic transition function P, a set of candidate states can be computed, by finding all successor states that have a probability of greater than zero.

As already pointed out in Section 2.5, different applications of the prediction process have different requirements on how to choose the relevant predictions from the set of possible successor states. In order to reflect a large variety of applications and to keep this description as general as possible, we assume the following abstract evaluation procedure: first, each predicted state is assigned a score which reflects its relevance for the application. Of course, this value is mainly based on the predicted probability for this state to be the next active state. However, other application-specific cost and benefit factors can be taken into account. This means, that a single prediction is characterized by a tuple (s, sc), where $s \in S$ is the state and $sc \in \mathbb{R}$[40] is its application-specific score. The second step consists of choosing the result set (i.e., the predictions that will be returned to the calling application) from the set of candidate predictions. Again, this procedure can be highly dependent on the characteristics of the application. E.g., an application might only need the $k>0$ best scoring predictions, all predictions with a score above a certain threshold, or as many best scoring predictions such that an additional constraint is fulfilled (for example that the results of the predicted queries fit into a limited cache space or that the predicted queries can be executed within a given timeframe based on a cost-model).

In the design of the prediction algorithm, we mirror this two-step evaluation process by means of two application-specific functions that are passed as parameters to the prediction algorithm:

- a scoring function $f : S \times [0;1] \rightarrow \mathbb{R}$. This function $f(s,p)$ computes the benefit score of a predicted state from information about the successor state s itself and its predicted probability p. The simplest case of such a function is the PROB function that returns the predicted probability of the state as the score. Thus, PROB$(s,p)=p$,
- an evaluation function $e:2^{(S\times\mathbb{R})} \rightarrow 2^{(S\times\mathbb{R})}$ that determines which predictions (tuples of state and score) should be returned to the application. Common general-purpose examples for such a function are:
  - TOP_N: this function chooses the $N>0$ states with the highest scoring value. Thus,

    $$c \in TOP_N(cset) :\Leftrightarrow c \in cset \wedge \left|\{c_i \in cset \,|\, c_i.prob \geq c.prob\}\right| \leq N$$

  - THRESHOLD_t this function chooses states which have a score of at least t. Thus,

    $$c \in \text{THRESHOLD}_t(cset) :\Leftrightarrow c \in cset \wedge c.prob > t$$

Both functions have to be provided by the application and are treated by our abstract algorithm as 'black-boxes' in the sense that we do not make any additional assumptions besides the signature of these functions. However, we will discuss later how knowledge about properties of these functions can be used for improving actual implementations of the algorithm.

[40] $\mathbb{R}$ denotes the set of real numbers.

```
01  struct MarkovPrediction {
02     state: MarkovState;
03     score: Score;
04  }
05
06  function Predict_Markov( MarkovModel M,
07                           SessionContext C,
08                           ScoringFunction f,
09                           EvaluationFunction e ): setof(Prediction)
10  {
11     // precondition: context C contains at least m states,
12     // where m is the order of the Markov Model M
13     PRECONDITION( length(C) >= M.order )
14     declare TempResultSet: setof(MarkovState);  // empty set for results
15     declare Candidate: Prediction
16     TempResultSet = ∅
17     forall s∈M.S where ( M.p(s, tail(C, M.order)) > 0) do
18     {
19        Candidate.state = s
20        Candidate.score = f(s,M.p(s,C))        // evaluate scoring function
21        TempResultSet = TempResultSet ∪ {Candidate}
22     }
23     return e(TempResultSet);
24  }
```

Figure 4.9 Pseudo Code for the General Markov Model Prediction

Figure 4.9 contains the according generic prediction algorithms as pseudo-code. The first step of the algorithm is to determine the set of potential successor states (line 17) for the current context i.e., the states that have a transition probability greater than zero. For each of these states, the scoring function has to be evaluated. The probability of the corresponding transition is passed to the scoring function as a parameter (line 20). The prediction consisting of the state itself and its score is added to the temporary result set (line 21). After processing each candidate successor, the evaluation function determines which predictions are contained in the result set that is being returned to the calling application (line 23).

This algorithm is the basis for defining the OLAP query prediction algorithm in the next sections. Considerations about the implementation and optimization of the algorithm, especially the data structures involved and an analysis of the runtime complexity can be found in Section 4.3.3.

4.3.2 The OLAP Query Prediction Algorithm

This section describes the prediction algorithm for predicting a set of OLAP queries using a combination of different Markov models. The parameters to the algorithm (cf. Figure 4.10) are

- the prediction profile $\Phi_C(m_s,(m_{v_1},\ldots,m_{v_n}))$ (as described in the previous section),
- the current session context $C=q_{\text{t-m},\ldots,}\, q_{\text{t-1}}$. These are the last m queries of the session in canonical form, where m is the largest order of the Markov models contained in the prediction profile, thus $m=\max(m_s,m_{v_1},\ldots,m_{v_n})$,
- a scoring function $f:\Theta_{C_\Psi}\times[0;1]\rightarrow\mathbb{R}$ that is passed to the algorithm by the application and which evaluates the relevance of a predicted query q for the application depending on the probability of the prediction p,

- an evaluation function that chooses the query predictions that are returned to the calling application from the set of candidate queries.

Notably, the OLAP prediction algorithm has the same interface as the general prediction algorithm for a single Markov model introduced in the previous section. Therefore, it hides all the details of the pattern model (the design decisions to use Markov models, the generalizations and the specific pattern schemata). This complies with our objective to keep the PROMISE framework modular in order to be able to exchange the pattern model without affecting the applications.

Internally, the algorithm uses the different prediction models contained in the prediction profile and combines their results. Basically, the algorithm calls the prediction method for each of the models of the prediction profile. Each of these models produces a set of predictions (called partial result set) that are then combined by the algorithm to the result set that contains predictions for complete canonical OLAP queries. Without any restrictions, the cardinality of the result set is the product of the cardinalities of the partial result sets and can contain a lot of predictions with very low probabilities. Although the algorithm exploits integrity constraints between the models (see above) to restrict the predictions to well-formed queries, this set can still be quite large. The scoring and the evaluation function passed to the OLAP prediction process as parameters are designed to reduce the overall result scoring and evaluating predictions of complete canonical queries. This means that they cannot be directly used to restrict the set of predictions for the generalized models.

Therefore, we internally use a threshold method based on the transition probabilities to restrict the cardinality of the partial result sets. The appropriate threshold *t* is an internal parameter to the algorithm and states the minimum probability that a partial prediction must fulfill in order to be considered for the combination process. A threshold of 0 means that all of the partial predictions are used. In Chapter 6, we will empirically evaluate the impact of this parameter on the performance of the algorithm.

The **initialization phase** of the algorithm (line 17-19 in Figure 4.10) computes the generalizations for the queries contained in the context *C* as these will be needed as parameters for predictions using the different prediction models. Line 17 computes the structural generalization for every query and line 18 computes the value-based generalizations for the queries of the context for each dimension. Additionally, the candidate set (variable *cand_set*) containing candidates for predictions is initialized to the empty set (line 19).

According to our observation that the query structure mirrors the macro structure of the analysis process, the **first prediction phase** of the algorithm predicts candidates for the structure of the next query using the structural prediction model $\Phi.spm$ (line 22). The structural generalization of the context (s_*cont*) is used as input to the basic prediction algorithm. The results are scored by their probability (function PROB) and all candidates that have a score (probability) above a certain THRESHOLD *t* (a parameter of the algorithm) are added to the set of structural candidates (variable *s_cand*).

The structural query prototypes predicted by the first phase contain information about the granularity vector *G* and the result measures *M* of the next query. What is missing is information about the restriction vector *R*. Consequently, the **second prediction phase** of the algorithm augments the structural predictions with predictions about restriction elements in order to generate complete canonical queries. This phase is executed separately for each prediction *s_pre* that is contained in the set of structural predictions *s_cand* (cf. line 23). The structural prediction *s_pre* contains a predicted structural prototype (*s_pre.state*) and its probability

score (*s_pre.prob*). Using the predicted structural prototype (*s_pre.state*) together with the structural context (i.e., the structural prototypes of the last m_{Struct} queries; variable *s_cont*), it is possible to determine the probabilistic change vector (ω, line 24).

```
00 struct QueryPrediction {
01   query:  canonical query;
02   sc: score;
03 }
04
05 PredictOLAPQuery(  Context C,  // c contains last m queries
06                    Profile Φ,  // prediction profile (Definition 4.7)
07                    Scoring function f,
08                    Evaluation function e): set of (Query Prediction)
09 {
10
07   declare cand_set: set of (QueryPrediction)
08   declare s_cont: structural prototype[1..(Φ.m_S)]
09   declare v_cont: value prototype[1..n][1..max(Φ.m_v,i)]
10   declare s_cand: set of (structural predictions)
11   declare v_cand: set of (value predictions)[1..n]
12   declare ω: probability[1..N]
13   declare query: CanonicalQuery
14   declare prob: probability
15
16 // generalize contexts
17   s_cont[i] = ℘_Struct(C[i]); 1 ≤ i ≤ m_Struct
18   v_cont[j][i] = ℘_val,j(C[i]); 1 ≤ i ≤ m_Val ; 1 ≤ j ≤ n
19   cand_set = ∅
20
21   // predict candidates for query structure
22   s_cand := Predict_Markov( Φ.spm, s_cont, PROB, THRESHOLD_t)
23   for each (s_pre ∈ s_cand){
24     ω := Φ.ω(℘(s_cont), s_pre.state)
25     v_cand[j] = ∅;   1 ≤ j ≤ n
26     for each dimension d_i do {
27       if (ω[i] ≥ t) {
28         if s_pre.state.Σ[i] == TRUE {  // d_i is selection dimension ?
29           v_cand[i] = Predict_Markov(Φ.vpm_i, v_cont[i],
30                                      PROB, LEVEL-THRESHOLD_t,s_pre.state.G[i])
31         else
32           v_cand[i] = Predict_Markov(Φ.vpm_i, v_cont[i],
33                                      PROB, RESULT-THRESHOLD_t,s_pre.state.G[i])
34         }
35       }
36       if ((1-ω[i]) ≥ t)
37         v_cand[i] = v_cand[i] ∪ {(v_cont[i][1], 1-ω[i])}
38     }
39     for each (v ∈ Πv_cand[i]) do {
40         query.M = s_pre.state.M
41         query.G = s_pre.state.G
42         query.R[i] = v[i].value ; 1≤i≤n
43         prob = s_pre.score * Πv[i].score;
44         cand_set = cand_set ∪ {(query, f(query,prob))}
45     }
46   } // for each s_pre
47   return e( cand_set );
48 }
```

Figure 4.10 Pseudo Code of the Prediction Algorithm

As the value change patterns of different dimensions are considered to be independent (cf. assumption 2 above), the restriction elements are predicted separately for each dimension of the cube schema (cf. line 26; the predicted restriction values for dimension d_i are stored in the variable *v_cand*[i]). The decision which method is used to predict the restriction value is dependent on the value of the probabilistic change vector for the corresponding dimension. If this value is above the threshold *t* (cf. line 27), the algorithm assumes that the restriction value has a good chance of changing. This means that the value-based prediction model is used to determine a set of candidate values. However, using the integrity constraint between the structural and the value-based model, it is possible to restrict the number of candidate states that have to be considered by the value-based prediction in the following way (cf. line 28):

- For a selection dimension $d_i \in \sigma(q)$, the restriction element must be from the same level as the predicted result granularity g_i (cf. case 1 in Figure 4.11). Consequently, we parameterize (cf. line 29/30) the generic Markov prediction algorithm with the evaluation function LEVEL-THRESHOLD $_{t,l}$ that is defined as follows:

$$c \in \text{LEVEL} - \text{THRESHOLD}_{t,l}(cset) :\Leftrightarrow = c \in cset \wedge c.prob > t \wedge level(c.state) = l$$

- For a result dimension $d_i \notin \sigma(q)$, it must be from a level which is greater than the level of the element according to the classification lattice of the dimension (cf. case 2 in Figure 4.11). This integrity constraint is reflected in the evaluation function RESULT-THRESHOLD $_{t,l}$ which is defined as follows (cf. line 32/33):

$$c \in \text{RESULT} - \text{THRESHOLD}_{t,l}(cset) :\Leftrightarrow = c \in cset \wedge c.prob > t \wedge level(c.state) >_{\Psi} l$$

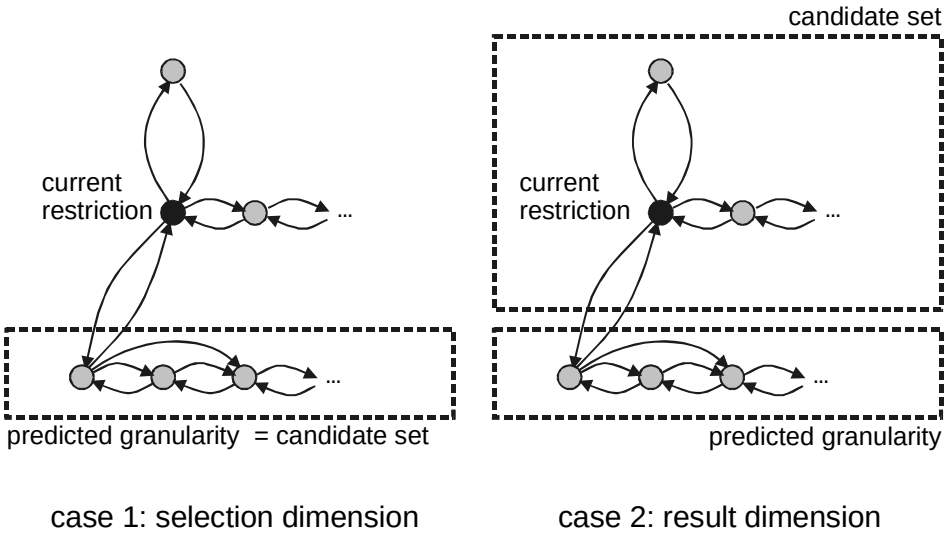

Figure 4.11: Restricting the Set of Candidate Successors using the Structural Prototype Information

The change vector contains the predicted probability that the according restriction element changes. That means that the predicted probability for the restriction element not to change is computed as (1-value of the value change vector). If this value is (also) above the threshold *t* (line 36), the algorithm (also) adds the current restriction element to the candidate set with the corresponding probability. Notably, it is possible that both a changed element and the original element are added to the set of candidate restrictions. E.g., if the threshold is set to 0.3 and the value of the probabilistic change vector is 0.4, both the change probability (0.4) and the probability for an unchanged value (0.6) are above the threshold.

After the value prediction has been done for all dimensions the algorithm starts the **combination phase**. A complete canonical query is constructed for every element of the cross product of the value-based partial prediction sets (cf. line 39). The result measures and the result granularity are copied from the structural prediction (cf. line 40/41) while the restriction value is copied from the corresponding value prediction (cf. line 42). The probability of the prediction is computed as the product of the prediction probabilities (line 43) and then the candidate set is passed to the application defined scoring function and added to the candidate set (line 44).

After performing these steps for all elements of the structural candidate set, the algorithm passes the candidate set to the evaluation function and subsequently returns it to the calling application.

Example 4.5 (Prediction Algorithm)

For reasons of simplicity let us assume that all Markov models are of order 1 and that the scoring function is PROB (probability as score). Additionally, the threshold parameter should be set to t=0.4. The last query that was executed by the user (the current session context C) should be query $q_{t-1}:=q_{example}$.

```
C[1] = q_example= ( {#repairs},
       ( all vehicles , 1999, Germany      , steering, all types ),
       ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

First, the structural (cf. Example 4.2) and value-based prototype of this query is computed (line 17/18).

```
s_cont[1] = ℘_Struct(q_example)= ℘_1
       ( {# of repairs},
         ( vehicle.all , year, geogr. region, assembly, type.all ),
         ( 1,1,0,1,1) )
v_cont = [ all vehicles , 1999, Germany      , steering, all types ]
```

The structural context *s_cont* is used for the prediction of candidates for the query structure using the spm shown in Figure 4.6. The current context corresponds to *state*($\wp_1$). Consequently, the structural prediction returns only one prototype that satisfies the threshold (with a score of 60%). This means after executing line 22, *s_cand* has one element:

```
s_cand ={ (℘_1, 0.6) }
```

This means that the next query is predicted to have the same prototype than the last. The next step of the algorithm is to predict all the restriction elements for the different dimensions. For this purpose, the algorithm looks up the probabilistic change vector for the sequence <state($\wp_1$), state($\wp_1$)> (line 24). In our example the change vector is (cf. Figure 4.8):

$$\varpi(\wp_1, \wp_1) = (0,\ 0.98,\ 0.1,\ 0.3,\ 0)$$

For dimension 1 (*repaired vehicle*), 3 (*repaired location*), 4 (*repaired part*) and 5 (*repair type*) the change probability is below the threshold t=0.4, this means that the algorithm does not query the value change prediction models (skipping lines 28-35) and assumes no change at all (cf. line 36) adding the restriction element of the previous query to *v_cand* (line 37). This results in the following partial result sets with one element each:

v_cand[1] = {(all vehicles, 1)}; *v_cand*[3] = {(Germany, 0.9)};

v_cand[4] = {(steering, 0.7)}; *v_cand*[5] = {(all types,1)}

For dimension 2 (*repair time*), the change probability is above the threshold *t* (line 27) and therefore, the corresponding value-based model (Figure 4.7, right hand side) is used for pre-

diction. As dimension 2 is predicted to be a result dimension, the set of candidates can be restricted to the level 'year' (line 29/30). The prediction returns two candidates that satisfy the threshold and the condition that they are from the level 'year':

v_cand[2] = {(1998, 0.55), (2000, 0.41)}

Then, the combination phase constructs the candidate query by computing the cross product of all value candidate sets v_cand[i] (i∈[1;5]). This set contains two elements:

```
Πv_cand[i] = {
   ( all vehicles , 1998, Germany      , steering, all types )
   ( all vehicles , 2000, Germany      , steering, all types ),
   }
```

Each element of this set is combined with the structural prototype $\wp_1$. This results in the following two queries:

```
q1:=   ( {#repairs},
       ( all vehicles , 1998, Germany      , steering, all types ),
       ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
q2:=   ( {#repairs},
       ( all vehicles , 2000, Germany      , steering, all types ),
       ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

◆

This section completely specified the OLAP prediction algorithm for the PROMISE/OLAP framework. The next section will discuss issues in the implementation of the algorithm and will analyze the runtime complexity.

4.3.3 Runtime Complexity and Implementation Issues

This section discusses the most important considerations of the implementation on a general level. A more technical description of the implementation we used for the evaluation of the approach can be found in Chapter 6 or [Sch01].

A central issue of the implementation is which data structure is used in order to represent a Markov model. When discussing the data structure in the following section, we assume that the Markov model is of order one. Nevertheless, our design can easily be applied to Markov models with orders larger than one, as we can use the transformation introduced in Theorem 4.1 to transform an order-*m* model to an order-*1* model.

Mathematically speaking, an order-1 Markov model is isomorphic to a labeled graph structure (the isomorphism maps states to nodes and transitions with a probability larger than 0 to edges). Therefore, central implementation decisions can be classified into decisions about the representation of nodes and decisions about the representation of edges (the adjacency matrix of the graph).

The first design decision concerns the representation of nodes and the access structure used to satisfy associative retrievals of states. For each node, we have to store the state information i.e., the attributes of the event which is being represented by the state. For example in the structural model, the result measures, the result granularities, and the selection vector have to be stored. In order to reduce the storage requirements for this information, the state information can be represented using a bitstring. As we assume that the multidimensional schema is static, we can assign a unique id to every classification level in a classification lattice. Analogously, every member of the domain of the lattice and every measure is assigned an id. Therefore, the storage space in bits needed to represent the state information for the structural model is:

$$\text{sizeof}(s)= \underbrace{\sum_{i=1}^{n}\left(\left\lceil \log_2\left(\left|L(\Psi|_{d_i})\right|\right)\right\rceil\right)}_{\substack{\text{number of bits to represent the}\\ \text{result granularity vector}}} + \underbrace{|M|}_{\substack{\text{number of bits to represent the}\\ \text{result measures}}} + \underbrace{n}_{\substack{\text{number of bits}\\ \text{to represent the}\\ \text{selection vector}}} \quad [\text{bits}]$$

For our example schema introduced in Example 3.16, this results in 17 bits to store the state information for a structural state. The storage requirement for a state of the value-based model is:

$$\text{sizeof}(s)=\left\lceil \log_2\left(dom\left(\Psi|_{d_i}\right)\right)\right\rceil \quad [\text{bits}]$$

The largest classification lattice (in terms of size of the size of the domain) in our example is the repaired part dimension with 1664 members, which would require 11 bits for the storage of a state. These results imply that it should be easily possible to maintain the whole models in main memory.

Having defined the representation for a single node, this section compiles some requirements regarding the data structure used to maintain and query the set of nodes. Although the set of all nodes of a Markov model is finite and known in advance, it is desirable to store only those states in the model that have actually been visited for reasons of space utilization. This means that the set of nodes dynamically grows (and maybe shrinks due to the aging process that discards states from the model if they have not been visited for a long time, cf. 4.4). During the prediction phase, two types of access to set of nodes occur:

- *associative* search operations that retrieve the node which represents the current session context (a session context uniquely corresponds to a single node in the order-1 model).
- *navigational access* finding the successors of a given node. However, this type of access is processed on the data structure used to represent the edges (see below).

According to our above analysis, the main requirements towards the data structure are that the structure is suited for main memory, allows efficient retrieval of single items of the set by means of a fully qualified reference (point-query) and that it allows the dynamic insertion and deletion of items. Therefore, favoring the access time requirements over maintenance times, we choose a hash table data structure to store the set of nodes. Hash tables allow for retrieval of elements in constant time in the best case. However, due to collisions (several elements being mapped to the same hash key) the performance can degrade to a retrieval performance linear in the number of elements stored in the hash table. Nevertheless, the ideal case (no collisions) can be realistically assumed if sufficient space is allocated for the hash table and an effective hashing function is chosen. In our case the maximum number of states stored in the model can be computed in advance. Additionally, the number of states currently stored in the model can be determined when the model is loaded into main memory at system startup time. This implies that a reasonable size for the hash table can be chosen. The hashing function can be based on the binary representation of states (see above). In the (unlikely) case that almost all states of a model have actually been visited, the binary state representation can serve as the hash key guaranteeing no collisions. Therefore, for our theoretical performance analysis, we will assume that a node can be retrieved in linear time from the hash table. Our measurements in Chapter 6 will confirm that this is a realistic assumption.

Alternatively, it would be possible to trade access times for improved maintenance characteristics by using a dynamic tree structure (for example a B^+-tree) to organize the set of nodes.

This would result in logarithmic retrieval times (best and worst case) but would guarantee better maintenance and storage utilization characteristics.

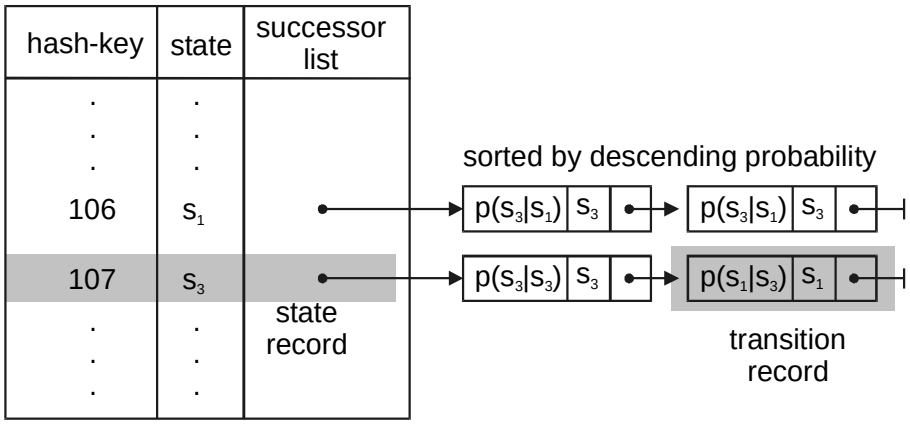

Figure 4.12: Data Structure Used to Store a Markov Model

The second design decision is how to represent the adjacency matrix of the graph. We choose to use the adjacency list approach instead of fully materializing the matrix, as we assume that the adjacency matrix of the graph is extremely sparsely populated. This means that we store a list of references to potential successor states (with the respective transition probability) with every node. We will show later that it is beneficial to keep the list sorted by probability. Figure 4.12 summarizes the considerations about the data structure design presenting the structure of the hash table that organizes the states and the list of transitions stored for every state.

Having outlined the data structure storing the Markov model, we can now perform a first analysis of the runtime complexity of the prediction algorithm. Before analyzing the OLAP prediction algorithm it makes sense to analyze the underlying Markov prediction algorithm. However, a precise analysis of the runtime complexity for the generic algorithm is not possible, as it can be largely influenced by the complexity of the 'black-box' functions. Nevertheless, we discuss the influencing factors in order to identify optimization potentials for an implementation. The runtime complexity is influenced by:

- the runtime complexity of the scoring and evaluation functions. Most of the functions used in practice have a constant complexity as they merely consist of evaluating a simple score formula. Thus, for this discussion, we assume a constant factor. However, this function is evaluated for every potential successor. Therefore, the overall complexity of the scoring phase is linear in the number of considered potential successor states.
- the method used to find the state that corresponds to the current session context and from there to determine the potential successor states (functionally described in line 16). We designed our data structure (see above) such that it allows the retrieval of the current state with constant complexity and the retrieval of each successor state also with constant effort (following the link to the next list element and retrieving the according state information). Thus, this step is also linear in the number of considered successor states.

A straightforward coding of this functional specification of the prediction algorithm requires to always consider all possible successors of a state. This leads to a worst case performance that is linear in the number of states of the Markov Model, as theoretically, each state (including the state *s* itself) can be potential successors of the current state *s*. Although, the average case complexity will be a lot better as the transition matrix of the Markov models is extremely sparsely populated.

For practical implementations however, certain properties of the scoring and evaluation function are known and can be used to greatly reduce (especially the worst-case) complexity of the algorithm: Typical scoring and evaluation functions are monotonous in the following sense: A scoring function is called monotonous if a larger probability value automatically ensures a larger score i.e.,

$$p_1 > p_2 \Rightarrow f(s_1, p_1) > f(s_2, p_2) \quad \forall s_1, s_2 \in S .$$

An evaluation function is called monotonous if for every prediction that is not contained in the result set, no query with a lower score exists that is in the result set. Formally, for two candidate predictions, the evaluation function *e* must fulfill the following property:

$$\forall pre_1, pre_2 \in Cand :$$
$$pre_1 = (s_1, p_1) \wedge pre_2 = (s_2, p_2) \wedge pre_1 \notin e(Cand) \wedge p_2 \leq p_1 \Rightarrow pre_2 \notin e(Cand)$$

Both, the TOP_N and the THRESHOLD_t function satisfy the monotony property. If both the scoring function and the evaluation function are monotonous, we can pipeline the scoring and the evaluation phase of the algorithm in the following way. Instead of computing all the score values first and then starting the evaluation process, it is possible to compute the score of a successor and immediately start the evaluation with a set of only one element. The algorithm terminates if the first prediction is rejected by the evaluation function (due to the monotony properties). Consequently, only those successors have to be regarded that are actually returned to the calling application. This results in a complexity that is linear in the size of the result set.

These results can be transferred to the complete OLAP prediction algorithm. It calls the basic prediction algorithm once with the structural model and then *n*-times for every structural prototype returned by the structural prediction. The structural prediction uses PROB and THRESHOLD as scoring respectively evaluation function. Therefore, the complexity of this call is guaranteed to be linear in the size of the result set *s_cand*. Additionally, the maximum number of successor states that satisfy the threshold *t* is bounded by $\lfloor 1/t \rfloor$ as the sum of all transition probabilities of a state is always less or equal to 1. However, for the value predictions, the special threshold functions used with the value-based prediction models do not satisfy the monotony property defined above such that the best case performance (linear in the size of the result set) cannot be guaranteed for the value-based prediction as the additional restriction of the successors (using their level) cannot be exploited. Nevertheless, the processing of candidate successors during the prediction process (when traversing the list) can still be stopped, when the probability value of a transition drops below the threshold *t*. This again leads to a worst case complexity of $\lfloor 1/t \rfloor$. Thus, the runtime complexity for the OLAP prediction algorithm (worst-case) is:

$$\underbrace{\lfloor 1/t \rfloor}_{\substack{\text{runtime complexity} \\ \text{for structural} \\ \text{prediction}}} + \underbrace{\lfloor 1/t \rfloor}_{\substack{\text{size of} \\ \text{structural} \\ \text{result set} \\ \text{(maximum)}}} \cdot (\underbrace{n}_{\substack{\text{number of} \\ \text{dimensions}}} \cdot \underbrace{\lfloor 1/t \rfloor}_{\substack{\text{complexity of} \\ \text{value based prediction}}}) = \lfloor 1/t \rfloor + n \cdot \lfloor 1/t \rfloor^2$$

This shows that the runtime complexity of the algorithm is constant in the number of states of the Markov model, linear in the number of dimensions (that are considered as static in our case) and quadratic in the number of possible successor states.

An analysis of the complexity of typical maintenance tasks can be found in the next section which covers the creation and maintenance of OLAP Markov models.

4.4 Inducing Interaction Patterns from OLAP Interaction Logs

The previous sections assumed that a prediction profile consisting of *n*+1 (where n is the dimensionality of the cube schema) Markov models exists before the prediction process starts. As already mentioned, such a profile can be generated by induction from former interactions or deduced (i.e., modeled) by experts. This section covers the induction of prediction profiles from a history of OLAP queries observed in the past (in machine learning, this step is called training phase). We assume that the sequence *past* contains all the sessions of the user or user group for which the profile should be built. The sequence is formulated according to the query based interaction model. That means, it contains a list of sessions observed in the past. Once again, we describe the algorithm for an order-1 model without restricting the generality of the presented approach due to the transformation presented in Theorem 4.1.

The induction algorithm is based on a simple frequency counting technique. The central assumption is that if the training set is large enough, the relative frequency for transitions asymptotically approaches the actual probability. Abstractly speaking, the algorithm simulates state transitions of the Markov model according to the sessions contained in the log file. For each transition from state s_1 to s_2, a count variable $F(s_1, s_2)$ records how often a transformation is being taken. The probability of a transition from s_1 to s_2 is then computed as:

$$p(s_2 \mid s_1) = \frac{F(s_1, s_2)}{\sum_{s \in S} F(s_1, s)}$$

The denominator of the above quotient is equal to the absolute frequency $F(s_1)$ of s_1 occurring in a session $Ses \in past$.

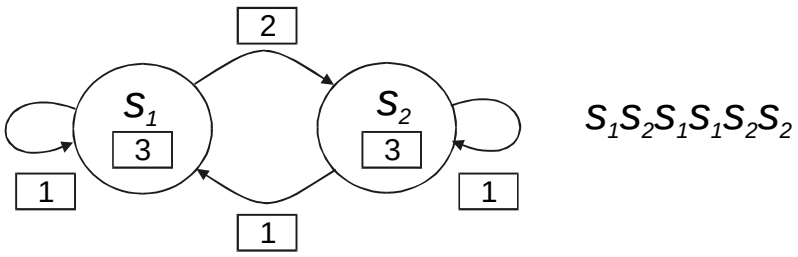

Figure 4.13: Storing Relative Frequency Counts with the Markov Model

The main advantage of this algorithm is that the frequency counts can easily be incrementally maintained online during the operation of the system with a very small overhead. For every transition and every state of the Markov model a count variable has to be stored. The transition variables $F(s_1, s_2)$ are incremented when the corresponding transition is taken by the model (cf. Figure 4.13). The count for the state is updated every time the state becomes active. The probability for a transition can then be determined by computing the following quotient:

$$p(s_2 \mid s_1) = \frac{F(s_1, s_2)}{F(s_1)}$$

Therefore, instead of storing the probability with the data structure described above, the frequency count variables can be stored and the probability can be computed 'on the fly'.

Another important requirement motivated in Section 2.5 was that the model should detect changes in user behavior over time and adapt itself to these changes. In this context, the problem with the relative frequency counting technique presented so far is that the weight for each

interaction is equal irrespective of the time of the occurrence of the event. This means that if a model has been trained with a large set of data, the frequency counts are quite large and newly arriving transaction can only marginally change the behavior of the model. This conflicts with the assumption that transactions perceived in the near past better represent the current behavior of the user. A possible solution to this problem is to use a sliding window technique that only takes the last $t>0$ interactions into account when computing the transition probabilities. Additionally, a weighting function w: $\mathbb{N}\rightarrow[0;1]$ can be used, where $w(\Delta t)$ computes the weight for events with an age of Δt. Different aging functions can be used to represent linear, exponential or logarithmic aging. The drawback of this solution is that the frequency counts cannot be incrementally maintained and that it requires storing the full history of t interactions (note that t must be sufficiently large in order to fulfill the assumption that the relative frequencies found in the sliding window adequately approximate the probabilities). Additionally, the probabilities have to be recomputed after every step of the logical timescale (because the age of the interactions changes).

As this 'accurate' approach is obviously not feasible, we use an approximation using $k>0$ age classes. Instead of using the actual age of the interaction event to determine the aging, we partition the sliding window (i.e., the logical timescale) into $k<<t$ intervals of equal size (Notably, for $k=t$, the approach is equal to the aging described above). Thus, an aging class A_i is defined as the following interval on the logical timescale:

$$A_i = [i \cdot \frac{t}{k}; (i+1) \cdot \frac{t}{k}], \text{ where } i \in [0;k\text{-}1]$$

Additionally, the weight function is generalized in a way that $w(i)$ computes the weight for all events of age class $i \in [0;k\text{-}1]$. An examples for a weighting function are $w(i) = \frac{1}{i+1}$, but alternative weighting strategies are possible, e.g., $w_i = \frac{1}{(i+1)^2}$

Let $F^i(s)$ denote the absolute frequency of state $s \in S$ occurring in sessions contained in *past* within the age-class $i \in \{0,\ldots,k\text{-}1\}$. Analogously, $F^i(s_1, s_2)$ denotes the absolute frequency of $<s_1,s_2>$ occurring is past sessions within the age class $i \in \{0,\ldots,k\text{-}1\}$. Thus, the probability function that takes the age classes and the weighting function into account is the following:

$$p(s_2 \mid s_1) = \frac{\sum_{i=0}^{k-1} w(i) \cdot F^i(s_1, s_2)}{\sum_{i=0}^{k-1} w(i) \cdot F^i(s_1)}$$

The advantage of the model is that it can be maintained incrementally, without storing the history of interaction events by storing a vector of k frequency counts in the Markov model. During the system's operation, only the count of age class 0 is being incremented. After k events, the counts for the age classes are shifted (cf. Figure 4.14) such that $F^0=0$ and $F^i = F^{i-1}$ ($i \in [1;k\text{-}1]$). This can be done during a reorganization phase. However, if such a phase is not possible, this update can be performed using a 'lazy' evaluation strategy, by additionally storing a logical timestamp for every frequency vector. If a frequency count vector is accessed it

can be determined, if the information is up to date by comparing the timestamp to the current logical timestamp. If it is not current it can be updated before accessing it.

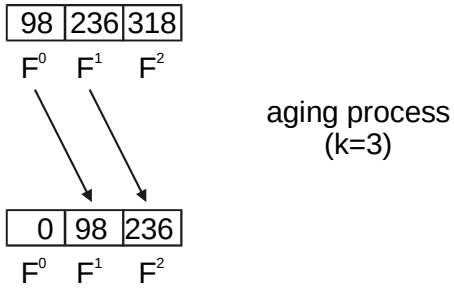

Figure 4.14: Aging the Relative Frequency Counts Using 3 Age Classes

In order to avoid storing a large number of transitions with marginal transition probabilities, it is also possible to deploy a garbage collection algorithm periodically that removes transitions with probabilities below a certain probability and subsequently removing states that are not referenced anymore. The garbage collection can also be done online during the aging process.

```
01  procedure UpdateFrequencyCount(       SessionContext C,
02                                         MarkovState successor,
03                                         MarkovModel M)
04  {
05    declare sr: StateRecord
06    declare tr: TransitionRecord
07    sr = findSessionContext (M,C)  // hash table lookup
08    if (not found(sr))
09      sr = insert(M, C)  // insertation in hash table
10    sr.F[0] = sr.F[0]+1 //increase frequency count
11    tr = findSuccessorState(sr.succesors,successor)  // list traversal
12    if (not found(tr))
13      tr = insertAtTail(sr.succesors, successor) // end of the list
14    tr.F[0]=tr.F[0]+1   // increase frequency count
15    findNewInsertationPoint() // reverse traversal of the list
18  }
```

Figure 4.15: Pseudo Code for Updating the Frequency Count

The algorithm described so far is independent of the OLAP approach and can be applied to any Markov model. When training the Markov models in a PROMISE/OLAP prediction profile, some peculiarities have to be taken into account. First, the according generalization functions have to be applied to the query descriptions characterizing the events (this is being done by the *state* mapping function). Additionally, the value change vector records the probabilities that a value changes between subsequent states of the structural model. Therefore, it can be updated using an equivalent technique as described here (although the change vectors are not represented as a Markov model). Another consequence is that the value prediction models do not need to store the information for transitions from a state to itself. Thus, only subsequences with changing restriction values need to be used to train the value prediction model.

In order to analyze the complexity of keeping the model up to date with the data structure proposed in the previous section, Figure 4.15 sketches the algorithm that updates the frequency count for a given session context *C* and the actually chosen successor state. First, the entry for the current session context (state record) has to be found (cf. line 7). This corresponds to a hash table lookup (constant complexity). If the state is not found, it has to be inserted (also constant complexity, line 9). Then, the frequency count for age class 0 of this state

(the current context) must be increased (cf. line 10). In the next step, the transition record *tr* for the successor state has to be found (cf. line 11). This corresponds to a traversal of the list of successor states which can be of linear complexity in the worst case (if the state is not found). In case the successor state is not in the list of successors for the current context, it has to be inserted at the end of the list. In any case, the frequency count of the transition record (age class 0) has to be increased by one. This increases its weighted transition probability and therefore, it might be necessary, to change the records position in the ordered list. As the probability is always decreased, the new position can be found by traversing the list backwards from the current record. In the worst case this also takes a complexity linear to the length of the list. Thus, the overall runtime complexity of updating a frequency count is linear in the number of successor states for the current context.

4.5 Summary and Discussion

This chapter constitutes the core of this thesis as it presented the pattern model and the according prediction process for OLAP queries. The design of this model required several design decisions that have been thoroughly discussed throughout the previous sections. Our basic design decision was to base the pattern model on the query based interaction model disregarding information about query transformations for the design of the pattern model. That means, the model does not 'hard-wire' any integrity constraints concerning two subsequent OLAP queries (for example requiring that they are similar because they were produced by a limited number of operations) into the pattern schema design. This makes our approach independent of the actual navigational capabilities supported by the actual user interface. However, restrictions in the navigational behavior imposed by the actual user interface are 'learned' by the system during the training phase because they can be expressed as pattern instances. E.g., if a front-end tool with a single execution model only allows the new restriction element of a slice transformation to be from the same level of granularity than the current restriction, this tool-imposed integrity constraint will be automatically mirrored in the patterns that are generated by the induction process in a way that the value-based Markov model only contains transitions between values of the same levels.

The pattern instances are represented using the formalism of Markov models. We showed that a standard mapping of OLAP user behavior to a Markov model is bound to fail because of the large state space and the non existent abstraction ('learning') abilities of this approach. Our solution to the problem is the combined usage of generalized patterns for the prediction. As an example, we introduced two generalizations of canonical OLAP queries: structural and value-based. The definition of these generalizations has been deduced from an analysis of the typical user interface and its query transformations and by analyzing analysis typical procedures. Therefore, it is heuristic and bases on the following assumptions:

1. Patterns concerning the MD presentation structure **are independent** of patterns concerning the restriction values for OLAP queries.
2. Patterns for changes in the restriction values in the different dimensions **are independent** of each other.
3. The probability which of the dimensions changes the restriction element between two queries **is dependent** on the structural session context.

The practical evaluation (cf. Chapter 6) will show that this generalization works very well in typical OLAP environments. An alternative way of deriving generalizations would be the analysis of a considerably large number of interactions (for example using clustering algo-

rithms). However, this means that the prediction process that must support a variable class of pattern schemata (using different generalizations). The challenges involved in designing such an algorithm are discussed in greater detail in Section 7.3.3.2.

Assumption two is a direct consequence of the assumption that the dimensions of a multi-dimensional schema are independent of each other. In case that this assumption is not met for a practical model, it is possible to combine the *k* dependent dimension in one value-based prediction model. The states of this combined model would then represent *k*-tuples of values.

The prediction algorithm presented in Section 4.3 combines n+1 (one structural and n value-based) Markov models for the prediction of a set of possible queries. The most important requirement for the specification of the algorithm itself was to keep it as generic as possible to support a large variety of applications for the prediction results. This requirement was fulfilled by using two black-box functions in the specification of the algorithm. This generality constituted the largest challenge for finding an efficient implementation of the algorithmic specification. The typical accesses of a prediction algorithm can be efficiently handled using a combination of a hash-table and ordered linked lists to represent the Markov models. We also presented an optimization of the prediction algorithm for a class of parameter functions that satisfy a monotony property. In this case, the complexity of the algorithm is linear in the size of the predicted query set.

Section 4.4 presented an algorithm to induce a prediction profile from past user behavior. Using an incremental frequency counting technique, the algorithm allows for maintaining the profiles online during the query processing with a minimal runtime and storage overhead. Additionally, the algorithm supports dynamic aging in order to give a higher weight to more recently observed query behavior.

In summary, the prediction and the induction algorithm provide an efficient framework for analyzing and predicting OLAP queries for different applications. As an example of such an application, the next chapter will discuss the impact of the availability of the prediction of future user behavior on the design of OLAP caching algorithms.

«Destiny is not a matter of chance,
it is a matter of choice;
it is not a thing to be waited for,
it is a thing to be achieved.»
-- William Jennings Bryan

«I have seen the future and
it's like the present, only longer.»
-- Dan Quisenberry.

Chapter 5 Predictive Caching for OLAP Systems

The previous chapter described an algorithm that is capable of predicting a set of queries that are likely to be executed in the near future. However, the prediction itself does not improve the system's performance but has to be applied in order to be useful. When looking at the applications of prediction results in related application areas (Section 4.1), we pointed out that the predominant application of prediction information is the class of prefetching techniques (which includes precalculation, presending and speculative execution). However, in order to store the prefetched results until they are (hopefully) actually requested, an intermediate storage (a cache) is necessary. Most of the approaches consider a separate cache for the prefetching results. However, in real systems the prefetching cache competes with the traditional cache management for resources (mainly main memory). Therefore, we find it essential to regard prefetching and caching in an integrated way (as also proposed in [CFK+95]). Additionally, caching algorithms can also benefit from predictions when making eviction and admission decisions. Thus, the presence of a prediction algorithm influences the caching strategy in two orthogonal ways (cf. Section 2.8):

- the objective of an ideal caching strategy is to store exactly the set of objects that offer the largest cost savings in processing future queries taking into account a space constraint for the maximum cache size. Eviction (and admission) algorithms dynamically adapt the cache content such that this objective is fulfilled. This requires an anticipation of the future reference probability for cached objects. Obviously, the prediction algorithm can provide this information thus augmenting or replacing the traditional heuristic approximations (for example LRU).
- traditional caches are passive in that they only consider objects that have been produced by past queries. An accurate prediction of user behavior enables the cache manager to autonomously initiate query processing (active cache). We call this technique predictive prefetching.

Analogously to the organization of the previous chapters, we start by surveying current state-of-the-art OLAP caching system approaches (Section 5.1). As our approach seeks to augment and improve existing caching strategies instead of replacing existing approaches (classified in Section 5.1.1), our proposal is based on an abstract definition of a query-level cache management system which we describe in Section 5.1.2. This model is general enough to subsume all

the current caching approaches such that our approach can be easily combined with each of these approaches. Then, a separate section is devoted to each of the two different applications: Section 5.2 discusses the design of admission and eviction algorithms that take predictions into account and Section 5.3 describes a prefetching algorithm for OLAP caches using the prediction information. We summarize the results in Section 5.4.

5.1 Semantic Caching for OLAP Systems

5.1.1 Survey of OLAP Specific Caching Approaches

The most successful approach to improve the query performance in the presence of very large raw data sets is to introduce redundancy in the form of materialized views (for example preaggregations). A lot of work has been done in the area of static materialization (e.g., [HRU96], [GHR+97], [BPT97], [Gup97], [YKL97], [TS97], [SDJ+96][41]). These approaches are called static because they assume that the query workload is known in advance and that the choice which views should be materialized is performed at predefined times and is not changed during the operation of the system. These approaches can marginally benefit from the PROMISE approach as the static workload description (which query is executed how often) can be derived from the pattern models. However, these approaches cannot benefit from the conditional reference probabilities (depending on the current session context) provided by PROMISE as they cannot dynamically adjust the set of materialized views.

Regarding these limitations, it is no surprise that recently several dynamic materialization techniques (caching techniques) for OLAP systems have been discussed (e.g., [SSV96], [SSV99], [DRS+98], [ABD+99], [DN00]). These techniques can fully benefit from applying results of the PROMISE prediction process. As the special focus of this chapter is to discuss these enhancements, the rest of this section compiles and compares the fundamental ideas of these dynamic caching techniques tailored specifically to the peculiarities of OLAP systems.

The unifying idea of specialized OLAP caches is to use semantics of the multidimensional data model (for example information about hierarchies and multidimensional transformations) in order to drive caching decisions. This implies that the cached objects are described on the semantic level of a query (query level cache). This results in several important differences in comparison with caching techniques applied in the area of (database) buffer management (for example [SSV96]):

- *Cached objects*. In buffer management, the number of pages is finite and thus each page can be described either by a logical number or by a physical address. However, the set of queries is infinite in general and different descriptions of semantically equal queries exist. This means that each caching strategy must define a canonical query form (cf. discussion in Section 3.2) that can be used to describe the cached objects.
- *Different size of cached objects*. As the cached object is a query result, the memory needed to store the object can largely differ for each object. In OLAP environments, the size even varies in several orders of magnitude. This means that a newly arriving object might well evict several other objects from the cache. This influences the eviction decisions and makes an admission algorithm necessary that evaluates whether a query result should be admitted to the cache by taking into account the benefit of the evicted objects ([SSV96]).
- *Different re-computation costs*. In buffer management, the costs of computing (loading) a page are constant for all pages. However, for a query level cache, each query has a differ-

[41] A very comprehensive overview of materialization techniques can be found in [Leh98b].

ent cost. Thus, every cached object has a different cost saving potential (i.e., the cost that can be saved by using the cached object instead of computing it from raw data).

- *Query rewriting problem*. In buffer management, the pages that are necessary to answer the query (according to the optimized execution plan) can be uniquely identified. This is not the case in query level caches, where each cached object represents a materialized view of the base data. These views are redundant in the sense that certain queries can be computed from different sets of views (often at different cost). This requires a query rewriting component that translates incoming queries using the cached objects (in an optimal way) and the base data. The query rewriting strategy largely influences the eviction strategy.

Not surprisingly, the current approaches mainly differ in how they address the above mentioned peculiarities (the description of cached objects respectively the class of queries being supported, the design of admission and eviction algorithms, the query rewriting strategy and the incorporating classification semantics). Table 5.1 compares the different approaches (described in more detail in the next paragraphs) by these criteria.

The first approach to using dynamic query-level caching in warehouse environments is the WATCHMAN approach ([SSV96], [SSV99]). The class of queries considered by the approach are star-join queries with an additional HAVING clause. Incoming queries are rewritten (split) into two queries: a data cube ([GCB+97]) query (called base query) and a query that computes the answer to the original query using the results of the base query (called slice query). The cache maintains complete result sets of both slice and base queries. Additionally to the cache content, a special data structure (called query containment graph) is maintained that records the derivability of queries. This containment graph is used for query subsumption testing. The subsumption test for the base queries (data cube queries) makes use of the derivability of different data cube queries (lattice structure). A query can benefit from the cache if either the split query is subsumed by a cached split query or if the base query can be derived from a cached data cube query according to the data cube lattice. Cache admission and replacement decisions are driven by an overall profit metric of the cache content which takes into account the execution cost of a query, the storage cost, reference frequency of the query (approximating the future benefit of a query) and the cost of updating the cache content in case that the raw data changes (cache coherence) and the frequency of changes to the raw data.

The most advanced caching schema is currently developed as part of the Cube Star project (for example [ABD+99]). The class of queries considered by this approach is defined by so called multidimensional objects (MOs) which can express hyper-rectangles of the MD space[42] on different levels of granularity (this corresponds to a star join query in the relational case). All objects managed by the cache are also MOs. The distinctive feature of this approach is that it does not only use one cached object to answer the incoming query (thus demanding a total containment) but also considers combinations of several cached objects and queries to the base data. Therefore, it requires a complex query rewriting algorithm (called patch working, [Leh98b]). As the problem in NP-hard, the project uses a greedy algorithm to determine the optimal (with respect to a cost model) rewriting for a query in the presence of a set of cached MOs.

The CubeStar cache replacement strategy uses the following factors to assess the benefit of an object: The *weighted relative reference density* of a cached result describes the number of accesses to an object within a period of time taking into account that only part of an object is used to satisfy the incoming queries. The *costs of re-computing* an object form the cache con-

[42] An MO is equally expressive as our notion of a canonical query.

tent and the raw data can be determined by the patch working algorithm. The *absolute benefit* of an object describes the cost saving of accessing a query from the cache instead of computing it from raw data (this measurement is independent of the current cache content). Additionally, the degree of relationship between the cached object and the last query is measured by the number of query transformations that are necessary to transform the current query to the cached query. This approach is a distinguishing feature of the CubeStar caching approach and is similar to our ideas of using a distance metric based on the query transformation definitions. All these measures are combined and divided by the size of the cached object to determine its score for eviction purposes.

[DRS+98] combines the ideas of a query level cache and the advantages of a fixed caching granule. Like the CubeStar approach it aims at using a combination of several cached objects to answer an incoming query. However, the approach does not cache query results but portions of the multidimensional space (called chunks) of predefined regular size (a similar approach is described e.g., in [Fur99]). This makes query rewriting more efficient and avoids the disadvantage of redundant data in the cache. Notably, the chunks require different storage space depending on the sparsity of the data. For each incoming star join-query, an algorithm determines the chunks that are present in the cache and the chunks which have to be computed from the raw data. The cache replacement algorithm is a variation of the CLOCK algorithm that incorporates a benefit function taking into account the cost of re-computing the chunk from the base data.

Approach	Cached Objects	Class of Queries	Query Rewriting/ Derivability	Admission/Eviction Criteria
WATCHMAN ([SSV96], [SSV99])	Query results	Starjoin query with HAVING clause	Query Split (slice query / data cube query) Total containment of query with cached result or total containment of base query.	cost/benefit function taking into account: ■ size of result ■ cost of re-computation ■ cost of maintenance (frequency of change) ■ benefit (reference rate since object is in cache)
CubeStar ([ABD+99], [AGL+99], [Leh98b])	Query results	MO (corresponds to Starjoin Query)	Patch Working	■ weighted relative reference density ■ degree of relationship ■ cost of re-computation ■ absolute benefit ■ size
Chunk Cache ([DRS+98] [DN00])	Chunk of MD space (regular hyper rectangles)	Starjoin Query	Intersection	Variation of CLOCK algorithm incorporating: ■ cost of recomputation ■ size of the chunk

Table 5.1: Comparison of different OLAP caching techniques

Despite their different solutions to the caching problem, these approaches have several properties in common. All of the algorithms use a benefit estimation for cached objects (for eviction and admission purposes). The main problem in computing the anticipated benefit for a cached object is that the future workload is not known. The algorithms therefore approximate the future benefit of an object by its reference count since it is stored in the cache. However, this technique only works for objects which have been in the cache for a certain amount of time, as

for newly arriving objects, no reference count is being maintained[43]. Another drawback of this estimation is that it does not take the current session context into account (i.e., the last queries actually executed) as it only considers a global reference count. Additionally, this heuristic only works for repetitive patterns (for example loops) and hot spots. Regarding the fetching strategy, all of the presented OLAP caching approaches assume a demand fetching strategy thus only considering objects that have been produced by recent queries.

As the results are still too new to judge their practical impact, it is essential to design our approach in a way that it can be combined with any of these caching techniques. All of the proposed algorithms show a great potential for improvement by speculative execution techniques (predictive prefetching strategies) as the current algorithms exclusively deploy demand fetching strategies and all of them use benefit estimations. Instead of extending a selected algorithm, in the next section, we will design an abstract model of a query level OLAP cache and will base our approach on this abstract model. In this way, we ensure the adaptability of our techniques to all the proposed algorithms.

5.1.2 An Abstract Model for Multidimensional Query Result Caches

This section describes an abstract process and data flow model for a query level cache in an OLAP system. The model is designed in a way, that the caching approaches presented in the previous section can be regarded as instances of the abstract framework. Thus, all of the PROMISE extensions described with respect to the abstract model can be applied to all of the caching approaches.

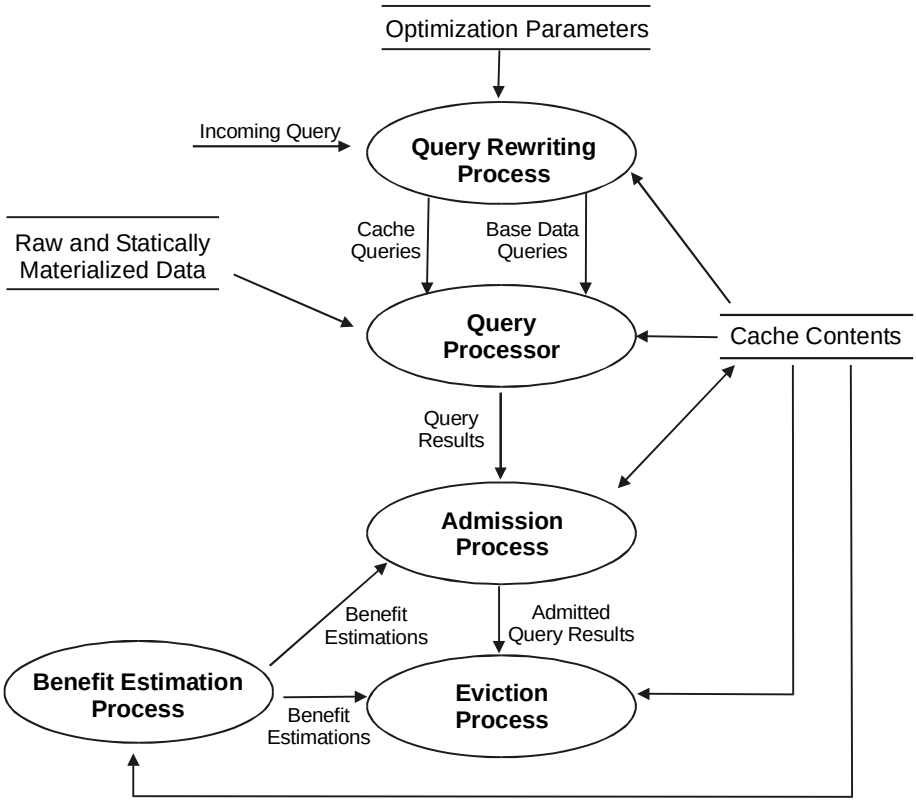

Figure 5.1: The Functional Architecture (Data Flow) of a Query Level Cache Management System

[43] The approach of [SSV96] tries to overcome this problem by maintaining reference information of objects that have already been evicted from the cache. This is a pragmatic approach of building access pattern information however not taking into account navigational user behavior.

Figure 5.1 shows the outline of the abstract caching framework. We start by describing the cache content (right hand side) i.e., the objects managed by the cache. As we have described in the previous section, all OLAP caches assume a query level cache. This means that the cache manages results of multidimensional queries[44]. As the query formalism is closed, a query result is formally also a cube instance (cf. Definition 3.13)[45]. Therefore, the cache manages tuples of canonical query descriptions and their according cube instances.

Definition 5.1 (OLAP Cache Object, OLAP Cache Content)

An OLAP cache object γ is defined as a tuple $\gamma=(q, \rho)$, where

- $q \in \Theta_C$ is the description of a canonical OLAP query
- $\rho \in \Xi(C)$ is a multidimensional cube instance containing the results of the query q_C

The content $\Gamma=\{\gamma_1,\ldots, \gamma_k\}$ of an OLAP cache is a set of OLAP cache objects γ_i ($i \in [1;k]$). ◆

For our approach, we assume a query level cache which has a limited size *MAXSIZE*. That means that the sum of the size of all cached objects must be smaller than the maximum cache size at all times. The size needed to store a cached object (q, ρ) is mainly determined by the size of the result ρ (see Section 5.1.3 for a calculation of the result of a canonical query).

The main task of the cache management framework is to reduce query execution times by using cached results to answer incoming queries. Therefore, for the description of the algorithms it is essential to define when a cached object can be used to answer a query. The simplest case (which all more complicated algorithms are based on) is total containment (or subsumption[46]). If a canonical OLAP query can be fully answered using a cached object, we say that the cached query result *subsumes* the query result (or short the cached query subsumes the query).

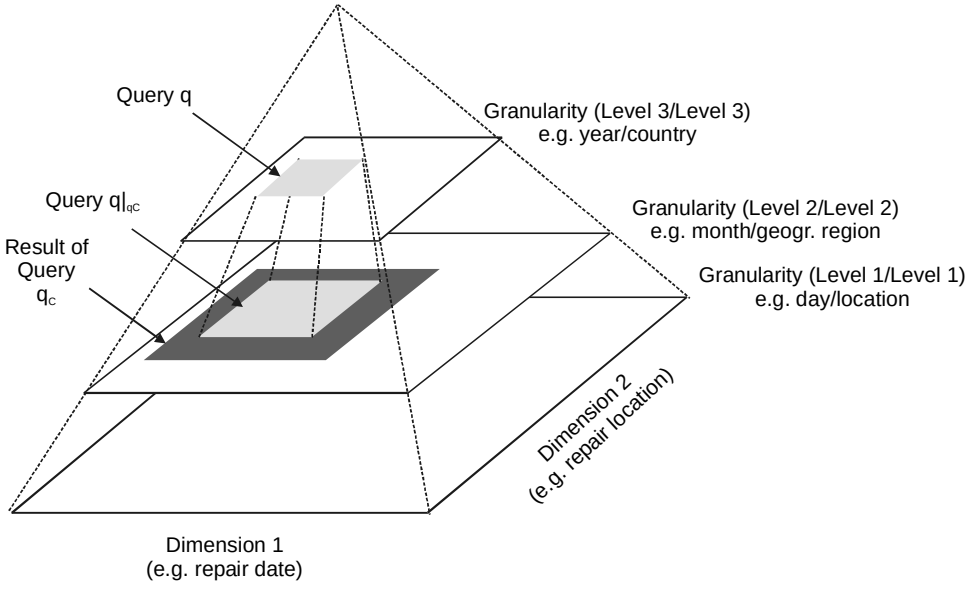

Figure 5.2: Subsubsumption of Queries in the Presence of Hierarchies

[44] Notably, the cached objects must not necessarily exactly correspond to queries that have been executed by the user, but may be the result of a rewriting process (see below).

[45] Although those query result cube instances use the same dimensions (classification levels from the classification lattice), their active domains may differ due to restrictions contained in the query.

[46] We deploy the terms query subsumption and query containment synonymously in this thesis.

In order to have this property, a cached query result q_C must have a finer or equal level of granularity (according to the classification lattice) than the incoming query q in each dimension. Otherwise, no aggregation of the cached data to the required granularity would be possible because no classification schema path exists. Additionally, it must contain at least all of the measures of the incoming query. The third property demands that the hyper-rectangles implicitly defined by the restriction elements on the base cube must fully overlap. Otherwise (in the case of a partial overlap), values would be missing from the aggregation leading to wrong aggregation results. Figure 5.2 visualizes a two-dimensional case where query q_C (result granularity for both dimensions is level 2, e.g., *month* and *geogr. region*) subsumes a query q (result granularity for both dimensions is defined on level 3, e.g., *year* and *country*). Vividly, the subsumption can be tested by 'projecting' the query box of the subsumed query to the result granularities of the subsuming query (depicted as $q|_{q_C}$ in Figure 5.2) and testing both query boxes for geometric containment. A formal definition of $q|_{q_C}$ will be given in Section 5.1.3.2. The following definition uses these properties to define the subsumption relationship between queries:

Definition 5.2 (OLAP Query Subsumption)

A (cached) query $q_C = \{M_C, (r_{C1},\ldots,r_{Cn}), (g_{C1},\ldots,g_{Cn})\}$ subsumes a canonical query $q = \{M, (r_1,\ldots,r_n), (g_1,\ldots,g_n)\}$ if and only if the following conditions are fulfilled:

- $M_C \supseteq M$
- $g_{Ci} \leq_\Psi g_i \quad \forall i \in [1;n]$
- $\text{active}_{d_i}(q_C) \supseteq \{x \in dom(g_{C_i}) \mid x \in \text{descendants}_\Psi(r_i)\} \quad \forall i \in [1;n]$ [47]

We express query subsumption using the relation symbol *subsumes*: $\Theta_C \times \Theta_C$ and write q_C subsumes q. ◆

Example 5.1 (OLAP Query Subsumption)

Let us once again regard our example query q_{example}.

```
q_example := ( {#repairs},
             ( all vehicles , 1999, Germany      , steering, all types ),
             ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

If the cache contains the results of the following query q_1, we can completely derive the results of q_{example} from the results of q_1, thus q_1 subsumes q_{example}.

```
q_1 :=    ( {#repairs, duration},
          ( all vehicles, all times, all locations, steering, all types ),
          ( vehicle class, month    , geogr. region, assembly, type.all  ) )
```

The subsumption can be checked using the three properties given in Definition 5.2: q_1 contains a superset of the measures of q_{example}. Additionally, the classification level determining the result granularity for q_{example} is larger or equal than q_1 for all of the dimensions. Finally, the data areas implicitly defined by the classification lattice instances (hierarchies) and the restriction elements overlap in the required way. E.g., in the *repair time* dimension (2), the active set of classification nodes contains all months for all of the years contained in the cube instance thus:

[47] The active set of classification nodes for a level is defined in Definition 3.17. Notably, if only one hierarchy is defined for a dimension, it is sufficient to check if the restriction elements of q are identical or descendants of the restriction elements of q_C. However in the general case the containment must be tested on the set of active classification nodes like stated in the condition.

$$active_{\text{repair time}}(q_1)=\{\text{Jan95, Feb95,…,Dez00}\}$$

The set of classification nodes on the result granularity of q_1 (Month) that are selected by the restriction element '*1999*' contains all the months of *1999* (Jan99, Feb99,…,Dec99) which is a subset of $active_{\text{repair time}}(q_1)$.

However, the following query q_2 does not subsume q_{example} because condition 3 is violated as the set of active classification nodes in dimension *repaired vehicle* (1) of q_2 only contains the element 'Truck', while q_{example} would need the values for all vehicle classes.

```
q₂:=    ( {#repairs, duration},
        ( trucks       , all times, all locations, steering, all types ),
        ( vehicle class, month    , geogr. region, assembly, type.all  ) )
                                                                        ◆
```

The cache manager is responsible that the performance gains achieved by exploiting query subsumption are maximized. It consists of different communicating processes (cf. Figure 5.1) which are functionally described in the next sections. The algorithms used to realize these components differ in the different approaches and are influenced by the actual caching strategy.

Notably, as we assume a read-only environment, where updates take place in a very controlled way, we do not address the problem of cache coherence here. Besides, all the standard algorithms for coherence can also be applied in the OLAP cache environment.

The user formulates queries (the incoming queries in Figure 5.1) with respect to the objects of the database (in our case the multidimensional cube instances). The task of the *query rewriting* process is to transform (rewrite) these incoming queries to a set of queries that are semantically equivalent to the original query. I.e., that the result of the rewritten queries can be combined to compute the results of the original query. Thus in the general case, the rewriting algorithm generates a set of queries that are executed separately and a special combination query that combines their results to the result of the original query. In the special case of an OLAP tool, the combination of the results can be done by performing a union operation of the results. Formally, each query result is a cube instance. The union $I_{\cup}$ of a set of cube instances $\{I_1,\ldots,I_k\}$ that all have the same *n*-dimensional cube schema *C* (with dimensions $d_1,\ldots,d_n$) is a cube instance with the same schema containing all the cells of the original cubes. Formally:

$$I_{\cup}\big((x_1,\ldots,x_n)\big)=\begin{cases} I_j(x_1,\ldots,x_n) & \text{for } x_i \in \text{actdom}_{\text{Ij}}(d_i) \quad \forall i \in [1;n] \\ \perp & \text{else} \end{cases} \quad \text{with } x_i \in dom(d_i)$$

The rewriting is performed such that each of the rewritten queries can be evaluated only using either a cached object (i.e., cached query results) or a database object (i.e., cube instances or statically materialized views). This allows the cache manager to make use of results even if none of the cached objects completely subsumes the rewritten query. Thus, the query rewriting process can be functionally described by a function *rewrite* with the following signature:

$$rewrite: \underbrace{\Theta_{C_\Psi}}_{\substack{\text{original}\\ \text{query}}} \times \underbrace{2^{\Theta_{C_\Psi}}}_{\substack{\text{cache}\\ \text{contents}}} \rightarrow \underbrace{2^{\Theta_{C_\Psi}}}_{\substack{\text{set of rewritten}\\ \text{queries}}}$$

The rewritten queries of an original query q_C against a cube schema *C* have the following properties:

- the combinaion (union) of the results of all rewritten queries is equal to the result of the original query for all possible cube instances I_C of a cube schema *C*. Thus,

$$\bigcup_{q_r \in rewrite(q,\Gamma)} q_r(I_C) = q_C(I_C) \quad \forall I_C$$

- each of the rewritten queries can be evaluated using either a single cached object or a single database object. Thus the set of rewritten queries can be partitioned into two sets: *cachequeries* containing all the queries that can be satisfied from the cache and *basedataqueries* containing all the queries that require base data for their computation. Formally, the sets are defined as follows (where Γ denotes the current cache content):

$$\text{cachequeries}_\Gamma(q) := \left\{ q_r \in \text{rewrite}(q,\Gamma) \mid \exists \gamma = (q_c, \rho) \in \Gamma : q_c \text{ subsumes } q_r \right\}$$

$$\text{basedataqueries}_\Gamma(q) := \text{rewrite}(q,\Gamma) - \text{cachequeries}_\Gamma(q)$$

In general, the rewriting process involves the generation of all semantically equivalent rewritings (which might be infinitely many) and a selection of the optimal rewriting with respect to an appropriate cost model and according optimization parameters that are passed to the algorithm. Both the cost model and the generation of rewritings is dependent on the deployed algorithm (we will give an example for a cost model in Section 5.1.3).

In order to illustrate the concept of the rewriting process, let us assume that the cache manager implements a total containment strategy. This means that the query is either rewritten completely to a cached object or evaluated from the base data. In this case, the rewriting process performs a subsumption test for all of the cached objects and either returns the complete query as a cache query or as a base data query (in this case the output set always consist of a single query). In the case of more a complex algorithm (for example the patch working algorithm [Leh98b]), the original query is subdivided into regions such that each region can be derived from a cached object or from base data (in this case, the rewriting algorithm actually returns a set of queries that are different from the original query).

In any case, the output of the rewriting (partitioned into cache and base data queries) is then passed to the *query processing process*. This process is normally part of the database system and is responsible for scheduling, optimizing and executing the queries. If the query rewriting process returns a set of queries, the query processor must also assemble the final result from the individual results. The query processor also decides which of the base data queries should be evaluated on the raw data (finest granularity) or which should be answered using statically materialized views.

Each query result that is produced by the query processor is passed to the admission process that decides whether the result should be stored in the cache. This involves an estimation of the benefit of the object for future queries. As future queries are not known in advance, the benefit has to be approximated using a sophisticated *benefit estimation process*. The details of the estimation process are largely dependent on the actual caching strategy (for example using weighted access frequencies, see Section 5.1.1 for a comparison of strategies). The admission process compares the computed benefit with the costs incurred in storing the object (e.g., taking into account the cache size needed for storage or the benefit of the objects that have to be evicted). If the admission process decides that the object should be added to the cache, the object is passed to the eviction process, which has to decide which of the objects are to be evicted from the cache (if the space in the cache is not large enough to hold the new object). Of course, the eviction decision is again based on a benefit estimation, this time concerning objects in the cache. Those objects that have the least benefit are evicted from the cache.

All of the processes in the caching framework rely on functions estimating the costs for executing queries and storing queries. These estimations are typically based on a cost model

combined with statistics about the data (if the system employs a cost based optimizer, large parts of this functionality can be re-used). In our abstract framework, we represent this cost model component by the function *executioncosts*: $2^{\Theta} \rightarrow \mathbb{R}$ that assigns execution costs to a set of multidimensional OLAP queries. Additionally, we assume a function *storagecosts:* $\Theta \rightarrow \mathbb{R}$, that computes the costs of storing the results of a query. The actual implementation of these functions depends on the OLAP system and its physical implementation strategy. As we do not assume any properties for these functions, any existing estimation function can be integrated with the described approach. However, in order to make our approach self-contained, we propose a simple cost model for the processing of canonical OLAP queries in Section 5.1.3.

Having defined this abstract functional model of the cache manager, we can now discuss how this system can be extended by the PROMISE/OLAP prediction framework. Section 5.2 discusses extensions to the Admission/Eviction process, while Section 5.3 will focus on predictive prefetching.

5.1.3 A Cost Model for Execution and Storage of Canonical OLAP Queries

As described in the previous section, many cost and benefit measures used in the caching process to drive decisions (for example eviction decisions) are based on the estimation of the storage cost of a canonical query result and on the execution costs of a canonical query result. We already pointed out that an accurate function must always take into account the specifics of the actual physical implementation (e.g., indexing, clustering, network situations). Significant research has already been done in this field for the different variations of multidimensional database implementations. However, our emphasis is not on researching the different physical implementation strategies for OLAP tools. Therefore, the next sections present a cost model that does not take implementation-specific aspects into account.

It is important to note that the only purpose of this model is to illustrate the concepts in our examples. Especially, it is not meant to be used in a real-world implementation nor does it claim to reflect the actual costs in a real world system. However, as our framework does not make any assumptions about the properties of the cost estimation function, any existing more sophisticated cost functions provided by a real system (for example provided by a query optimizer) can be used in implementations of the framework.

5.1.3.1 Storage Costs

Storage costs of a query result are determined by the amount of memory that is needed to store the result in the cache[48]. Like base data, cached results (and statically preaggregated views) can be stored using an array-like data structure or a table-like structure (a set of occupied cells). In terms of storage utilization, the array approach is better suited for dense data while the set of cells approach is better suited to sparsely populated result sets. It is also possible to use a hybrid approach for different partitions of the data cube. However, the different approaches lead to different formulas for the size of the data that has to be processed.

For the array data structure the full data space has to be materialized (regardless of the number of occupied cells). This leads to the following storage requirements for the results of a query $q_C = \left(M_q, (r_1, \ldots, r_n), (g_1, \ldots, g_n)\right)$, where *sizeof*($M_q$) denotes the space needed to store the result measures (size of result cell content):

[48] As the purpose of cost estimations is the relative comparison of different processing plans, virtual units are used for the cost estimation. In our examples we use the unit of bytes for storage costs.

$$\text{storagecosts}_{\text{array}}(q) = \prod_{i=1}^{n} \left|\text{descendants}(r_i) \cap \text{dom}(g_i)\right| \cdot \text{sizeof}(M_q)$$

For a set based approach, the size of a query result is determined by the number of cells that are filled. This size can easily be computed for objects that are already in the cache, but has to be estimated for queries that are to be prefetched. If we assume that the data is uniformly distributed, we can approximate the size of the set of cells by using a sparsity factor. Notably this sparsity factor varies for each combination of granularities. It is therefore denoted as a function of *G*. It is defined as the number of filled cells divided by the total number of cells:

$$\text{sparsity}_{\text{I}_\text{C}}(G) = \frac{\left|\{x \in dom(g_1) \times \ldots \times dom(g_n) \;\middle|\; I_C(x) \neq \perp\}\right|}{\prod_{i=1}^{n} \left|\text{dom}(g_i)\right|}$$

Therefore, the storage size of the result of query $q_C = \left(M_q, (r_1, \ldots, r_n), (g_1, \ldots, g_n)\right)$ using the cell-set method is

$$\text{storagecosts}_{\text{set}}(q) =$$

$$\prod_{i=1}^{n} \left|\text{descendants}(r_i) \cap \text{dom}(g_i)\right| \cdot \text{sparsity}_{\text{I}_\text{C}}(G) \cdot (\textit{sizeof}(M_q) + \text{sizeof}(\text{coordinate}))$$

where sizeof(M_q) denotes storage space for a tuple of result measures and sizeof(*coordinate*) denotes the storage space for a coordinate in the n-dimensional cube[49]

This estimation assumes uniformly distributed data. The quality of the estimation can be additionally enhanced by using multidimensional histograms (e.g [FM00]). Advanced systems use compression techniques to reduce the amount of storage needed for the results. However, depending on the compression technique deployed, an estimation of the result size becomes difficult.

Example 5.2 (Storage Cost for OLAP Query)

Let us consider the query q_{example} introduced in Example 3.16.

```
q_example := ( {#repairs},
              ( all vehicles , 1999, Germany      , steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

Additionally, we assume that each of the measure values requires 4 bytes of storage and that Germany is divided into 14 geographical regions. Then the storage requirements for the array based approach are calculated as follows:

$$\text{storagecosts}_{\text{array}}(q_{example}) = 1 \cdot 1 \cdot 14 \cdot 1 \cdot 1 \cdot \underbrace{4}_{\substack{\text{sizeof}(M) \\ [bytes]}} = 56$$

If we assume that the coordinate for each of the 5 dimensions is represented as a 4 byte value, a coordinate requires 20 bytes of storage. Let us additionally assume a sparsity factor of sparsity(G_q)=0.3 (which seems realistic as the result granularity is defined on very high levels in all dimensions; on lower levels, the sparsity will be several orders of magnitude smaller), the corrsponding storage requirements for the set-based storage are as follows:

[49] The storage space for a coordinate is dependent on the encoding and the cardinality of the domain of the lowest classification level.

$$\text{storagecosts}_{\text{set}}(q_{example}) = 1 \cdot 1 \cdot 14 \cdot 1 \cdot 1 \cdot \underbrace{0.3}_{sparsity(G_q)} \cdot (\underbrace{4}_{\text{sizeof}(M)} + \underbrace{20}_{\text{sizeof}(\text{coordinate})}) = 100.8$$

◆

5.1.3.2 Execution Costs

After the rewriting process, each of the rewritten queries q_r can be evaluated using either a single cached object, a single cube instance from the database or a single statically materialized view. This source object of the query can itself be represented as a canonical query $q_{Source} = (M_{Source}, R_{Source}, G_{Source})$[50]. If the source object is a base cube I_C with schema C that is stored in the multidimensional database, we denote the query that reads the whole cube as $q(I_C)$. For our cost model, we estimate the execution costs of a query $q = (M_q, R_q, G_q)$ by the size of the data that has to be processed i.e., the size of the data that has to be read from the cached object, cube instance or statically materialized view. This corresponds to the result size of the following query, that extracts the data using the granularity of the query source:

$$q|_{q_{Source}} = (M_q, R_q, G_{Source})$$

Additionally, data in an OLAP system is held on different levels of a complex storage hierarchy (for example in the OLAP storage manager's disk and in cache memory). The components of this hierarchy exhibit different characteristics regarding data access and data transfer times. We reflect this in our model, by introducing an access factor for each component in the architecture that stores data (We assign a factor of *1* to cache accesses). This is a simplification of real systems as the size of data that can be retrieved from a component (for example the storage manager) in a given timeframe is influenced by a lot of other factors that differ from component to component. E.g., the 'reading performance' of a relational database system using index structures is additionally influenced by the selectivity of the query. However as already mentioned, our cost model does not aim at mirroring these technology specific influences as it is only intended for illustrative purposes in our examples. For implementations of our framework more sophisticated models can easily be deployed. Considering the above, the estimation function for the execution cost is defined as follows:

$$\text{executioncosts}_C(q) = \text{storagecosts}(q|_C) \cdot \text{access factor}_C$$

The definition of execution costs can easily be extended to sets of queries by summing up the execution costs of the individual queries contained in the set.

Example 5.3 (Execution Cost)

The execution cost of q_{example} against the raw data stored in the instance I_{repair} of the cube schema C_{repair} (cf. Example 3.16) is estimated by calculating the size of the result of the following query:

```
q_example|q(I_repair):= ( {#repairs},
        ( all vehicles, 1999, Germany , steering, all types ),
        ( vehicle     ,  day, location, part    , repair type) )
```

To calculate the size of the result, we need to know the selectivity of the restriction elements regarding the base classification level which is defined by the fan-out of the hierarchy trees. E.g., the number of days in *1999* is *365* and the number of parts in the *steering* assembly is *20*. We additionally assume a sparsity factor of 10^{-6} and a set based storage of the raw data. We

[50] Of course, we assume that q_{Source} subsumes q (which is e.g. guaranteed by the rewriting algorithm).

assign an access factor of 100 to the storage manager, assuming that it take 100 times longer to retrieve data from the storage manger that from the main memory cache. The size of data that needs to be processed is calculated as follows:

$$\text{executioncosts}_{C_{\text{repair}}}(q_{example}) =$$

$$\underbrace{1000}_{\text{vehicle}} \cdot \underbrace{365}_{\text{time}} \cdot \underbrace{500}_{\text{location}} \cdot \underbrace{20}_{\text{part}} \cdot \underbrace{2}_{\text{type}} \cdot \underbrace{10^{-6}}_{\text{sparsity factor}} \cdot \underbrace{24}_{\text{size of coordinate/ measures}} \cdot \underbrace{100}_{\text{access factor}} = 1.752 \cdot 10^{7}$$

Calculating the same query based on the results of a subsuming query q_2 stored in the cache (main memory) in array format leads to the following cost estimation:

```
q1:=    ( {#repairs, duration},
        ( all vehicles, all times, all locations, steering, all types ),
        ( vehicle class, month    , geogr. region, assembly, type.all  ) )
```

$$\text{executioncosts}_{C_{q_1}}(q_{\text{example}}) = \underbrace{4}_{\text{vehicle}} \cdot \underbrace{12}_{\text{time}} \cdot \underbrace{14}_{\text{location}} \cdot \underbrace{1}_{\text{part}} \cdot \underbrace{1}_{\text{type}} \cdot \underbrace{1}_{\text{access factor}} = 672$$

This demonstrates that the overall benefit of the query level cache (in the order of 10^5 in this example) has two sources: (1) faster media access due to data storage in a higher part of the storae hierarchy (10^2 in this example) and (2) the reduced size of data that has to be read and aggregated (10^3 in this example). ◆

5.2 Predictive Admission and Eviction Strategies

The functional reference architecture of our caching framework (cf. Figure 5.1) shows that the admission and the eviction algorithm both heavily rely on an estimation of the benefit of an object for future queries. This benefit is then compared to the costs that are incurred by keeping/admitting the object. Which cost factors or (for example the recomputation costs of an object) are considered is dependent on the caching strategy. As the *costs* for storing and retrieving a query are independent of the anticipated workload, all techniques proposed for cost estimation can still be applied. Therefore in this section, we concentrate on estimating solely the *benefit* of an object. Current approaches make benefit estimations based on statistics gathered since that point in time when the cache became operational (for example the reference rate of a cached object). Most importantly, the algorithms do not explicitly make use of the information about the session context (i.e., which queries were executed last)[51]. This sections presents two different approaches to use information about the current session context in order to improve the benefit estimation:

- *based on the PROMISE prediction* (Section 5.2.1): As already motivated, a prediction of the workload based on the current session context can be used for improving the benefit estimations. Figure 5.3 shows the integration of the PROMISE prediction algorithm into the general caching framework (Added, respectively altered components are shaded in gray). The prediction algorithm sends the set of predicted queries (the anticipated workload) to the benefit estimation process such that it can be used by the admission and the eviction process. We describe this approach in Section 5.2.1.

[51] Caching algorithms use implicit information about the timing of the last access to an object (for example using reference densities) but the do not use relationships between a cached object and the object requested last, if these objects are distinct.

- *based on information about the navigational capabilities of the OLAP front-end*: The benefit of an object can also be measured using information about the last query and the interactions (query transformations) that can be executed using the OLAP front-end. We describe the approach in Section 5.2.2 although it is not strictly related to prediction because it has so far been neglected by the OLAP caching algorithms (with the exception of [ABD+99]) and nicely matches the PROMISE objective that aims at making use of knowledge about the user interface.

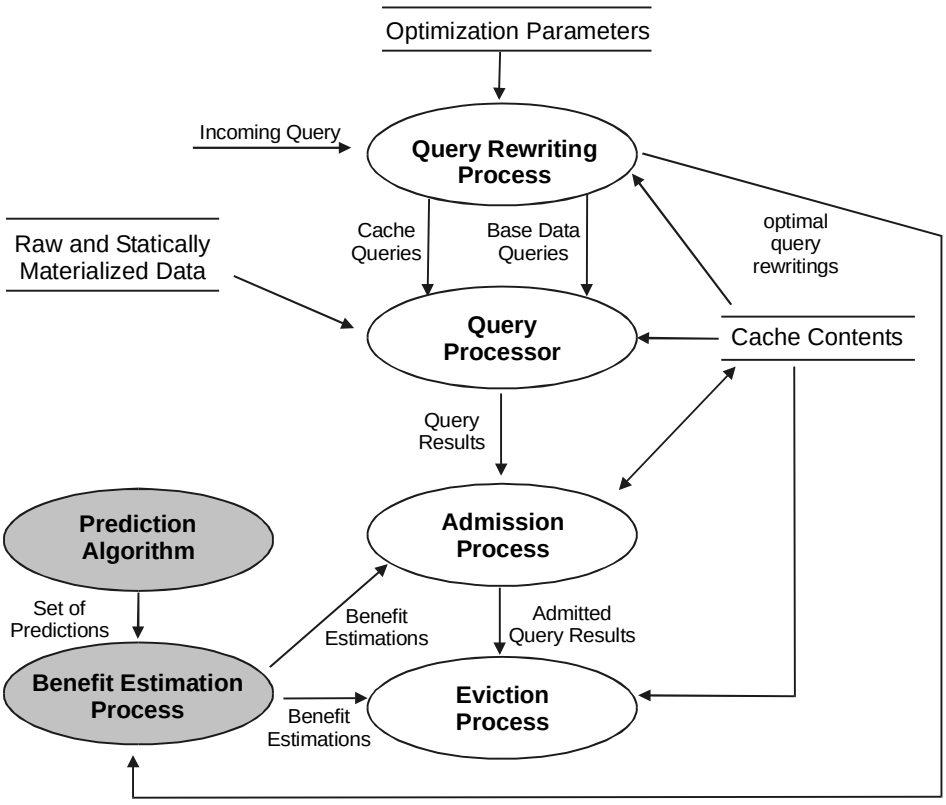

Figure 5.3: Extending the Caching Framework by Predictive Admission and Eviction

Naturally, both methods to measure the benefit can be combined with other 'classical' benefit measures like the aged number of references. The purpose of a mixed benefit approximation strategy is to better represent a mixture of workloads in user behavior. The typical way is to use a linear combination of the different benefit measures. The parameters of the linear-combination can be used to weigh the influence of the different benefit estimation methods.

5.2.1 A Predictive Benefit Model

This section describes an approach to measure the benefit of an object with respect to the workload predicted by the prediction process. Generally speaking, the benefit of an object for a workload of queries can be measured by the cost savings that can be achieved using the object to answer the queries of the workload. Although the exact future workload is not known, a prediction algorithm can offer a probabilistic hypothesis about which set of queries is likely to be executed next. The benefit of objects can thus be measured with respect to these predictions.

In order to assess the benefit of an object (q_C,ρ) that is contained in the cache (for the purpose of finding eviction candidates) for an anticipated query q, we measure the cost savings that can be achieved by using the object for answering the query q. When calculating these cost savings, the rest of the cache content must be taken into account because the cache might

contain further results from which q can be derived that will be used if (q_C,ρ) is not cached anymore. The benefit for a workload (weighted set of queries) is calculated as the sum respecting the weights (probabilities) of the queries. This is mirrored in the following definition:

Definition 5.3 (Probabilistic Benefit of an OLAP Cache Object)

Let Γ denote an OLAP cache content (i.e., a set of materialized query results). The *benefit* benefit_Γ of a cached object $(q_C,\rho)\in\Gamma$ for the execution of a query q is defined as the cost savings of caching q_C compared to not caching q_C. Thus:

$$\text{benefit}_\Gamma(q_C,q) := \text{executioncosts}\big(\text{rewrite}(q,\Gamma-(q_C,\rho))\big)-\text{executioncosts}(\text{rewrite}(q,\Gamma))$$

Calculating the *probabilistic benefit* of a query q_C for a predicted set of queries $P=((q_1,\ldots,q_k),(w_1,\ldots,w_k))$ is done by summing up the weighted benefits for each query in the set:

$$\text{benefit}_\Gamma(q_C,P) := \frac{\sum_{i=1}^{k} w_i \cdot \text{benefit}_\Gamma(q_C,q_i)}{\sum_{i=1}^{k} w_i}$$

◆

Notably, this benefit definition takes into account other objects in the cache that subsume the query result. This is done by comparing the cost of execution based on the query rewriting using the original cache content and the cache content without the query result (q_C,ρ). In the extreme case, this can result in a benefit value of zero although the query q can be derived from (q_C,ρ). Additionally, it is important to note that the benefit of all cached objects dynamically changes with the currently anticipated workload (which is dependent on the current session context). This requires the re-evaluation of the benefit directly before any admission/eviction decision.

Example 5.4 (Probabilistic Benefit of an OLAP Cache Object)

Let us assume that the cache contains results for queries q_1 and q_2 (see below). Thus, the cache content is defined as $\Gamma=\{(q_1, C_1),(q_2,C_2)\}$ with

```
q1:=    ( {#repairs, duration},
        ( all vehicles  , all times, all locations, steering, all types ),
        ( vehicle class , month    , geogr. region, assembly, type.all  ))
```

```
q2:=    ( {#repairs },
        ( all vehicles  , all times, all locations, steering, routine ),
        ( vehicle class , month    , geogr. region, assembly, type    ) )
```

The prediction algorithm has predicted the following workload $P=\{(q_{example},0.7)\}$ containing only $q_{example}$ as a prediction for the next query with a probability (weight) of 0.7. In order to evict a query result form the cache, the eviction algorithm evaluates the probabilistic benefit of the cached objects for the predicted workload. The following example details the benefit calcuclation for q_1. The benefit value is computed by comparing the execution costs of $q_{example}$ with and without the results of q_1 in the cache:

```
qexample:= ( {#repairs},
        ( all vehicles , 1999, Germany      , steering, all types ),
        ( vehicle.all  , year, geogr. region, assembly, type.all  ) )
```

$$\text{benefit}_{\Gamma}(q_1, q_{\text{example}})$$
$$= \text{executioncosts}(\text{rewrite}(q_{\text{example}}, \{(q_2, C_2)\}) - \text{executioncosts}(\text{rewrite}(q_{\text{example}}, \Gamma)$$

Let us assume a rewriting algorithm that uses parts of the cached results (like the patch working algorithm [Leh98b]). If the results of q_1 are not contained in the cache, q_{example} can be evaluated from a combination of the results of q_2 and a query to the raw data. Thus the rewriting algorithm returns the following queries: *rewrite*(q_{example}, {(q_2,C_2)})={q_3,q_4} with

```
q3 :=   ( {#repairs},
        ( all vehicles , 1999, Germany      , steering, routine ),
        ( vehicle.all  , year, geogr. region, assembly, type ) )

q4 :=   ( {#repairs},
        ( all vehicles , 1999, Germany      , steering, critical ),
        ( vehicle.all  , year, geogr. region, assembly, type ) )
```

As q_3 is subsumed by q_2 (which is contained in the cache) and q_4 has to be evaluated using raw data, the execution cost of the rewritten query set are computed as follows[52]:

$$\text{executioncosts}(\text{rewrite}(q_{\text{example}}, \{(q_2, C_2)\})) =$$
$$\text{executioncosts}(q_3 | C_2) + \text{executioncosts}(q_4 | C_{repair}) = 672 + 8760000$$

The execution cost of q_{example} using the cached results of q_1 have already been calculated in Example 5.3 as 672. Therefore, the benefit measure of q_1 is

$$\text{benefit}_{\Gamma}(q_1, q_{example}) = 8760672 - 672 = 8.76 \cdot 10^6$$

The benefit of q_2 in this example is 0, as the rewriting of q_{example} will contain queries that are evaluated against results of q_2 as long as the results of q_1 (that subsumes q_{example}) are containted in the cache. This demonstrates that the benefit measure is not only dependent on the predicted workload but also on the current cache contents. The conseqence would be that the results of q_2 will probably be evicted first. However, the actual eviction decision must also consider the storage costs incured by the cached object.

◆

For the admission algorithm, it is important to compute the additional benefit of an object. This benefit is measured analogously by considering the costs savings that can be achieved by adding the object to the cache.

Definition 5.4 (Additional Probabilistic Benefit)

Let Γ denote an OLAP cache content (i.e., a set of materialized query results). The *benefit* benefit_X of a cached object $(q_a,\rho) \in \Gamma$ for the execution of a query q is defined as the cost savings of caching q_C compared to not caching q_C. Thus:

$$\text{addbenefit}_{\Gamma}(q_a, q) := \text{executioncosts}(\text{rewrite}(q, \Gamma)) - \text{executioncosts}(\text{rewrite}(q, \Gamma \cup (q_a, \rho)))$$

[52] The execution costs of q_4 are half the execution costs of q_{example} (cf. Example 5.3) because the queries only differ in the *repair type* dimension which has two elements (*critical* and *routine*). Query q_4 reads only half of the elements (critical) compared to q_{example}. For the rest of the examples in this chapter, we will omit the details of the benefit calculations and only present the results thus leaving the calculation as an exercise to the interested reader.

Calculating the *additional probabilistic benefit* of a query q_C for a predicted set of queries $P=((q_1,\ldots,q_k), (w_1,\ldots,w_k))$ is done by summing up the weighted benefits for each query in the set:

$$\text{addbenefit}_\Gamma(q_C, P) := \frac{\sum_{i=1}^{k} w_i \cdot \text{addbenefit}_\Gamma(q_C, q_i)}{\sum_{i=1}^{k} w_i}$$

◆

This formula implies that the additional benefit of a query result is zero if it is already contained in the cache. Notably, this benefit measure consciously does not consider the space limitations for the cache. That means that the benefit of an object is independent from the fact whether the free cache space is large enough to hold the object or if other objects have to be evicted. Taking this into account is the task of the cache admission algorithm, that typically compares the benefit of cached objects with the cost incurred when caching the object. The cost measure should mirror the fact that objects have to be evicted from the cache in order to make room for the newly arriving object[53].

Both benefit measures described in this section can be used to enhance the admission and eviction algorithms of OLAP query level caches. They constitute the link between the prediction algorithm and the cache manager. Although this approach itself already improves the performance of OLAP caches (see Chapter 6), prefetching techniques can additionally improve the caching performance. However, we will show in Section 5.3 that the prefetching algorithm can make use of the benefit definitions given above.

5.2.2 A Transformation Based Benefit Model

While the previous section described a method of measuring the benefit of objects for the anticipated workload by predicting the workload, this section describes an alternative benefit measure for cached OLAP objects based on knowledge about the interactive capabilities of the OLAP front-end. As described in 3.2.4, OLAP interfaces offer a fixed set of query transformations that are used to derive the next query from the previous one. Therefore, queries that can be produced with a smaller number of query transformations from the last query seen can have a higher probability of being executed next. This approach can be combined with the predictive approach and other techniques or can be used alone.

In order to evaluate the benefit of cached objects, we define a metric space for queries which is based on the definition of query transformations. The distance measure for two queries q_1, q_2 in this *query transformation space* is defined as the minimum number of transformations that is needed to transform q_1 into q_2. We can then assume that the query behavior exhibits locality properties regarding this space. Consequently, the benefit of a cached object is defined inversely to the distance of the cached object from the last executed query.

Definition 5.5 (Transformation Distance between Two OLAP Queries)

Let T denote a complete (cf. Theorem 3.5) set of query transformations. The set of minimal transformation sequences $\text{MIN}_T(q_1,q_2)$ with respect to T that transform a canonical OLAP query $q_1 \in \Theta_C$ into $q_2 \in \Theta_C$ is defined as follows:

[53] For example the approach of [SSV96] only admits objects to the cache if the benefit of the object is larger than the benefit of the objects that have to be evicted.

$$\mathrm{MIN}_T(q_1,q_2) := \left\{ t \in T^* \mid t(q_1) = q_2 \wedge \neg\exists t' \in T^* : \left(t \neq t' \wedge t'(q_1) = q_2 \wedge |t'| < |t|\right) \right\}$$

The transformation distance $|q_1 - q_2|_T$ between two canonical OLAP queries q_1 and q_2 with respect to T is defined as the length of the minimal sequence of transformation operations that transforms q_1 into q_2.

$$|q_1 - q_2|_T := |t| : t \in \mathrm{MIN}_T(q_1,q_2)$$

◆

Example 5.5 (Transformation Distance Between OLAP Queries)

To calculate the transformation distance between the queries q_{example} and q_{trans}, we have to find the shortest sequence that transforms q_{example} into q_{trans}.

```
q_example := ( {#repairs},
              ( all vehicles , 1999, Germany      , steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type.all  ) )

q_trans :=   ( {#repairs},
              ( all vehicles, 1999 , USA          , steering, all types),
              ( vehicle.all , month, geogr. region, assembly, type.all ) )
```

The following sequence of transformations applied to q_{example} produces the query q_{trans}. It should be obvious, that no shorter sequence (one transformation) exists.

$$(\textbf{\textit{rotate-selection}}_{\text{repair time,month}} \circ \textbf{\textit{slice}})(q_{\text{example}}) = q_{\text{trans}}$$

Therefore, the transformation distance of the two queries $|q_{example} - q_{trans}|_T$ is 2. ◆

Without providing a formal proof, we remark that for our set of transformations presented in 3.2.4 the shortest transformation sequence can be computed efficiently. As each transformation only affects one dimension (or the set of measures), the length of the sequence can be computed dimension-wise. Although using a distinct set of transformations, [ABD+99] presents performance measurements that show that the transformation based benefit measurement can greatly improve the caching performance in the presence of typical OLAP workloads.

5.3 Predictive Prefetching for OLAP Caches

The purpose of this section is to extend the abstract caching system presented in Section 5.1.2 by a component that allows for speculatively executing OLAP queries based on the results of the prediction process. Figure 5.4 shows the necessary extensions and communication links that are necessary for extending the general caching framework. Our proposed solution makes use of the extensions already introduced in the previous section. The added/modified components are shaded in gray.

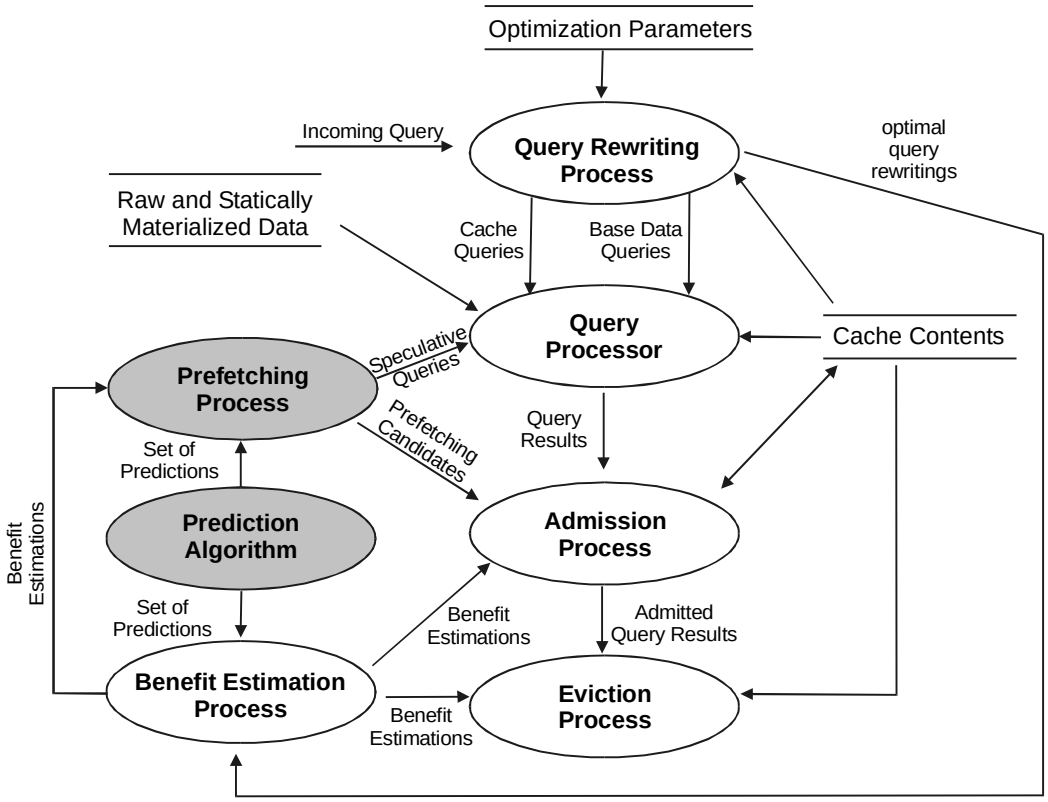

Figure 5.4: Data Flow in a Caching System Extended for Predictive Prefetching

5.3.1 The Predictive Prefetching Algorithm

At any point during a session (typically after finishing the processing of the last query), the *prefetching process* (cf. Figure 5.5) starts the *prediction process* that computes a set of anticipated queries based on the current session context (line 6). Based on this *set of predictions*, the prefetching process must decide what data should be actually prefetched and in which order it should be prefetched. Due to the subsumption (respectively derivability) property of query results, it is not necessarily optimal to speculatively execute the exact queries from the predicted set of queries in order to speed up the query processing. Depending on the actual rewriting strategy, it might be more beneficial to fetch sub- or supersets of the predicted queries. Additionally, it might be more cost efficient to prefetch partial results that are of potential benefit for several queries of the anticipated workload. This means the prefetching algorithm must first determine a set of *prefetching candidates* (line 7) that have the highest cost/benefit ratio. We will discuss the influencing factors of the benefit measure for the prefetching process in Section 5.3.2.2. However, it should be obvious now that the cost saving potential for the predicted workload is an influencing factor. This means that the prediction algorithm can make use of the predictive benefit estimation process (cf. Figure 5.4) that has been described in Section 5.2.1 (see data flow diagram, Figure 5.4). The candidates are then processed in order of descending benefit values (line 9) as long as the next query has not been executed (i.e., during the consideration time, cf. line 10)

```
01  procedure Prefetching(  SessionContext C,
02                          MarkovModel M)
03  {
04     declare workload: set of (OLAP query prediction)
05     declare candidates: set of (OLAP query)
06     workload = PredictOLAPQuery(C, M)
07     candidates = GeneratePrefetchingCandidates( workload )
08     sort candidates by PrefetchingBenefit()
09     while (!NextQuery())
10        get next q in candidates
11        if Admission (q) = TRUE
12           QueryProcessing(q, SPECULATIVE)
13     end while
14  }
```

Figure 5.5: Pseudo Code for the Prefetching Algorithm

For each candidate, it is important to decide if it should actually be prefetched. The reason for this is that a prefetching request uses system resources (query processing time, network bandwidth but also cache storage place). It is obvious that a very aggressive prefetching strategy does not only cost large amounts of system resources (network bandwidth, CPU time) but is also likely to nullify the caching strategy by evicting large amounts of cached objects to make room for the prefetched objects. In our cache model, we assume that the query processing system can handle requests that are marked as speculative in a way that they are only processed if the system is idle (for example by assigning a lower priority to speculative execution requests). Therefore, for the prefetching decision, we only consider the caching relevant cost factors (e.g., size of result and benefit of evicted cache objects). Deciding whether an object can increase the overall benefit of the cache (depending on the caching strategy) is the task of the admission process. Thus, each candidate is presented to the admission algorithm (line 11). If the admission process returns a positive result, the candidate is passed to the query processing as a speculative query (line 12). After the execution, the speculative query result is inserted into the cache by the query processor (calling the admission algorithm before the execution ensures that the result will actually be admitted).

This coarse description of the prefetching process shows that in addition to the extensions already described Section 5.2, we have to design a benefit measure *PredictionBenefit* (cf. line 8) and an algorithm that generates the best prefetching candidates according to this measure function (function *GeneratePrefetchingCandidates*, cf. line 7). In the next section we will show, that this problem can be reduced to the general view selection problem and is therefore too complex to be solved online. Consequently, we will discuss heuristic approaches to this problem.

5.3.2 Generating and Evaluating Prefetching Candidates

This section will discuss the problem of selecting optimal prefetching candidates. This problem is a step in the overall prefetching algorithm discussed in the previous section. In Section 5.3.2.1, we will show that this problem can be formalized as a variation of the view selection problem and conclude that an analytical solution to the problem is far too complex for online computation. When presenting the prefetching algorithm in the previous section, we already motivated that an evaluation function is necessary to determine the order in which the prefetching requests should be executed. Therefore, we suggest several heuristic construction methods for the candidate set in 5.3.2.3. These heuristics aim at producing candidates that are good (optimality cannot be guaranteed due to the heuristic approach) with respect to a prefetching benefit measure. This means that the benefit measure has to be defined before dis-

cussing the heuristics. Therefore Section 5.3.2.2 presents a benefit measure that evaluates candidates produced by the candidate generation process. This measure is used to determine the order in which the prefetching requests are passed to the admission algorithm (cf. Figure 5.5; line 8). The combination of the heuristic candidate generation and the application of the benefit measure are illustrated in the comprehensive example in Section 5.3.3.

5.3.2.1 Problem Formalization

The problem of determining an optimal set of prefetching candidates is a special case of the view selection problem with additional constraints. For the relational case, the view selection problem (e.g., [TS98], [SDN98]) is defined as follows: For a given workload (a set of relational queries and corresponding weights) and a given set of base relations, a set of views has to be constructed such that the execution costs for a rewritten workload exploiting the materialized views and the base relations are minimized. Additionally, the set of materialized views must fulfill a space constraint (i.e., the space available for materialization is limited by S)[54]. The problem in its general form has been proven to be computationally intractable [SDN98]. Several heuristic solutions have been proposed for this problem (e.g., [HRU96], [SDN98]) giving solutions for restricted types of view definitions (for example only considering SPJ views) and using greedy algorithms to traverse the search space. However all these algorithms are too complex to be performed online (as needed for dynamic caching decisions) as their execution time may last for several hours or even days in realistic scenarios.

We are aiming at minimizing the execution costs of the anticipated probabilistic workload by changing the current cache content Γ (set of currently materialized views) to Γ'. The execution costs of the probabilistic workload P (predicted queries and their weights) using the (modified) cache content Γ' are defined as follows:

$$\text{executioncosts}_{\Gamma'}(P) = \sum_{i \in [1;k]} \left(\text{executioncosts}\left(\text{rewrite}(q_i, \Gamma')\right) \cdot w_i \right) \qquad \text{(OPT)}$$

The optimization algorithm should find a cache content Γ' such that these costs are minimized taking into account the following constraints: Γ' must not exceed the maximum cache space, thus:

$$\left|\Gamma'\right| \leq \text{MAXSIZE} \qquad \text{(C1)}$$

An additional important and interesting peculiarity of our prefetching environment is that the sequence of the materialization and the time needed for the materialization is important as only views which are materialized before the execution of the next user query can be exploited for query speed-up. That means that the following constraint has to be satisfied:

$$\text{executioncosts}(\Gamma' - \Gamma) \leq \textit{ConsTime} \qquad \text{(C2)}$$

where *ConsTime* denotes the consideration time between the last and the next query. As this time is not known in advance (and also not predicted by our time-unaware prediction process), it is not possible to use a deterministic analytical model to find the optimal prefetching set, but a statistical model has to be deployed. In this model *ConsTime* is modeled as a random variable. In general, the distribution (respectively the probability density function) of this variable is not known in advance which additionally increases the complexity of the problem.

[54] The problem has been specialized in different ways introducing additional constraints, for example that the set of materialized views should be self-maintainable or that the maintenance cost of the views should be also minimized.

5.3.2.2 Evaluating the Prefetching Benefit of OLAP Queries

The previous section formally described the candidate generation problem. The conclusion was that an analytically optimal solution is not feasible. Therefore, we will use a heuristic algorithm to solve the problem. In order to decrease the complexity of the algorithm, we solve the problem using two steps: a heuristic candidate generation algorithm that transforms the queries of the workload in a heuristic way and an evaluation step, that evaluates the transformed workload (i.e., the prefetching candidates) using a special benefit function. Candidates with a high evaluation score will be prefetched first. Before discussing the candidate generation procedure in the next section, we first have to derive a measure for evaluating the benefit of generated candidates. This measure is used to order the generated candidates because the most promising candidates should be processed first.

First of all, 'good' candidates must have a high additional benefit (a high potential of reducing the cost in executing several queries of the anticipated workload) as this is the main optimization goal (OPT).

Constraint (C1) of the prefetching candidate selection problem formulation (cf. 5.3.2.1) favors small result sets. However, our strategy is to propagate decisions involving estimations of the storage costs to the cache admission algorithm. This may lead to the generation of suboptimal intermediate prefetching candidates. However, these candidates will be pruned by the cache admission algorithm before they are executed. The advantage of this approach is that we can fully reuse the existing admission algorithm and therefore benefit from all progress made in this area. Additionally, the separation of concerns ensure that our algorithm can be easily integrated will all caching approaches. This means that our benefit measure must not take (C1) into account.

Because of constraint (C2) the benefit measure should favor results that are fast to compute (having a high probability of being available before the end of the consideration time). As the prediction model is not time-aware (i.e., no predictions about the actual time of the next request are available), we can only use the monotony of the probability density function, assuming that a query that can be executed quicker has a higher probability of being materialized before the next event. Therefore, we define the prefetching benefit of a query as the quotient of the additional benefit of caching the query result divided by the execution costs of the query (used to estimate the execution time).

Definition 5.6 (Prefetching Benefit of OLAP Query)

Let Γ denote an OLAP cache content and $q_p \in \Theta_C$ a canonical OLAP query. The *prefetching benefit* prefetch-benefit$_\Gamma$ of query q_p regarding a set of predictions $P=((q_1,\ldots,q_k), (w_1,\ldots,w_k))$ is defined as:

$$\text{prefetch}-\text{benefit}_\Gamma(q_p,P)=\frac{\text{addbenefit}_\Gamma(q_p,P)}{\text{executioncosts}(q_p)}$$

◆

Example 5.6 (Prefetching Benefit)

Let us assume that the prediction process generated the following predictive workload $P=\{(q_1,q_2), (0.5,0.45)\}$ and that the rewriting algorithm cannot find a rewriting of either $q_{example}$ or q_2 using cached objects (for example because the cache is empty). Both queries only differ in the restriction elements of the *repair time* and *repair type* dimension. Figure 5.6

shows a projection of the query rectangles that is defined by both queries to the classification levels year and type.

```
q1:=   ( {#repairs},
       ( all vehicles , 1999, Germany      , steering, all types ),
       ( vehicle.all  , year, geogr. region, assembly, type      ) )

q2:=   ( {#repairs},
       ( all vehicles , all times, Germany      , steering, routine ),
       ( vehicle.all  , time.all , geogr. region, assembly, type    ) )
```

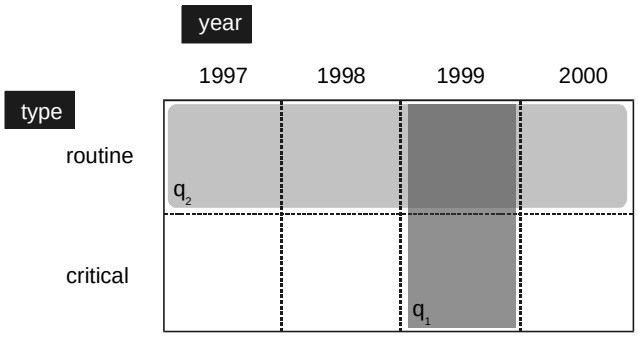

Figure 5.6: A Projection of the Query Rectangles Implicitly Defined by q_1 and q_2 on the level type/year

The prefetching benefit for the query q_1 can be calculated as follows: the additional benefit of q_1 for executing q_1 is (q_1 has the same execution cost as $q_{example}$, as both queries only differ in the result granularity of dimension *repair type* and the result granularity does not influence our estimation of execution costs, cf. Example 5.3 and Example 5.2):

$$\text{addbenefit}(q_1, q_1) = 1.752 \cdot 10^7 - 112 = 1.7519828 \cdot 10^7 \approx 1.75 \cdot 10^7$$

However, the results of $q_{example}$ can also be partially used to answer q_2 (cf. Figure 5.6). Assuming that the rewriting algorithm can fully exploit this relationship, the additional benefit of $q_{example}$ for q_2 is calculated as follows (the exact calculation are omitted here):

$$\text{executioncosts}_{C_{\text{repair}}}(q_2) = 3.5 \cdot 10^7$$

$$\text{executioncosts}_{C_{\text{repair}}}(\textit{rewrite}(q_2, \{q_1\}) = 2.628 \cdot 10^7 + 56$$

$$\text{addbenefit}(q_1, q_2) = 0.8719944 \cdot 10^7 \approx 0.87 \cdot 10^7$$

Thus, the additional benefit of the results of q_1 computes to:

$$\text{addbenefit}_{\Gamma}(q_1, P) = \frac{0.5 \cdot 1.75 \cdot 10^7 + 0.45 \cdot 0.87 \cdot 10^7}{0.95} = 1.33 \cdot 10^7$$

The execution cost of $q_1$against raw data (cube C_{repair}) are $1.75 \cdot 10^7$. Therefore, the prefetching benefit of q_1 is:

$$\text{prefetchbenefit}_{\Gamma}(q_1, P) = \frac{1.33 \cdot 10^7}{1.75 \cdot 10^7} = 0.76$$

The according prefetching benefit of q_2 (detailed computation omited here) is:

$$\text{prefetchbenefit}_{\Gamma}(q_2, P) = \frac{2.12 \cdot 10^7}{3.5 \cdot 10^7} = 0.61$$

This means that even though the additional benefit of q_2 for the predicted workload P is larger than the additional benefit of q_1, the prefetcing process will opt to first prefetch results of q_1, as the execution costs for q_2 are overproportianally larger than the additional benefit.

◆

5.3.2.3 Heuristic Candidate Generations Methods

This section discusses possibilities to generate prefetching candidates i.e., queries that should be executed in order to optimize the cache content. The aim of this step in the overall prefetching process (cf. Figure 5.5) is to produce a sorted list of queries that have a high prefetching benefit (as defined in the previous section). We will discuss the following heuristic strategies for the candidate generation process:

- Considering the original queries of the predicted workload
- Considering super-queries for queries with similar values
- Considering common sub-queries for queries with similar structure
- Using the rewriting process to enhance the candidate generation process

In Section 5.3.3, we will show the (combined) application of these strategies using an example.

Considering workload queries. Probably the most straightforward approach is to use each original query of the workload as a prefetching candidate. This coarse grained approach has the advantage that it can realize the optimal cost savings if one of the predictions is 100% accurate as this query can then be answered directly from the cache. However, this strategy may lead to prefetching very large query results and it does not take into account common subexpressions of the queries in the workload. Thus, the strategy is best suited for situations, where the predicted workload is either small (i.e., the number of queries is low) or heterogeneous (i.e., no common subexpressions) and the consideration time is large compared to the execution times of the predicted queries.

Considering super-queries. A typical situation when using the PROMISE/OLAP approach is that the set of predicted queries contains several similar queries that e.g., only differ in the restriction element of one dimension. These similar queries are predicted when the value-based prediction model contains two successor states with comparable probability values (for example the previous and the next year). In these situations, it is a good idea to consider prefetching one query that subsumes all these similar queries. We call this query the *super-query* of the set. It can be constructed from the individual queries in the following way (cf. Figure 5.7 for a visualization of the process): the result measure set of the super-query is obtained by using the union of the individual result sets. The new result granularity for each of the dimensions must be chosen such that any of the result granularities of the original queries can be aggregated from this level. Additionally, the 'super-granularity' should be as 'coarse' as possible to reduce the size of the query result. Therefore, the super-granularity is chosen as the greatest lower bound for all original granularities in the dimension's classification lattice[55]. Finally, the new restriction element is chosen such that it is a parent to all the original restriction elements. Again, if more than one element is possible, we chose the element on the smallest possible classification level in order to reduce the size of the query result. This construction process for the super-query leads to the following formal definition:

[55] In the special case of a single hierarchy this is the smallest result granularity of all the queries in the set for which the super-query should be computed.

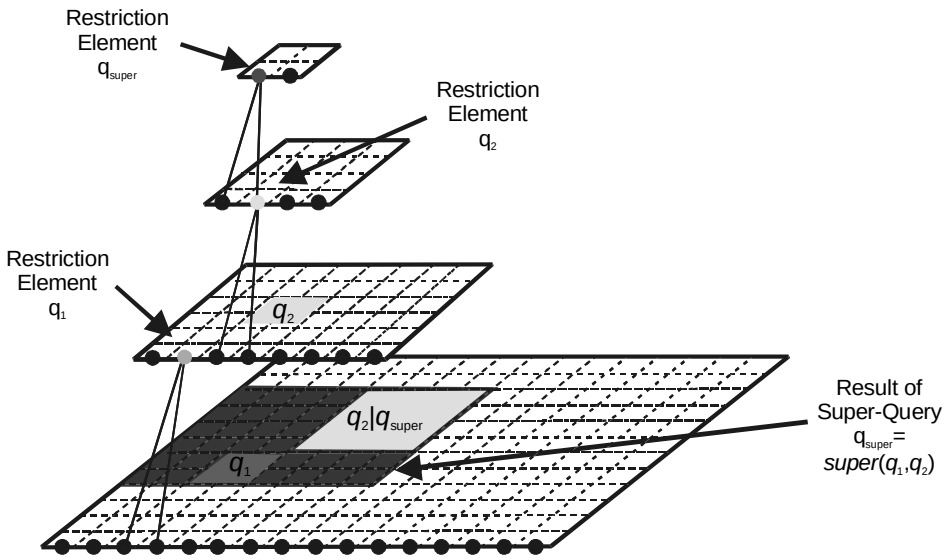

Figure 5.7: Visualization of the Super-query for a Set of Two Queries Q={q_1,q_2}

Definition 5.7 (Super-Query for a Set of Canonical OLAP Queries)

Let $Q=\{q_1,\ldots,q_k\}$ with $q_i=\left(M_i,(g_{i,1},\ldots,g_{i,n}),(r_{i,1},\ldots,r_{i,n})\right)$ denote a set of k canonical OLAP queries ($i\in[1;k]$). The super-query $super(Q)$ of Q is defined as follows:

$$super(Q)=\left(M_S,(g_{S,1},\ldots,g_{S,n}),(r_{S,1},\ldots,r_{S,n})\right) \text{ with}$$

- $M_S=\bigcup_{i\in[1;k]} M_i$ being the result query measure set of the super-query,
- $g_{S,i}=GLB_{\Psi|d_i}\left(\bigcup_{j=1}^{k}\{g_{i,j}\}\right)\ \forall i\in[1;k]$ being the result granularity for dimension i and
- $r_{S,i}$ being the classification node of the super query for dimension i that fulfills the following conditions[56]:

$$r_{S,i}\in\bigcap_{j\in[1;n]}\left(ancestors(r_{i,j})\cup\{r_{i,j}\}\right)\wedge$$
$$\neg\exists r'_{S,i}\in\bigcap_{j\in[1;n]}\left(ancestors(r_{i,j})\cup\{r_{i,j}\}\right): level(r'_{S,i})<_{\Psi} level(r_{S,i})\quad\forall i\in[1;k]$$

◆

Example 5.7 (Super Query for a Set of OLAP Queries)

Let us assume that we want to compute the *super-queries* for the query set $\{q_{example}, q_{example2}\}$ with

```
q_example  := ( {#repairs},
              ( all vehicles , 1999, Germany      , steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type.all  ) )

q_example2 := ( {#repairs, duration },
              ( all vehicles , Jan 1999, Germany      , steering, all types ),
              ( vehicle.all  , month, geogr. region, assembly, type.all  ) )
```

[56] Notably, such an element always exists, as the special member 'all' is ancestor of all elements in the domain of the lattice by definition.

Both queries only differ in the result set and the granularity and restriction element in the time dimension. The granularity of the super-query is *month* as both granularities are from the same classification schema path and *month* is smaller than *year*. The restriction element of the super-query is the element *1999*, as this is the common parent of *1999* and *Jan 1999*.

```
super({q_example, q_example2}):=({#repairs, duration},
       ( all vehicles , 1999, all locations, steering, all types ),
       ( vehicle class, month    , geogr. region, assembly, type.all  ) )

common({q_example, q_example2}):=({#repairs},
       ( all vehicles , Jan 1999, all locations, steering, all types ),
       ( vehicle class, month   , geogr. region, assembly, type.all  ) )
```

◆

The super-query has the useful property that it subsumes all the queries of the query set. This property is formulated by the following theorem:

Theorem 5.1 (Subsumption of Super Query)

Every member of a set of canonical OLAP queries Q is subsumed by the super-query for this query set *super*(Q). Thus,

$$super(Q) \text{ subsumes } q \quad \forall q \in Q$$

◆

Proof 5.1 (Subsumption of Super Query)

The full proof can be found in Appendix A and is a direct consequence of the construction of the super-query (Definition 5.7). ◆

The rational for considering the super-query in addition to (or instead of) the original queries is that it is likely to have a high benefit value as it subsumes all the queries of the sub-workload (see definition of benefit). Additionally, our assumption, that the query execution costs are proportional to the number of cells selected on the base data does not always hold in real systems. The reason for this is that queries with a high selectivity can benefit from special access structures (for example a multidimensional index structure, cf. [MZB99]) while queries with a low selectivity require that the base data (for example the fact table) must be entirely read which results in query execution costs that are only marginally dependent on the number of tuples read[57] if the selectivity drops below a certain threshold. Consequently, fetching a larger portion of the data only increases the execution costs by a small amount while the benefit increase is considerable. These effects are properly reflected by our prefetching benefit measure that was presented in Section 5.3.2.2. However if the workload is too heterogeneous, the super queries can become costly to store because the query result size quickly grows. In this case, super queries should be rejected by the cache admission algorithm. This is why we call the admission algorithm prior to executing the query (cf. Figure 5.5).

Considering common sub-queries. A subset of the predicted workload might be similar in the sense that they all address a common portion of the data. We call the query that retrieves these common data sets the common sub-query for a query subset. This is the query (with the largest result set) that can be used in answering all queries in the subset. The common sub-query can be constructed from a set of queries in the following way (cf. Figure 5.8): The result set of the common sub-query is chosen to be the intersection of all result measure sets of the individual queries. The results granularity of the common sub-query is determined in the same way as the result granularity of the super-query (as the greatest lower bound of the individual

[57]Of course, the aggregation process (CPU-bound) is still more expensive for larger data volumes.

result granularities in the classification lattice), because it must be possible to aggregate (partial) results of all queries from this level. The restriction element for each dimension is chosen such that the query rectangle implicitly defined by the restriction vector on the chosen result granularity (of the sub-query) is completely contained in every query rectangle (on this granularity) of all queries in the set. As the results of the common sub-query should be as large as possible (because this increases their benefit for the predicted workload), we chose the restriction element on the largest possible level[58]. In contrast to the super-query, it is not always possible to find a common sub-query for a set of queries. If the set of result measures is empty or if no restriction element can be found that fulfills the above mentioned properties, the common sub-query is undefined.

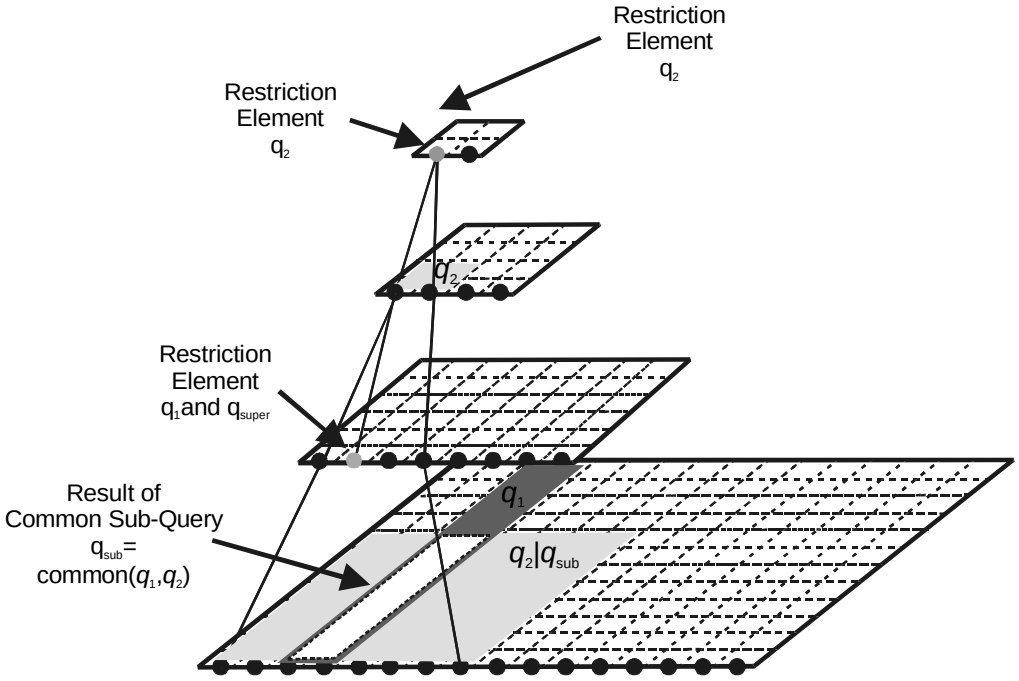

Figure 5.8: Visualizing the Common Sub-Query of Two Canonical Queries q_1 *and* q_2

These considerations lead to the following definition of a common sub-query:

Definition 5.8 (Common Sub-Query of a Set of Canonical OLAP Queries)

Let $Q=\{q_1,\ldots,q_k\}$ with $q_i=\left(M_i,(g_{i,1},\ldots,g_{i,n}),(r_{i,1},\ldots,r_{i,n})\right)$ denote a set of k canonical OLAP queries ($i\in[1;k]$). The common sub-query *common*(Q) of Q is defined as follows:

$$common(Q)=\left(M_C,(g_{C,1},\ldots,g_{C,n}),(r_{C,1},\ldots,r_{C,n})\right)$$

- $M_C=\bigcap_{i\in[1;k]}M_i$ being the result query measure set of the common sub-query,
- $g_{C,i}=GLB_{\Psi|d_i}\left(\bigcup_{j=1}^{k}\{g_{i,j}\}\right)\ \forall i\in[1;k]$ being the result granularity for dimension i and

[58] In the case of a simple hierarchy, the restriction element of the common sub-query is the smallest restriction element if all restriction elements are from the same path in the hierarchy tree (classification instance). If no path exists where that contains all restriction elements, the common sub-query is empty.

- $r_{C,i}$ is chosen such that the following condition holds:

$$\left((descendants(r_{C,i}) \cup \{r_{C,i}\}) \cap dom(g_{C,i})\right) \subseteq \bigcap_{j \in [1;n]} \left(descendants(r_{i,j}) \cup \{r_{i,j}\} \cap dom(g_{C,i})\right) \quad \forall i \in [1;k]$$

◆

Common sub-queries typically occur in the PROMISE/OLAP framework if the structural prediction model contains several successors with comparable probabilities. In this case, it is useful to add common sub-queries of these sub-workloads to the set of prefetching candidates. All (sub-)workload queries can benefit from the common sub-expression (assuming that the rewriting algorithm can make use of partial results) and the execution costs of these queries are smaller than the execution costs of the original queries. This is especially useful in situations where the consideration times are short compared to the execution times of queries contained in the workload or where the set of predictions is relatively large. However, sub-queries are only useful if the rewriting algorithm can make use of the partial containment (for example the patch working algorithm).

Using the rewriting process. All of the candidate generation methods discussed so far do not consider the cache content. However, the prefetching benefit of a candidate decreases if it 'overlaps' with an object in the cache. That means if the candidate can be rewritten using the current cache content to answer part of the query, prefetching the complete candidate is not necessary. Instead only those parts should be considered as prefetching candidates that have to be evaluated against raw data. Therefore, the candidate list generation process can use the rewriting process to increase the benefit of the candidates. In this case, all candidates are passed to the rewriting algorithm and are replaced by the resulting set of base data queries.

Of course, all of these strategies can be combined by sequentially applying them to the candidate set (starting with the original set of predicted queries). The comprehensive example presented in the next section illustrates the combination of the above heuristic strategies. It additionally demonstrates the combination of the candidate generation with the evaluation and admission process.

5.3.3 A Comprehensive Example

The following example illustrates the prefetching process using a simple example scenario that is an extension of the examples presented so far.

Example 5.8 (Prefetching Algorithm)

Let us assume that the prediction process generated the following predictive workload $P=\{(q_{example},q_2), (0.5,0.45)\}$ and that the cache contains the results of a single query q_3. The rewriting algorithm cannot find a rewriting of either $q_{example}$ or q_2 using cached objects (for example because the cache is empty).

```
q_example := ( {#repairs},
              ( all vehicles , 1999, Germany      , steering, all types ),
              ( vehicle.all  , year, geogr. region, assembly, type      ) )

q_2 :=       ( {#repairs},
              ( all vehicles , all times, Germany      , steering, routine ),
              ( vehicle.all  , time.all , geogr. region, assembly, type    ) )

q_3 :=       ( {#repairs},
              ( all vehicles , 1999, Germany      , steering, critical ),
              ( vehicle.all  , year, geogr. region, assembly, type     ) )
```

This configuration is shown in Figure 5.9.

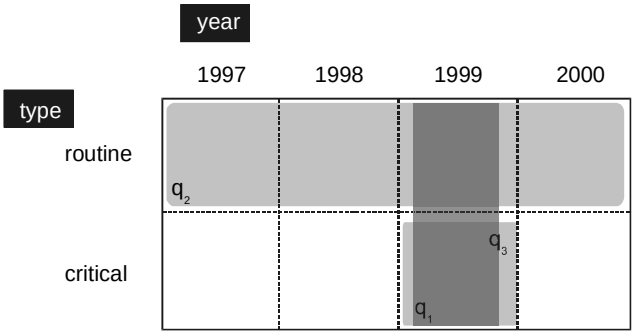

Figure 5.9: Visualization of the Cache Content (q_3) and the Predicted Workload (q_1,q_2)

The first step of the prefetching process after calculating the predicted workload P is to generate a set of prefetching candidates. If we assume that this process uses all of the techniques proposed above except the rewriting method (i.e., basic queries, super and common-sub-queries), the candidate set contains 4 queries: $\{q_1, q_2, q_{common}, q_{super}\}$, where q_{common} denotes the common sub-query of q_1 and q_2 and q_{super} the respective super-query.

```
q_common:= ( {#repairs},
           ( all vehicles , 1999, Germany       , steering, routine ),
           ( vehicle.all  , year , geogr. region, assembly, type    ) )

q_super:=  ( {#repairs},
           ( all vehicles , all times, Germany       , steering, all types ),
           ( vehicle.all  , year     , geogr. region, assembly , type    ) )
```

The next step is to compute the prefetching benefits. Table 5.2 shows the computed prefetching benefits and storage costs for all queries of the candidate set. Please note, that the prefetching benefit of q_1 is lower than in Example 5.6. This is an effect of the cache content (q_3 can be used to partially answer q_1).

Query q	**Additional Benefit** $addbenefit_\Gamma(q)$	**Execution Costs** $executioncosts(q)$	**Prefetching Benefit** $prefetch\text{-}benefit_\Gamma(q)$	**Storage costs (Array)**	**Benefit/ Storage Cost**
q_1	$0.87 \cdot 10^7$	$1.75 \cdot 10^7$	0.50	112	$7.7 \cdot 10^4$
q_2	$2.12 \cdot 10^7$	$3.5 \cdot 10^7$	0.61	224	$9.4 \cdot 10^4$
q_{common}	$1.70 \cdot 10^7$	$0.875 \cdot 10^7$	1.49	56	$30.4 \cdot 10^4$
q_{super}	$2.58 \cdot 10^7$	$7 \cdot 10^7$	0.37	448	$5.7 \cdot 10^4$

Table 5.2: Benefit and Cost Measures for the List of Prefetching Candidates

This evaluation shows that q_{common} has the highest prefetching benefit and will therefore be processed first. I.e., q_{common} is passed to the admission algorithm to make sure that the prefetched result will be admitted to the cache before speculatively executing q_{common}. Although, the actual implementation of the admission is dependent on the caching approach, it will usually compute the storage requirements for the results of the query (see Table 5.2; column 5) to check whether the cache has enough room for the result and if not which objects have to be evicted. The eviction decision is driven by comparing the benefit/cost ratio of the new object (see Table 5.2; column 6) with the benefit/cost ratio of the cached results (Table 5.3 computes this ratio for q_3 which is stored in the cache).

Notably, the eviction of objects from the cache can influence the benefit values of the prefetching candidates. This effect is not addressed in our heuristic algorithm. This simplification

is justified, as we assume that the cache is large enough, so that the objects can be evicted from the cache that have no effect on the benefit of the prefetching candidates (i.e., a benefit value of 0).

Cached Query	Probabilistic Benefit	Storagecosts (Array)	Benefit/Cost
q_3	$0.46 \cdot 10^7$	56	$8.2 \cdot 10^4$

Table 5.3: Evaluations of the Costs and Benefit of the Cache Content

◆

5.4 Summary and Conclusions

This chapter thoroughly discussed the impact of the PROMISE/OLAP algorithms to OLAP caching systems. Therefore, it completes the core of this thesis by demonstrating the usefulness of the prediction approach for an important exemplary application.

Our starting point was the observation, that caching strategies can apply predictions of future queries in two ways: in order to improve eviction and admission strategies through predictive benefit estimations and for speculative execution of queries. A comparison of existing caching approaches has shown that all OLAP caches apply a query level cache which is a variation of the idea of semantic caches ([DFJ+96]). The main differences of the algorithms are their approach to the query rewriting problem and to estimating the benefit of objects (for admission and eviction decisions).

A main objective of the PROMISE extensions is that the approach should be applicable with all the currently proposed OLAP caching solutions. To this end, we developed an abstract caching model which subsumes all the approaches discussed in 5.1.1. This model is not only useful for the discussion of our approach but can serve as a general framework for the comparison of query-level caching approaches beyond the scope of this thesis (for example for the combining different caching approaches). An elegant integration of our extensions, reusing components of the algorithms and other extensions, indicates the quality of the modular design. A model for the comparison of costs for different execution plans using cached objects completed the description of the abstract PROMISE caching framework. However, due to the modular nature of the framework, it can be easily integrated and extended with systems that already deploy highly sophisticated estimation mechanisms (for example cost based optimizers of database systems).

The first extension proposed in Section 5.2 was to use the predictions in order to estimate the benefit of cached objects. To this end, we introduced two benefit measures that are based on the predicted workload and discussed an additional method to increase benefit estimations by using knowledge about OLAP user interface transformations. The common innovative feature of both methods is the usage of dynamic information about the session context for benefit estimation. However, both benefit estimation methods can be combined with each other and further traditional methods of benefit estimation.

The second extension (Section 5.3) discussed the usage of speculative execution techniques based on the predicted workload. We algorithmically described the integration of the extension into the caching framework and concluded that a prefetching candidate generation process incorporating a special prefetching benefit measure has to be designed. We formalized the prefetching problem as a variation of the view selection problem and proposed different heuristic methods for its solution. Notably, we based the definition of the prefetching benefit

measure upon the benefit estimation functions presented in Section 5.2. In this way, the effort for implementing the extensions can be reduced.

Although we described the extension of the caching framework specifically for OLAP systems, the results of this chapter can be applied in a much wider area. Semantic query level caches have also been discussed for relational database systems ([DFJ+96]). Our general framework and algorithms do not rely on any assumptions about the definition of query subsumption and the rewriting process. Therefore, the functional cache description framework, the predictive benefit measures, the definition of the prefetching candidate generation problem and its heuristic solutions can be regarded as a foundation for systematically researching the combination of prefetching techniques with query level caches in database systems in general. This means that our results can be applied to a much wider context.

At this point, the description of algorithms, formalisms and techniques covered by this thesis is complete. The rest of this thesis will be devoted to the empirical evaluation of the algorithms using a trace-driven environment (Chapter 6) and the discussion of possible extensions (Chapter 7).

«A theory is something nobody believes,
except the person who made it.
An experiment is something everybody believes,
except the person who made it.»
-- Albert Einstein, attributed

«In theory, there is no difference
between theory and practice.
In practice, however, there is.»
-- Unknown

Chapter 6 Empiric Evaluation of Prefetching Techniques

This chapter is devoted to the practical evaluation of the framework that has been described in the core of the thesis. The design of the pattern model and the prediction and caching algorithms in the previous chapters are heuristic in the sense that they are based on certain assumptions about the behavior of the user and the OLAP data. Therefore, a performance evaluation of the framework has to be done empirically.

In order to show that our basic assumptions about the characteristics of OLAP systems can be found in real-world applications, we present an analysis of OLAP behavior characteristics found in a real OLAP environment in the Section 6.1. For the performance measurements of the PROMISE/OLAP approach, we use a simulation of user behavior based on a real data warehouse schema measured against real-world data warehouse data. The architecture of the evaluation framework and the description of the simulation scenario is detailed in Section 6.2. The results of the measurements are presented in Section 6.3. Finally, Section 6.4 summarized the results and draws conclusions.

6.1 Empiric Study: Characteristics of OLAP User Behavior

Our approach is based on the assumption that the system workload posses the following characteristics:

- *Session Oriented Navigational Behavior*. Sessions consist of a number of consecutive queries. These sessions must have a sufficient length in order for prediction to work effectively. Additionally, longer sessions are an indicator for navigational user behavior characteristics.
- *Sufficient time for prefetching.* Additionally, the time between two queries must be long enough such that a significant percentage of the next query can be prefetched (computed) during this period. The pure prefetching assumption ([CKV93]) postulates that the time between two requests is long enough to precompute the whole query.

To show that these two prerequisites can be found with typical OLAP workloads, we analyzed the query interactions of a productive OLAP system. The system being analyzed is an SAP BW® system supporting the distribution logistics of the material management division in a large chemical company in Germany. The interaction behavior of *18* users was monitored over a two-month period including *260* sessions containing *3150* queries.

The log file contains an entry for every query the user executes using his OLAP front-end. For each query the starting time and the execution time are being logged. It further contains information about the logon and logoff procedures of an user. A session is defined as all the queries that are issued by a single user between logging into the system and logging off the system.

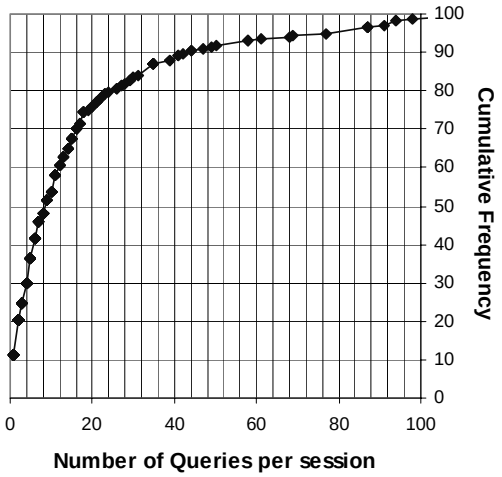

Figure 6.1: Cumulative Distributions of Consideration Time and Number of Queries per Session

The hypothesis that OLAP users work in an explorative navigational way is confirmed by an analysis of the number of queries (cf. multidimensional operations) an user executes per session. Figure 6.1 shows the cumulative frequency distribution of the number of queries per session. Although the system is mainly used for static reporting purposes, only *11*% of all the sessions consist of executing a single query (simple reporting). Some sessions contain more than *100* queries. *63,8*% of the queries contain *5* or more navigation steps. The median for the number of navigation steps is *9*.

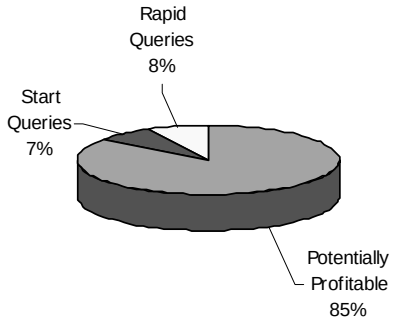

Figure 6.2: Percentage of Queries for which the Pure Prefetching Assumption is Fulfilled

In order for speculative materialization to be successful, the time between two subsequent navigation steps (the user's consideration time) must be relatively long compared to the execution time. An analysis of the log files reveals that *7*% of the queries cannot benefit from prefetching because they are the first query in a session (as sessions typically start with a predefined report, they are likely to benefit from traditional caching). For only *8*% of the queries, the pure prefetching assumption (consideration time larger than execution time) was not fulfilled. This means that it is fulfilled for *85*% of the queries (cf. Figure 6.2). A further analysis

showed, that consideration time in most of the cases was orders of magnitude longer than the execution time of the next query.

An analysis of the distribution of consideration times can be seen in Figure 6.3. It plots the cumulative frequency distribution of the consideration time (time spent between two navigation steps). The median of the distribution is *121* seconds. Over *81*% of the queries have a consideration time of more than *10* seconds and *70*% have a consideration time of more than *45* seconds. Additionally, it can be seen that the consideration time follows a logarithmic distribution.

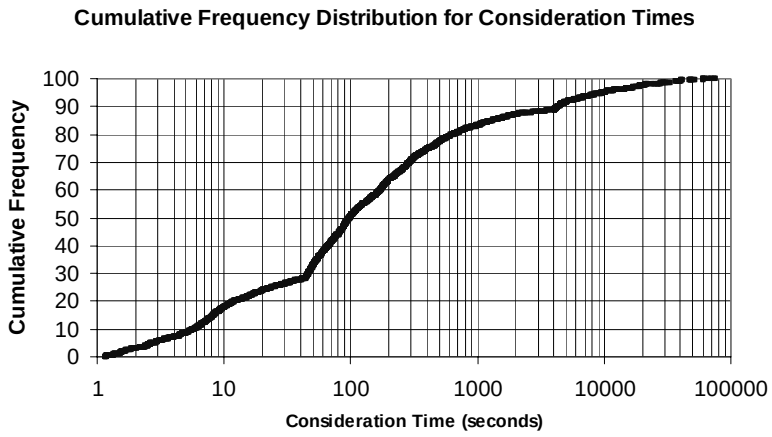

Figure 6.3: Cumulative Frequency Distribution for Consideration Times

Thus, these observations confirm that our basic assumptions about the navigational behavior and the consideration times of OLAP users can actually be found in real world systems.

6.2 The PROMISE/OLAP Evaluation Environment

The purpose of this section is to describe the software architecture of the evaluation environment (Section 6.2.1), the structures of the data (Section 6.2.2), and the user simulation (Section 6.2.3) that were used in the performance measurements.

6.2.1 Architecture

This section briefly describes the architecture of the system that we used in our performance measurement experiments. A more detailed description of the component's design and implementation including the object-oriented model for the MD schema and the OLAP engine can be found in [Sch01].

The PROMISE/OLAP prototype evaluation environment (cf. Figure 6.4) consists of the following components:

- *User Simulation(Trace Generator).* This component simulates the behavior of a user according to a *user model*. This model can be parameterized by a set of *simulation parameters*. By varying them different types of users can be simulated. Section 6.2.3 describes the generation process and its parameters in more detail. The output of the generator component is a user trace file that contains a description of the sessions including the according consideration times. This output consists of a *training set* with a predefined number of sessions that are used to train the model prior to the measurements and an *evaluation set* that is used for performance measurements. This trace file is the only communication be-

tween the simulation and the measurement environment. Especially, the user profile used for simulation purposes is not known to the measurement environment. The MD schema, which is needed for the query generation (and the OLAP engine, see below) is stored in a file that is read at system startup time.

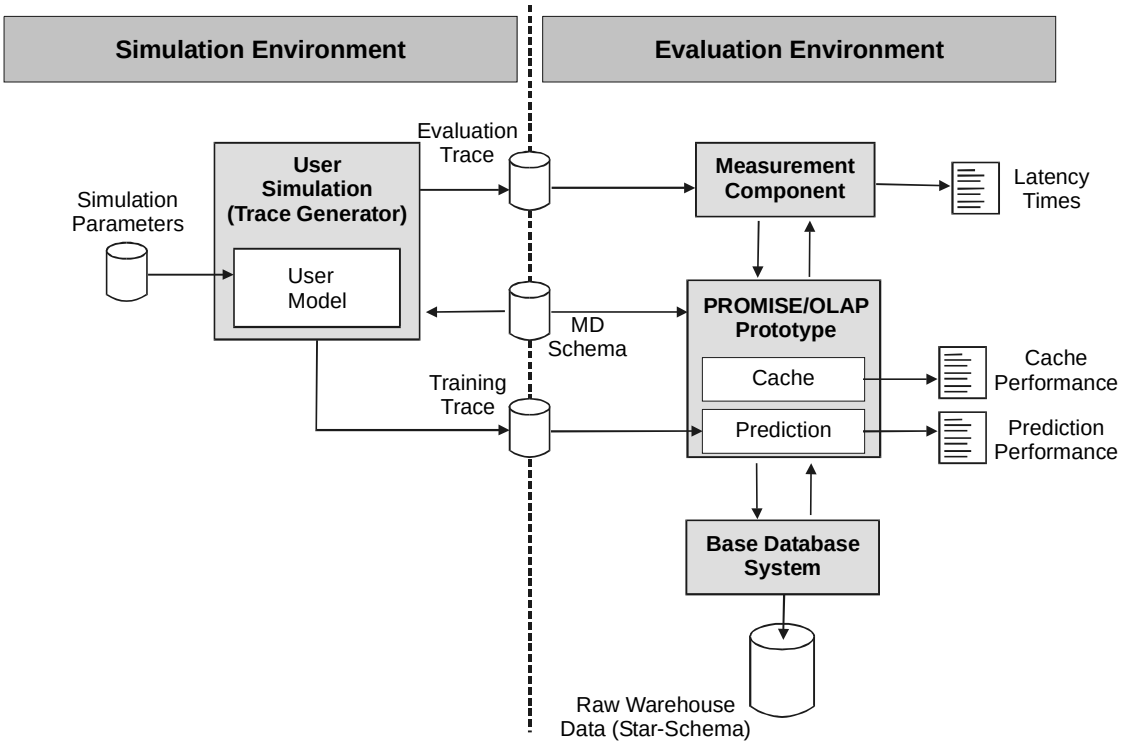

Figure 6.4: The Architecture of the PROMISE/OLAP Evaluation Environment

- *Measurement Component*. This component is a simulation of the front-end component and the user. It uses the query trace file generated by the generator component to execute queries according to this specification and measures the result times. It communicates with the PROMISE/OLAP prototype using a simple interface allowing for the execution of canonical OLAP queries.
- *PROMISE/OLAP Prototype*. This component is a prototypical implementation of an OLAP engine which includes the PROMISE/OLAP extensions. It contains an implementation of a general caching system (see Section 5.1.2) with according measurement facilities. The caching strategies used in the evaluation are detailed in Section 6.3.3. The rewritten queries that have to be evaluated against raw data are translated to SQL and send to the base database system.
- *Base Database System*. The base database system takes the role of the storage manager in the OLAP reference architecture, processing incoming queries. For the measurements, we used the relational database system MS SQL Server Version 7.0. However, in order to study the interrelationships of multidimensional indexing techniques with our approach, we also performed selected measurements using the Transbase Hypercube product which offers UB-tree indexing (cf. Section 6.3.3.5).

All custom-made components of the framework (most importantly the PROMISE/OLAP prototype) are designed using the object-oriented paradigm and coded in C++. The code was compiled using MS Visual C++ 6.0. Physically, the components are distributed using a two-tier architecture, where the base database system runs on a dedicated server. Consequently, the PROMISE prototype and the measurement component run on a separate machine. The server machine is a two processor machine equipped with two Intel Pentium III 400 MHz processors,

768 MB of RAM running Microsoft Windows NT Server 4.0. The client machine is equipped with an Intel Pentium III 500 MHz processor, 256 MB of RAM, and running Microsoft Windows NT Workstation 4.0.

6.2.2 Test Data Schema and Volumes

The evaluation uses a real-world scenario. The data is an extract from a real-world data warehouse of a large German market research company. It contains data about the sales of non-food products in different stores. The cube schema (cf. Figure 6.5) has three dimensions: *Product*, *Time* and *Segment*. Each of the dimensions is structured in a hierarchical way. The numbers shown in the diagram denote the cardinality of the domain of the classifications levels.

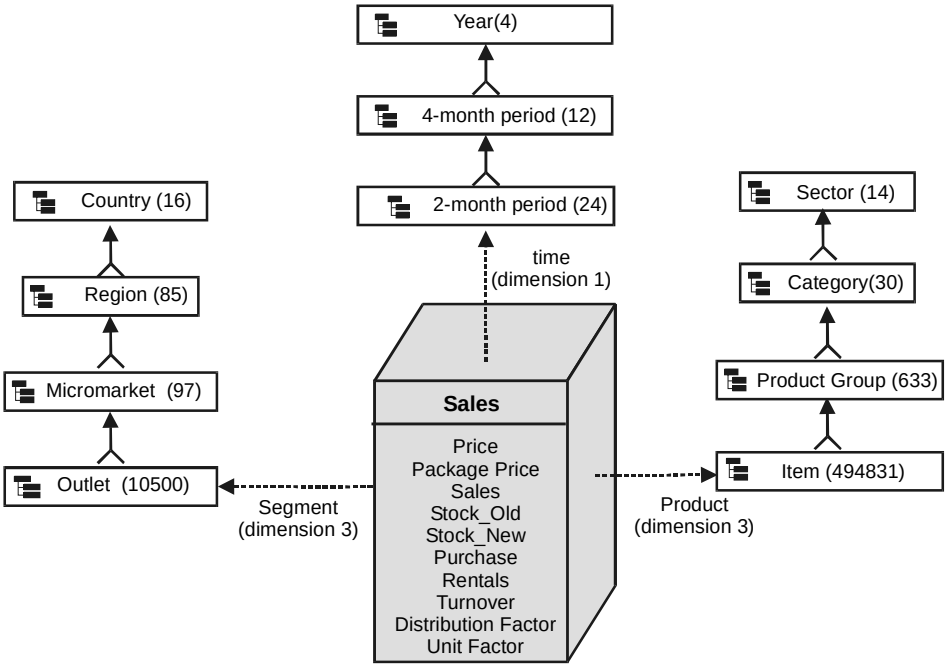

Figure 6.5: The Multidimensional Schema of the Data Used for the PROMISE Evaluation

The data is physically stored in a relational star schema with one fact and three dimension tables. The size of the fact table is *42* million tuples which corresponds to *2.3* GB of raw data. The foreign keys between the fact table and the dimension tables are realized using compound surrogates that take into account the structure of the classification hierarchy ([MRB99]). The fact table is indexed using a compound clustering index over the surrogates of *Product*, *Segment* and *Time*.

6.2.3 Simulating Dynamic User Behavior

Instead of using real users in the system evaluation, we use a simulation generating user interaction traces to drive the evaluation process. This allows us to influence the user behavior and vary different parameters (e.g., session length, number of random queries) in a controlled way. This is very helpful in studying the impact of different kinds of workload characteristics on the PROMISE techniques. The simulation is based on a set of Markov models that is similar in structure to the models used in the prediction. Note however, that the actual models used for the generation are not known to the PROMISE evaluation framework. Instead, the evaluation framework builds its own model from the training and evaluation traces. Simulation state transitions in a Markov model works similar to the prediction algorithm: Starting with the

current state, the system randomly picks a successor state taking into account the respective transition probabilities. Analogously to the prediction process, the simulation process is driven by the structural prediction model. Figure 6.6 shows the structural prediction model (including the according change vectors) that is used in the simulations. It models a typical briefing book scenario. States *1*, *2* and *3* correspond to the pages (predefined reports) of the briefing book that are normally processed sequentially. Therefore, transition probabilities according to the sequence of the reports (state *1* → state *2*, state *2* → state *3*) are higher than the probability for traversals in the opposite direction. When looking at a report, the analyst typically changes one of the selection parameters. This is mirrored by transitions from the according state to itself. The change vector has a strong preference for changing restriction values in a specific dimension. E.g., when analyzing the report represented by state *1*, the *4-Month Period* is typically changed. Another interactive possibility is to perform a drill-down operation in order to investigate a single value. This behavior is represented by states *4* and *5*. The transition from state *3* to state *1* models the fact that the analyst restart the process to perform his analytical task again.

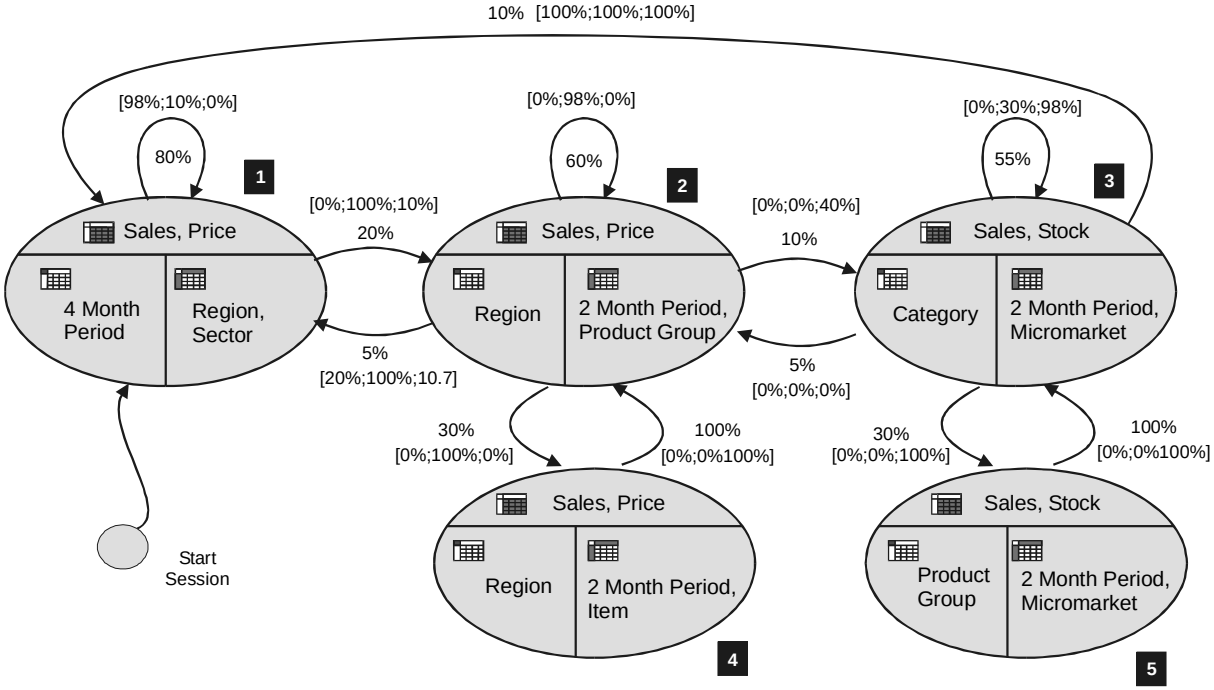

Figure 6.6: The Structural Prediction Model Used for the User Simulation

The value-based models for the simulation are generated automatically from the hierarchical structure of the nodes in the classification lattice of the dimensions. They contain transitions from each node to its siblings and to its parent and the first of its children. For example, the node '*1999*' contains transitions to '*1998*', '*2000*', *Time.all* and '*Jan99*'). This models the navigational changing of restriction elements along the hierarchical structures (cf., section 3.3.2). Additionally, the simulation environment automatically adjusts the value-based simulation models, if transitions in the structural model require value-changes that are not contained in the current model. The structure- and value-based simulation models are the most important parameters to the simulation process. Additionally, the following parameters can be varied:

- *Parameters for Consideration Time Generation*: For each query, the simulation generates a random consideration time that will be used by the measurement component in order to simulate the timing of the query execution. The simulation supports different distributions (Gaussian, Logarithmic, Exponential) which are used to determine this consideration time.

The parameters of the according distribution (e.g. expectation and standard-deviation) are passed as a parameter to the simulation component.

- *Number of Sessions and Session Length*: The number of sessions generated by the simulator is a fixed number passed as a parameter, The session length is randomly determined according to a random distribution function. Analogously to the random generation of consideration times, the distribution parameters for the session length are parameters of the simulation process.
- *Random Query Probability*: In order to reflect the fact that the user will not always follow the pattern defined in the profile, we introduce a random query probability parameter that can be set for the simulation. It gives the probability that the simulator generates a random query instead of using the pattern profile. The random query is generated based on the last query in the session randomly choosing a multidimensional query transformation (cf. section 3.3.2) and its parameters.

The generated queries together with the timing information are written to a file which can be read by the measurement component. The following section presents selected results of measurements with different parameter sets for the simulation and the prediction and caching algorithm.

6.3 Measurement Results

The overall performance of the system that is observed by the user is a combination of the performances of the different components of the PROMISE framework and their interaction. In order to enable a detailed analysis it is necessary to separately measure the performance of the different components with respect to different influencing factors. Therefore, we start by testing the runtime performance of the prediction and induction algorithm (Section 6.3.1) under different parameters. Then, we evaluate the accuracy of the isolated prediction algorithm (Section 6.3.2). Finally, we measure the combined performance of the prediction and caching algorithms with and without speculative execution (Section 6.3.3).

Primarily, it is interesting to evaluate the absolute performance of the respective component to prove the practical feasibility of the PROMISE concepts. Additionally, we investigate the impact of varying parameter settings on the performance in order to prove the applicability of the approach to different environments and to identify the main influences on the performance. We can distinguish two types of parameters: the parameters of the user simulation process (cf. previous section) mirror different types of workload characteristics. They are *external parameters* to the PROMISE approach, as they cannot be influenced in a real world application environment. We perform experiments with different external parameter settings in order to evaluate how well our approach performs under varying sorts of user behavior. Additionally, the PROMISE framework and the caching algorithms also have a set of *internal parameters*. These can be changed in a running system (for example by the system administrator). The goal of experiments with different internal parameter settings is to verify our theoretical considerations and to gain insights into the impact of these parameters on the overall performance. Using these results, it is possible to give hints and heuristics on finding the optimal parameters settings. The internal parameters of PROMISE prediction algorithm are:

- *Prediction Threshold.* The central parameter of the PROMISE prediction algorithm is the prediction threshold that determines which of the Markov model successor states are considered for building the result set. For our experiments, we assume that the same threshold is used for both, structural- and value-based prediction. The theoretical range of the threshold is]*0*;*1*]. However, in our experiments, we will only use the reduced range

[*0.1*;*0.7*] as smaller thresholds lead to unrealistically large result sets and higher thresholds produce empty result set for most of the predictions.

- *Order of the Markov Models*. We used Markov models of order 1 in all examples and algorithm descriptions throughout the thesis. However, the prototype implementation supports higher order models making use of the transformation described in Theorem 4.1. It also allows for using different orders for structural and value-based models. In our experiments, we used models of order *1* and *2*. Thus, four different combinations were measured: *1*/*1* (structural model of order *1*; value-based models of order *1*), *2*/*1*, *1*/*2* and *2*/*2*.
- *Number of Queries used for Training.* The PROMISE algorithm dynamically builds and adapts the pattern information by training the Markov models with the observed interactions. The quality of the predictions is dependent on the number of sessions that are used to train the model. Therefore, we varied the size of the training-set form from *0* queries (empty model) to *10.000* queries.

Additionally, to these PROMISE parameters, the caching algorithm has the following parameters:

- *Maximum Cache Size*. The cache of our prototype implementation uses array structures to store the cached data. For our experiments, we varied the size of the cache between *100* KB and *2* MB.
- *Fetching Strategy*. In order to compare our solution to the state-of-the-art techniques, we performed experiments with demand fetching strategies and speculative fetching strategies.

Of course the parameter space is far too large to allow for exhaustive experiments using all parameter combinations. Besides, not all combinations are sensible for all measurements. The following Table 6.1 gives an overview of the parameters and measurements that we performed and the results that will be presented in the next sections.

Evaluation Category	Measured Values	Influencing Parameters
Execution Performance (Section 6.3.1)	■ Execution Time Prediction ■ Execution Time Training	■ Threshold Value ■ Markov Model Order
Accuracy of the Prediction (Section 6.3.2)	■ Prediction Hit Rate Next Query ■ Avg. Size of Prediction Set ■ Avg. Probability of Predictions	■ Threshold Value (6.3.2.1) ■ Markov Model Order(6.3.2.2) ■ Number of Training Sessions (6.3.2.3) ■ Rate of Random Queries (6.3.2.4)
Performance of the Caching Algorithm (Section 6.3.3)	■ Cache Hit Rate ■ Latency Time Reduction	■ Threshold Value (6.3.3.1) ■ Maximum Cache Size (6.3.3.2) ■ Consideration Time (6.3.3.3) ■ Rate of Random Queries (6.3.3.4)

Table 6.1: Measures Values and Varied Parameters for the Evaluation Experiments

6.3.1 Runtime Performance of the Prediction and Training-Algorithm

In the PROMISE framework, both prediction and training processes are carried out online during the 'normal' operation of the system. Therefore, it was an important design goal of these algorithms that they can be executed in a very short time and that the execution time is

stable against changes of the parameters. Therefore in this section, we measure the absolute times for the execution of the prediction and the training algorithm in order to ensure that the OLAP system performance will not be affected by this additional overhead.

Additionally, we investigate the scalability of the algorithms by measuring the performance under different parameter sets. The theoretical analysis in Section 4.3 concluded that the prediction performance is influenced by the dimensionality of the model and the prediction threshold. The prediction threshold determines the worst case complexity of the basic Markov prediction (for one Markov model) and also determines the number of times, the prediction algorithm will be called for each of the prediction models. Therefore, we measured the prediction performance for different threshold values. The training performance behaves independent of the threshold variation.

It is important for the scalability of the algorithm, that its performance is constant in the size of the Markov model and in the order of the Markov model. To show that this claim holds in a practical implementation, we measured the performance for predictions with Markov models of different orders. A higher order model automatically has a larger number of states (the number of states grows exponentially with the order), thus it is not necessary to vary the size of the models.

The results of these experiments are shown in Figure 6.7 for the prediction algorithm (*a*) and the training algorithm (*b*). We used a training set of *100* session with *100* queries each to train the model. Then for another *100* sessions with *100* queries, the prediction was performed. The diagrams plot the average execution times for the algorithms under these conditions. The horizontal axis shows the different values for the threshold parameter, while the different lines in the diagram represent measurements with different Markov model orders (for example *2/1* stands for order *2* for the structural prediction and order *1* for the value-based prediction).

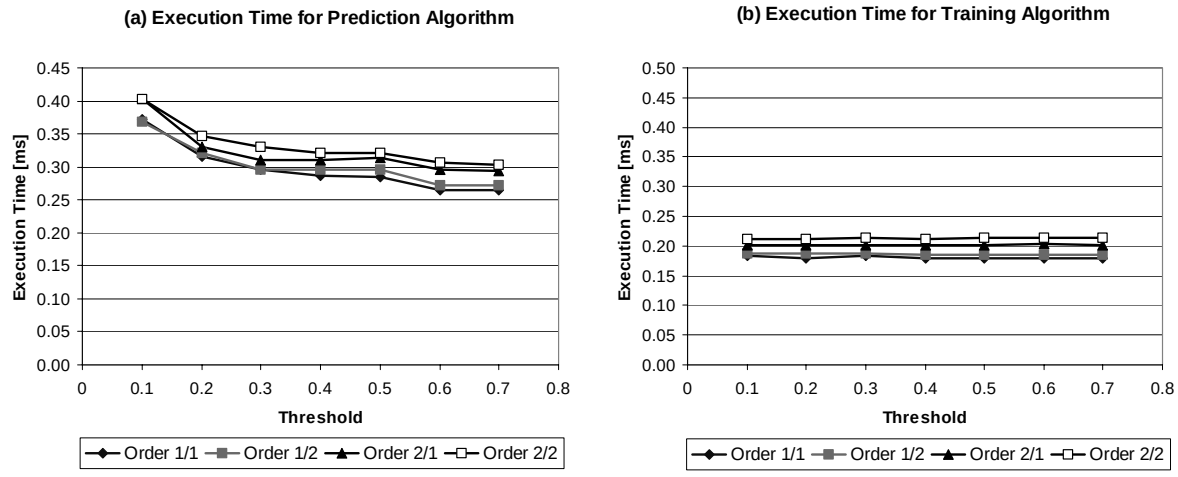

Figure 6.7: Execution Times for the Prediction and Training Algorithms

The first observation concerning the absolute range of the execution times shows that execution times in our setup (Intel Pentium® III, 500 MHz Processor) are between *0.45* ms and *0.3* ms for the prediction and around *0.2* ms for the training. This means that the prediction during the query processing will not slow down the overall system's performance.

The measurements also confirm that the prediction performance grows with decreasing threshold. The reason is that a smaller threshold increases the average result set size for the

structural prediction (cf. Figure 6.8 (*b*)). As for each structural prediction, a value-based prediction has to be performed, the average result set size for the overall prediction grows linearly with the size of the set.

Not surprisingly, the measurements (cf. Figure 6.7 (*b*)) show that training performance is independent of the prediction threshold, as transition information is updated irrespective of the threshold value. Another theoretical observation that is confirmed by the measurement is that execution times of both prediction and training algorithm behave constant in the model size and state space size (higher order models have a much larger state space size). This behavior is a consequence of organizing the state space as a hash table and ensures that the prediction algorithm scales well to larger models with a larger number of states.

6.3.2 Accuracy of the Prediction

The accuracy of the predictions provided by the prediction algorithm is a critical factor for the performance of the overall system. The number of correct predictions limits the number of cases in which the caching algorithm can make use of the results. Therefore, in this section, we analyze the accuracy of the prediction/training framework without the caching framework. To this end, we executed *100* sessions with *100* queries each (*10000* queries). After each query, we predicted the set of probable next queries using the prediction algorithm. We then compared the next query contained in the trace file with the set of predicted queries, classifying it as a *prediction hit* when the actual query is in the set of predictions. We use the *prediction hit rate* as an indicator for the quality of the prediction accuracy.

However, a good prediction hit-rate may be achieved by producing a large set of queries, each with a small predicted probability. A large size of the result set can nullify the benefit of a large hit accuracy for the application because this algorithm has to process a large amount of predictions. For example, the speculative execution algorithm has to prefetch all the candidates in order to be sure that the hit rate of the prediction algorithm is also achieved by the caching algorithm. Therefore we additionally measure the *average number of predictions* contained in the prediction result and their *average probability*. This gives further indications about the behavior of the prediction algorithm.

Our experiments investigated how these measures are affected by changes in the prediction threshold (Section 6.3.2.2), the Markov model order (Section 6.3.2.2), the size of the training set (Section 6.3.2.3) and the rate of random queries (Section 6.3.2.4).

6.3.2.1 Impact of the Prediction Threshold

The most important parameter for the performance of the algorithms is the prediction threshold. Figure 6.8 shows the results of a measurement with a training set of *100* sessions training and a Markov model order of *1* for both structural and value-based prediction. For a threshold value of *0.1*, the *prediction hit rate* (diagram *a*) is above *85%*. It decreases to about *55%* for medium threshold values (*0.3* to *0.5*) and further decreases to below *30%* for high threshold values.

When examining the *average size* of the set of queries that are predicted (diagram *b*), it can be observed that the set size decreases rapidly for threshold values under *0.3* from *5* queries to *1* query. It is also obvious (diagram *a*) that the average probability of the predicted queries increases with higher thresholds as expected. That means that low threshold parameters produce large sets of predictions with a lower average predicted probability, while high thresholds produce very small sets of predictions with high significance.

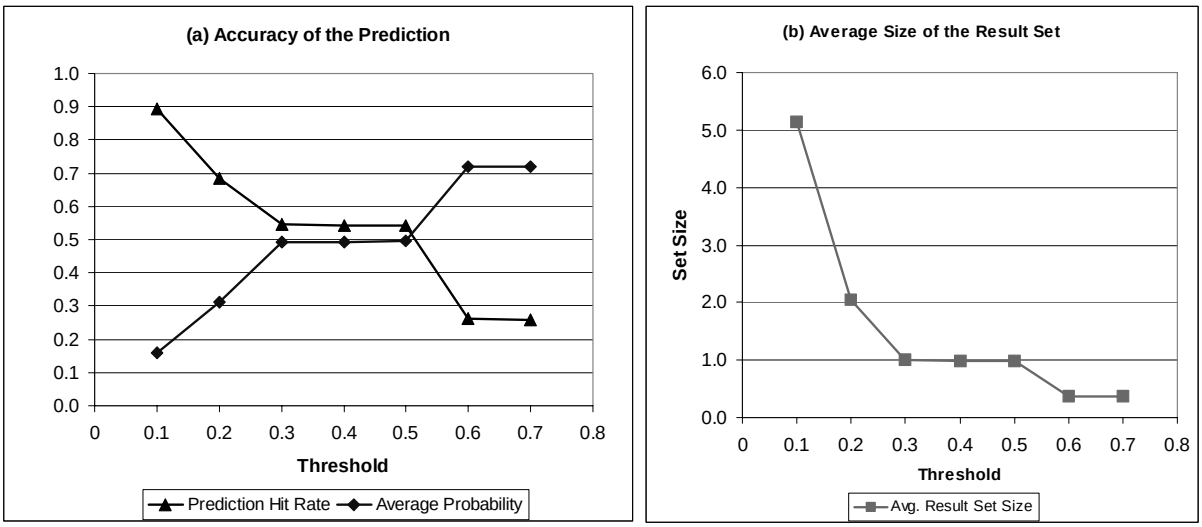

Figure 6.8: Behavior of the Prediction Algorithm under Different Threshold Parameters

All measure values show a constant behavior between *0.3* and *0.5*. This can be explained by examining the distribution of the transition probabilities of the different prediction models after the training phase. Figure 6.9 plots histograms for the structural prediction model (SPM) and the value-based prediction models (VPMs) for the different dimensions. It is obvious that all the distributions have peaks around *0.2* and *0.7* but only a small number of transition probabilities lie in the range [*0.3*;*0.5*].

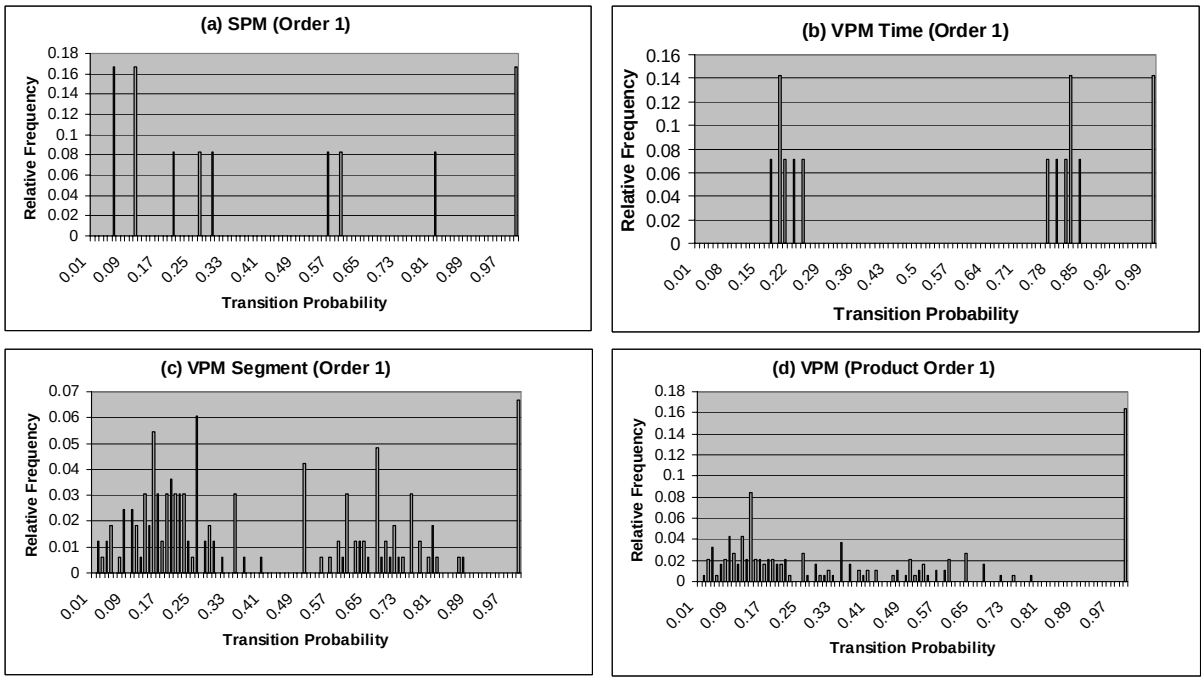

Figure 6.9: Distribution of the Transition Probabilities for the Different Prediction Models

The ideal choice of the prediction threshold is of course dependent on the application of the prediction results (for example speculative execution). It is a tradeoff between a high prediction hit rate (for low threshold values) and a small set of highly probable predictions (high threshold). The ideal point depends mainly on the ability of the application to handle large result sets because the number of results increases overproportionally compared to the hit rate

with shrinking thresholds. In section 6.3.3, we will analyze the impact of different thresholds for speculative cache environments.

However, a valuable mean to determine the correct setting is the analysis of the transition probability distributions. That way, the administrator can choose an appropriate threshold. An interesting topic for future research would be to develop techniques that dynamically adjust the threshold parameter to model by automatically analyzing the histograms.

6.3.2.2 Impact of Markov Model Order

The order of the Markov models deployed in the PROMISE prediction, represent the number of previous queries that are taken into account when predicting the next possible query (look-back window size). It can be independently chosen for the structural and the value-based model. However, a higher order increases the number of transitions that have to be managed by the system. In order to examine the impact of using different orders for Markov models within the PROMISE approach, we performed measurements with orders *1* and *2* for both structural and value-based models (yielding in 4 combinations). The experiments used a training set size of *100* sessions.

Figure 6.10 contains the measurement results for the three key performance metrics: prediction hit rate (*a*), predicted probability (*b*) and prediction set size (*c*). The first observation is that the increase in Markov model order does not have a significant impact on the properties of the prediction algorithm. However, as we used an order-1 model for user simulation purposes, this was not to be expected. Higher order models perform slightly better for high prediction thresholds.

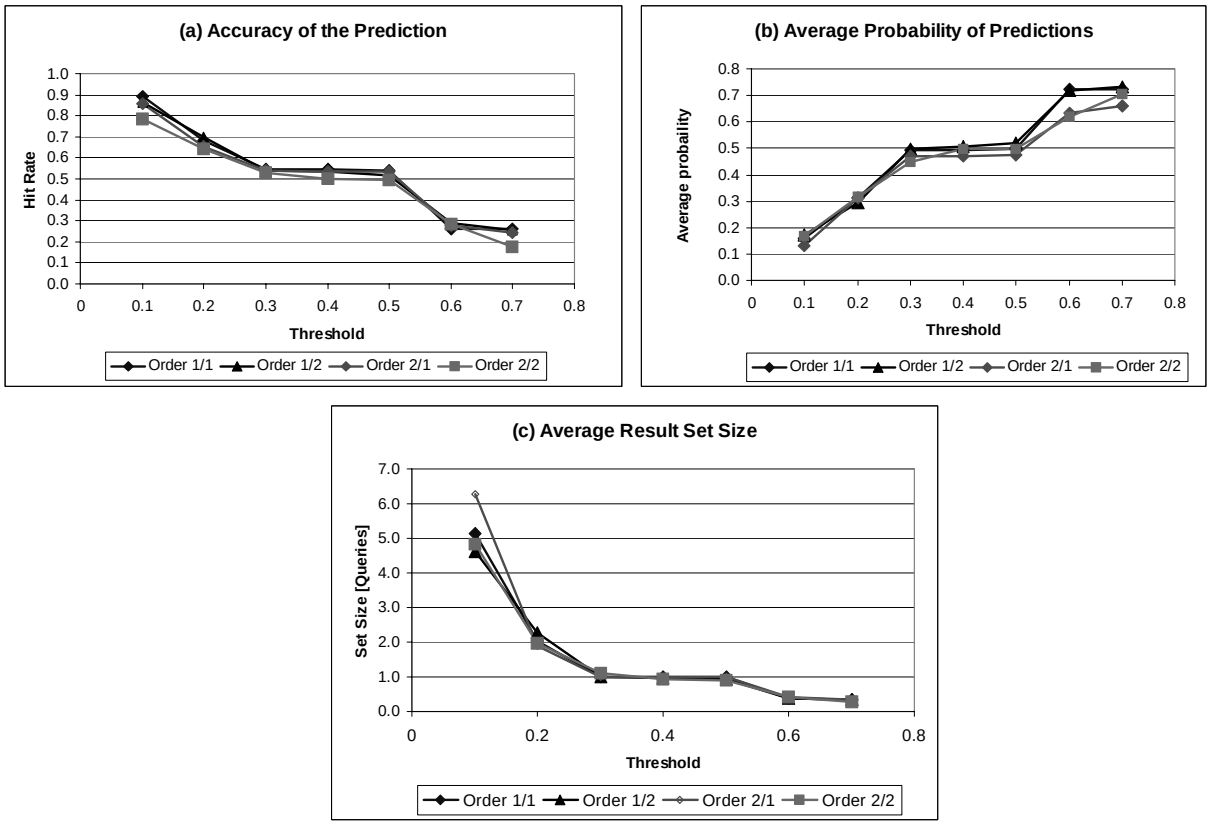

Figure 6.10: The Impact of Choosing Different Markov Model Orders

Figure 6.11 shows the frequency distributions for the transition probabilities of the order *2* models. By comparing these diagrams to Figure 6.9, it can be observed, that the peaks of the distribution are at the same probability values, which explains the similar behavior of the order *1* and order *2* models.

The conclusion for practical applications of the PROMISE framework is, that the ideal order can be determined by (manually or automatically) comparing the distributions for different orders and choosing the highest order that still leads to significant changes in the distribution.

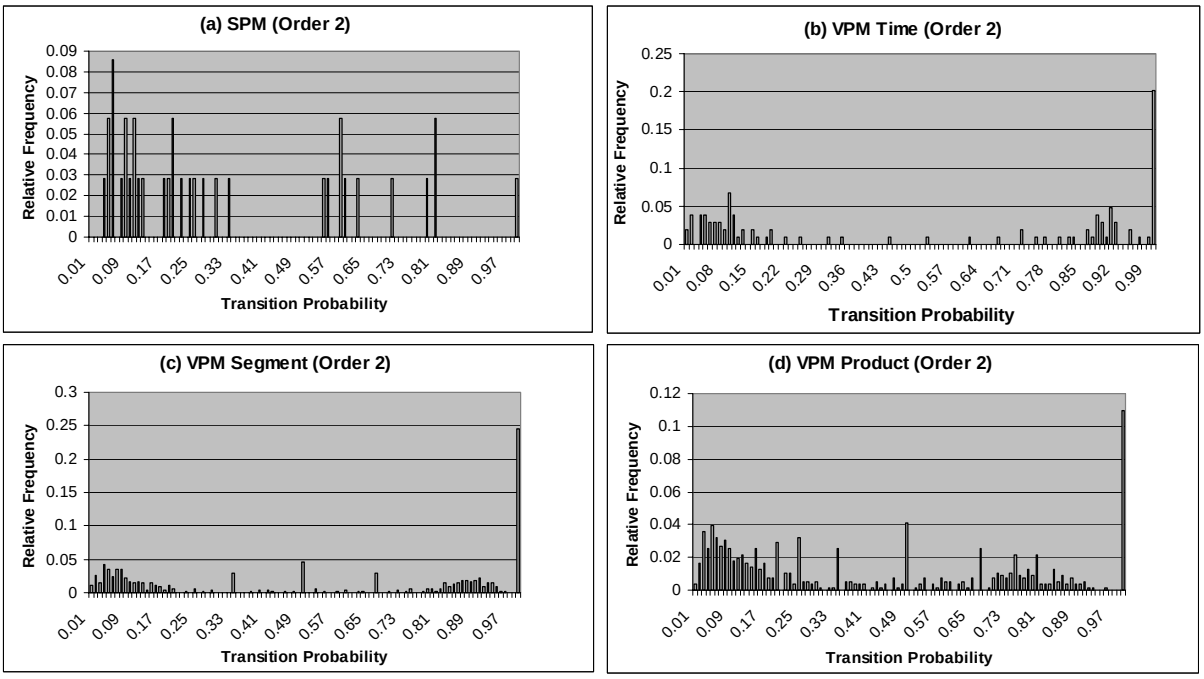

Figure 6.11: Histograms of the Transition Probabilities for Order 2 Models

6.3.2.3 Impact of Training

In order to evaluate the performance gained by training the model with sessions observed in the past, we measured the *prediction hit rate* after a varying number of training sessions. During normal operations, the model is constantly trained with every arriving query. However, to analyze the impact of different training set sizes on the prediction accuracy, we disabled this feature. This means that the model is only trained using the training set and remains unchanged throughout the whole evaluation phase.

Figure 6.12 summarizes the results of these experiments for *100* experimental sessions. Diagram (a)-(c) plots the average prediction hit rate measured after a varying number of training sessions for different threshold values. The different lines in diagrams represent different Markov model orders. As expected, the hit rate without training is *0* as no predictions can be performed. However, the hit rate quickly grows to the maximum rate. A performance within *90*% of the maximum performance is already achieved after *25* sessions of training (*2500* queries). Notably, systems with a higher threshold value reach this stable state earlier than systems with a low threshold. This is a consequence of the frequency counting algorithm: The high prediction hit rate of low threshold systems stems from the fact that they also consider transitions with low probabilities. However, these transitions are rarely traversed which requires a larger set of training queries in order to contain enough traversals of these transitions.

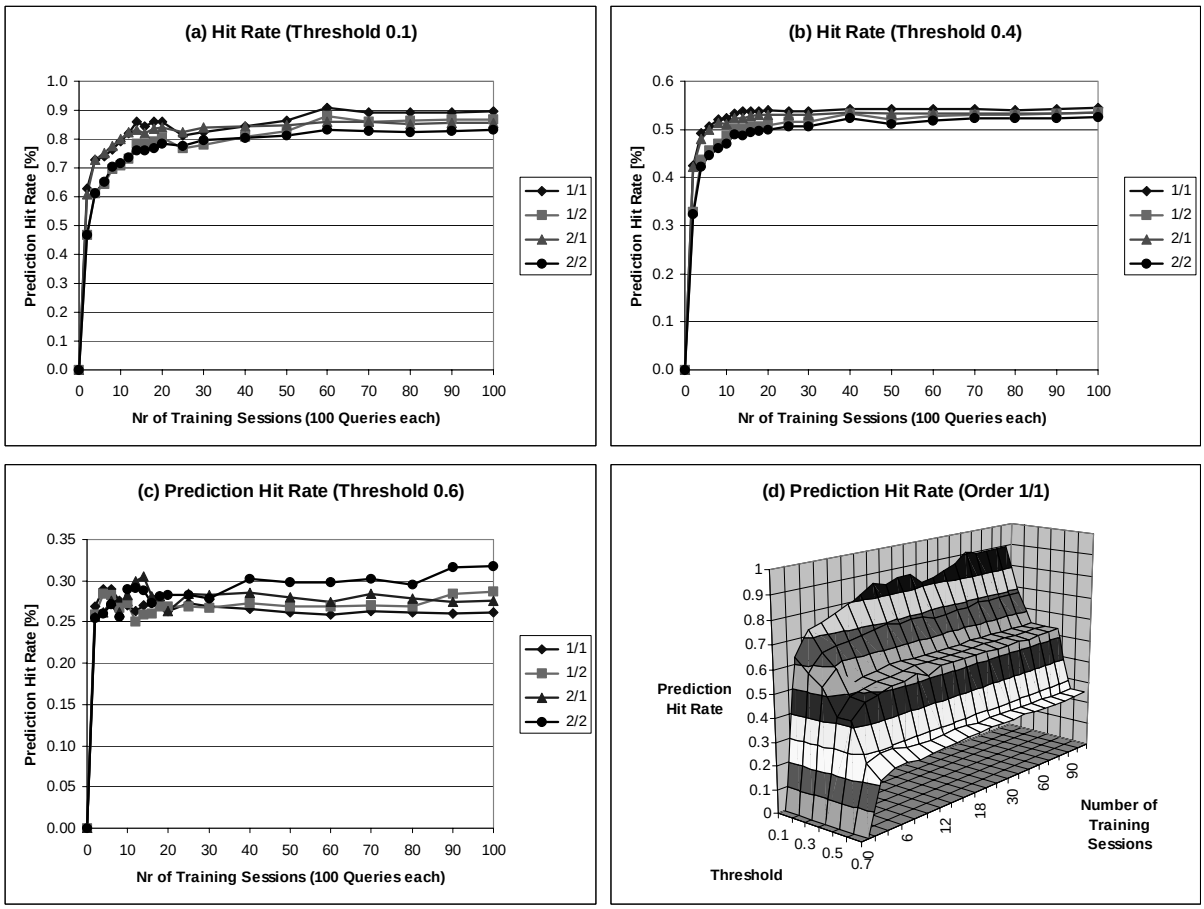

Figure 6.12: Prediction Accuracy for Different Training Sizes, Threshold Values and Model Orders

Additionally, effects of training are more stable for threshold *0.4* than for *0.1* and *0.6* which is mirrored by a smoother curve in Figure 6.12 (b) compared to (a) and (c). Again, we can explain this effect by considering the frequency distribution of the prediction models in Figure 6.9 and Figure 6.11. All the models exhibit transitions with a probability of around *0.1* and the SPM and the VPM for dimension *Segment* shows peaks around *0.6*. Consequently, in early phases, single queries can change the relative frequency on a way such that a transition is or is not considered by the according threshold. Transitions that have a low traversal frequency are especially effected by this, as they show a high variance in early phases of the process because each increase in the frequency counter has large effects on the transition probability. However, these transitions are contained in the results set of systems with a low threshold value, which leads to a more dynamic performance behavior in early phases for these thresholds.

Comparing the learning speed of systems with different Markov model order, it is confirmed that Models with higher order need more training compared to order-*1* models. This is a consequence of the higher number of state transitions that require more training before the relative frequencies counts adjust to the actual transition probabilities.

Figure 6.12 (d) summarizes the combined influence of training sessions and threshold parameter for models of order *1*. The conclusion of these experiments is that the prediction models adjust quickly to the user behavior, already offering reasonable performance after a *10* to *15* sessions of training.

6.3.2.4 Impact of Random User Behavior

All query simulations that were used so far, were generated solely on the basis of the simulation profile described in Section 6.2.3. However, in real world environments, not *100*% of the user queries follow the structure of the analytical workflow. Instead a certain percentage of the queries exhibits random characteristics. In order to evaluate the performance of our algorithms in the presence of such mixed workloads, we performed experiments with different random query rates (RQR). Section 6.2.3 describes the procedure of generating the mixed workloads.

For the experiments, we varied the rate of random queries between *0*% (no random queries) to 60%. Additionally, we varied the threshold between *0.1* and *0.7*. Figure 6.13 shows the results of measuring the prediction accuracy and the size of the predicted set for different combinations of these parameters. Figure 6.13 (*a*) shows the variation of the prediction hit rate under different RQR settings. As expected, it decreases with increasing RQR. However, for low random RQRs this decrease is linear. Surprisingly, the hit rate starts to slightly increase respectively the rate of decrease gets smaller. This point is reached earlier for low threshold values. We will explain the effect in the next paragraph when analyzing the size of the predicted sets. The important result of this experiment is that using prediction techniques still yields acceptable hit rates (over *60*%) for random query rates of over *50*%.

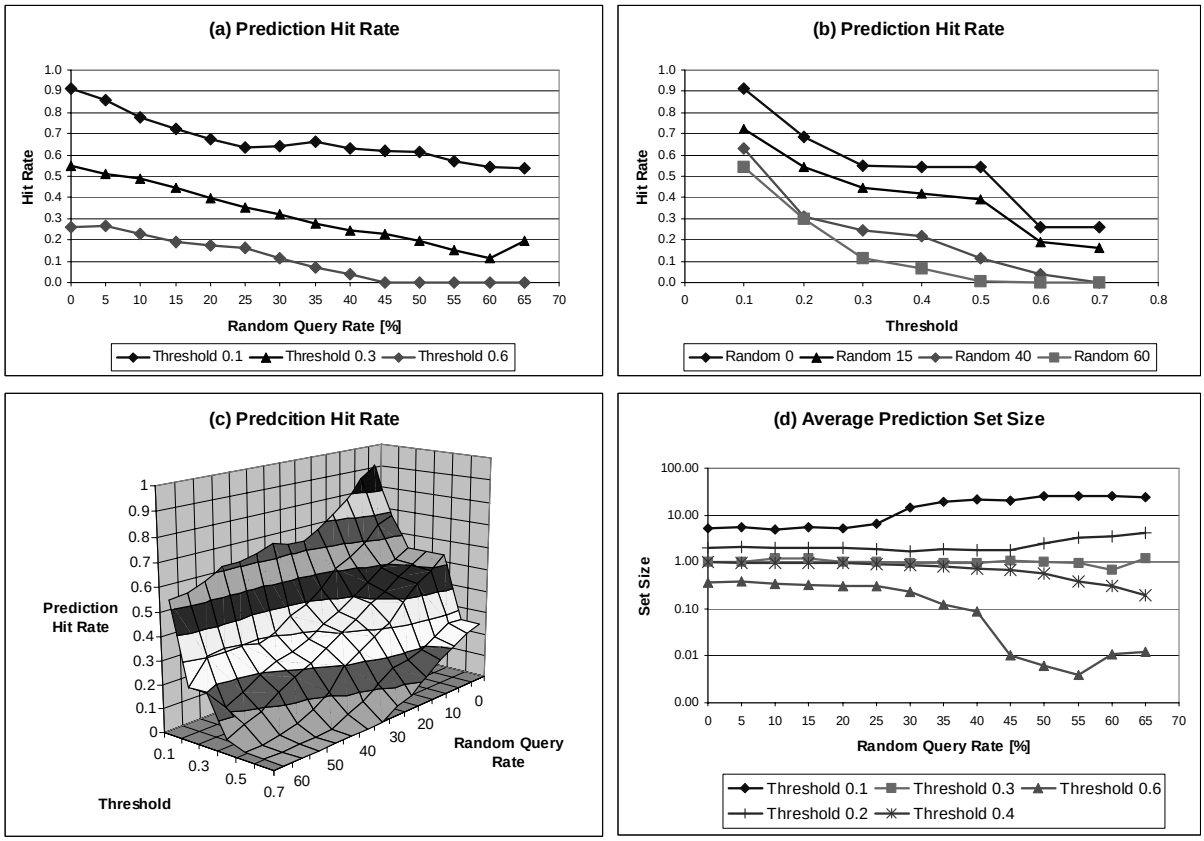

Figure 6.13: Prediction Performance for Different Rates of Random User Behavior

Figure 6.13 (*b*) shows the effect of random queries on the characteristic behavior of the prediction hit rate in depending on the threshold. For increasing random query rates, the curve gets 'smoother', as the probability distributions of the models also contain values for the random queries leading to a more uniform distribution. Figure 6.13 (*c*) summarizes the combined effects of different thresholds and different random query rates.

Figure 6.13 (*d*) shows another interesting effect of random queries. This figure plots the average size of the prediction set against a logarithmic scale. For low rates of random queries, the set size stays stable or slightly decreases, but from a certain point on, the size dramatically increases. The increase is especially large for low thresholds (therefore we used the logarithmic scale). Additionally, it can be observed that the random query rate from which on the set size increases is lower for lower threshold values. The decrease of the set size that is expected, as training the model with the random queries introduces new states and transitions into the model, which decreases the transition probabilities the rest of the transitions. This means that especially for high thresholds, the set of transitions added to the result set decreases. However if the rate of random queries increases, the newly introduced transitions reach probabilities that are above the threshold. This means that the states corresponding to random queries itself are added to the result sets, which in turn increases the set size. Lower threshold values are more prone to this effect. This effect is in turn responsible for the improvement of the prediction hit rate mentioned above, as at this point, random queries are being correctly predicted. For example, when observing the prediction hit rate for threshold *0.1* in diagram (*a*), it slight increases for a random query rate of *25*%. This is exactly the point from which on the prediction set size increases (cf. diagram (*d*)).

6.3.3 Performance of the Predictive Caching Framework

This section analyzes the performance of the integrated framework consisting of the PROMISE prediction framework and the speculative execution techniques making use of the predictions as described in Section 5.3. As an indicator for the performance of the caching system, we measure the cache hit rate and the maximum and average reduction in latency times observed by the user. For the cache hit rate, we distinguish between exact hits (i.e., the addressed query is already contained in the cache) and subsuming hits (i.e., a query is in the cache from which the addressed query can be derived).

The cache implementation for these experiments uses a total containment rewriting strategy using our subsumption definition (cf. Definition 5.2). I.e., the rewriting process checks if the user query is subsumed by a cached object. If this is the case, the query is directly answered from the cache, otherwise the original query is passed to the query executor. The according cache admission algorithm tests if a newly arriving query result fits into the cache at all (comparing its size to the maximum cache size) and if a query is already contained in the cache that subsumes the newly arriving query. In the latter case, the query result is not admitted to the cache. The eviction algorithm uses a last recently used (LRU) strategy to evict as many objects as need to make room for the inserted object. The prefetching algorithm uses the predicted set of queries as prefetching candidates and deploys the benefit estimation introduced in Definition 5.6 in order to determine the sequence of prefetching requests. The admission algorithm is called prior to executing the prefetch request (cf. Section 5.3.1). The cache is flushed between sessions. This simulates the fact that the base data is being updated between sessions or that the time period between sessions is large.

The performance of the integrated framework is dependent on the internal parameters of the prediction algorithm as discussed in the previous sections (threshold, Markov model order, Number of training sessions). Additionally, the maximum size of the cache and the consideration times of the user have an impact on the performance. In order to study the influence of these factors in an isolated fashion, we performed three series of experiments. Section 6.3.3.1 describes measurements with unlimited cache size and unlimited consideration time and therefore allows for statements about the maximum performance under idealized conditions. Section 6.3.3.2 describes experiments with limited cache size thus investigating the impact of

different cache sizes on the performance. Section 6.3.3.3 additionally introduces constraints on the consideration time. Section 6.3.3.4 investigates the effect of random user behavior. In Section 6.3.3.5, we briefly investigate the effects of combining the speculative execution strategies with conventional tuning mechanisms by measuring against a database using a multidimensional clustering index.

6.3.3.1 Unlimited Cache Size/ Unlimited Consideration Time

This section presents experiments where the cache size and the consideration time were not limited. That means, the speculative execution component prefetches all results that are produced by the prediction algorithm and all results are added to the cache (unless, a subsuming result is already in the cache, which is tested by the admission algorithm). No queries are evicted from the cache during a session. This setting can be used to assess the performance of the algorithm under optimal conditions and to determine the maximum performance increases that are possible.

Figure 6.14 shows the cache hit rate of the caching algorithm with speculative execution strategy. The models were trained with *100* sessions and the model order is *1* for structure- and value-based prediction model. The results show average figures for *100* sessions (i.e. *10.000* queries).

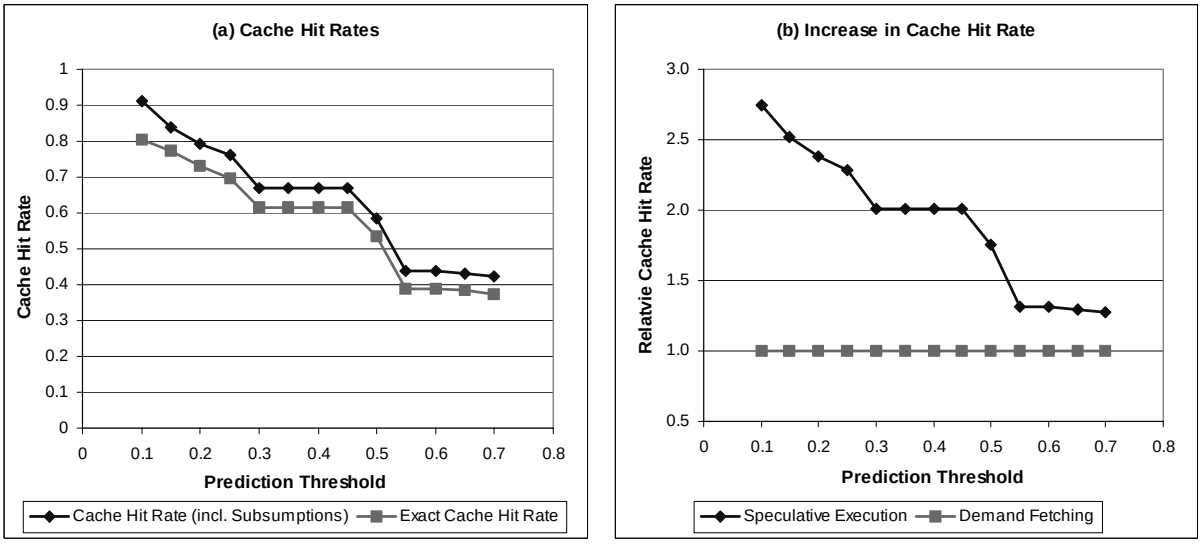

Figure 6.14: Absolute and Relative (Compared to Demand Fetching) Increases Caching Hit Rates

Diagram (a) confirms that the cache hit rate behaves similar to the prediction hit rate (cf. Figure 6.8). Comparing the exact cache hit rate and the subsuming cache hit rate shows that the caching performance is mainly influenced by exact hits due to prediction. The additional improvement of combining the speculative techniques with subsumption testing is about *10*% for all threshold values. Diagram (*b*) compares the performance of the speculative execution strategy with a traditional demand fetching strategy that is employed by current OLAP caching systems. Although this strategy also considers query subsumption, it performs badly in our environments (cache hit rate is only *33*%[59]). This is a consequence of the navigational workload that contains only very little repetitive patterns on the query level (i.e. the same query is scarcely executed twice during a session). The diagram plots the quotient of the PROMISE cache hit rate and the demand cache hit rate. Using PROMISE techniques, the cache hit rate

[59] This figure is not shown in the diagram.

can be increased by a factor of *4.5*. Even for medium threshold values, the PROMISE caching algorithms still outperforms the traditional demand fetching algorithm by a factor of *2*.

The decrease of average latency times perceived by the user is even larger, as the measurements results that are depicted in Figure 6.15 show. The diagrams compare the average latency time using the demand fetching algorithm and the PROMISE algorithm. Diagram (*a*) shows the absolute values, while (*b*) plots their quotient showing the speed-up that can be achieved by the PROMISE algorithm. The maximum speed-up factor is *6.3*, but medium thresholds still offer performance increases of factor *2.5*. Notably, the maximum speed-up for single queries is orders of magnitude higher, as answering a query from the cache instead of answering it from the database typically yields performance increases of factors in the order of *100*.

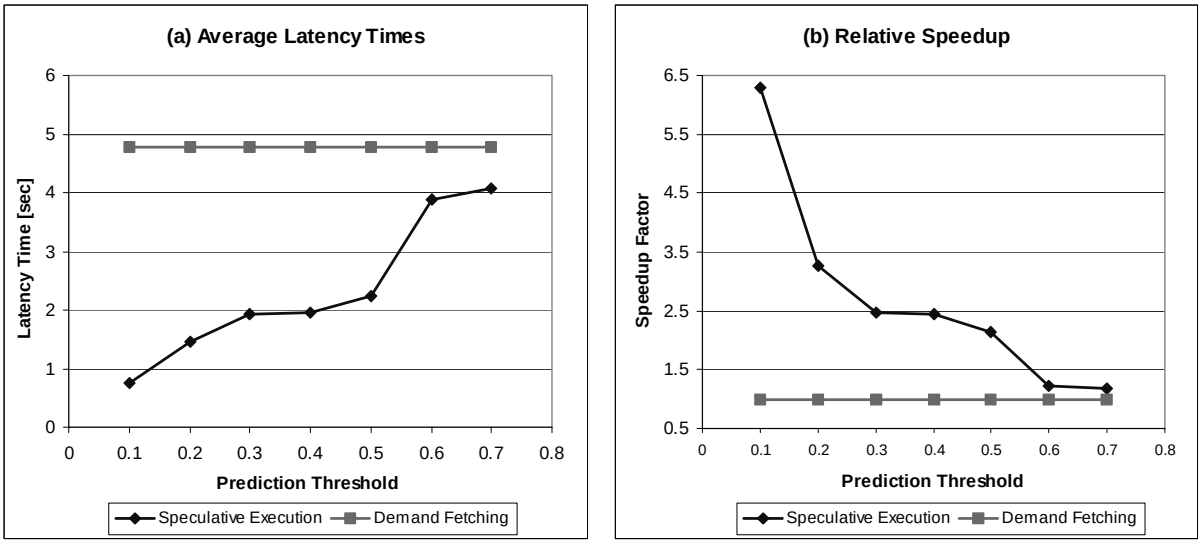

Figure 6.15: Latency Time Reductions Perceived by the User

In order to analyze, how realistic these achievements are in practice, we also measured the needed cache size and the time needed for prefetching all queries. Figure 6.16 (*a*) shows the cache size needed by the unlimited speculative algorithm compared to the demand fetching algorithm under different threshold values. For a threshold value of *0.1*, the necessary cache space is increased by a factor of *4.57*. However, with growing threshold, the cache size quickly approaches the size of the demand fetching cache. For a threshold value of *0.3*, the increase in cache size is only *1*%. This means that with 1% of additional cache space, a doubling of the cache hit rate and a decrease of the latency time by factor *2.5* is possible using speculative execution.

Figure 6.16 (*b*) shows the average time that was used to prefetch the predicted query results for one query. This gives an indication how much consideration time is needed by the system to achieve the maximum speed-up. For a threshold parameter of *0.1*, the average prefetching time is *19.08* seconds. A comparison of this figure to the average query execution time for the whole workload (*8* seconds) shows that the prefetching time falls below the execution time for threshold values over *0.2*. Additionally, it is interesting to note that the average prefetching time is of the same order as the query execution time.

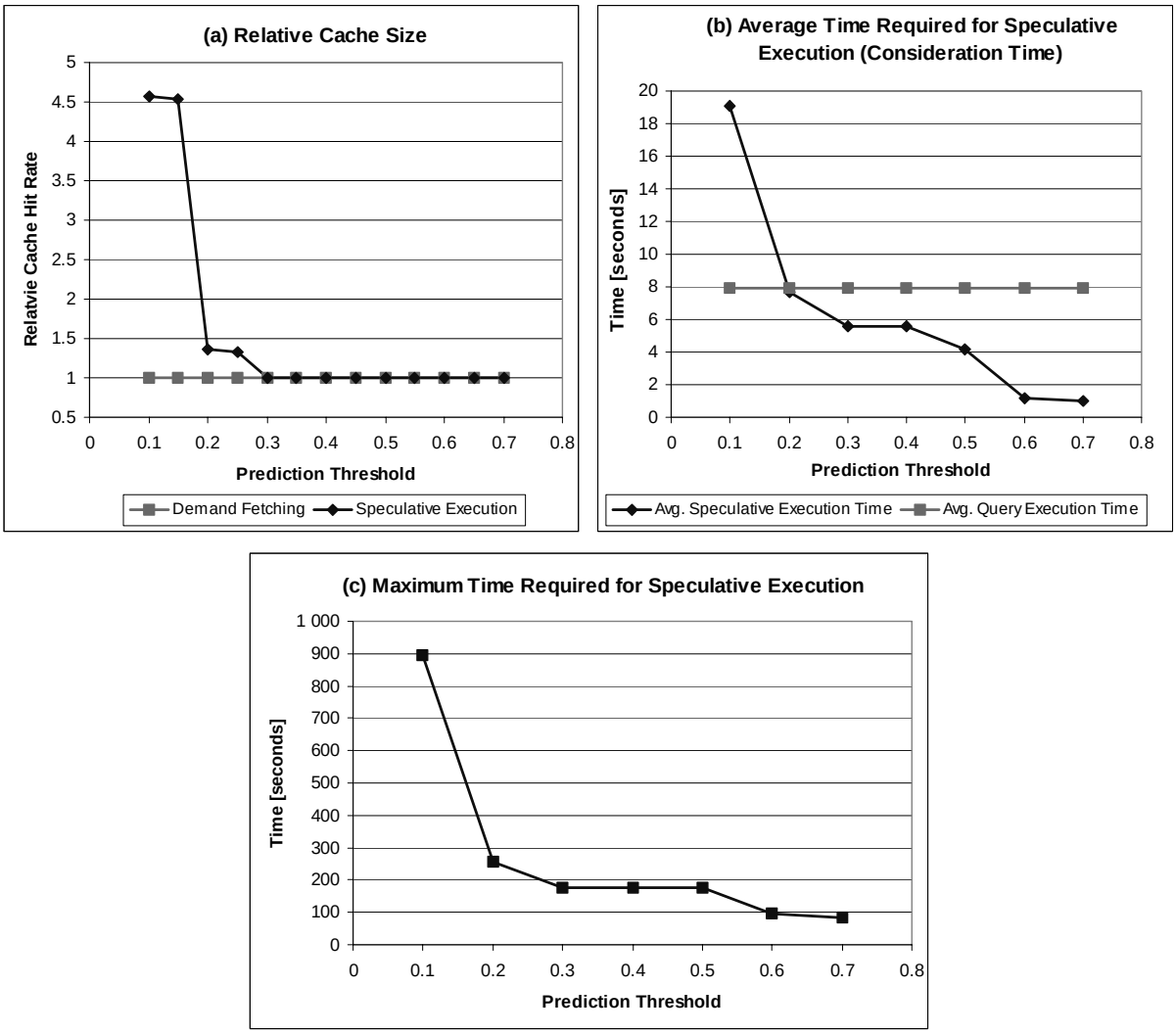

Figure 6.16: Comparison of Cache Size and Consideration Time Requirements

Figure 6.16 (*c*) shows the maximum time required for speculative execution. Analogously to the average time, this time decreases with increasing thresholds. Additionally, it is important to note that the maximum time is approximately *100* times higher than the average time. This indicates that only a small number of queries produce prediction sets that need very high prefetching times. Therefore, we can assume that the cache performance will decrease gently when introducing consideration time limits. Our measurements in Section 6.3.3.3 confirm this assumption.

While this section investigated the maximum speed-up that can be achieved by the system, the next sections will systematically investigate the behavior of the framework in the presence of different constraints. The next section starts by examining the impact of different cache sizes.

6.3.3.2 Limited Cache Size/Unlimited Consideration Time

This section investigates the performance of the integrated prediction/caching framework with constraints in the maximum available cache size. The goal is to investigates the loss in cache performance compared to the ideal case presented in the last section, if the available cache size is limited. Additionally, we examine the behavior of the PROMISE algorithm compared to the demand fetching algorithm under these conditions. To this end, we performed experi-

ments with different maximum cache sizes ranging from *100* Kb to *2* MB. Cache eviction is performed used a traditional last recently used (LRU) strategy for both the speculative and the demand fetching strategy.

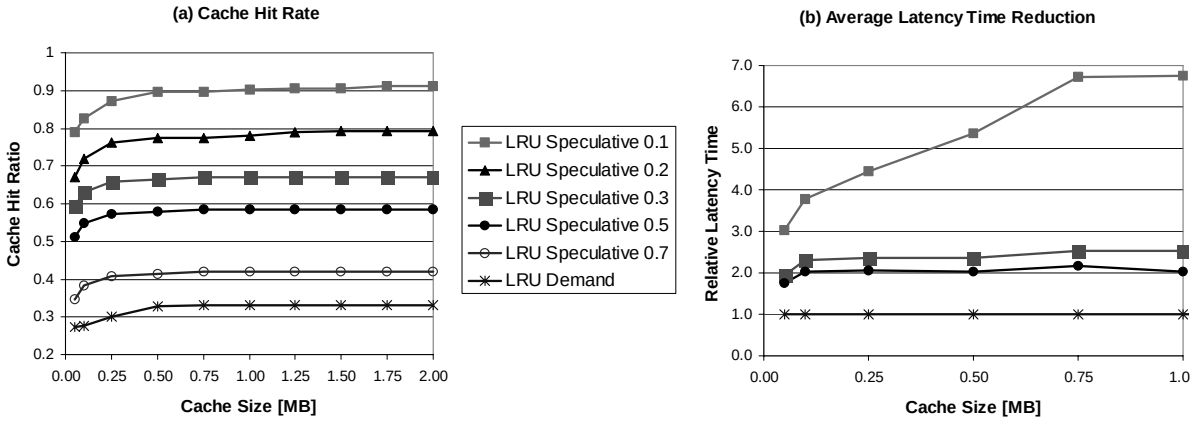

Figure 6.17: Impact of Different Maximum Cache Sizes

The results of this measurement are plotted in Figure 6.17. Diagram (*a*) shows the cache hit rate for different threshold and cache sizes. As expected, the cache hit rate decreases with decreasing cache sizes. However, the hit rate is already within *95*% of the optimal rate for cache sizes of over *250* KB and within *99*% of the optimal rate for cache sizes over *1* MB. Both speculative execution and demand cache performance degrades for very small cache sizes under *0.5* MB. However, the speculative execution algorithms still clearly outperform the demand fetching algorithm.

Figure 6.17 (*b*) plots the relative speed-up of the latency time compared to the demand fetching algorithm. This shows that the effects of decreasing cache size are more severe for the speculative algorithm, the speed-up facto is still up to *3* for cache sizes of *100* KB, quickly increasing to speed-ups of over *6* for cache sizes of over *0.5* MB. Additionally, it can be observed that small increases in the cache hit rate have considerable effects on the latency time reduction. This is because increased cache size allows for prefetching larger query results. These queries are typically more expensive to compute. Therefore, caching these queries overproportionally decreases the latency time in the case of a cache hit.

The average size of a query result for our simulated workload is *51* KB. Thus, with a maximum cache size of about *10* times the average result size, it is possible to achieve cache hit rates of nearly *90*% and latency reductions of over factor *5*.

6.3.3.3 Limited Cache Size/Limited Consideration Time

The previous section already researched the effects of reduced cache sizes showing that near optimal performance can already be achieved with caches of realistic size (in the order of *1* MB). However, these experiments still assumed that the time between two queries is long enough to prefetch all predicted queries. The purpose of this section is to drop this assumption and to investigate the behavior of the algorithm in the presence of limited consideration times. To this end, we performed experiments with consideration times of *10*, *20*, *60* and *120* seconds. The prefetching order of the candidates is determined by the prefetching benefit measure defined in Definition 5.6. After the predefined consideration time, the execution of the speculative query is aborted. However, in case that the currently executed speculative query is equal to the actual next query, the execution is continued.

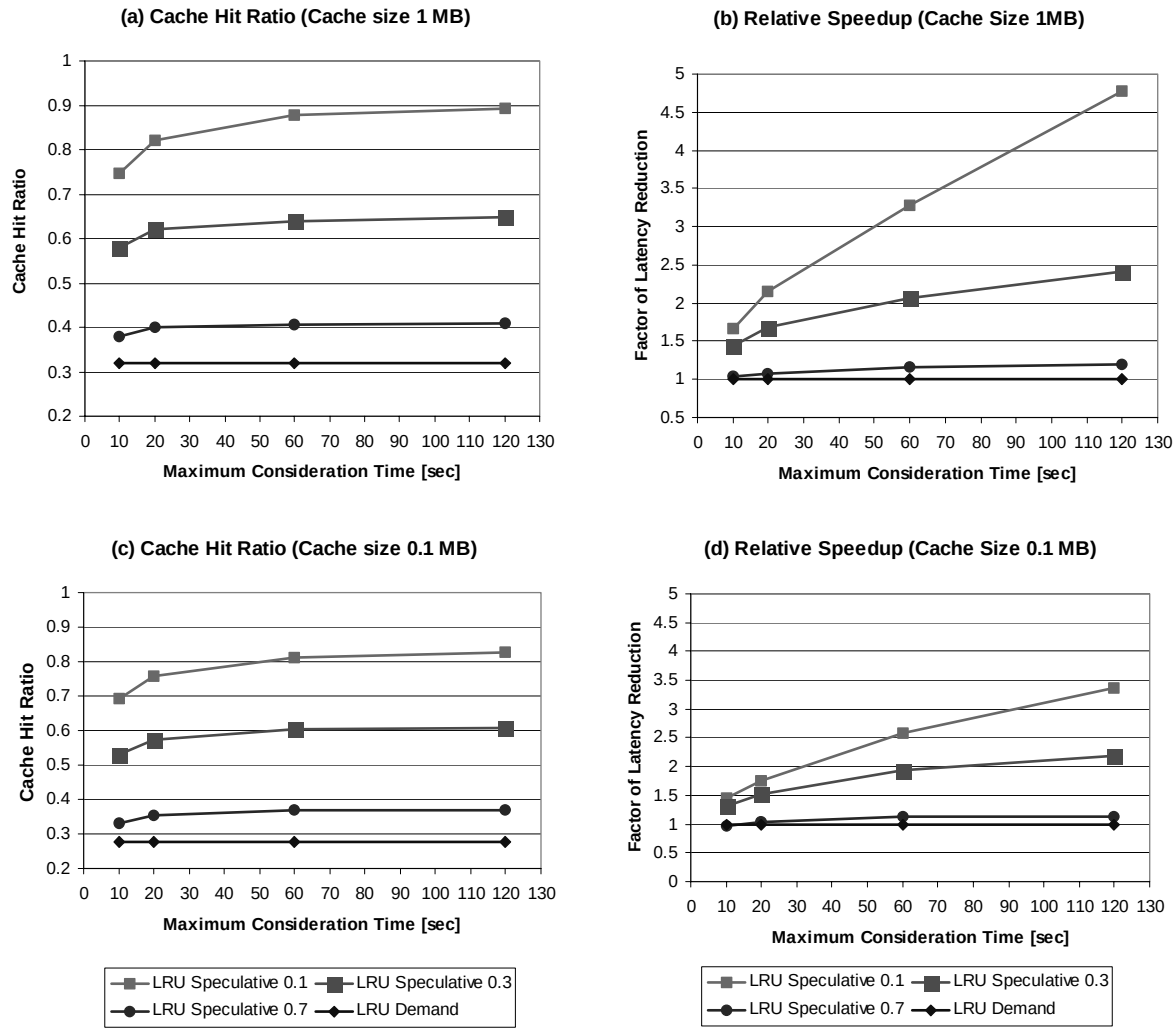

Figure 6.18: Varying Performance of the Caching Algorithm for Different Consideration Times

Figure 6.18 summarizes the results of our measurements. Diagram (*a*) and (*b*) show measurements of the cache hit rate and the relative speed-up for a cache size of *1* MB and diagram (*c*) and (*d*) show the same measurements for a cache size of *100* KB. As expected, the effectiveness of the PROMISE approach is better for longer consideration times, as more speculative results can be materialized. For a consideration time of *120* seconds, the cache hit rate is within *97*% of the optimal cache hit rate for the respective cache size (cf. Figure 6.17) for all threshold values. However, for a consideration time of *20* seconds, the cache hit rate is still over *90*% of the maximum value for all threshold values. For consideration times of *10* seconds, the values are within *80*% of the optimum and still considerably better than the demand fetching algorithm. Qualitatively, these observations also apply to the reduction factor for latency times. We perceived an latency reduction factor of over *4.5* (*1* MB cache, *120* seconds consideration time and Threshold *0.1*). But also for small consideration times (*20* seconds), latency can still be reduced by a factor of *2*.

Of course, the absolute value for the consideration times are dependent on the actual environment. However, the quotient of consideration time and average query execution time is a measurement for the average number of queries that can be prefetched during a consideration phase. The average query execution time for our workload and environment is *8* seconds. This means that with a consideration time in the same order, the latency time can be reduced by a

factor of *1.5* with a cache hit rate of *80*%. With a consideration time that is *7* times as large as the average query execution time, speed-ups of a factor of *3* are possible (using a cache size of *1* MB).

6.3.3.4 Impact of Random User Behavior

The goal of this section is to experimentally investigate the behavior of the caching algorithms for different rates of random user behavior. We have already analyzed the impact of the existence of a random component in the user's behavior on the prediction hit rate (cf. Section 6.3.2.4). It can be expected that the cache hit rate behaves analogously to the prediction hit rate. However, the prediction hit rate only considers hits, where the next query is equal to the predicted query. In contrast to this, a cache hit is achieved if the predicted query is actually executed by the user while the results of the speculative prefetching are still in the cache (the exact amount of time is dependent on the cache size and the eviction strategy). Thus, we expect that the cache hit rate will not decrease as fast as the prediction hit rate in the presence of random user behavior.

Figure 6.19 (*a*) shows the results of the experiment conforming this expectation. It plots the cache hit rate for a cache size of *1* MB. For a random query rate of *0*%, the prediction hit rate and the cache hit rate are equivalent (cf. Figure 6.13). However, e.g. for a random query rate of *20*%, the cache hit rate is *74*% (threshold *0.1*) while the prediction hit rate is *67*%. The increase in prediction hit rate due to the increased prediction set size starting from *30*% (threshold 0.1) can also be observed in both diagram (*a*) and (*b*). Figure 6.19 (*b*) shows the same measurements for a cache size of *0.1* MB. This shows that extremely small caches are more prune to random user behavior, as the random queries and their predictions evict prefetched result sets before they can are actually referenced by the user. In other words, the effective lookback window size for which the cache can keep the results is decreased.

The speculative strategy still perform significantly better than the demand fetching strategy (especially for lower threshold values). The cache hit rate achieved by the different strategies converge for high random query rates.

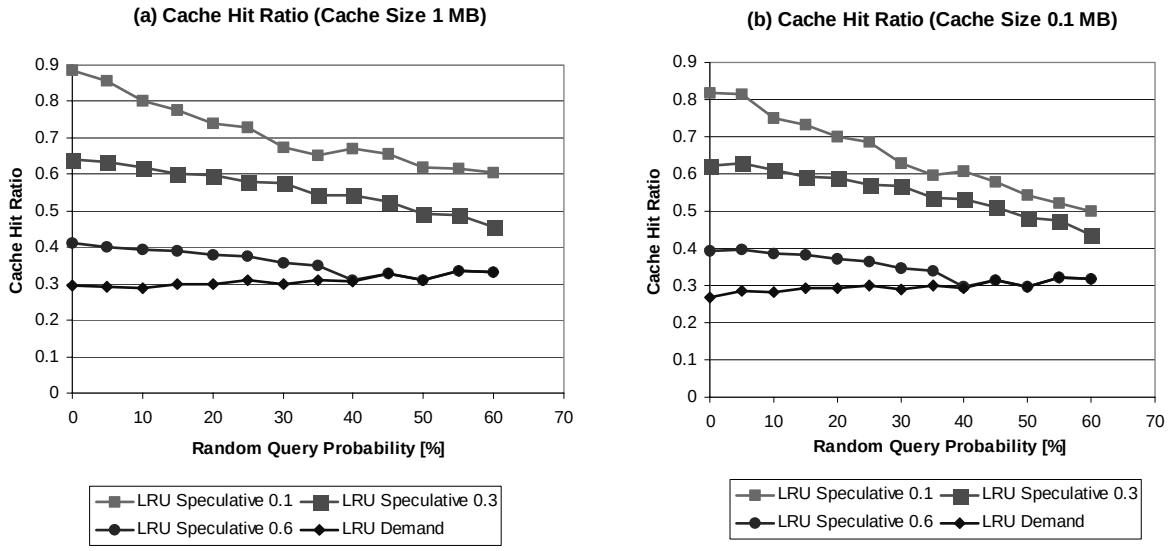

Figure 6.19: Cache Hit Rates in the Presence of Random User Behavior

6.3.3.5 Combining Speculative Execution and Multidimensional Indexing

Different approaches have been proposed to speed up query processing in OLAP systems. Multidimensional indexing has been prove to effectively reduce response times in OLAP environments ([MRB99]). In order to evaluate the effects of combining the PROMISE approach with these techniques, we modified our evaluation environment using the UB-tree, a multidimensional clustering index, to organize the base data. To this end, we replaced the SQL Server database management system by the Transbase Hypercube system that implements the UB tree in its kernel. This results in a decrease of the average query execution time for our workload from *8* seconds to *1.4* seconds.

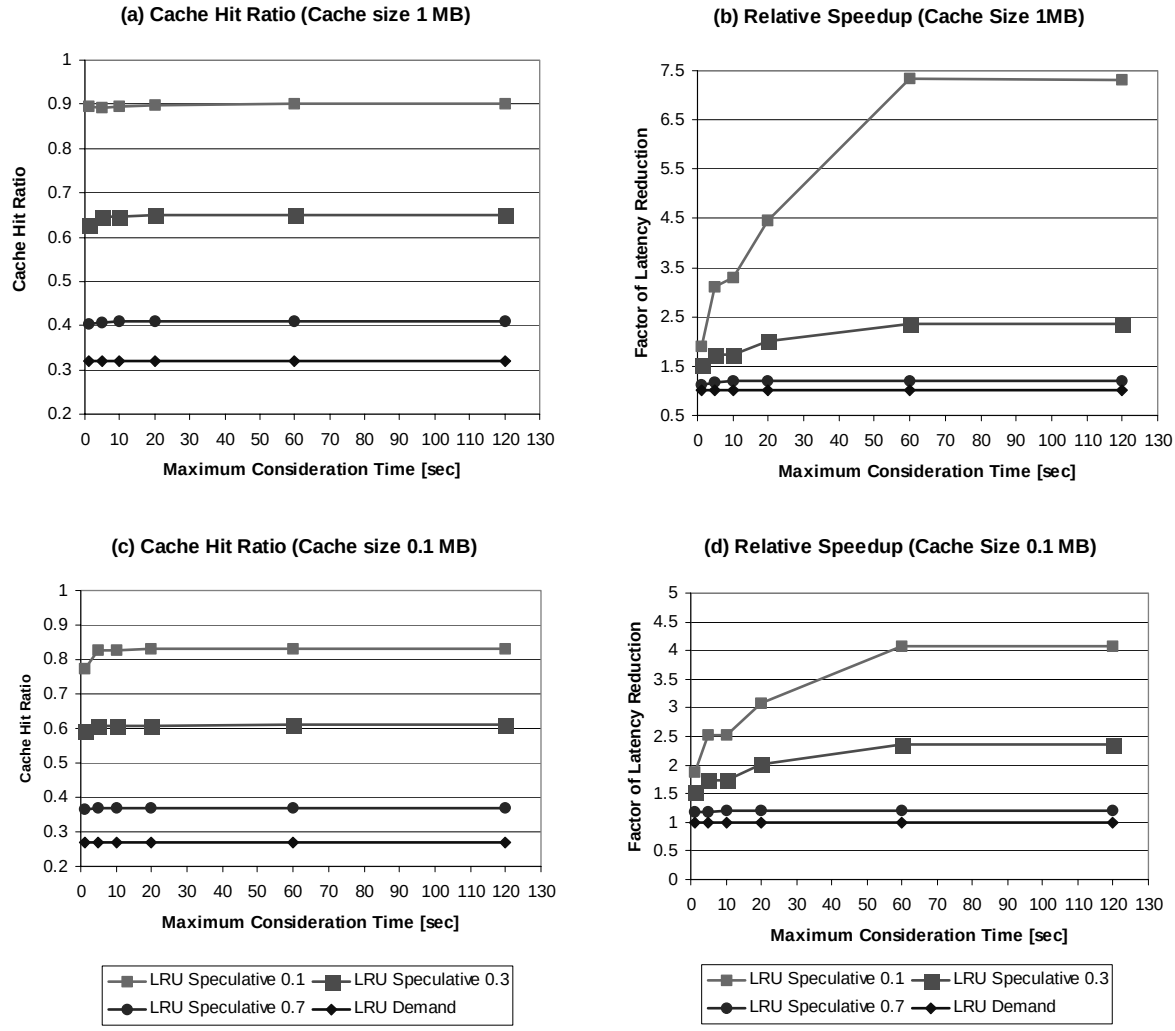

Figure 6.20: Impact of Better Query Performance through a Multidimensional Clustering Index

This reduced query execution time allows for prefetching more queries during the same consideration time. This can be observed in Figure 6.20, which shows the same measurements that were presented in Section 6.3.3.3 (limited cache size/ limited consideration time). By comparing this figure to Figure 6.18, it is obvious that the increase in latency time reduction is much faster in this environment. This means the PROMISE algorithm already delivers much better performance for the same consideration times. In this environment, speed-ups of over factor *7* compared to the demand fetching algorithm. Notably this speed-up is achieved additionally to the speed-up of approximately factor *5* that stems from using the multidimensional clustering indexing technique. Thus, speculative execution techniques and other performance

tuning tools that decrease the execution time of queries are not in conflict but can be applied in a combined way and additionally offer symbiotic effects.

6.4 Summary and Conclusions

This section described our empirical studies concerning OLAP system user behavior and the performance of the PROMISE framework and its application to OLAP caches. We started by analyzing the workload characteristics of a real-world data warehouse system (Section 6.1). The most important result confirmed the basic assumptions of our approach:

- User behavior in OLAP systems shows navigational characteristics. A considerably large part of the sessions contain over *5*0 queries.
- The consideration time between two queries is long enough for speculative materialization techniques to be beneficial. Over *80*% of the examined queries satisfied the pure prefetching assumption.

Additionally, we evaluated the performance of the PROMISE framework under varying conditions. To this end, we used an experimental framework using data from a real-world warehouse. The behavior of the user is generated by a simulation component that was designed according to the findings of our user behavior analysis (Section 6.2.3) and allows for generating workload with different characteristics. The most important observations were:

- The absolute runtime performance of the algorithms for prediction and training are very low (under *1 ms*) thus allowing for online training and prediction.
- The runtimes of the prediction algorithm are independent of the size of the Markov model and its order. The influencing factor for the runtime of the prediction algorithm is the threshold parameter that determines the size of the result set. This confirms the good scalability of the approach.
- We observed a prediction accuracy of over *9*0% for low threshold values. For mid-range threshold values, the accuracy is still in the range of *60*%. This high accuracy allows prediction applications (like speculative materialization techniques) to improve the performance of OLAP systems.
- The cache hit rate of an OLAP cache can be increased by a factor of up to *3.75* by using speculative execution techniques based on the PROMISE predictions. The latency time perceived by the user can be reduced by a factor of up to *6.5*.
- The performance is already nearly optimal for low cache sizes. In our experiments, we achieved results with *99*% of the optimum for a maximum cache size of *1* MB (which corresponds the storage requirements of *20* average size queries in our environment).
- A critical factor for the performance of the speculative execution strategies is the length of the user's consideration time. Our measurements achieved average speed-up factors of over *2* (*1* MB cache size, threshold *0.1*) even for small consideration times of *20* seconds (which is about *2.5* times the average query execution time in our environment) . Nearly optimal average speed-ups are already achieved for consideration times starting with *60* seconds, which seems realistic in an analytical environment.
- The PROMISE techniques can also be applied to workloads that contain a certain amount of random query behavior. For example, for workloads containing *25*% random queries, a cache hit rate of *73*% percent could be achieved, which corresponds to an increase of factor *2.4* compared to state-of-the art demand fetching techniques.

- A combination of tuning methods decreasing the average query execution time (e.g. multidimensional indexing) and the PROMISE framework leads to a higher effectiveness of the approach for smaller consideration times.

Thus, our measurements with the prototype implementation strongly indicate the practical usability and effectiveness of our algorithms in achieving significant performance gains for different typical classes of OLAP workloads.

When I am working on a problem
I never think about beauty.
I only think about how to solve the problem.
But when I have finished,
if the solution is not beautiful, I know it is wrong.
-- Buckminster Fuller

Do not go where the path may lead,
go instead where there is no path and leave a trail.
-- Ralph Waldo Emerson

Chapter 7 Discussion and Conclusions

This chapter concludes the thesis by critically assessing and discussing the PROMISE approach. To this end in Section 7.1, we first compile the contributions of the approach and compare them to the objectives defined in Section 1.2. A discussion of similarities, differences, and interrelationships of our approach compared to work done in related fields (Section 7.2) constitutes a conclusion to the observations regarding state-of-the-art practices presented in the different chapters of this thesis. Section 7.3 will discuss extensions of the PROMISE approach and thus point to interesting areas of future research that can benefit from the fundamental work done in this thesis. Section 7.4 contains some speculations about the impact of our research work on OLAP product development.

7.1 Contributions

This section summarizes the most significant contributions of this thesis and compares them to the objectives defended in Section 1.2.

- *General Prediction Framework*

  We modeled and formally described the structure of a framework that uses patterns in user behavior to dynamically predict future user behavior (cf. Chapter 2). The framework consists of different communicating processes. This includes a formal definition of the data structures (namely the general user interaction and user interaction pattern model) used for communication and the functional description of the processes and the issues involved in the design of these processes. This framework description is not only the basis for systematically applying the framework to the OLAP application area, but can be used to compare our approach with other approaches thus satisfying our general objective of having a general approach and achieving comparability to related work.

- *Comprehensive Model for Patterns OLAP User Behavior*

  We provided a comprehensive formal description of the multidimensional model which constitutes a best-of-breed combination of previously proposed data model descriptions. This description of the static part of an OLAP system is augmented by a description of dy-

namic OLAP user behavior. Compared to other existing approaches, our user behavior model incorporates the special role of the graphical OLAP user interface. This is mirrored in a declarative canonical query form that describes an isolated interaction with the system. The navigational aspect of iteratively formulating queries is mirrored by the definition of a set of query transformations that iteratively transform queries thus resulting in navigational sequences. These query transformations are explicitly distinct from the operations used to algebraically formulate a query. This is a distinctive feature of our approach remedying shortcomings of existing approaches in this area.

However, the description of user behavior is not presented for its own sake, but builds the toolkit for describing navigational patterns in OLAP user behavior using sequential probabilistic rules defined on OLAP sessions. We solved the problem of a prohibitively high number of rules due to the large number of possible canonical OLAP queries by applying generalization techniques. To this end, we introduced two orthogonal generalizations for canonical OLAP queries (structure and value-based) that mirror the peculiarities of typical OLAP analysis processes. Each set of generalized patterns can be efficiently represented as a Markov model. Considering both, the peculiarities of the OLAP analysis process including the graphical iterative query formulation and the prerequisites for an efficient induction and prediction process, this pattern model is one of the central contributions of this thesis.

- *Efficient Prediction Algorithm for OLAP Queries*

  Based on our pattern model, we presented an algorithm that can predict the next interactions from the current session context. We also proposed a data structure representing Markov models in a way such that the execution of the prediction algorithm is efficient. A theoretical performance analysis shows that the execution time of the algorithm is linear in the size of the predicted result set which can be influenced by a threshold parameter passed to the algorithm. Practical measurements have not only confirmed these theoretical results but have also shown that the absolute execution time of the algorithm is short enough (around 1 *ms*) to execute it without even marginally affecting the performance of the overall system. The practical evaluation has also confirmed that the prediction accuracy of the algorithm is high enough (up to 80% in our experiments) such that significant performance improvements are possible using the prediction results.

- *Efficient Algorithm for the Acquisition of Knowledge about User Behavior*

  The presented PROMISE framework incorporates an induction process that enables the system to dynamically acquire (learn) patterns from the actual user behavior being perceived by the system. We presented a frequency counting technique for updating Markov models from a sequence of user interactions. A theoretical performance analysis and practical measurements has shown, that the training of the model can be done online during the classical query processing using the last interaction performed by the user. The presented induction algorithm additionally considers k ($k \geq 1$) age classes to reflect changes of user behavior in the model.

- *Methods to Improve Dynamic OLAP Materialization Strategies Techniques by Prediction*

  One of the central objectives of this work was the demonstration that prediction results can be used to improve the performance of an OLAP system. We chose to concentrate on the impact of predictions on the performance of dynamic materialization strategies (OLAP system caching). To this end, we proposed two extensions to existing OLAP query-level caching systems:

- improvement of the benefit estimations needed for admission and eviction decisions by considering the benefit for the predicted workload.
- improvement of the cache hit rate by using speculative materialization's based on the set of predicted queries.

The specification and usage of an abstract caching framework allows for combining both extensions with all existing OLAP caching schemes and analytical models for cost estimation. A prototype implementation has shown, that both techniques can significantly improve the cache hit rate of OLAP systems. Our measurements have shown cache hit rates of over *90%*, which is *375%* of the cache hit rate of a traditional LRU demand fetching cache. The average latency time of the user could be reduced up to a factor *6.5*. However, the systematical description of integrating prefetching techniques into semantic query-level caching systems is not restricted to OLAP caches. Therefore, formally described problems and the proposed solutions apply to query-level caches in general. Therefore, the discussion of predictive materialization strategies is one of the central contributions of the thesis.

7.2 Interrelationship with Related Work

To the best of our knowledge, the approach of modeling query sequences and predicting query behavior is unique in the area of OLAP and Data Warehousing. Therefore, we cannot directly compare our work to other approaches following the same methodology. Instead, this section will discuss the similarities and differences with selected approaches that also aim at reducing the latency time in interactive information systems and especially OLAP systems. This section is a conclusion of the more detailed survey of state-of-the-art techniques presented at the beginning of Chapters 3 to 5.

7.2.1 Predictive Prefetching for the WWW and Other Domains

In Section 4.1, we compared different approaches predicting navigational user behavior in different application domains (for example WWW based systems) with the goal of decreasing the overall system's latency time by prefetching, presending and speculative execution. The methodologies applied by these approaches were a model for the development of the PROMISE and PROMISE/OLAP framework. Therefore, our approach is similar in using frequency techniques for pattern induction. However, we have extended this approach by aging techniques in order to take into account changing user behavior. Another similarity is the usage of Markov models as the base formalism for our pattern model. The main differences between the PROMISE approach and the discussed predictive prefetching techniques are due to the following special requirements of OLAP user behavior:

- As calculated in Section 3.2.3, the number of different canonical OLAP queries against a simple example cube is in the magnitude of 10^{15}, which is significantly higher than the number of prediction entities in other areas (for example the number of pages for a web server or the number of pages in a database). This means that the Markov models – in contrast to other application areas – cannot be described on the finest granularity of data access (i.e., a query, see Section 4.2.2.2). Additionally, the typical analytical workflow in OLAP system produces patterns on a generalized level rather than on the query level. This is an additional argument not to build pattern models on the most detailed level. Consequently, we proposed using different generalizations for OLAP queries (i.e., a structural and a value-based) and use a set of Markov models each representing generalized patterns

according to the different generalization procedures. The result of the overall prediction is assembled from the results of the generalized predictions.

- Adjusting OLAP caches to incorporate predictions is significantly more complicated than in other application domains (for example a file buffer). This is a consequence of the derivability relation between the cached objects (query results can be computed from other results). This requires that the cache manager employs a complex rewriting process. The consequence for the predictive materialization strategy is that the interrelationships of prefetching candidates with already cached objects have to be considered. Our approach addresses this by giving benefit estimation functions incorporating the effects of rewriting queries using derivability (cf. 5.3.2.2). Additionally, the derivability property allows for substantially more flexible prefetching strategies because the algorithm is not restricted to only prefetching predicted results. It can instead choose from the large set of queries from which the predicted query can be (partially) derived. We reflect this in our framework by including an explicit candidate generation step and giving heuristic solution strategies (cf. Section 5.3.2.3).

7.2.2 OLAP Caching Approaches

The solutions described in Chapter 5 are extensions of current OLAP caching algorithms. The following two sections summarize the similarities and differences of our approach compared to the two most prominent OLAP caching approaches: the WATCHMAN Project (Section 7.2.2.2) and the Cube Star Project (Section 7.2.2.1).

7.2.2.1 The CubeStar Project

In our opinion, the caching algorithms developed in the Cube Star Project (e.g., [Leh98b], [ABD+99], [LAH00]) at the University of Erlangen-Nuremburg are currently the most advanced techniques regarding OLAP caching. They deploy a sophisticated rewriting strategy that answers the query from a set of cached objects and queries to the raw data. The caching granularity are multidimensional objects (so called MOs) that are produced by queries to the database system. Consequently, a query can be described by an MO respectively a tree with MOs at its leafs for more complex queries.

The definition of our data model and terminology has benefited from numerous discussions with members of the Cube Star group. Thus not surprisingly, our basic query formalism (a canonical query) is of comparable expressiveness as the MOs. However, the Cube Star projects put great emphasis on the support of so called features (an extension of the multidimensional model) which we do not consider in our model (cf. Section 3.1.1.4 for a more detailed discussion).

Another similarity shared by the PROMISE and the CubeStar approach is the idea of using the iterative nature of query formulation (i.e., the transformation operations) for improving the cache eviction decisions (cf. Section 5.2.2). CubeStar also measures the similarity of queries by the number of transformations (respectively operations in the CubeStar terminology) that are needed to transform one to the other. However, our definition of transformations differs in that it takes into account operations that are supported by current graphical front-ends. The measurements proving the feasibility of the Cube Star approach also rely on the simulation of interactive user behavior ([ABD+99]). However, the CubeStar simulation is based on a single Markov model where each transition corresponds to a multidimensional query transformation (e.g., slice, drill-down). The corresponding parameters are randomly generated. This means that the CubeStar model mirrors patterns like e.g., that after a drill-down, a slice operation is carried out with probability of 80%. If we apply our reference framework to this approach,

this can be regarded as a generalization for queries building equivalency classes by omitting the parameters to the operations. However, the resulting model in not suited for prediction (it is not intended for this purpose) and does not take any domain specific user behavior into account nor does it consider the induction of patterns.

Naturally, the main difference between the PROMISE and the CubeStar approach is that the CubeStar cache deploys a strict demand fetching strategy not taking into account any speculative materialization techniques. Additionally, as no prediction algorithm is integrated into the caching engine, the benefit estimation has to rely on the reference density, the recomputation cost of an object and the distance to the last query of the workload. Therefore, the CubeStar approach could benefit from an integration of the PROMISE concepts.

7.2.2.2 The WATCHMAN Project

The WATCHMAN approach (cf. [SSV96],[SSV99], Section 5.1) caches query results and uses a splitting algorithm combined with a total containment strategy for query rewriting. The most interesting features of the approach regarding PROMISE are the consequent usage of cost/benefit metrics for cache admission and eviction decisions that also take into account storage costs and recomputation costs. This approach has influenced our design of the PROMISE cost/benefit metrics which can be seen in a similarity of the benefit definitions. However, our benefit definition has been significantly extended to take the predictions for the current session context into account.

In contrast to the driving ideas of PROMISE, the WATCHMAN approach does not consider typical properties of OLAP query formulation (graphical tools) and analysis process (navigational patterns). The benefit measures are solely based on a relative frequency count of past access weighted by the time passed since the last access. This leads to difficulties in estimating the benefit of newly arriving objects as stated in ([SSV96]). Additionally, the employed strategy is exclusively demand fetching.

7.2.3 Relations to Physical Tuning Approaches

Being targeted at decreasing the latency time perceived by the user, the PROMISE approach indirectly competes with physical tuning techniques that aim at decreasing query execution times which also leads to shorter latency times. Therefore, this section discusses bilateral influences when combining these techniques with the PROMISE approach.

7.2.3.1 Static Materialization Techniques

Incorporating static materialization techniques with the PROMISE/OLAP caching approach does not involve any changes to the basic approach. This is because the rewriting process of a system with statically materialized views already considers these views when rewriting an incoming query. Additionally, the execution cost model must consider the shorter evaluation time of a base data query against a materialized view compared to execution against base data. Our cost model takes this effect into account as the amount of data cells to be read from the preaggregated view is smaller, thus the estimated execution time will also be smaller. Such a combination naturally has an effect on effectiveness of the PROMISE approach: Average execution costs of queries that cannot be satisfied from the cache are decreased. On the one hand, this means that the potential benefit in latency time reduction by using cached results is decreased. However, this effect applies to dynamic materialization techniques in general. On the other hand, the speculative execution benefits from a reduced average execution times of speculative queries. This in turn will increase the cache hit rate as a larger number of specula-

tive execution request can be completed during the consideration time thus increasing the probability for a cache hit.

The other way around, algorithms for determining the set of statically materialized views can benefit from the pattern information gathered by the PROMISE framework, as the Markov models can be used to calculate a workload characteristic that is needed as an input to the view selection algorithms.

7.2.3.2 Special Indexing techniques

Multidimensional data clustering in combination with multidimensional access techniques (for example [MRB99]), bitmap indexes (for example [WB98]) and foreign column indexes (for example [Inf96]) speed up the query processing by reducing the costs of the join or increasing the efficiency of the tuple selection. Again, this speeds up the cost of executing queries on raw data, thus decreasing the potential for reductions in latency time through prefetching. However not only the user's actual queries profit from these techniques, but also the speculatively executed queries. This means that in an environment, where consideration times are quite small compared to the executions times of the queries, it is possible to increase the cache hit rates and to decrease the latency times by combining indexing and speculative execution. We have practically investigates this effect in Section 6.3.3.5. For example, for consideration times of *20* seconds, the speed up achieved by the PROMISE approach is doubled additionally to the speed-up that is achieved by applying the multidimensional indexing technique.

Thus, we conclude that the PROMISE approach is not in conflict with these tuning approaches. In contrary, symbiotically effects can be achieved. Therefore, predictive prefetching techniques should become an common instrument in the administrator's tuning-toolbox for advanced OLAP system.

7.3 Extending the PROMISE/OLAP approach

Being the first step into a new research area, the range of topics covered in this thesis cannot be exhaustive and leaves plenty of room for further research and improvements. This section compiles some directions for extensions of our approach that we believe to be interesting and challenging but which could not be addressed in the scope of this work. Wherever possible, we will outline ideas on how the presented issues could be approached.

7.3.1 Impact on the OLAP Application Design Process

In the motivation in Section 1.1, we showed that the characteristics of user behavior are not only interesting for OLAP system design as demonstrated in Chapter 5 and Chapter 6 but also deserve more attention when designing OLAP applications. In fact, the starting point for the PROMISE research work came from experiences in applying and developing a methodology for OLAP application design.

Currently, OLAP application and data warehouse design is mainly concerned with designing a static multidimensional schema representing the application domain (data warehouse design additionally involves designing mapping and transformations from the operational base data sources to this schema). However, in early phases of the development process, when this schema is still unclear, it is beneficial to model the workflow of the analyst (together with a domain expert). This approach has several advantages:

- As end users tend to understand the dynamic workflow better than the static data model, a model adequately representing the universe of discourse can be found faster respectively the model's quality can be improved.
- A first version of the static data model (the logical OLAP database schema) can be derived from a description of the workflow, because these workflows contain the granularities that the user is referring to and the necessary measures.
- Design decisions during the physical design can benefit from information about the dynamic user behavior. E.g., denormalization or vertical fragmentation decisions require information about which data is often accessed together.

This means that an integration of the PROMISE pattern model into an OLAP system design methodology seems promising. We already presented graphical representations mechanisms for the PROMISE patterns which can be used as part of a graphical modeling notation. Especially the structure based prediction model can be used to interactively specify the analytical workflow of the user for a scenario (see [Sap99]).

The following section sketches how the PROMISE approach can be integrated into a design methodology for Data Warehouse and OLAP Design. The core of the BabelFish approach ([BSH00], [BSH99], [Bla00], [SBH+99]), to which the author significantly contributed, is a comprehensive object-oriented model of a data warehouse containing all the details that are necessary to specify a data ware-house (e.g., the data cube names, a description of their dimensions, the classification hierarchies, a description of the data sources, the tool-specific database schema). We refer to the object oriented schema of the warehouse model as warehouse metamodel (for example containing a class dimension). Such a metamodel is far too complex to use it for modeling purposes or graphical representations. Therefore, we follow the view based approach which has already been successfully deployed for Object Oriented Software Engineering tools. I.e., we defined certain subsets (views) of the warehouse model as part of the BabelFish method design. The designer indirectly manipulates the warehouse model through graphical visualizations of these views. Each view represents a certain aspect of the warehouse design, e.g.

- *static data model view* describing the conceptual schema of the warehouse (i.e., a multidimensional model of the data that originate from the business processes). It is visualized using the ME/R notation (cf. [SBH+99]).
- *functional view* specifying the functional interrelationship between data (e.g., transformations from source to target or how to compute complex derived measures).
- *data source view* describing the static structure of the data sources and their interrelationship to the static warehouse modeli.e, a specification of the data transformation and loading process.

A graphical view definition consists of two components: a query defined with respect to the metamodel selecting a part of the warehouse model (for example selecting all the conceptual measures, dimensions, cubes, and their relationships) and the specification of how to actually display the selected parts (for example stating that a dimension level should be represented as a node in the view graph and depicted as a rectangle which is labeled with the name of the level, cf. [HSB00]).

The model can be easily extended by adding classes and relationships to the metamodel and defining according views. In order to allow for modeling dynamic user behavior, a new view (called *dynamic view*) would have to be introduced. The graphical representation is equivalent to the graphical representation of the structural prediction model.

The metamodel concept allows for expressing complex integrity constraints and relationships between objects taking part in different views. Furthermore, a single object of the warehouse model can participate in different views (possibly as node, edge or label). Consequently, interrelationships between the different aspects of warehouse design can be formulated in an elegant way. E.g., objects of the class *classification level* are part of the static data model view and the new dynamic data model view. Whenever a modeler creates a new dimension level (for example by adding a node to the ME/R representation of the static data model view), this dimension level is automatically available in the dynamic system view for the description of user query behavior. This can also be used to enforce inter- and intra-view consistency. E.g., if a new dimension level is being added in the dynamic system view as a part of a report, it is automatically also available in the static system view. An integrity constraint defined for the static view states that every dimension level must be connected to a fact. Thus, when checking the consistency of the static view, the system can automatically remind the modeler to update the static view.

Of course a full integration into the methodology would also have an impact on the process model of the development process introducing phases where user behavior is modeled.

7.3.2 Extensions to the Predictive Caching Techniques

An issue that has not been addressed in this thesis is the analysis of the prefetching performance of the approach in multi-user environments. For our measurements, we have assumed that the user can use all the resources of the system at a time. This assumption is not unrealistic for data warehouse architectures, where e.g., a small number of analysts access a topic-oriented data mart. However, when looking at information systems that are used by a large number of users (for example accessing the system via Internet), it is interesting to research extensions to the current framework.

One of the effects of concurrent data accesses is that the idle times of the system that are used by speculative execution techniques are decreased. This will consequently decrease the average number of prefetching candidates that can be processed during the user's consideration time which will lead to a decreasing latency reduction for the user. On the other hand, sharing the cache between different users can potentially increase the benefit of cached objects, as these objects can be used in answering several queries from different users.

Therefore, it is scientifically interesting to systematically study of the effects of concurrent data access on the speculative execution strategies, both theoretically and empirically. A possible consequence might be that the benefit estimation function would have to be extended in order to take concurrent sessions on a shared cache into account.

7.3.3 Improving the Prediction and Induction Process

Currently, the prediction and induction process relies on a set of assumptions concerning the pattern model i.e., especially on the assumption that patterns are represented as Discrete Time Markov Models, that all patterns are sequential patterns (relating a session context to the next query) and on the properties of the structural and the value-based generalization function as described in Chapter 4. For the PROMISE approach it was necessary to fix these design decisions before designing the prediction process. However, interesting research directions arise from dropping some of these assumptions and discussing the impact to the overall framework. The following section exemplarily discuss some of these topics.

7.3.3.1 Using Variations of Markov Models

Several variations and extensions of Markov models have been proposed. We chose to represent our patterns using Discrete Time Markov Models (DTMMs). However, for future work it is interesting to incorporate more sophisticated variations of Markov models. To give an impression, which extensions are likely to be scientifically challenging, we will shortly discuss the impact of *Continuous Time Markov Models* ([Nel95],[Tij94]) and *Hidden Markov Models* (for example [Rab89]) on the PROMISE approach.

So far, we have assumed a discrete time model, that means that the system changes its state at predefined times. As long as we use a logical timescale where the logical units of time are defined by the user interactions, this constitutes no restriction. Nevertheless, for the prefetching process, it might be interesting to make predictions about the length of the consideration time. This can be achieved by a *Continuous Time Markov Model* (CTMM). This model assumes that the time value that the system stays in one state follows the same statistical distribution (usually an exponential distribution) for all states. Nevertheless, the parameters for the distribution can be varied for each state transition. In the domain of navigational information systems, this models the fact that the duration of the cognitive process of the user is typically dependent on the results of the last interaction. For the Markov model this means that for each state of the model, the individual parameters of the distribution have to be stored. The concept of CTMMs has already been successfully applied in the field of large Multimedia Digital Libraries ([KW98]). This extended model does not only have an impact on the prediction algorithm, but also on the induction and prefetching algorithm. The implications for the speculative execution are interesting because the component picking prefetching candidates would have additional statistical information about the probability that a prefetching candidate can actually be fully prefetched. This factor could be incorporated into the benefit estimation.

Hidden Markov Models are typically applied in speech recognition ([Rab89]) and computational biology ([KBM+94]). They extend the classical Markov model by assuming that not all states (and thus state transitions) of the system can be observed from outside the system. This nicely models the situation in interactive analysis environments, where the state of the user's analysis process can change without the user executing a query that is perceivable by the system. Mathematically, this is achieved by defining a set of states and a set of observation symbols. When the system is in a specific state, there is a certain possibility that an observation symbol (in our case corresponding to a query) is generated. This probability is given by a probability distribution that is defined for each state. Although these models model the nature of the analytical process more adequately, they have some challenging problems in their application for prediction arising from the fact that observations and states do not directly correspond: The training of Hidden Markov Models is considerably more complicated compared to DTMMs as the training sequence is a sequence of observations that can possibly be generated by a large set of state transition sequences and it is not directly obvious how the parameters of the system have to be adjusted. The same problem arises when predicting the next observation based on the current session context (this is also a sequence of operations). Additionally, the prediction process will be more complex, as a larger set of possible transitions and observation generations must be tested. Therefore, an adaptation of the PROMISE/OLAP framework using Hidden Markov Models seems to be an interesting scientific challenge.

7.3.3.2 Using Variable Generalizations and Pattern Schemata

For the PROMISE approach we have defined two generalizations for canonical queries. The design of these generalizations was heuristic and guided by a thorough analysis of the characteristics of the OLAP analysis process. However, it is possible that better generalizations exist

for specific application areas and also for all applications areas. Currently, a change to the generalizations influences the structure of the OLAP specific prediction algorithm that has to combine the prediction results of the different models. However, the underlying pattern induction and partial prediction algorithms are independent of the generalizations, as their states correspond to events that have already been generalized. An interesting and at the same time scientifically challenging work would be to extend the prediction algorithm in a way, that it can be parameterized with different generalizations and their interrelationships. This would e.g., allow for the domain specific adjustment of generalizations.

In the extreme case this would mean that generalizations are automatically generated and modified by the system. Clustering algorithms from the area of knowledge discovery in databases could be adapted such that they dynamically build and modify equivalency classes based on an analysis of the previous sessions observed. The criteria for building clusters would be a measurement for the adequacy of the resulting pattern model. Designing such a clustering algorithm is another significant challenge.

Of course, a further extension would be to drop the assumption that all patterns are sequential patterns. For example, restriction values for one dimension may influence the probabilities for the restriction values in another dimension of the same query (described as simple interaction patterns in Section 2.4.2). An incorporation of additional respectively different pattern schemata into the framework does require changes to the prediction algorithm. These changes also affect the basic Markov model algorithms if the pattern formulated according to the new pattern schemata cannot be represented as Markov models. The ultimate goal would be the design of a prediction algorithm that is fully generic and can be parameterized with the pattern schemata. This would require that patterns are e.g., described as logical rules and that the prediction algorithm uses an inference mechanism.

7.4 Concluding Remarks

This thesis started with the premise that prediction techniques for navigational data access can be applied to predict the query behavior for users of OLAP systems. Our theoretical analysis of the problem has shown, that the OLAP area exhibits certain properties (mainly the number of different queries and the query subsumption problem) that makes prediction as well as the application of prediction results significantly more complicated than e.g., in hypertext environments. On the other hand, limitations of OLAP domain compared to database systems in general (semantically rich multidimensional data model containing classification semantics, graphical query formulation and thus a limited class of queries with locality properties) can be exploited such that prediction still makes sense. Our theoretical and practical evaluations have proven that the PROMISE approach makes sense, in environments where the user behavior exhibits navigational characteristics. However, it’s probably still a long way until parts of these techniques will really have an impact on OLAP products.

When starting our work (in the beginning of 1997) a strong discussion was going on in the scientific community (and also in the commercial community), if the logical multidimensional data model should be regarded as a full-fledged data model of its own. At this time, industry was well ahead of scientific research in this point as OLAP products were already sold on a large scale, while the research community did not perceive the MD model as a significant topic. Meanwhile, a consensus has been reached in both communities that the MD paradigm can be applied on the conceptual, logical and physical layer, for application areas that exhibit multidimensional characteristics. Thus, we believe that the MD model will take its place as an alternative to relational and object-oriented techniques and that a broader understanding of the

peculiarities of OLAP modeling will emerge. Currently, it can be observed, that the technical innovations of OLAP vendors (who formerly were ahead of research) are getting scarce. On the other hand, a large body of research work has been done based on the MD model that has not been incorporated into products yet. Thus, the research community has meanwhile regained the initiative in the OLAP field.

Personal discussions with representatives of OLAP system vendors and consultants in this field have shown that currently, there is only a small awareness of the potential benefits of addressing user behavior in OLAP system design. However, we are optimistic that the necessity will be recognized by the industry in the near future. The main two reasons for this are:

- OLAP system vendors are extending their classical market (management information systems) and are offering their base technology for entering the market for web portals and eCommerce solutions. This means that the MD data model will in future not be restricted to business analysis modeling, but will be applied for a broader range of applications. In these areas, even more applications for knowledge about user behavior will be possible. E.g., the classification of users according to their typical navigation paths requires mechanisms to induce and represent these patterns.
- A paradigm shift for decision makers from analyzing static reports to interactively analyzing data online can already be perceived and will certainly be increased in future. The main driving factor for this process are that a larger availability of OLAP tools in company increases the awareness for the potentials of this technology. The second factor is that the willingness to use modern information technology in management is highly increasing with new generations of mangers.

In retrospective, we regard this thesis as a first step towards incorporating user behavior into the design of interactive information systems. We hope, that this work will form the nucleus for further research in this innovative area of research that will eventually lead to innovations in products.

«"Inventive man" has invented nothing -- nothing "from scratch."
If he has produced a machine that in motion overcomes the law of gravity,
he learned the essentials from the observation of birds.»
-- Dorothy Thompson (1894-1961)
"The Courage To Be Happy," 1957

«When you steal from one author, it's plagiarism,
if you steal from many, it's research.»
-- Wilson Mizner

Bibliographic References

[ABD+99] J. Albrecht, A. Bauer, O. Deyerling, H. Günzel, W. Hümmer, W. Lehner, L. Schlesinger: *Management of Multidimensional Aggregates for Efficient Online Analytical Processing*. In Proceedings of the 1999 International Database Engineering and Applications Symposium (IDEAS 99), Montreal, Canada, 1999, pp. 156-164.

[Alb01] J. Albrecht: *Anfrageoptimierung in Data-Warehouse-Systemen auf Grundlage des multidimensionalen Datenmodells* (in German). Dissertation Thesis, Universität Erlangen-Nürnberg, 2001.

[AGL+99] J. Albrecht, H. Günzel, W. Lehner: *Set-Derivability of Multidimensional Aggregates*. In Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery (DaWak 99), Florence, Italy, LNCS Vol. 1667, pp. 133-142, Springer Verlag, 1999.

[AGS97] R. Agrawal, A. Gupta, S. Sarawagi: *Modelling Multidimensional Databases.* In Proceedings of the 13th International Conference on Data Engineering (ICDE 97), Birmingham, UK, pp. 232-243, IEEE Computer Society, 1997.

[ANZ99] D.W. Albrecht, I. Zukerman, A.E. Nicholson: *Pre-sending Documents on the WWW: A Comparative Study*. In Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 99), Stockholm, Sweden, pp. 1274-1279, Morgan Kaufmann, 1999.

[Bau99] P. Baumann: *A Database Array Algebra for Spatio-Temporal Data and Beyond*. In Proceedings of the 14th International Workshop on Next Generation Information Technologies and Systems (NGITS 99), Jerusalem, Israel, LNCS Vol. 1649, pp. 76 – 93, Springer Verlag, 1999.

[BC96] A. Bestavros, C. Cunha: *Server-Initated Document Dissemination for the WWW.* In Data Engineering Bulletin Volume 19, Number 3, IEEE Computer Society, 1996.

[Bes96] A. Bestavros: *Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time*. In Proceedings of the 12th International Conference on Data Engineering (IDCE 96), New Orleans, Louisiana, USA, pp. 180-187, IEEE Computer Society, 1996.

[BG00] A. Bauer, H. Günzel: *Data Warehouse* (in German). dpunkt-Verlag, Heidelberg, 2000.

[BL97] A, Bauer, W. Lehner: *The Cube-Query-Language (CQL) for Multidimensional Statistical and Scientific Database Systems*, Proceedings of the 5th International Conference on Database Systems for Advanced Applications (DASFAA 97), Melbourne, Australia, pp. 253-262, World Scientific, 1997.

[Bla00] M. Blaschka: *FIESTA- Framework for Schema Evolution in Multidimensional Database Systems*, Dissertation Thesis, Technische Universität München, 2000.

[BPT97] E. Baralis, S. Paraboschi, E. Teniente: *Materialized Views Selection in a Multidimensional Database*. In Proceedings of the 18th International Conference on Very Large Data Bases (VLDB 97), Athens, Greece, pp. 156-165, Morgan Kaufmann, 1997.

[BSH98] J.W. Buzydlowski, I.Y. Song, L. Hassell: *A Framework for Object-Oriented Online Analytical Processing*. In Proceedings of the 1st International Workshop on Data Warehousing and OLAP (DOLAP'98), Washington DC, USA, pp. 10-15, ACM Press, 1998.

[BSH99] M. Blaschka, C. Sapia, G. Höfling: *On Schema Evolution in Multidimensional Databases*. In Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery (DaWak 99), Florence, Italy, LNCS Vol. 1667, pp. 153-164, Springer Verlag, 1999.

[BSH00] M. Blaschka, C. Sapia, K. Hahn: *BabelFish: A Framework for Conceptual Warehouse Design – How to end the babylonic confusion of tongues in data warehouse design* FORWISS Report 2000-03, FORWISS, Munich, 2000. http://www.forwiss.de/~system42/publications/

[CFK+95] P. Cao, E.W. Felten, A.R. Karlin, K.Li: *A Study of Integrated Prefetching and Caching Strategies*. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Ottawa, Canada, Performance Evaluation Review 23(1), pp. 188-197, 1995.

[CKV93] K.M. Curewitz, P. Krishnan, J.S. Vitter: *Practical Prefetching via Data Compression*, In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C, USA, pp. 257-266, ACM Press, 1993.

[CMS99] R. Cooley, B. Mobasher, J. Srivastava: *Data Preparation for Mining World Wide Web Browsing Patterns*. Knowledge and Information Systems, Volume 1, Number 1, February 1999, pp. 5-32, Springer Verlag, 1999.

[CCS93] E.F. Codd, S.B. Codd, C.T. Salley: *Providing OLAP (Online Analytical Processing) to User-Analysts. An IT Mandate.* Arbor Software Corporation, 1993.

[Cog99] Cognos Corp.: *PowerPlay –Administrator's Guide*, Version 6.5, 1999.

[CT97] L. Cabibbo, R. Torlone: *Querying Multidimensional Databases.* In Proceedings of the 6th International Workshop on Database Programming Languages (DBPL), Estes Park, Colorado, USA, LNCS Vol. 1369, pp. 299-318, Springer Verlag, 1997.

[CT98a] L. Cabibbo, R. Torlone: *A Logical Approach to Multidimensional Databases*. In Proceedings of the 6th International Conference on Extending Database Technology (EDBT' 98), Valencia, Spain, LNCS Vol. 1377, pp. 183-197, Springer Verlag, 1998.

[CT98b] L. Cabibbo, R. Torlone. *From a Procedural to a Visual Query Language for OLAP*. In 10th IEEE International Conference on Scientific and Statistical Database Management (SSDBM '98), Capri, Italy, pp. 74-83, IEEE Computer Society, 1998.

[DFJ+96] S. Dar, M. Franklin, B. Jónsson, D. Srivastava, M. Tan: *Semantic data caching and replacement.* In Proceedings of the International Conference on Very Large Databases (VLDB), Bombay, India, pp. 330-341, Morgan Kaufmann, 1996.

[DKP+99] S. Dekeyser, B. Kuijpers, J. Paredaens, J. Wijsen: *Nested Data Cubes for OLAP*, In Y. Kambayashi et. al. (Eds.): *Advances in Database Technologies*. Proceedings of the International Workshop on Data Warehouse and Data Mining (DWDM 98), LNCS Vol. 1552, pp. 129-140, Springer Verlag, 1999.

[DN00] P. Deshpande, J.F. Naughton: *Aggregate Aware Caching for Multi-Dimensional Queries,* In Proceedings of the 7th International Conference on Extending Database Technology (EDBT 2000), Konstanz, Germany, LNCS Vol. 1777, pp. 17-182, Springer-Verlag, 2000.

[DRS+98] P. Deshpande, K. Ramasamy, A. Shukla, J.F. Naughton: *Caching Multidimensional Queries Using Chunks.* In Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, pp. 259-270, ACM press, 1998.

[DSHB98] B. Dinter, C. Sapia, G. Höfling, M. Blaschka: *The OLAP Market: State of the Art and Research Issues,* In Proceedings of the 1st International Workshop on Data Warehousing and OLAP (DOLAP'98), Washington DC, USA, pp. 22-27, ACM Press, 1998.

[DSV+97] B. Dinter, C. Sapia, M. Vrca, G. Höfling: *Der OLAP Markt – Architekturen Produkte, Trends* (in German). FORWISS Technical Report, 1997.

[DT97] A. Datta ,H. Thomas: *A Conceptual Model and an Algebra for On-Line Analytical Processing in Data Warehouses,* In Proceedings of the Workshop on Information Technologies and Systems (WITS'97), Atlanta, Georgia, USA, 1997.

[FB99] P. Furtado, P. Baumann: *Storage of Multidimensional Arrays based on Arbitrary Tiling*, In Proceedings of the 15th International Conference on Data Engineering (ICDE 99), Sydney, Australia, pp. 480-489, IEEE Computer Society, 1999.

[FCL+99] L. Fan, P. Cao, W. Lin, Q. Jacobsen: *Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance*, In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems '99, Atlanta, Georgia, USA, pp. 178-187, ACM Press, 1999.

[FM00] P. Furtado, H. Madeira: *Vmhist: Efficient Multidimensional Histograms with Improved Accuracy*, In Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWak 2000), Greenwich, UK, LNCS Vol. 1894, pp. 431-236, Springer Verlag, 2000.

[FPS+96a] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.): *Advances in Knowledge Discovery and Data Mining*, AAAI Press & MIT Press, Menlo Park, USA, 1996.

[FPS+96b] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth: *From Data Mining to Knowledge Discovery: An Overview*, in [FPS+96a], pp. 1-36, 1996.

[Fur99] P. Furtado: *Storage Management of Multidimensional Arrays in Database Management Systems*, Dissertation Thesis, Technische Universität München, 1999.

[GA94] J. Griffioen, R. Appleton: *Reducing File System Latency using a Predictive Approach*. in Proceedings of the USENIX Summer 1994 Technical Conference 1994, Massachusetts, USA, pp. 197-207, 1994.

[GA96] J. Griffioen, R. Appleton: *The Design, Implementation, and Evaluation of a Predictive Caching File System*, Technical Report No. CS-264-96, University of Kentucky, 1996.

[GCB+97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh: *Data Cube: A Relational Operator Generalizing Group-By, Cross-Tab and Sub-Totals*. Data Mining and Knowledge Discovery Volume 1, Number 1, pp. 29-53, March 1997.

[GHR+97] H. Gupta, V. Harinarayan, A. Rajaraman, J.D. Ullman: *Index Selection for OLAP*. In Proceedings of the 13th International Conference on Data Engineering (ICDE 97), Birmingham, UK, pp. 232-243, IEEE Computer Society, pp. 208-219, 1997.

[GL97] M. Gyssens, L.V.S. Lakshmanan: *A Foundation for Multi-Dimensional Databases*. In Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 97), Athens, Greece, pp. 106-115, Morgan Kaufmann, 1997.

[GMR98] M. Golfarelli, D. Maio, S. Rizzi: *The Dimensional Fact Model for Data Warehouses*, International Journal on Cooperative Information Systems (IJCIS) Volume 7, Numbers 2-3, pp. 215-247, World Scientific, September 1998.

[GR98] M. Golfarelli, S. Rizzi: *A Methodological Framework for Data Warehouse Design*, In Proceedings of the 1st International Workshop on Data Warehousing and OLAP (DOLAP'98), Washington DC, USA, pp. 3-9, ACM Press, 1998.

[Gup97] H. Gupta: *Selection of Views to Materialize in a Data Warehouse*. In Proceedings of the 6th International Conference on Database Theory (ICDT'97), Delphi, Greece, LNCS 1186, pp. 98-112, Springer Verlag, 1997.

[HBD+97] G. Höfling, M. Blaschka, B. Dinter, P. Spiegel, T. Ringel: *Data Warehouse Technology for the Management of Diagnosis Data* (in German). in Dittrich, Geppert (Eds.): Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'97), Springer Verlag, 1997.

[HH99] A. Harren, O. Herden: *Conceptual Modeling of Data Warehouses*. In Proceedings of Demonstrations and Posters ER99, Paris, France, 1999.

[HMR97] M.S. Hacid, P. Marcel, C. Rigotti, *A rule based data manipulation language for OLAP systems*. In Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases (DOOD'97),Montreux, Switzerland, pp. 417-418, 1997.

[How60] R.A. Howard: *Dynamic Programming and Markov Processes*, John Wiley, 1960.

[HPP+98] B.A. Huberman, P.L.T. Pirolli, J.E. Pitkow, R.M. Lukose: *Strong Regularities in World Wide Web Surfing*. In Science Vol. 280 (April 1998), pp. 95-97, 1998.

[HRU96] V. Harinarayan, A. Rajaraman, J.D. Ullman: *Implementing Data Cubes Efficiently*. In Proceedings of the 25th International Conference on Management of Data (SIGMOD'96), Montreal, Canada, pp. 205-126, ACM Press, 1996.

[HSB00] K. Hahn, C. Sapia, M. Blaschka: *Automatically Generating OLAP Schemata from Conceptual Graphical Model*s. In Proceedings of the 3rd International Workshop on Data Warehousing and OLAP (DOLAP'00), Washington DC, USA, ACM Press, 2000.

[Inf98] Informix Corporation: *Data Warehouse Administrators Guide, MetaCube ROLAP Option for Informix Dynamic Server Version 4.0.* Informix Press, Menlo Park, USA, January 1998.

[Inf96] Informix Corporation: *Informix Decision Support: Indexing the Enterprise Data Warehouse*. White Paper, http://www.informix.com, Menlo Park, USA, 1996.

[Inm96] W.H. Inmon: *Building the Data Warehouse*. John Wiley & Sons, Inc., 1996.

[Jac92] I. Jacobsen: *Object-Oriented Software Engineering – A Use Case Driven Approach*, ACM Press, 1992

[JLS99a] H.V. Jagadish, L.V.S. Lakshamanan, D. Srivastava: *What can Hierarchies do for Data Warehouses?*, In Proceedings of the 25th International Conference on Very Large Databases (VLDB'99), Edinburgh, Scotland, pp. 530-541, Morgan Kaufmann, 1999

[JLS99b] H.V. Jagadish, L.V.S. Lakshamanan, D. Srivastava: *Snakes and Sandwiches: Optimal Clustering Strategies for a Data Warehouse*, In Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, Philadephia, Pennsylvania, USA, pp. 530-541, 1999.

[JLV+99] M. Jahrke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis: *Fundamentals of Data Warehouses*, Springer Verlag, 1999

[KBM+94] A. Krough, M. Brown, I.S. Milan, K. Sjölander, D. Haussler: *Hidden Markov Models in computational biology. Applications to Protein Modeling*. In Journal of Molecular Biology, 235, pp. 1501-1531, 1994.

[KL96] Tom M. Kroeger, Darrell D. E. Long: *Predicting Future File-System Actions From Prior Events*. in Proceedings of the USENIX Annual Technical Conference 1996: pp.319-328, 1996.

[KW98] A. Kraiss, G. Weikum: *Integrated document caching and prefetching in storage hierarchies based on Markov-chain predictions*. In The VLDB Journal, Vol. 7, pp. 141-162, Springer-Verlag, 1998.

[KZ96] W. Klösgen, J.M. Żytkow: *Knowledge Discovery in Databases Terminology*. In [FPS+96a], pp. 573-592, 1996.

[LAH00] W. Lehner, J. Albrecht, W. Hümmer: *Divide and Aggregate: Caching Multidimensional Objects*. In Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses (DMDW 2000), Stockholm, Sweden, 2000.

[Leh98a] W. Lehner: *Modeling Large Scale OLAP Scenarios*. In Proceedings of the 6th International Conference on Extending Database Technology (EDBT' 98), Valencia, Spain, LNCS Vol. 1377, pp. 153-167, Springer Verlag, 1998.

[Leh98b] W. Lehner: *Adaptive Preaggregations-Strategien für Data Warehouses*. (in German) Dissertation Thesis, University of Erlangen-Nuremburg, 1998.

[LRT96] W. Lehner, T. Ruf, M. Teschke: *CROSS-DB: A Feature-Extended Multidimensional Data Model for Statistical and Scientific Databases*, In Proceedings of the 5th International Conference on Information and Knowledge Management (CIKM '96), Rockville, Maryland, USA, pp. 253-260, ACM Press, 1996.

[LS97] H.-J. Lenz, A. Shoshani: *Summarizability in OLAP and Statistical Data Bases*. In Proceedings of the 9th IEEE International Conference on Scientific and Statistical Database Management (SSDBM 97), Olympia, Washington, USA, pp. 132-143, IEEE Computer Society, 1997.

[LW96] C. Li and X.S. Wang: *A data model for supporting on-line analytical processing.* In Proceedings of the 5th International Conference on Information and Knowledge Management (CIKM '96), Rockville, Maryland, USA, pp. 81-88, ACM Press, 1996.

[Mar99] V. Markl. MISTRAL: *Processing Relational Queries using a Multidimensional Access Technique,* Dissertation Thesis, TU München, 1999, published by infix Verlag, St. Augustin, DISDBIS 59, 1999.

[Mic98] Microsoft Corp., *OLEDB for OLAP Specification 1.0*, 1998. *http://www.microsoft.com/oledb/*

[MRB99] V. Markl, F. Ramsak, R. Bayer. *Improving OLAP Performance by Multidimensional Hierarchical Clustering*. In Proceedings of the 1999 International Database Engineering and Applications Symposium (IDEAS '99), Montreal, Canada, pp. 165-177, IEEE Computer Society, 1999.

[MZB99] V. Markl, M. Zirkel, R. Bayer. *Processing Operations with Restrictions in Relational Database Management Systems without external Sorting*. In Proceedings of the 15th International Conference on Data Engineering (ICDE 99), Sydney, Australia, pp. 562-571, IEEE Computer Society, 1999.

[NZA98] A.E. Nicholson, I. Zukerman, D.W. Albrecht: *A Decision-Theoretic Approach for Pre-sending Information on the WWW*. In Proceedings of the 5th Pacific Rim International Conference on Artificial Intelligence (PRICAI 98), LNCS 1531, pp. 575-586, Springer Verlag, 1998.

[Nel95] R. Nelson: *Probabilistic Stochastic Processes, and Queuing Theory – The Mathematics of Computer Performance Modeling.* Springer Verlag, 1995.

[OLA98] OLAP Council *OLAP Council API Specification*, Version 2.0, 1998. http://www.olapcouncil.org/

[OOM87] G. Ozsoyoglu, M. Ozsoyoglu, V. Matos: *Extending Relational Algebra and Relational Calculus with Set-Based Attributes and Aggregation Functions*. In ACM Transactions on Database Systems (TODS), Vol. 12, Nr. 4, ACM Press, 1987.

[Ora98] Oracle Corp. *Oracle Express Administrators Guide – Version 6.2*, 1998

[PC00] N. Pendse, R. Creeth: *The OLAP Report,* Business Intelligence Ltd., 2000. http://www.olapreport.com/

[PJ99] T. B. Pedersen, C. S. Jensen: *Multidimensional Data Modeling for Complex Data*. In Proceedings of the 15th International Conference on Data Engineering (ICDE '99), Sydney, Australia, pp. 336-345, IEEE Computer Society, 1999.

[PM96] V. N. Padmanabhan, J. C. Mogul: *Using Predictive Prefetching to Improve World Wide Web Latency*. In Proceedings of ACM SIGComm, 1996, pp.26-36, ACM Press, 1996.

[PP99a] P. Pirolli, J. Pitkow,: *Distributions of Surfers' Paths through the World Wide Web. Empirical Characterization*. In World Wide Web Journal, Vol 2(1999) 1-2, pp. 29-45, Baltzer Science Publishers, 1999.

[PZ91] M. Palmer, S.B. Zdonik: *FIDO: A Cache That Learns to Fetch*. In Proceedings of the 17th International Conference on Very Large Data Bases (VLDB), Barcelona, Spain, pp. 255-264, Morgan Kaufmann, 1991.

[Rab89] L.R. Rabiner: *A tutorial on hidden Markov models and selected applications in speech recognition*. Proceedings of the IEEE, 77(2), pp.257-286, 1989.

[Sap99] C. Sapia *On Modeling and Predicting User Behavior in OLAP Systems*, In Proceedings of the 1st International CaiSE Workshop on the Design and Management of Data Warehouses (DMDW 99), Heidelberg, Germany, 1999.

[Sap00] C. Sapia: *PROMISE: Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP systems*, In Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWak 2000), Greenwich, UK, LNCS Vol. 1894, pp. 224-233, Springer Verlag, 2000.

[SBH+99] C. Sapia, M. Blaschka, G. Höfling, B. Dinter, *Extending the E/R Model for the Multidimensional Paradigm*, In Y. Kambayashi et. al. (Eds.): *Advances in Database Technologies*, Proceedings International Workshop on Data Warehouse and Data Mining (DWDM 98), Singapore, LNCS Vol. 1552, pp. 105-116, Springer Verlag, 1999.

[SBH99] C. Sapia, M. Blaschka, G. Höfling: *An Overview of Multidimensional Data Models*, FORWISS Technical Report FR- 1999-001.

[SBH00] C. Sapia, M. Blaschka, G. Höfling, *Grammi: Using a Standard Repository Management System to build a Generic Modeling Tool*, In Proceedings of the Hawaiian International Conference on System Sciences (HICSS'00), Maui, Hawaii, USA, IEEE Computer Society, 2000.

[Sch01] E. Scherer, *Evaluating the Performance of Predictive Prefetching Algorithms in OLAP Environments*, Master Thesis, Technische Universität München, 2001.

[SDJ+96] D. Srivastava, S. Dar, H. V. Jagadish and A. Y. Levy: *Answering Queries with Aggregation Using Views*. In Proceedings of the 22nd International Conference on Very Large Databases (VLDB'96), Bombay, India, pp. 318-329, Morgan Kaufmann, 1996.

[SDN98] A. Shukla, P. Deshpande, J.F. Naughton: *Materialized View Selection for Multidimensional Datasets,* Proceedings of the 24th International Conference on Very Large Databases (VLDB '98), New York, USA, pp. 488-499, Morgan Kaufmann, 1998.

[SKN89] X. Sun, N. Kamel, L. Ni: *Solving implication problems in database applications*. In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'89), Portland, Oregon, USA, pp. 185-192, ACM Press, 1989.

[SSV96] P. Scheuermann, J. Shim, R. Vingralek: *WATCHMAN: A Data Warehouse Intelligent Cache Manager*. In Proceedings of the 22nd International Conference on Very Large Databases (VLDB '96), Bombay, India, pp. 51-62, Morgan Kaufmann, 1996.

[SSV99] J. Shim, P. Scheuermann, R. Vingralek: *Dynamic Caching of Query Results for Decision Support Systems*. In Proceedings of the 11th International Conference on Scientific and Statistical Database Management (SSDBM), Cleveland, Ohio, USA, 1999. pp. 254-263, IEEE Computer Society, 1999.

[SZ00] G. Specht, P. Zoller: *HMT: Modeling Temporal Aspects in Hypermedia Applications.* In Proceedings of the 1st International Conference on Web Age Information Management (WAIM), Shanghai, China, 2000.

[TS98] D. Theodoratos, T. Sellis: *Data Warehouse Schema and Instance Design*, In Proceedings of the 17th International Conference on Conceptual Modeling (ER'98), LNCS Vol. 1507, pp. 363-376, Springer Verlag, 1998.

[TS97] D. Theodoratos, T. Sellis: *Data Warehouse Configuration*. In Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 97), Athens, Greece, pp. 126-135, Morgan Kaufmann, 1997

[Tij94] H.C. Tijms: *Stochastic Models - An Algorithmic Approach*. John Wiley & Sons, 1994.

[Vas98] P. Vassiliadis: *Modeling Multidimensional Databases, Cubes and Cube Operations*. In 10th IEEE International Conference on Scientific and Statistical Database Management (SSDBM '98), Capri, Italy, pp. 53-62, IEEE Computer Society, 1998.

[VS99] P. Vassiliadis, T. Sellis: *A Survey of Logical Models for OLAP Databases*. SIGMOD Record, Volume 28, Number 4, pp. 64-69, ACM Press, 1999.

[Vas00] P. Vassiliadis: *Gulliver in the Land of Data Warehousing*. In Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses (DMDW 2000), Stockholm, Sweden, 2000.

[WB98] M.-C. Wu, A.P. Buchmann: *Encoded Bitmap Indexing for Data Warehouses*. In Proceedings of the 14th International Conference on Data Engineering (ICDE 98), Orlando, Florida, USA, pp. 220-230, IEEE Computer Society, 1998.

[YKL97] J. Yang, K. Karlapalem, Q. Li: *Algorithms for Materialized View Design in Data Warehousing Environment*. In Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97), Athens, Greece, pp. 136-145, Morgan Kaufmann, 1997.

[ZAN99] D.W. Zukerman, I., Albrecht, D., and Nicholson, A., *Predicting Users' Requests on the WWW*. In Proceedings of the 7th International Conference on User Modeling (UM 99), Banff, Canada, Springer-Verlag 1999.

[ZRT+98] Y. Zhao, K. Ramasamy, K. Tufte, J.F. Naughton: *Array-Based Evaluation of Multi-Dimensional Queries in Object-Relational Databases Systems*. In Proceedings of the 14th International Conference on Data Engineering (ICDE 98), Orlando, Florida, USA, pp. 241-249, IEEE Computer Society, 1998.

Appendix A: Detailed Proofs for Theorems

Theorem 3.1 (Validity of Classification Lattice)

Every Classification Lattice $\Psi|_{base} = (L'_\Psi, class'_\Psi)$ defined on the classification schema $\Psi = (L_\Psi, class_\Psi)$ has the following properties:

- $(l,l) \in class'_\Psi \quad \forall l \in L'_\Psi$ (1)
- $(l_1,l_2) \in class'_\Psi \wedge (l_2,l_3) \in class'_\Psi \Rightarrow (l_1,l_3) \in class'_\Psi \quad \forall l_1,l_2,l_3 \in L'_\Psi$ (2)
- $(l_1,l_2) \in class'_\Psi \Rightarrow (l_2,l_1) \notin class'_\Psi \quad \forall l_1,l_2 \in L'_\Psi$ with $l_1 \neq l_2$ (3)
- Two levels $l_1 \in L'_\Psi$ and $l_2 \in L'_\Psi$ have a least upper bound (LUB)
- Two levels $l_1 \in L'_\Psi$ and $l_2 \in L'_\Psi$ have a greatest lower bound (GLB)

Proof:

The general idea for proofs (1-3) is to prove the required property for classification levels which are in the lattice as well as in the classification schema, thus, $l \in L_\Psi \cap L'_\Psi$ by showing that the construction of the lattice did not harm the appropriate property (which is fulfilled in the classification schema by definition). Additionally, the property has to be proven for the special level *l.all.*

Prerequisite (P): $\Psi=(L_\Psi,class_\Psi)$ is a valid classification schema, therefore $class_\Psi$ defines a partial order on L_Ψ.

(1) This condition is fulfilled for $l \in L_\Psi \cap L'_\Psi$ because of (P) and for $l=l.all$ because of the construction of class: $l \leq l.\text{all} \; \forall \; l \in L'_\Psi$

(2) This condition is fulfilled for l_1, l_2, $l_3 \in L_\Psi \cap L'_\Psi$ because of (P). If $l_3=l.all$, the implication is automatically fulfilled because every level l_1 can be classified according to *l.all.* If either l_1 or l_2 are equal to *l.all,* it can be automatically concluded that $l_3=l.all$ as *l.all* is the only level that can be classified according to *l.all*.

(3) This condition is fulfilled for l_1, $l_2 \in L_\Psi \cap L'_\Psi$ because of (P). If $l_1 = l.all$ the left side of the implication always evaluates to false as no $l_2 \neq l.all$ according to which the all level can be classified. This means that the implication is always true. If $l_2 = l.all$, the right side of the implication always evaluates to true. Therefore, the implication is also always fulfilled.

(4) By definition *l.all* is an upper bound for every l_i, l_j as $l \leq l.all \; \forall \; l \in L'_\Psi$. That means that the set of upper bounds for any l_i, l_j always contains the level *l.all* which is also an upper bound for all members of the set of upper bounds. Consequently, a (unique) least upper bound can be found in the set.

(5) The base level *base* of the lattice $\Psi|_{base}$ is by definition a lower bound for any combination $l_1,l_2 \in L'_\Psi$ of levels as $l \leq l.all$. With argumentation analogous to case (4), a greatest lower bound can be found.

◆

Theorem 5.1 (Subsumption of Super Query)

Every member of a set of canonical OLAP queries Q is subsumed by the super-query for this query set *super*(Q). Thus,

$$super(Q) \textbf{ subsumes } q \quad \forall q \in Q$$

◆

Proof:

Let us assume the Q=$\{q_1,\ldots,q_k\}$ is a set of canonical OLAP queries and that each of the queries in Q has the following form ($i \in [1;k]$):

$$q_i = \left(M_i, (g_{i,1},\ldots,g_{i,n}), (r_{i,1},\ldots,r_{i,n})\right)$$

The elements of *super*(*Q*) are denotes as follows:

$$super(Q) = (M_S, \left(M_S, (g_{S,1},\ldots,g_{S,n}), (r_{S,1},\ldots,r_{S,n})\right)$$

In order prove that the required *super*(*Q*) **subsumes** *q* holds for all q_i, we have to prove, that the following three conditions are fulfilled (Definition 5.2) for all $i \in [1;k]$

- $M_S \supseteq M_i$ (1)
- $g_{S,j} \leq_\Psi g_{i,j} \quad \forall j \in [1;n]$ (2)
- $\text{active}_{dj}(\text{super}(Q)) \supseteq \{x \in dom(g_{S,j}) \mid \text{x} \in \text{descendants}(r_{i,j})\} \quad \forall j \in [1;n]$ (3)

The proof for (1) is a simple consequence of the definition of *super*(*Q*). According to Definition 5.7, M_S id defined as follows:

$$M_S = \bigcup_{i \in [1;k]} M_i \quad \Rightarrow \quad M_S \supseteq M_i$$

Property (2) can also be directly concluded from the definition of $g_{S,i}$ in Definition 5.7 and the definition of the greatest lower bound:

$$g_{S,j} = GLB_{\Psi|d_i}\left(\bigcup_{i=1}^{k}\{g_{i,j}\}\right) \quad \Rightarrow \quad g_{S,j} \leq_\Psi g_{i,j} \quad \forall j \in [1;n]$$

According to Definition 3.17, the left side of the set inequation (3) can be transformed as follows:

$$\text{active}_{dj}(\text{super}(Q)) = \left\{m \mid m \in dom(g_{S,j}) \wedge group_{g_j, level(r_j)}(m) = r_{S,j}\right\}$$

which can be transformed to the following statement using the definition of the descendants function in Definition 3.16:

$$\text{active}_{dj}(\text{super}(Q)) = \left\{m \mid m \in dom(g_{S,j}) \wedge \text{descendants}(r_{S,j})\right\} \quad \text{(I)}$$

According to Definition 5.7, the following condition holds for $r_{S,i}$:

$$r_{S,i} \in \bigcap_{j \in [1;n]} \left(\text{ancestors}(r_{i,j}) \cup \{r_{i,j}\}\right) \quad \text{(II)}$$

Using the definition of the *descendants* function, we can conclude the following from this condition

$$\text{(II)} \Rightarrow \text{descendants}(r_{S,j}) \supseteq \text{descendants}(r_{i,j}) \qquad \text{(III)}$$

This means that we can further transform (I) in the following way:

$$\text{(I),(III)} \Rightarrow \text{active}_{dj}(\text{super}(Q)) \supseteq \left\{ m \mid m \in dom(g_{S,j}) \wedge \text{descendants}(r_{i,j}) \right\} \qquad \text{q.e.d}$$

◆

Appendix B: Indexes and Tables

Table of Figures

Table of Definitions

Table of Theorems

Table of Proofs

Appendix C: Glossary

Term	Description	Symbol(s)	Definition
Briefing book	Several OLAP report templates definitions can be combined to a briefing book, which is a set (mostly ordered in as a linear sequence) of predefined reports. The composition of a briefing book corresponds to the workflow of the analysis process.	--	--
Benefit			
Prefetching -	Benefit of prefetching a →*canonical OLAP query q* in the presence of a predicted workload *P* and an →*OLAP cache* contents Γ	$prefetch\text{-}benefit_{\Gamma}(q,P)$	Definition 5.6
Probabilistic -, additional	Additional benefit of adding an →*OLAP cache* object (q,ρ) to the →*OLAP cache contents* Γ for answering a predicted workload *P*.	$addbenefit_{\Gamma}(q,P)$	Definition 5.4
Probabilistic -	Anticipated benefit of an →*OLAP cache object* (q,ρ) already contained in the →*OLAP cache contents* Γ for answering a predicted workload *P*.	$benefit_{\Gamma}(q,P)$	Definition 5.3
Canonical OLAP			
- Query	Canonical form of a query against an OLAP system considering the capabilities of the OLAP →*GUI*	q	Definition 3.12
- Query Space	The set of all well-formed canonical queries that can be formulated according to an →*MD cube schema*	Θ_{C_Ψ}	Definition 3.12
Classification			
- Instance	Instance of a classification schema containing the →*grouping functions* mapping classification nodes to its parent node	I_Ψ	Definition 3.5
- Lattice	Subgraph of the →*classification schema* defined by a base level containing a special 'all' level.	$\Psi\vert_{\vert}$	Definition 3.6
- Level	Represents a classification criterion (e.g. vehicle group). Part of a →*classification schema*	l	Definition 3.1
- Level Domain	Set of →*classification nodes* that belong to the →*classification level*	$dom(l)$	Definition 3.1
- Node	Member of a →*classification level domain*	x	Definition 3.1
- Schema	Graph of →*classification levels* containing information about the classification relationships.	Ψ	Definition 3.2
- Schema Domain	Union of the →*classification level domains* for all levels belonging to the schema	$dom(\Psi)$	Definition 3.4
- Schema Path	Path in the →*classification schema* (resp. → *classification lattice*) defining a hierarchy of →*classification levels*	*cpath*	Definition 3.3

Term	Description	Symbol(s)	Definition
Concatenation of functions	$a \circ b := b(a(x))$	$\circ$	--
Domain			
Active Dimension-	→classification nodes of a dimension that reference at least one occupied cell in an →MD cube instance	$\mathbf{actdom}(d_i)$	Definition 3.9
Dimension	Part of the →*MD cube schema*. Each dimension is chosen from the set of →*classification levels* contained in the →*classification schema*	d	Definition 3.7
Selection -	set of dimensions in a →canonical OLAP query q, where the level of the restriction element is equal to the result granularity.	$\sigma(q)$	Definition 3.14
Result -	dimensions that are not →selection dimensions of a →canonical OLAP query	--	Definition 3.14
Event			
- atomic-	In the context of the PROMISE/OLAP approach, an atomic interaction event is the execution of a →*canonical OLAP query* by the user. This query may be predefined or derived from the results of the previous query.	e,q	Definition 2.2
- composite	A composite event is a sequence of atomic events. In the context of the PROMISE/OLAP approach, a composite interaction event is a session.	Ses	Definition 2.3
- /state mapping	A bijective function mapping each event to the state of a →Markov model M	$\equiv_M$	Definition 4.2
Grouping function	Function mapping elements of a →*classification level domain* to another domain that classifies this level	$group_{l_1,l_2}$	Definition 3.5
GUI	Graphical User Interface	--	--
Markov Model	Data structure used in PROMISE to represent a set of probabilistic sequential rules. Central part of the →*prediction profile*	$DTMM$	Definition 4.1
- order	Number of previously visited states that is taken into account by the transition probability function of the Markov model.	m	Definition 4.1
MD Cube			
- instance	The instance of an →*MD cube schema C*. Function mapping the coordinates to cell values	I_C	Definition 3.8
- instance space	Set of all cube instances for a given →*MD cube schema C*	Ξ_C	Definition 3.8
- schema	Description of the structure of a multidimensional cube containing the measure set and the dimension description	C_Ψ	Definition 3.7

Term	Description	Symbol(s)	Definition
MD presentation	Graphical visualization method used to phrase a →*canonical OLAP query q* and to present the results of *q*		
- granularity	Represents the information which granularity is chosen for the →result dimensions of q	$presgran_q$	Definition 3.15
- structure	Represents the information which of the dimensions are chosen as →result dimensions or →selection dimensions	$presstruct_q$	Definition 3.15
MDX	**M**ulti**d**imensional E**x**pressions. The query language of the →OLEDB for OLAP standard	--	--
OLAP		--	
- application	The application build on top of an →OLAP system. An OLAP application is specified by configuring the OLAP system (for example defining a multidimensional schema)		Figure 1.1
- session	a sequence of →*canonical OLAP queries* executed by the same user solving an analytical task. Used as the scope of prediction. A →*composite interaction event* in the OLAP →*user interaction model*	*Ses*	Definition 2.3
- system	The software system which is used to build an →*OLAP application*		Figure 1.1
OLAP Cache			
- contents	Set of cache objects contained in an OLAP cache	Γ	Definition 5.1
- object	Description of an object managed by an OLAP cache. A combination of a →*canonical OLAP query* and an →*MD cube instance* describing the results of the query.	γ	Definition 5.1
OLAP Query			Definition 5.2
- Prototype	Generalization of a →*canonical OLAP query*. In PROMISE we distinguish between structure- and value-based prototypes.	$\wp_{Struct}$ $\wp_{Val,i}$	Definition 4.4; Definition 4.5
Subquery, common	Largest common query that can be used in answering all →*canonical OLAP queries* contained in the set *Q*	**common**(*Q*)	Definition 5.8
- Subsumption	Relationship between →*canonical OLAP queries*. If query $q_1$subsumes q_2, this means that q_2 can be answered from the results of q_1.	$subsumes(q_1,q_2)$	Definition 5.2
Superquery	Smallest →*canonical OLAP query* that can →*subsumes* all queries contained in the set *Q*	**super**(*Q*)	Definition 5.7
- Transformation	Function mapping a →*canonical OLAP query* to another canonical OLAP query. Abstractly describes the interaction of the user with the OLAP →*GUI*.	$\tau(q)$	Definition 3.18-Definition 3.22
- Transformation Distance	Minimum number of transformation operations needed to transform the →*canonical OLAP query* q_1	$\lvert q_1\text{-}q_2\rvert_T$	Definition 5.5
OLEDB for OLAP	Interface proposed by Microsoft to define and access multidimensionally structured data. See [Mic98].	--	--

Term	Description	Symbol(s)	Definition
Pattern			
- Instance	Formal intensional description of a subset of interesting objects. In the case of PROMISE/OLAP this is a description of properties of →OLAP sessions	P	Definition 2.5
probabilistic -	A →*pattern instance* with a probability value	--	Definition 2.6
- Representation	Formal Language used to describe →*pattern schemata* and →*pattern instances*	L	Definition 2.5
- Schema	Expression of the →*pattern representation* language containing free variables	--	Definition 2.5
Prediction Profile	Central data structure used to represent →patterns in the PROMISE/OLAP framework and to predict →canonical OLAP queries. For an *n*-dimensional →*MD cube schema*, it consists of *n*+1 Markov models. One model uses → structural query prototypes, while *n* models use →value-based-query prototypes.	Φ_{C_Ψ}	Definition 4.7
PROMISE	**Pr**edicting User Behavi**o**r in **M**ultidimensional **I**nformation **S**ystem **E**nvironments. The name of the approach presented in this thesis ☺	--	--
Real numbers	The infinite set of all real numbers	$\mathbb{R}$	--
Session Context	The sequence of *m* →*events* that occurred directly before an interaction event in a session *Ses*.	$cont_m^{Ses}$	Definition 2.4
State Sequence	The sequence of →*Markov model* states that is produced by mapping each →*event* of a session *Ses* to a state of the Markov model *M* using the →event/state mapping.	$stateseq_M(Ses)$	Definition 4.3
User Interaction			
- Model	A meta model for designing description formalisms for interactions between users and interactive navigational information systems. Consists of →event types, attributes and integrity constraints.	IM	Definition 2.1
- Pattern Model	A meta model for the description of pattern representations. Bases on an →interaction model IM, the pattern model contains →patterns, →pattern schemata and generalization functions for the →events of the interaction model *IM*.	PM_{IM}	Definition 2.7