

Extending OLAP Querying To External Object Databases

Torben Bach Pedersen
Department of Computer
Science, Aalborg University
tbp@cs.auc.dk

Arie Shoshani Junmin Gu
Lawrence Berkeley National
Laboratory
shoshani,jgu@lbl.gov

Christian S. Jensen
Department of Computer
Science, Aalborg University
csj@cs.auc.dk

ABSTRACT

On-Line Analytical Processing (OLAP) systems based on a multidimensional view of data have found widespread use in business applications and are being used increasingly in non-standard applications. These systems provide good performance and ease-of-use. However, the complex structures and relationships inherent in data in non-standard applications are not accommodated well by OLAP systems. In contrast, object database systems are built to handle such complexity, but do not support OLAP-type querying well.

This paper presents the concepts and techniques underlying a flexible, “multi-model” federated system that enables OLAP users to exploit simultaneously the features of OLAP and object database systems. The system allows data to be handled using the most appropriate data model and technology: OLAP systems for multidimensional data and object database systems for more complex, general data. Additionally, physical data integration can be avoided. As a vehicle for demonstrating the capabilities of the system, a prototypical OLAP language is defined and extended to naturally support queries that involve data in object databases. The language permits selection criteria that reference object data, queries that return combinations of OLAP and object data, and queries that group multidimensional data according to object data. The system is designed to be aggregation-safe, in the sense that it exploits the aggregation semantics of the data to prevent incorrect or meaningless query results. A prototype implementation of the system is reported.

1. INTRODUCTION

On-Line Analytical Processing (OLAP) systems have become increasingly popular in many application areas, as they considerably ease the process of analyzing large amounts of enterprise data. Designed specifically with the aim of better supporting the retrieval of higher-level summary information from detail data, these systems offer better performance for aggregate queries than do traditional DBMSs. The multidimensional data models employed in OLAP systems enable visual querying, as well as support the semantics of the model. As an example, most OLAP systems support *automatic aggregation* [20, 14], meaning that the systems know which aggregate functions to apply when retrieving different higher-level summaries.

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM 2000, McLean, VA USA
© ACM 2000 1-58113-320-0/00/11 . . . \$5.00

Almost all OLAP systems are based on a *dimensional* view of data, in which measured values, termed facts, are characterized by descriptive values drawn from a number of dimensions; and the values of a dimension are typically organized in a containment-type hierarchy. While the dimensional view of data is particularly well suited for the aggregation queries performed in OLAP analysis, this view also limits the abilities of OLAP systems to capture complex relationships in the data. As a result, an OLAP database only captures some of the structure available in the data from which it derives. Furthermore, it is often difficult or impossible to combine data from an OLAP system with data from other sources.

In contrast, object database (ODB) systems excel at capturing and querying general, complex data structures. These systems offer semantically rich data models and query languages that include constructs such as classes, inheritance, complex associations between classes, and path expressions. However, ODB systems do not support aggregate queries well. For example, the complex data structures tend to make it hard to formulate correct queries that aggregate the data in the ODB. Also, ODB systems are optimized to perform more general types of queries, so the performance for aggregate queries is usually not satisfactory.

Federated database systems [22] support the *logical* integration of autonomous database systems, without affecting the functioning of the individual autonomous database systems. When integrating data from databases based on different data models, the traditional approach has been to map all data into one common data model and federate the (logically) transformed data rather than the original data. In this paper, we adopt an alternative approach that combines data from summary databases (SDBs) and object databases using a federated database approach¹, where data is handled using the most appropriate data model and database technology: SDB systems for summary data and ODB systems for complex, general data. No attempt is made to “shoehorn” the data into one common format, which is unlikely to fit all the data.

Our focus is on enabling OLAP-style queries over *existing* SDBs to also include data from *existing, external* ODBs, without jeopardizing the benefits of OLAP queries and without having to integrate the data physically. As a first step in demonstrating the capabilities of the system, a prototypical, user-oriented query language for SDBs, termed SumQL, is defined. The concept of a *link*, which enables the connection of SDBs to ODBs in a general and flexible manner, is then integrated into SumQL along with object features, yielding an extended language, termed SumQL++.

With this language as a vehicle, it is shown how the system enables the use of path expressions for referencing data in SDBs in *selection criteria*. Queries over SDBs may return ODB data along

¹Although the paper’s contributions are applicable to almost all current OLAP systems, we use the term SDB instead of OLAP DB to emphasize the focus on aggregate queries over summary data.

with the aggregate results, i.e., the result of an OLAP query may be *decorated* with object data. Finally, SDB data may be grouped based on ODB data. The approach can easily be applied to queries over external relational databases that allow path expressions in queries, e.g., as proposed in the SQL:1999 standard [11].

A prototype has been built [8] that supports the execution of SumQL++ queries over a federation of SDBs and ODBs.

The most related previous work is a system based on the nD -SQL language [6]. This system enables the querying of a federation of solely relational data sources, which are treated symmetrically, using nD -SQL. In contrast, we extend OLAP-style queries on an identified SDB to object databases with related data. Further, nD -SQL supports neither dimension hierarchies nor the aggregation semantics that enable safe aggregation and shield the users from incorrect results. Other existing middleware offerings such as DataJoiner [10], Cohera [5], and Oracle Gateways [17] exhibit the same limitations, which renders the formulation of distributed OLAP queries cumbersome and error prone in comparison to this paper’s proposal.

More specifically, we believe this paper to be the first to consider the integrated querying of data from independent summary and object databases without prior physical integration, with the objective of giving OLAP users enhanced, aggregation-safe query capabilities. Surveys of OLAP data models and languages [18, 23, 24] indicate that this issue has not been addressed previously. To our knowledge, the paper is also the first to demonstrate a “multi-paradigm” (or “multi-model”) federation [1, 9], where one of the data models is a summary data model.

The remainder of the paper is structured as follows. Section 2 motivates the paper’s contribution and presents a case-study. Section 3 introduces prototypical data models and query languages for the SDBs and ODBs to be federated. Section 4 then describes the notion of a link that connects SDBs to ODBs. Section 5 proceeds to describe the federated data model, which incorporates links, and the extended SumQL query language, which enables queries to access information in both SDBs and ODBs. Section 6 describes the prototype implementation of the system that implements the concepts and techniques presented. The last section summarizes and offers research directions.

2. MOTIVATION

In this section, we discuss why it is a good idea to federate existing summary and object databases and present a real-world case study that is used for illustration throughout the paper.

2.1 Reasons for Federation

Many reasons exist for federating *existing* SDBs and ODBs, as opposed to physically integrating these. The generic arguments for federation include leveraging existing technology, accessing the most current information, and allowing the autonomous existence of the systems being federated. These arguments also apply in this case, so we concentrate on the advantages specific to summary and object databases.

First, in many situations, SDBs only contain summary data and do not contain the base data from which the summary data is derived. For example, summary databases provided by the Ministry of Health do not permit access to base data, because the base data is unavailable or considered too sensitive for general disclosure, e.g., diagnosis information. The same situation arises in census databases, where only high-level information is disclosed publicly. Such databases are controlled by organizations that allow access to the summary data, but do not allow them to be extended with additional data. In such cases, federation with external databases is the only option.

Second, in the federated approach, an SDB needs not contain all objects, attributes, and relationships in the base database, but only the elements relevant to summary querying. This is attractive, as capturing all information in the SDB unnecessarily complicates the model of the SDB system. Indeed, most OLAP systems that implement SDBs do not have the necessary facilities to support object modeling concepts, e.g., category inheritance [13], to support this extra information. The federated approach allows the SDB to remain simple, while still allowing access to relevant external object databases.

Third, OLAP-type systems provide *better performance* of summary queries than do general-purpose DBMSs. The former type of system typically employs specialized, performance enhancing techniques, such as multidimensional storage and pre-aggregation. If the OLAP database is large, this performance gain can often outweigh the performance loss due to the fact that the data is not physically integrated, meaning that a federated system can have comparable (or even better) performance without the limitations incurred by physical integration.

Fourth, it is *easier to formulate summary queries* in an SDB system than in a general (relational or object) DBMS. This is because an SDB query language is designed exclusively for expressing summary queries over categories, taking advantage of, e.g., the automatic aggregation implied by the summary database semantics. Even when extending an SDB language to access object data (as we do in Section 5), it is easier to pose summary queries in the extended language than in a general database query language such as OQL or SQL.

Fifth, an SDB system can be designed to support the formulation of summary queries that return *correct, or meaningful, query results*. When building an SDB, the data may be shaped in order to satisfy summarizability conditions [14]. Briefly, a summary query satisfies summarizability conditions if the query result is correct w.r.t. the real world. For example, summarizing the populations over cities to get summaries for states will produce incorrect results if the populations in towns and farms outside cities are not accounted for. As another example, if some patients have several diseases, and we summarize over all diseases to get the total number of patients, we obtain a wrong result, as some patients are counted more than once. We may enrich an SDB system with information that enables the system to ensure correctness. In a general-purpose DBMS, no mechanisms for ensuring correct summary results are available. Using the federated approach permits the continued support of summary semantics while linking to external object databases.

Finally, the federated approach offers additional *flexibility* when query requirements change. SDBs may be huge, and therefore rebuilding them may be time consuming. Restructuring an SDB, e.g., adding new types of information, may require a total or partial rebuild of the database. In contrast, a new link can be added in a matter of minutes, yielding much faster access to newly acquired information.

The above reasoning suggests that in many cases, it is advantageous to logically federate existing OLAP and object databases instead of forcing physical integration.

2.2 Case Study

The case study concerns data in three different databases, each managed by a separate organization. Each database serves a different purpose, but the databases contain related data. A graphical illustration of the databases is seen in Figure 1.

The databases are modeled using the Unified Modeling Language (UML) [21]. Compound boxes denote classes. The class name is in boldface in the top part of the box, while class attributes

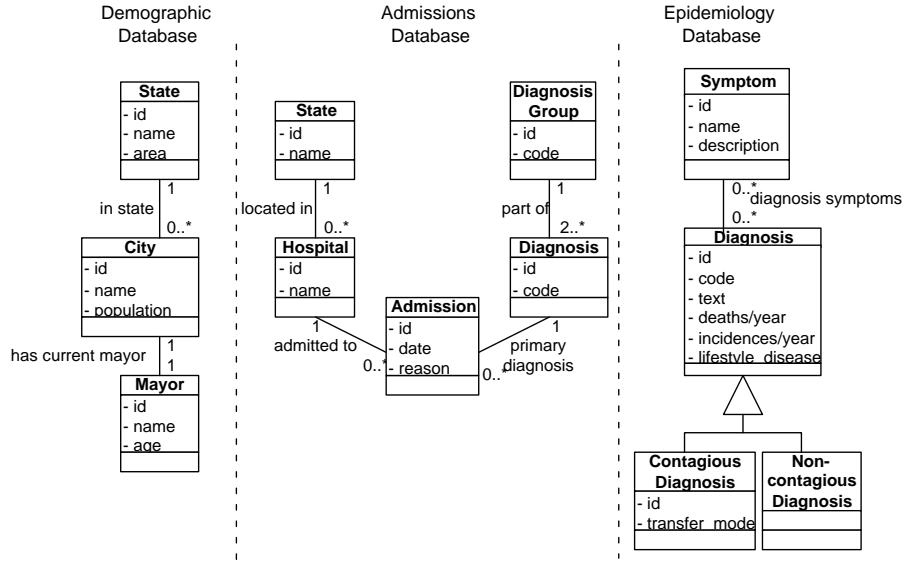


Figure 1: UML Schema of Case Study

are listed in the middle part. Associations, i.e., relationships, between classes are represented by lines tagged with an association name. The cardinality of an association is shown by the numbers at the ends of the association line. Either a single cardinality or a range of cardinalities are specified. A “*” denotes any natural number.

The *demographic database* is maintained by the Department of the Interior and offers central access to demographic data for all cities and states in the country. Data is collected for *states*, for which name and area are stored, and for *cities*, for which name and population are recorded. The database also contains information about the current *mayor* of a city. There are zero or more cities in each state, and each city has exactly one current mayor.

Next, the *admissions database* is maintained by the Department of Health and provides an overview of the admissions patterns for all hospitals nationwide. For an *admission*, the date of admission and the reason for admission, e.g., accident, are recorded. Additionally, we record which *hospital* the patient is admitted to and the primary *diagnosis* that caused the admission, e.g., Insulin Dependent Diabetes. For hospitals, the name and the *state* where the hospital is located are recorded. For diagnoses, we record an alphanumeric code, determined by a standard classification of diseases, e.g., the World Health Organization’s International Classification of Diseases (ICD-10) [25]. The classification also determines how the diagnoses are grouped into *diagnosis groups*, e.g., Diabetes. Diagnosis groups consist of at least 2 related diagnoses and a diagnosis belongs to exactly one diagnosis group. For diagnosis groups, we record an alphanumeric code, determined by the classification.

The last database is an *epidemiology database* maintained by a medical school for research purposes. Data are collected from hospitals, practicing physicians, and insurance companies to obtain a rich overview of the occurrence of diseases. The database is organized around the *diagnoses* in the standard disease classification also used in the admissions database, but more information is recorded. In addition to the alphanumeric code and an additional descriptive text, the database also records the number of incidences per year, the number of deaths per year, and whether the disease is dependent on the lifestyle of the patient. The Diagnosis class has two subclasses, *Contagious Diagnosis* and *Non-contagious Diagnosis*. For contagious diagnoses, we additionally record the mode

of transfer of the disease, e.g., by air. The *symptoms* of diseases are also recorded. For symptoms, we record a name and a description of the symptom, e.g., Cough or Fever.

The three databases were built and are used separately, which explains the differences in their information contents. We want to use them together, to exploit information from the demographic and epidemiology databases in queries against the admissions database. We consider the Admissions Database to be the SDB, and the other two DBs to be the ODBs that we wish to link to in a federation.

3. FEDERATION DATA MODELS AND QUERY LANGUAGES

This section defines a prototypical multidimensional data model and query language used for the SDB component in the federation; and it briefly presents the data model and query language of the federation’s ODB components.

The multidimensional model precisely and concisely captures core multidimensional concepts such as categories, dimensions, and automatic aggregation. The model and query language are equivalent in expressive power to previous approaches such as the ones proposed by Cabbibo et al. [2] and Jagadish et al. [12]. The ODB data model and query language is the ODMG data model and OQL query language.

3.1 Summary Data Model

The model has constructs for defining the *schema*, the *instances*, and the *aggregation properties*.

An *n-dimensional fact schema* is a two-tuple $\mathcal{S} = (\mathcal{F}, \mathcal{D})$, where \mathcal{F} is a *fact type* and $\mathcal{D} = \{\mathcal{T}_i, i = 1, \dots, n\}$ is its corresponding *dimension types*. A fact type is a *name* describing the type of the facts considered.

EXAMPLE 1. In the case study we will have *Admissions* as the fact type, and *Diagnosis*, *Place*, *Reason*, and *Time* as the dimension types.

A dimension type \mathcal{T} is a four-tuple $(\mathcal{C}, \leq_{\mathcal{T}}, \top_{\mathcal{T}}, \perp_{\mathcal{T}})$, where $\mathcal{C} = \{\mathcal{C}_j, j = 1, \dots, k\}$ are the *category types* of \mathcal{T} , $\leq_{\mathcal{T}}$ is a partial order on the \mathcal{C}_j ’s, with $\top_{\mathcal{T}} \in \mathcal{C}$ and $\perp_{\mathcal{T}} \in \mathcal{C}$ being the top and bottom element of the ordering, respectively. The intuition is

that one category type is “greater than” another category type if each member of the former’s extension logically contains several members of the latter’s extension, i.e., they have a larger element size. The top element of the ordering corresponds to the largest possible element size, that is, there is only one element in its extension, logically containing all other elements. We say that \mathcal{C}_j is a category type of \mathcal{T} , written $\mathcal{C}_j \in \mathcal{T}$, if $\mathcal{C}_j \in \mathcal{C}$. We assume a function $Anc : \mathcal{C} \mapsto 2^{\mathcal{C}}$ that gives the set of immediate ancestors of a category type \mathcal{C}_j .

EXAMPLE 2. Diagnoses are contained in Diagnosis Groups. Thus, the *Diagnosis* dimension type has the following order on its category types: $\perp_{Diagnosis} = Diagnosis < Diagnosis\ Group < \top_{Diagnosis}$. Thus, $Anc(Diagnosis) = \{Diagnosis\ Group\}$. Other examples of category types are *Day*, *Month*, and *Year*. Figure 2, to be discussed in detail in Example 5, illustrates the dimension types of the case study.

A category \mathcal{C}_j of type \mathcal{C}_j is a set of *dimension values* e . A *dimension* D of type $\mathcal{T} = (\{\mathcal{C}_j\}, \leq_{\mathcal{T}}, \top_{\mathcal{T}}, \perp_{\mathcal{T}})$ is a two-tuple $D = (C, \leq)$, where $C = \{\mathcal{C}_j\}$ is a set of categories \mathcal{C}_j such that $Type(\mathcal{C}_j) = \mathcal{C}_j$ and \leq is a partial order on $\cup_j \mathcal{C}_j$, the union of all dimension values in the individual categories.

The partial order is defined as follows. Given two values e_1, e_2 then $e_1 \leq e_2$ if e_1 is logically contained in e_2 , i.e., e_2 can be considered as a set containing e_1 . We say that \mathcal{C}_j is a category of D , written $\mathcal{C}_j \in D$, if $\mathcal{C}_j \in C$. For a dimension value e , we say that e is a dimensional value of D , written $e \in D$, if $e \in \cup_j \mathcal{C}_j$.

We assume a partial order \leq_C on the categories in a dimension, as given by the partial order $\leq_{\mathcal{T}}$ on the corresponding category types. The category \perp_D in dimension D contains the values with the smallest value size. The category with the largest value size, \top_D , contains exactly one value, denoted \top . For all values e of the categories of D , $e \leq \top$. Value \top is similar to the *ALL* construct of Gray et al. [7]. We assume that the partial order on category types and the function Anc work directly on categories, with the order given by the corresponding category types.

EXAMPLE 3. The *Diagnosis* dimension has the following categories, named by their type. *Diagnosis*, *Diagnosis Group*, and $\top_{Diagnosis}$ which contain the actual dimensions values, e.g., *Diagnosis* could contain the value “52” which represents the diagnosis with the code “N12” and the text “Pneumonia.” The *Diagnosis Group* category contains the value “61” (Infections) and the partial order specifies that $52 \leq 61$ to indicate that the Pneumonia diagnosis is part of the Infections Diagnosis Group.

ID	Code	GroupID
50	E10	60
51	E11	60
52	N12	61

Diagnosis Table

ID	Code	Text
60	E1	Diabetes
61	N1	Infections

DiagnosisGroup Table

Table 1: Diagnoses and Diagnosis Groups

Let C_1, \dots, C_n be categories and T a domain that includes the special value *null*. A *measure* for these categories and this domain is a function $M : C_1 \times \dots \times C_n \mapsto T$. We say that M is measure for the set of dimensions $D = \{D_1, \dots, D_n\}$, if M is a measure for the categories $\perp_{D_1}, \dots, \perp_{D_n}$. Every measure M has associated with it a *default aggregate function* $f_M : T \times T \mapsto T$. The default aggregate function must be distributive. The null value is used to indicate that no data exists for a particular combination of category values. As with SQL, the aggregate functions ignore null values. The intuition behind a measure is that it “measures” what has happened at one

particular point in the multidimensional space, e.g., the total sales for a particular product and store.

EXAMPLE 4. In the case study we have one measure, *TotalAdmissions*, which is the total number of admissions by *Diagnosis*, *Place*, *Time*, and *Reason*. The default aggregate function is SUM.

For different kinds of measures, different aggregate functions are meaningful. For example, it is meaningful to sum up numbers of admissions; and because this data has a metric on it, it is also meaningful to compute the average, minimum, and maximum values. In contrast, in at least some situations, it may not be meaningful to compute the sum (over time) of measures such as the number of patients hospitalized (because each patient would be counted several times in the result), but it remains meaningful to compute the average, minimum, and maximum values. Next, it makes little sense to compute these aggregate values on data that do not have any ordering defined on them, such as diagnoses. Here, the only meaningful aggregation is the count of occurrences. Whether or not an aggregate function is meaningful also depends on the dimensions being aggregated over. For example, patient counts may be summed over the *Place* dimension, but not over the *Time* dimension. For additional discussion of these issues, we refer to reference [14].

An n -dimensional *summary database* (SDB) is a 3-tuple $S = (S, D, M)$, where S is the schema, $D = \{D_1, \dots, D_n\}$ is a set of dimensions, and $M = \{M_1, \dots, M_k\}$ is a set of measures for the categories $\perp_{D_1}, \dots, \perp_{D_n}$.

EXAMPLE 5. The case study has a 4-dimensional SDB with *Diagnosis*, *Place*, *Reason*, and *Time* as dimensions. There is one measure, *TotalAdmissions*, as described above. The SDB is illustrated in Figure 2.

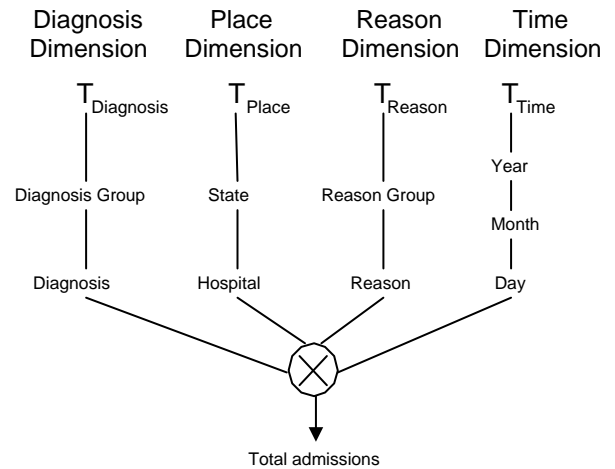


Figure 2: Summary Model for the Admissions Database

3.2 The Summary Query Language

The query language of the SDB components is termed SumQL and is meant to make it easy for the user to pose aggregate queries over SDBs by taking advantage of the semantic properties of the summary data model. We have chosen to define a separate summary language rather than attempting to augment the object query language, OQL, for querying SDBs because we wish to refer explicitly to the special data structures in SDBs. Using OQL, or some variant thereof, for querying SDBs would mean that we would have to introduce OLAP constructs such as measures, dimensions, and

hierarchies, which would conflict with the generality of the object model.

SumQL is reminiscent of SQL, but includes constructs that reflect SDB concepts such as measures, dimensions with hierarchically organized categories, and automatic aggregation, thus supporting naturally the expression of aggregate queries over SDBs. With SumQL we can concisely define the extensions for referencing object data.

The general format of a SumQL query is given below. The symbol “+” indicates one or more occurrences and square brackets denote optional parts. The formal syntax and semantics of SumQL are given elsewhere [19].

```
SumQL query ::= SELECT measure+
                INTO summary_database
                BY_CATEGORY category+
                FROM summary_database
                [ WHERE predicate_clause ]
```

The SELECT clause contains a list of measures for which a result is to be computed. Unlike in SQL, aggregate functions such as SUM need not be specified; rather, the default aggregation function specified in the schema is automatically applied to aggregate the data. An INTO clause follows that specifies the SDB into which the query result is stored. Thus, SumQL queries take SDBs as arguments and return an SDB.

The BY_CATEGORY clause specifies the aggregation level at which the measures are to be computed. For each dimension *not* mentioned in this clause, the measures are aggregated over the whole dimension, which is the same behavior as in SQL’s GROUP BY clause.

The FROM clause specifies the SDB from which to aggregate. The optional WHERE clause specifies predicates that are applied to the SDB before aggregation occurs. The predicates can include standard constructs such as comparison, set, and string operators. These constructs are equivalent or similar to those found in SQL and OQL [3].

EXAMPLE 6. The following SumQL statement computes the “Total Admissions” measure from the “admissions” SDB, aggregated to the level of Year and State, for the years after 1997. The resulting SDB is called “testdb.”

```
SELECT TotalAdmissions INTO testdb
BY_CATEGORY year, state
FROM admissions
WHERE year > 1997
```

3.3 The Object Model and Query Language

This section briefly reviews the object data model and query language used by the ODB components of the federation. We use the Object Data Management Group’s object data model, ODMG 2.0 [3], and its associated query-language, OQL. The ODMG data model includes constructs such as object class definitions, attributes, object identifiers, set-valued attributes, reference attributes, tuple attributes, inverse attributes, inheritance structures, and object class unions. An in-depth coverage of the ODMG data model and the OQL language may be found in the literature [3].

The OQL query language has constructs such as path expressions and class selectors. Path expressions are used to navigate through *reference attributes* to other classes using dot-notation, while class selectors restrict queries to operate only on a certain subclass. In Section 5 we will use these constructs for extending SumQL to allow its queries to refer to external ODB structures.

EXAMPLE 7. The following query uses a path expression to select the city name only for cities where the current mayor is more than 40 years old. The path expressions navigates from cities to mayors via reference attribute “current_mayor.”

```
SELECT C.name FROM C IN City
WHERE C.current_mayor.age > 40
```

EXAMPLE 8. The next query navigates from symptoms to the diagnoses that exhibit those symptoms using a path expression. It then applies a class selector (the square brackets) to select the attribute “transfer_mode” of the Diagnosis subclass “Contagious Diagnosis.” Thus, only transfer modes for contagious diagnoses with the the symptom “Cough” are returned.

```
SELECT S.diagnoses[ContagiousDiagnosis]transfer_mode
FROM S IN Symptom
WHERE S.name = “Cough”
```

4. LINKING DATABASES

We proceed to define the links that are used to connect SDBs and ODBs. As mentioned in the introduction, we use explicit links to connect the databases, rather than relying solely on implicit knowledge of relationships among the databases when formulating queries.

Explicit links are preferable for several reasons. First, even if the data in the SDB is derived from source data in an ODB, the complete mapping may be unknown because of substitutions for missing data and other types of data cleansing, interpolation, etc. Second, explicit links are needed when linking an SDB to an unrelated ODB, i.e., an ODB other than the base data from which the SDB was extracted.

Links are considered as separate from the databases being federated, to better capture their special semantics and to aid the optimization of queries involving links. However, links can be physically implemented as part of these databases.

Formally, a *link* L from a category C to an object class O is a relation $L = \{(c, o)\}$, where $c \in C$ and $o \in O$. All links have a *name* to distinguish them. This is because each category and even pair of category and object class may have several links.

Links may be specified in several ways. An *equivalence link* is specified by a predicate $C = O.a$, where C is a category, O is an object class, and a is an attribute of O that uniquely identifies instances of O , i.e., a is a candidate key for O in relational database terms. Equivalence links occur when a category in the SDB represents the same real-world entities as does some object class in an ODB. An *attribute link* is specified by the same type of predicate, the only exception being that a does not uniquely identify instances of O . An *enumerated link* is given by an a link relation $L = \{(c, o)\}$, where pairs of dimension values in C and object ids from class O are explicitly enumerated. Therefore multiple dimension values may be assigned to the same object. Enumerated links are typically used for linking a category in an SDB and an object class that do not represent the same real-world entities.

EXAMPLE 9. In our case study, we can specify an equivalence link between the Diagnosis category in the Admissions SDB and the Diagnosis Class in the Epidemiology ODB by the predicate “Diagnosis = Diagnosis.Code,” as the values of the Diagnosis category are the codes of the diagnoses. In subsequent examples, we term this link “diag_link.”

EXAMPLE 10. An enumerated link from the Hospital category in the SDB to the City class in the Demographic ODB may be specified by explicitly assigning hospitals to cities based on where the

hospitals are located. An example of instances of such a link relation is $L = \{(\text{“Alta Bates”}, \text{“Berkeley”}), (\text{“Portland General Hospital”}, \text{“Portland”}), (\text{“Portland Kaiser”}, \text{“Portland”})\}$. We will use the name “city_link” for this link.

The *cardinality* of a link is important, as the cardinality may affect summarizability. This cardinality is one of $[1 - 1], [1 - n], [n - 1]$, or $[n - n]$. For some link properties, only the cardinality of the object side of a link is interesting. As a short-hand notation, we say that the cardinality of a link is $[-1]$ if the cardinality is $[1 - 1]$ or $[n - 1]$. Similarly, the cardinality of a link is $[-n]$ if the cardinality is $[1 - n]$ or $[n - n]$. For example, the cardinality of link “diag_link” is $[1 - 1]$ and the cardinality of “city_link” is $[n - 1]$.

It is also necessary to capture whether some dimension values or objects do not participate in a link. For that purpose, we define that a link $L = \{(c, o)\}$ from category C to object class O *covers* C if $C = \pi_C(L)$. Similarly, L covers O if $O_i = \pi_O(L)$, where O_i is the set of object ids for O . If L covers both C and O , L is *complete*; otherwise, L is *incomplete*. For example, the “diag_link” link is complete, while the “city_link” link covers the Hospital category, but not the City class.

The next section explores the effect of these link properties on the semantics of queries. We shall see that incomplete links and $[-n]$ links, which are analogous to non-summarizable hierarchies, require special attention. Interestingly, an attribute link always has a link cardinality that is $[-n]$, while an equivalence link always has a $[1 - 1]$ cardinality.

5. THE FEDERATED DATA MODEL AND QUERY LANGUAGE

Having described the data models and query languages of the SDB and ODB components to be federated, as well as a minimal mechanism for linking SDBs and ODBs, the next step is to provide language facilities that enable OLAP-type queries across the entire federation. Specifically, we extend SumQL.

The federation approach presented here has the distinguishing feature that it uses the aggregation semantics of the data to provide *aggregation-safe* queries, i.e., queries that do not return results that are incorrect or meaningless to the user. This section describes how the previously defined concepts of aggregation types, summarizability, link cardinality, and link coverage combine to provide aggregation-safety for queries.

5.1 The Federated Data Model

The federation consists of a collection of independent components, supplemented with additional information and components that enable the functioning of the federation. Specifically, the federation consists of an SDB, a number of ODBs, and links that interrelate information in the different databases. Formally, a federation F of an SDB S and a set of ODBs $O = \{O_1, \dots, O_n\}$ is a triple $F = (S, O, L)$, where $L = \{L_1, \dots, L_m\}$ is a set of links from categories in the dimensions of S to classes in O_1, \dots, O_n .

We assume a single SDB only. Permitting multiple SDBs introduces additional challenges not covered here, e.g., the matching of categories and dimensions. The case of a single SDB is very useful, as typical queries to a federation naturally centers around one SDB: Typical queries concern SDB measures, grouped by SDB categories, and involving selection criteria relating to data from the ODBs; or queries retrieve ODB data along SDB data; or, in some cases, actually *group* SDB data by categorical ODB data.

Rather than requiring that the SDB and ODB data comply with one common data model, the federation adopts a *multi-paradigm* approach [1, 9], where the data remain in their original data models.

Allowing multiple data models (or paradigms) to co-exist in the federation enables us to exploit the strengths of the different data models and query languages when managing and querying the data. In particular, the availability of multiple paradigms allows a problem solution to take advantage of the fact that certain subsets of a problem are often well suited for one paradigm, while other problem subsets are better suited for other paradigms.

Just as the arguments to queries are federated databases, the results are federated databases, i.e., query results may have SDB, ODB, and link components. This closure property mirrors those of the well-known relational, object, and multidimensional data models and query languages, and it permits the result of one query to be used in a subsequent query. We allow the sets of ODBs and links, O and L , to be empty. Thus, an SDB is in itself a federation.

5.2 The SumQL++ Language

Because the objective is to allow more powerful OLAP queries over SDBs by letting the queries refer to data from ODBs, we take SumQL at the outset and extend this language. The new, extended language is termed “SumQL++” as it introduces object-oriented concepts into its predecessor, akin to the C++ successor to the C programming language.

We are interested in extending the typical OLAP queries that select a set of measures from an SDB, grouped by a set of categories. Three extensions to SumQL are useful in this respect. First, we introduce path expressions in selection predicates, in order to integrate ODB data. Second, we introduce so-called *decorations* [7] of SumQL results, which enable ODB data to be returned along with the SumQL result. Third, SumQL is extended to enable SDB data to be grouped by data belonging to ODBs, i.e., attributes of object classes, rather than just the built-in SDB categories.

5.2.1 Extended Selection Predicates

The first extension of SumQL is to allow selection predicates that reference ODB data. This is achieved by extending the ‘category’ in the selection predicate into a *category expression*. The basic idea is to allow the use of standard OQL *path expressions*, as described in Section 3.3, in the category expressions in the selection predicates, using standard dot-notation for path expressions.

The link that is used to reach the ODB is included in the category expression. A category expression always starts with an SDB category, and is followed by an optional part consisting of the link name and a path expression. Inside the path expressions, *class selectors* may occur that restrict predicates to work on selected (sub)classes. The syntax is shown below. The square brackets in single quotes in the “class_connector” rule denote (sub)class selection and are part of the language being defined. Nonterminals not defined below are strings.

```

cat_exp      ::= category [ . link obj_path attr ]
obj_path     ::= class_conn | path_list
class_connector ::= . | '[' class ']'
path_list    ::= class_conn elem | path_list elem
elem         ::= reference_attribute class_conn

```

EXAMPLE 11. We want to use the Epidemiology ODB to get the total admissions by year for only the diagnoses for which cough is a symptom. We use the “diag_link” link to do so in the following the SumQL++ statement.

```

SELECT TotalAdmissions INTO testdb
BY_CATEGORY Year FROM Admissions
WHERE Diagnosis.diag_link.symptoms.name = “Cough”

```

EXAMPLE 12. We use a class selector in the Epidemiology ODB to get the total admissions by year for only contagious diag-

noses with the transfer mode “Air,” with the following SumQL++ statement.

```
SELECT TotalAdmissions INTO testdb
BY_CATEGORY Year FROM Admissions
WHERE Diagnosis.diag_link[Contagious-
Diagnosis]transfer_mode = “Air”
```

Informally, the cardinality of a category expression is the combination of the cardinalities that we encounter as we go through the link and the subsequent (possibly set-valued) reference-attributes, i.e., going through a $[-1]$ relationship in a link or a reference attribute does not change the running cardinality, but a $[-n]$ relationship causes the total cardinality to be $[-n]$.

Given a category expression $E = C.L.P.O.a$, where C is a category, L is a link, P is an object path, O is an object class, and a is an attribute, we informally define L' to be the whole of C seen as a link from C to O . Following the definitions given for links, we say that E covers O , does not cover O , covers C , does not cover C , is complete, and is incomplete, if L' covers O , does not cover O , covers C , does not cover C , is complete, or is incomplete, respectively. Above, O is the object class that a is an attribute of, i.e., the last object class reached in the category expression. We say that O is the final class of E . C is the category in the beginning of E . We say that C is the starting category of E .

EXAMPLE 13. The cardinality of the category expression “Hospital.city_link.locatedin.name” is $[n - 1]$ as we only traverse $[n - 1]$ relationships and the state name is a key attribute. The cardinality of the category expression “Diagnosis.diag_link.symptoms.name” is $[n - n]$ because the “symptoms” reference attribute is set-valued.

The cardinality and covering properties of a category expression affect the meaning of a SumQL++ statement. If the cardinality is $[-1]$, the predicate will only reference one attribute value and the meaning is clear. However, if the cardinality is $[-n]$, the predicate will reference more than one attribute value, leading to several possible semantics for the query.

For example, the category predicate “Diagnosis.diag_link.symptoms.name = “Cough” in Example 11 has a $[-n]$ cardinality. One possible interpretation of this is that *all* the referenced attribute values must match the predicate, e.g., that *all* symptoms must have name “Cough.” Another interpretation is that *at least one* attribute value must satisfy the predicate, e.g., that at least one symptom has name “Cough.” Because this is the interpretation chosen in the OQL language, we adopt this interpretation.

Similar problems may arise when a category expression E does not cover its starting category C , because L' then will be undefined for the uncovered dimension values of C . However, if we adopt our previous interpretation, that *at least one attribute value* must match the predicate, the meaning is well-defined. The values in C not covered by E will then be excluded from the selection. There are no problems if E does not cover its final class O , as L' will be defined for all the instances of O referenced by E .

5.2.2 Decorating the Query Result

It is often desirable to display additional descriptive information along with the result of an SDB query. This is commonly referred to as *decorating* the result of the query [7]. For example, when asking for the number of admissions by hospital, it may be desirable to display the name of the city and the name of the city’s mayor along with the hospital name.

This can be achieved by extending the SumQL with features for decorating the result. One possibility is to allow category expressions with path expressions in the SELECT clause, but this makes

it unclear which parts of the SELECT clause refer to measures and which parts refer to decorations. Instead, we extend SumQL with an optional “WITH” clause. The extended syntax is shown below.

```
SumQL++ query ::= SELECT measure+
INTO summary_database
BY_CATEGORY category+
[ WITH expression+ ]
FROM summary_database
[ WHERE predicate_clause ]
```

EXAMPLE 14. Using this extension, we select the number of admissions by hospital, decorated with the names of the city and its mayor.

```
SELECT TotalAdmissions INTO testdb FROM Admissions
BY_CATEGORY Hospital WITH Hospital.city_link.name,
Hospital.city_link.current_mayor.name FROM Admissions
```

It only makes sense to decorate the result with data that is correlated to the original query result, so the categories referenced in the WITH clause *must* be part of the BY_CATEGORY clause. The decoration data is returned in the ODB and link parts of the federation and is not integrated into the result SDB. This loose coupling of decoration data and SDB data is essential in avoiding semantic problems.

5.2.3 Grouping By Object Class Attributes

The last extension allows the measures of an SDB to be grouped by attribute values in ODBs, thus enabling aggregation over hierarchies outside the SDB. Specifically, *category expressions* instead of just categories are allowed in the BY_CATEGORY clause. The syntax of the extension is given below. The only difference from the previous syntax is that the BY_CATEGORY clause is now a list of category expressions rather than just categories. Recall that a category expression is either a category or a category followed by a link, an object path, and an attribute.

```
SumQL++ query ::= SELECT measure+
INTO summary_database
BY_CATEGORY expression+
[ WITH expression+ ]
FROM summary_database
[ WHERE predicate_clause ]
```

EXAMPLE 15. The number of admissions grouped by symptoms may be retrieved as follows.

```
SELECT TotalAdmissions INTO testdb
BY_CATEGORY Diagnosis.diag_link.symptoms.name
FROM Admissions
```

This type of SumQL++ queries will return SDBs where one new dimension is added for each category expression in the BY_CATEGORY clause, thereby reflecting the hierarchy specified by the category expression, and aggregation will occur over these new dimensions.

Although the extensions to SumQL were described separately above, they can be used together in the same SumQL++ statement. Assuming an SumQL++ statement that contains all three extensions, query evaluation proceeds as follows. First, the rules for handling grouping by object attributes are used, producing a statement without object attribute grouping. This statement is then processed using the rules for the WITH clause described above, resulting in a statement without a WITH clause, which can then be evaluated using the rules for extended selection predicates. The statement produced by the extended predicate rules is a pure SumQL statement, which may be evaluated following standard SumQL semantics.

6. IMPLEMENTATION OVERVIEW

This section briefly describes a prototype federated system implementing SumQL++ queries. The overall architecture of the system is seen in Figure 3. The parts of the system handling object and link data are based on the commercially available OPM tools [15] that implement the Object Data Management Group's (ODMG) object data model [3] and the Object Query Language (OQL) [3] on top of a relational DBMS, in this case the ORACLE8 RDBMS. In-depth descriptions of the OPM toolset exist in the literature [4]. The OLAP part of the system is based on Microsoft's SQL Server OLAP Services using the Multi-Dimensional eXpressions (MDX) [16] query language. The graphical user interface (GUI) is implemented as Java classes running in a standard Web browser for optimal flexibility. A description of the user interface may be found elsewhere [8].

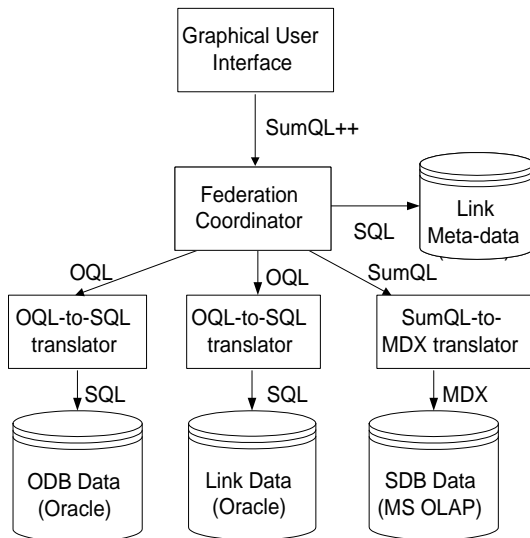


Figure 3: Architecture of the Federated System

The system has six major components: the GUI, the ODB systems, the link DB system, the SDB system, the federation coordinator, and the metadata database. The ODB, link DB, and SDB components are treated as independent units by the federation system; only their published interfaces are used, and no assumptions about their internal workings are made. The link component stores enumerated links and is placed in an independent "link" DB, as it cannot generally be assumed that these links may be stored in some ODB component. Should this be possible, we can choose to do so, e.g., to obtain better performance. The operation of the prototype is entirely based on federation metadata specified in the metadata database. This allows for a very flexible system that may adapt quickly to changes. For example, if a new connection to an outside ODB is desired, appropriate links just need to be specified and stored as metadata, after which queries can start using the new ODB.

Queries are generated by the GUI. A Query is first sent to the federation coordinator, which parses the query. Based on the query's content, the system looks up the relevant metadata (link specifications, ODB names, etc.) in the metadata database and processes the query according to the metadata by issuing queries to the DB components. Example 16 below explains how a particular query is processed.

EXAMPLE 16. The query below selects the total admissions by diagnosis, state, and year, restricted to diagnoses with "Cough" as

a symptom and to years after 1997.

```

SELECT TotalAdmissions INTO testdb
BY_CATEGORY Diagnosis,State,Year FROM Admissions
WHERE (Diagnosis.diag_link.symptoms.name ="Cough")
AND (Year > 1997)
  
```

To process the query, the Federation Coordinator (FC) parses the query and identifies the link and ODB parts of the query. Based on the link name (diag_link), the FC looks up in the metadata which ODB, object class, and attribute the link is connected to and the type of the link, i.e., equivalence, attribute, or enumerated. For this query, the ODB is the "Epidemiology" DB, the class is "Diagnosis," the attribute is "code," and the link type is "equivalence." The object path to be followed is ".symptoms," and the final attribute is "name." Based on this, the FC forms the OQL query seen below.

```

SELECT code = @n1
FROM @n0 IN SUMDB:Diagnosis, @n1 IN @n0.code
WHERE @n0.symptoms.name ="Cough";
  
```

The OQL query is then executed against the Demographic ODB, giving as result the single diagnosis code "N12" (for example database instances, only parts of which are given in the paper). Based on the result of the OQL query, the FC now forms the SumQL query seen below, which is executed against the SDB component of the federation to obtain the final result. The reason for using the intermediate SumQL statements is to isolate the implementation of the OLAP data from the FC. As an alternative, we have also implemented a translator into SQL statements against a relational "star schema" design.

```

SELECT TotalAdmissions INTO testdb
BY_CATEGORY Diagnosis,State,Year FROM Admissions
WHERE (Diagnosis IN ( 'N12' ) AND Year > 1997)
  
```

The SumQL query is now translated into the MDX statement seen below and executed against the SDB managed by MS SQL Server OLAP Services.

```

SELECT [Measures].[TotalAdmissions] ON COLUMNS,
INTERSECT (CROSSJOIN (CROSSJOIN([Diagnosis].[N12],
[Place].[State].MEMBERS), [Time].[Year].MEMBERS),
CROSSJOIN(CROSSJOIN([Diagnosis].[Diagnosis].MEMBERS,
[Place].[State].MEMBERS), FILTER([Year].MEMBERS,
[Time].CURRENTMEMBER.NAME > "1997"))))
ON ROWS FROM Admissions
  
```

The example above illustrates the amount of work a user must go through without the aid of the user interface and the federated translation tools. In particular, it reveals the usefulness of the OLAP-object database links in generating the combined result. Also, the users are shielded from the verbosity of MDX (which is hidden from them).

7. CONCLUSION AND FUTURE WORK

Motivated by the increasingly widespread use of OLAP technology, we have presented the concepts and techniques underlying a prototype system that federates data in OLAP databases with data from outside object databases, without requiring physical integration of the data.

Summary data is best handled using OLAP technology, while complex detail-level data structures are best handled with object database technology. This enables the handling of the data using the most appropriate data model and technology, while still allowing queries to reference data across the different databases and data

models. No attempt is made to map data into one common data model, which would be sub-optimal for some of the data. To our knowledge, this is the first example of a “multi-model” federation that includes a dedicated summary data model. In contrast to earlier works, the approach presented here uses the aggregation semantics of data to guard against meaningless or incorrect queries.

As a vehicle for presenting the paper’s contributions, a high-level language for summary databases, SumQL, is introduced. This is then extended to support queries that reference data in separate object databases. The resulting language, SumQL++ embodies the concept of *links* that connect an SDB to ODBs in a general and flexible way, in addition to object-oriented concepts. SumQL++ permits *selection criteria* that reference data in the ODBs using path expressions, facilities for *decorating* the aggregate results of SDB queries with external object data, and the ability to *group* data in the SDB according to object data. Also covered is the extension of aggregate queries over SDBs to include data from ODBs. It is possible to use other languages such as SQL, OQL, and MDX in place of SumQL++ once these are enriched with the necessary SumQL++ constructs they lack.

Interesting research directions include extending the approach to handle federations with several SDBs, as well as the federation with XML databases, which offer less structure than object databases and thus may benefit even more from the enforcement of aggregation semantics by the federation. Another interesting direction is to consider the optimization of queries over the federation. For example, in some situations it may be advantageous to perform aggregation before selection, to take advantage of OLAP techniques such as pre-aggregation.

Acknowledgements

This research was supported in part by the Danish Academy of Technical Sciences, contract no. EF661, by the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC03-76SF00098, by the Danish Technical Research Council through grant 9700780, and by a grant from the Nykredit corporation.

8. REFERENCES

- [1] T. Barsalou and D. Gangopadhyay. M(DM): An Open Framework for Interoperation of Multimodel Multidatabase Systems. In *Proceedings of ICDE*, pp. 218–227, 1992.
- [2] L. Cabibbo and R. Torlone. Querying Multidimensional Databases. In *Proceedings of DBPL*, pp. 319–335, 1997.
- [3] R. G. G. Cattell et al. (editors). *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.
- [4] I. A. Chen and V. M. Markowitz. The Object-Protocol Model (OPM) Version 4.1. Lawrence Berkeley National Laboratory TR LBNL-32738 (revised), 1996.
- [5] Cohera Corporation. Cohera Data Federation System. <www.cohera.com/datasheets.html>. Current as of Sep 7, 2000.
- [6] F. Gingras, Laks V. S. Lakshmanan: nD-SQL: A Multi-Dimensional Language for Interoperability and OLAP. In *Proceedings of VLDB*, pp. 134–145, 1998.
- [7] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–54, 1997.
- [8] J. Gu, T. B. Pedersen, and A. Shoshani. OLAP++: Powerful and Easy-to-Use Federations of OLAP and Object Databases. In *Proceedings of VLDB, demo track*, 2000.
- [9] D. K. Hsiao and M. N. Kamel. The Multimodel, Multilingual Approach to Interoperability of Multidatabase Systems. In *Proceedings of RIDE*, 1991.
- [10] IBM Corporation. DB2 DataJoiner. <www-4.ibm.com/software/data/datajoiner/>. Current as of Sep 7, 2000.
- [11] International Standards Organization. *ISO/IEC 9075-1:1999 Information Technology — Database Language — SQL — Part 1 — Framework (SQL/Framework)*, ISO, 1999.
- [12] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can Hierarchies do for Data Warehouses? In *Proceedings of VLDB*, pp. 530–541, 1999.
- [13] W. Lehner. Modeling Large Scale OLAP Scenarios. In *Proceedings of EDBT*, pp. 153–167, 1998.
- [14] H. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Proceedings of SSDBM*, pp. 39–48, 1997.
- [15] V. M. Markowitz et al. OPM Home Page. <gizmo.lbl.gov/DM_TOOLS/DMTools.html>. Current as of Sep 7, 2000.
- [16] Microsoft Corporation. OLE DB for OLAP Version 1.0 Specification. Microsoft Technical Document, 1998.
- [17] Oracle Corporation. Oracle Gateways <www.oracle.com/gateways>. Current as of Sep 7, 2000.
- [18] T. B. Pedersen and C. S. Jensen. Multidimensional Data Modeling for Complex Data. In *Proceedings of ICDE*, pp. 336–345, 1999.
- [19] T. B. Pedersen, A. Shoshani, J. Gu, C. S. Jensen. Extending OLAP Querying to External Object Databases. Technical Report R-00-5002, Department of Computer Science, Aalborg University, 2000.
- [20] M. Rafanelli and A. Shoshani. STORM: A Statistical Object Representation Model. In *Proceedings of SSDBM*, pp. 14–29, 1990.
- [21] Rational Corporation. UML 1.1 Notation Guide. URL: <www.rational.com/uml/resources/documentation/notation/index.html>. Current as of Sep 7, 2000.
- [22] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *Computing Surveys*, 22(3):183-236, 1990.
- [23] E. Thomsen. *OLAP Solutions: Building Multidimensional Information Systems*. Wiley, 1997.
- [24] P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. In *SIGMOD Record*, 28(4):64–69, 1999.
- [25] World Health Organization. *International Classification of Diseases (ICD-10)*. Tenth Revision, 1992.