# Constructing an OLAP Cube from Distributed XML Data

Tapio Niemi[1]
Helsinki Institute of Physics
CERN Offices, CH-1211 Genève
Switzerland
tapio@cs.uta.fi

Marko Niinimäki
Helsinki Institute of Physics
CERN Offices, CH-1211 Genève
Switzerland
marko.niinimaki@cern.ch

Jyrki Nummenmaa
[1]Dept of Computer and Information Sciences
FIN-33014 University of Tampere
Finland
jyrki@cs.uta.fi

Peter Thanisch
IBM UK
St. Andrew Square, Edinburgh
Scotland
thanisch@uk.ibm.com

## ABSTRACT

On-Line Analytical Processing (OLAP) is a powerful method for analysing large data warehouse data. Typically, the data for an OLAP database is collected from a set of data repositories such as e.g. operational databases. This data set is often huge, and it may not be known in advance what data is required and when to perform the desired data analysis tasks. Sometimes it may happen that some parts of the data are only needed occasionally. Therefore, keeping the OLAP database constantly up-to-date is not only a highly demanding task but it also may be overkill in practice.

This suggests that in some applications it would be more feasible to form the OLAP cubes only when they are actually needed. We present such a system. As the data sources may well be heterogeneous, we propose an XML language for data collection. Our system also has a facility, where the user may pose a query against a "universal" OLAP cube using the MDX language. The query is analysed to determine which data is required for the desired OLAP cube.

## Categories and Subject Descriptors

H 2.4 [**Database Management**]: Systems
—*Distributed databases*;
H 3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

## General Terms

Design, Management

## Keywords

OLAP, XML, distributed data warehousing


*DOLAP'02*, November 8, 2002, McLean, Virginia, USA.


## 1. INTRODUCTION

The contents of OLAP databases are typically collected from other data repositories, such as operational databases. For a well-defined and targeted system, where the information needs are well known, it may be straightforward to collect the right data at the right time. However, there is more and more data generally available, and also the information needs develop. Consequently, it gets more and more difficult to anticipate the needs of OLAP users. This leads to a situation, where it is increasingly difficult to know in advance, what data is required and when for the desired data analysis tasks. Sometimes it may happen that some parts of the data set are only needed occasionally.

Because of the huge volumes of data, it is very hard and expensive to prepare in advance for a potentially wide selection of OLAP queries. It appears that collecting the right data on demand might be a better alternative for some applications. This way the data is also up-to-date, as it is collected when it is needed.

A further problem may well be that the data repositories or data warehouses involved in data collection are often heterogeneous, yet their information should be integrated in the OLAP database. XML appears to be a suitable solution for this problem. For example, relational data can be easily transformed to an XML [2] presentation and an XML sublanguage can be translated to other sublanguages using XSLT (Extensible Stylesheet Language Transformations). In a similar way, the OLAP data can be easily transformed into a form suitable for an OLAP server. This enables us to use different server products for data analysis, provided that they are able to read data in XML form.

We have developed a system for distributed OLAP data collection from heterogeneous data sources. The data is collected in XML form (although other forms of data collection may also be feasible in some cases). This means that the OLAP database can be tailored to suit the users' needs. Construction of a new cube is not supposed to happen as 'on-line' as answering OLAP queries. However, we believe that construction of a new cube enables the system to answer much faster to users' actual queries. This is possible since a smaller cube is more efficient to process, for example it can contain relatively more precalculated data than a

large cube.

Sometimes, the users may be able to directly express the structure of the desired OLAP database but this is not generally something that the users are prepared for. Therefore, our system has a facility whereby the user can pose MDX (MultiDimensional Expressions) [8] queries against a virtual "universal" OLAP cube schema, and the system will analyse these queries to find out the structure of the required OLAP database and what data needs to be collected. From this information, the system is able to perform the data collection to build the desired OLAP database. The data used in our examples and to test our prototype implementation contains several million entries of world trade data distributed in three different databases.

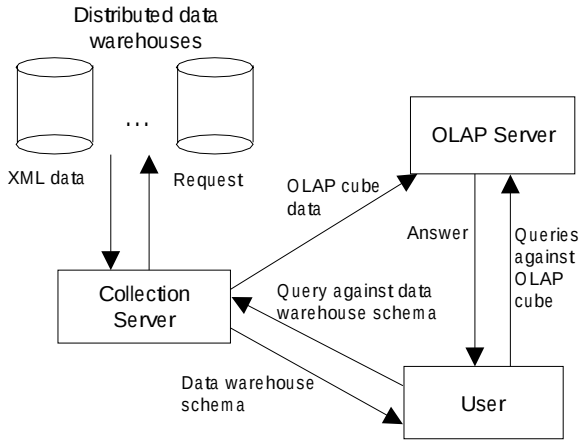


**Figure 1: The system architecture**

In Figure 1, the idea of the system is shown. In practice, making OLAP queries works as follows:

1. The virtual "universal" data warehouse schema representing all possible analysis data is shown to the user and the user poses a query using an MDX query based method [10].
2. The Collection Server analyses the query to discover measures, dimensions, and constraints for the cube.
3. The Collection Server sends requests to distributed data warehouses according to the distribution model.
4. The data warehouses return the desired data in XML form, and the Collection Server collects the data.
5. The Collection Server makes required modifications to the data and possibly performs some necessary aggregations.
6. The Collection Server sends the data to the OLAP Server in order to construct a real OLAP cube.
7. The user can pose queries against the OLAP cube in the OLAP Server using tools provided by the OLAP Server.

The rest of this paper is organised as follows. In the next section, we briefly review related work. In Section 3, we give a formal presentation for the OLAP cube and cube schema and study how an OLAP cube can be presented using XML. After that, in Section 4, we explain how users can define data to be included in the OLAP cube. The data collection is studied in Section 5 and the implementation of the system in Section 6. Finally, the conclusions are given in Section 7.

## 2. RELATED WORK

In order to achieve scalability, commercial OLAP server products have been designed to exploit distributed computing in a number of ways. For example, Microsoft's OLAP architecture copes with a large number of concurrent users by offloading aggregate processing and replicating cubes, or parts thereof, as 'local' cubes on client hosts [5]. Apart from distributing the processing load, this approach can also reduce network traffic by caching results on the client for subsequent re-use.

The use of XML is starting to spread to OLAP processing through the XML for Analysis Specification [3]. At present, the use of XML is confined to describing the structure of the result of an OLAP query, as well as providing the mechanism for transmitting the query and the results over the Internet. By contrast, our approach uses XML to describe the cube structure itself.

In Microsoft Analysis Services, when you create a new cube, it is stored in units called partitions [5]. Distributed partitioned cubes are stored on multiple servers. All of the metadata is stored on one of these servers and the partitions stored on the other servers are called remote partitions. This architecture facilitates coarse-grained parallel processing since query processing is performed on all servers containing relevant partitions.

Jensen et al. [7] study how an OLAP cube can be specified from XML data on the web. They also propose a UML (Unified Modelling Language) based multidimensional model for describing and visualising the logical structure of XML documents. Finally, they study how a multidimensional database can be designed based on XML data sources. Their method is also capable of integrating relational and XML data. The aims of Jensen et al. are quite different from ours, since they concentrate on how to find the possible multidimensional structure of the pre-existing XML data. We, instead, assume that the data was originally intended for use in multidimensional analysis; it is mostly distributed for technical reasons.

Ammoura et al. [1] have designed a system that retrieves data from distributed data sources according to user's query and displays the results using virtual reality methods. XML is used in representing meta data and communication with remote data sources. To represent user's query against data sources, they have developed an XML query language for OLAP called XMDQL. It has some similarities to MDX [8]. The system has some similarities to ours but the aim is still quite different: their system is a query and presentation tool while ours is for designing and populating OLAP cubes.

## 3. OLAP MODEL AND XML REPRESENTATION OF THE OLAP DATA

First of all, we formalise our notion of an OLAP cube. We use the relational model in our formalisation, but this does not mean that the implementation needs to be relational OLAP.

A dimension schema is a set of attributes. An OLAP cube

schema $C = D_1 \cup D_2 \cup ... \cup D_n \cup M \cup I$, where $D_1 ... D_n$ are dimension schemata, $M$ is the set of measure attributes, $I$ is a set of measure identification attributes, and $D_1 \cup D_2 \cup ... \cup D_n$ is a superkey for $C$. An OLAP cube $c$ is a relation over the OLAP cube schema $C = D_1 \cup D_2 \cup ... \cup D_n \cup M \cup I$.

If $D$ is a dimension schema, then a relation $d$ over $D$ is called a dimension. It is generally assumed that each dimension schema $D_k$ is chosen in such a way that there exists a single-attribute key $K_k$ for it, although theoretically this would not need to be so. Also, if this is not the case, then an artificial key can be formed with values concatenated from key attribute values, but this is certainly a complication in practice.

Let $K_1, ..., K_n$ be the respective single-attribute keys for dimension schemata $D_1, ..., D_n$. Now $K_1, ..., K_n$ is a superkey for $C$. This means that tuples over the relation schema $K_1, ..., K_n$ can be used as coordinates for the OLAP cube measure items, i.e. to identify the OLAP cube measure items.

We assume that the measure items can also be identified independently of the dimension information, that is, we assume that also $I$ contains a superkey for $C$. This is a fairly natural assumption, as we generally think that the measurements can be somehow directly identified (that is, they can be identified without the classification data using e.g. date and time information about the measurement). However, we allow the sets $I$ and $D$ to intersect. In the extreme case, the set $I$ may be a subset of $D$ meaning that no different measure IDs actually exist.

According to our formal model, an OLAP cube is structurally a conventional relation, where the set of dimension attributes forms a superkey for the relation. In addition, we assume that the measure attributes in $M$ are not on the left hand side of any functional dependency, that is, no measure attribute determines any dimension attribute.

This model easily supports distribution, assuming that the values of $I$ are stored along with the measure attribute values. As the values of $I$ form a superkey, they can be used to support both horizontal and vertical fragmentation in distribution. In general, $K_1, ..., K_n$ would not be equally good to use for fragmentation, as it is not so likely that all data repositories would contain information about all dimensions.

Although our formal model is based on relational model, an XML language is used to represent the actual data. The OLAP cube relation could be presented as such an XML document that each tuple of the relation would correspond to one row in the XML file. However, this kind of an XML document may contain a lot of redundancy, since information on dimension hierarchies is repeated in every row. Therefore, we partly normalise the OLAP relation and use the so the called *star schema* style XML formalism to represent the OLAP cube schema and to store OLAP cube data.

An XML star schema contains one fact table containing the measure values, the measure IDs, and dimension keys, and several dimension tables containing dimension hierarchies. This model not only ensures a more economical use of the storage space but also enables us to handle dimension information separately. For example, in our world trade data application, the country dimension is now classified according to geographical facts. However, our system enables us to get the hierarchy for the country dimension from some external data source. For example, it would be possible to classify the countries according to standard of living statistics that are available in a different database. A star schema representing our example application can be seen in Figure 2.

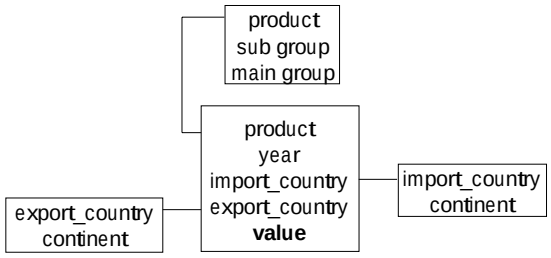


**Figure 2: Star schema for our example OLAP cube**

Figure 3 shows a part of the XML definition for the OLAP schema of our example data warehouse. The "*" symbols in the schema represent arbitrary strings that can appear as the values of the attributes. The fact table contains the measure value(s) and the key of each dimension in the cube. In this data, no specific measure ID attributes exist because of the nature of the data. Other parts are organised to represent the dimensions and they match the names of the attributes in the fact table. An example about presenting the OLAP data of our world trade example is seen in Figure 4.

```
<olap_cube name="*">
   <fact_table>
      <row value          ="*"
           product        ="*"
           export_country="*"
           import_country="*"
           year           ="*"/>
   </fact_table>
   <product>
      <row product="*" sub_group="*"
       main_group="*"/>
   </product>
         :
</olap_cube>
```

**Figure 3: A part of the example OLAP cube schema in XML**

# 4. DEFINING CONTENTS OF OLAP CUBE BY USING MDX QUERIES

The database / data warehouse schema is represented as an OLAP schema to the user. For simplicity, we assume that it is always possible to construct one integrated universal data warehouse schema for the whole distributed data warehouse. We use a simplified version of the query based OLAP cube design method [10]. According to the method, the user defines the contents of the OLAP cube by forming queries againts the conceptual schema of the data warehouse. In principle, the user could use several queries but to keep the system simple, we assume that only one query is given.

To present the query, we use a simplified MDX (Multidimensional Expressions) language. MDX is a declarative query language for multidimensional databases used in Microsoft's OLAP product [8]. The MDX query produces an

```
<olap_cube name="trade">
   <fact_table>
      <row value="200" product="fine paper"      export_country="Finland"
           import_country="UK"  year="1988"/>
      <row value="256" product="stainless steel" export_country="Finland"
           import_country="USA" year="1989"/>
   </fact_table>
   <product>
      <row product="fine paper"      sub_group="paper" main_group="forest"/>
      <row product="stainless steel" sub_group="steel" main_group="metal"/>
   </product>
           :
</olap_cube>
```

**Figure 4: A part of the example OLAP cube in XML**

OLAP cube (originally, a tabular presentation of a cube) given dimension specifications as input.

In the example in Figure 5, a simple MDX query concerning trade data can be seen. The query is used in analysing world trade figures of the main product group '0' imported in continents '2' or '5' during years 1980 and 1990.

```
SELECT {import_coutry.continent.[2],
        import_coutry.continent.[5]},
       export_country.MEMBERS,
       product.main_group.0.CHILDREN,
       {year.[1980], year.[1990]}
FROM trade
WHERE value
```

**Figure 5: An example MDX query concerning the World Trade database**

Each axes specification in the SELECT clause describes how to produce one of the dimensions (axes) in the result. This includes specifying the point in the dimension hierarchy of the original cube at which aggregation should be performed. The axis or dimensions are specified by giving a set of members. MDX has several functions of performing it, for example the MEMBERS function returns all members of the dimension. The WHERE clause allows the user to limit dimension values that will be used in the calculation. This is also called 'slicing'. The measure values to be viewed are also selected using the WHERE statement.

The MDX query is parsed by the system to find out the measure and dimension attributes and possible limits for the dimension values. The dimensions and the roll up operations related to the dimensions can be found in the SELECT clause and the measure value and possible slicing operations in the WHERE clause. This information is enough to construct an OLAP cube schema and to populate the cube. In our example, we get import continent, export country, main product group, and year as dimensions and we limit our consideration only to import continents '2' and '5', main product group '0', and years 1980 and 1990.

# 5. COLLECTING DATA FROM DISTRIBUTED DATA REPOSITORIES

The distribution is assumed to be transparent to the user. For simplicity, we assume that data is not replicated. We emphasise that the collection process is not performed 'online', though the user input is used to carry it out. The cube construction process may take some time but after that the actual queries of the user should be possible to be answered faster than in the case of a bigger cube that contains lots of irrevant data for current information needs. All the user queries are answered by the OLAP server and only possible updates to the OLAP cube in the server are made by the data collection parts of the system.

We assume that the OLAP cube is stored as a star schema, that is, it consists of (logically) one fact table and one dimension table for each dimension. Each of these tables can be stored in multible sub databases. The distribution can be horizontal or vertical. In vertical distribution each relation has to contain a superkey of the original OLAP cube relation. Because of space limitations, we do not study vertical distribution further.

The horizontal distribution of the fact table is simply described as predicate expressions by using the XML language as the following example shows.

```
<fact_table_distribution>
<source year="1980"
 database="trade1.cern.ch/db1980"/>
<source year="1981"
 database="trade2.cern.ch/db1981"/>
:
</fact_table_distribution>
```

In our example data, the import country dimension can be distributed according to the continent.

```
<import_country_distribution>
<source continent="Europe"
 database="database_1"/>
<source continent="North America"
 database="database_2"/>
<source continent="other"
 database="other_world_database"/>
</import_country_distribution>
```

To determine which database contains needed values, we use an algorithm implemented in the collection server. The algorithm gets an OLAP cube schema with possible constraints and a distribution table as input. The algorithm first groups the cube schema according to the constraints. We illustrate the process by a query shown in Figure 6.

We first form groups according to the given constraints. From the above example, we get four groups:

```
1: year=1980; import_country.continent=2;
   product.main_group=0;
2: year=1980; import_country.continent=5;
   product.main_group=0;
3: year=1990; import_country.continent=2;
   product.main_group=0;
4: year=1990; import_country.continent=5;
   product.main_group=0;
```

```
Selection constraints:
  year = {1980, 1990},
  import_country.continent = {2, 5},
  product.main_group = {0}

Level constraints:
  import_country.continent,
  product.main_group
```

**Figure 6: Parsed notation of the query in Figure 5**

We assume that the distribution model is as follows:

```
year<1990:
   database = trade1.cern.ch/tradedb
year>1989 and import_country.continent>3:
   database = trade2.cern.ch/tradedb
year>1989 and import_country.continent<4:
   database = trade3.cern.ch/tradedb
```

We notice that the following databases must be accessed for the groups of constraints:

```
group 1: database=trade1.cern.ch/tradedb
group 2: database=trade1.cern.ch/tradedb
group 3: database=trade3.cern.ch/tradedb
group 4: database=trade2.cern.ch/tradedb
```

According to our example query, we are interested in products as the main group level. This means that the measure values must be aggregated before sending them to the OLAP server. To decrease the network traffic and to distribute the calculation, the aggregation should be done in the source data repository. However, if groups, according to which aggregations should be performed, are stored on different servers, the whole aggregation calculation cannot be done locally. Further, since we do not assume that the remote database has any aggregation functions available (it can be just an XML document on the web), the aggregation is done by our collection server. The server first fetches the raw data data from databases and then performs necessary aggregations before sending the data to the OLAP server. The manipulation of the XML OLAP cube could be done by using standard XML tools, e.g. XSLT transformations. However, in our example implementation we use the Java language. In addition to aggregation, some other manipulation of the data may be needed, for example changing the names of attributes, if the databases have different names for the same attributes.

The collection process is implemented as follows. The collection server receives queries of the form of Figure 5 from the user and them to the form of Figure 6. Since the data in the collection server is in the form of Figure 3, the received queries can be transformed in e.g. XSLT statements that aggregate the data, if the queries require that. The (aggregated) data is then sent to the OLAP server. A part of the aggregated data that corresponds with the query in Figure 6 is shown below:

```
<row value="501011"
 import_country.continent="North America"
 export_country="Finland"
 product.main_group="forest"
 year="1980"/>
```

# 6. EXAMPLE IMPLEMENTATION

The system is being implemented using the Java language and Spitfire [11] software. As an OLAP server, we are using IBM's DB2 OLAP Server but the system is not particularly product dependent.

Figure 7 illustrates the implementation of the system. The system performs the following tasks:

- Concluding attributes and constraints in the query.
- Constructing the OLAP cube schema and populating the cube by transfering the data from different databases according to the distribution model.
  - Determining which remote databases contain the required data.
  - Sending SQL requests to remote databases using Spitfire software.
  - Collection of the XML OLAP cube and performing necessary aggregations and selections.
  - Transforming the XML OLAP data to a form understood by the OLAP server and sending the data to the OLAP server.
- After the cube is populated, the user can pose queries against it by using tools offered by the OLAP server.

## 6.1 Example Data

We use world trade data to illustrate the system. The data contains pairwise import/export figures for ten years of more than one hundred countries classified according to product groups. To present the OLAP cube schema, we use a conceptual description method based on (functional) dependencies between concepts [9]. The model is capable of representing OLAP dimensions and facts easily. The model also enables us to use dependency based methods in design of OLAP cubes. In Figure 8 we can see the conceptual model of our example database.

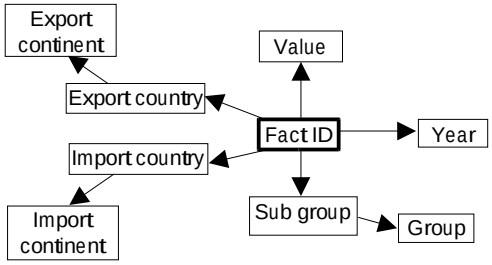


**Figure 8: OLAP schema of our example database**

## 6.2 Fetching Data

In our example system, the data is distributed according to years in such a way that each year is stored in different relations. Each dimension table is stored in a single relation.

We use the Spitfire software, which was developed in association with European Data Grid Project [4]. Spitfire provides HTTP/HTTPS based services for accessing relational databases (see [6]). In practice, a Spitfire installation is a web server whose central component is Oracle corporation's XSQL Servlet. Upon receiving a request that contains an SQL query from a web client, the XSQL Servlet returns a response in XML format (see: `http://technet.oracle.com/tech/xml/xdk_java/content.html`). Other components of

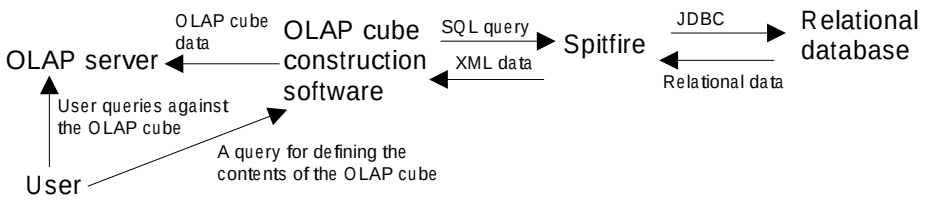


**Figure 7: System implementation**

Spitfire contain, for example, a user authorisation system that enables us to use certificates in database access [6].

Spitfire enables us to use XSLT stylesheets to define the format of the answer. If the databases are homogenous only one stylesheet is needed but in the case of heterogenous databases, a different stylesheet may be needed for each database to ensure that they all return the data in the same OLAP cube format. It would also be possible to distribute the aggregation calculation by giving suitable XSLT stylesheets to remote Spitfire nodes.

After applying the distribution model, the requests are sent as SQL queries to respective databases. If the requests contain selection conditions to the detailled data, these conditions are put in the WHERE clause of the SQL query. The remote servers format the answers to our XML presentation using XSLT stylesheets. The collection server collects these answers, perform required selections related to higher levels of dimension hierarchies, and calculates necessary aggregations. Finally, the collection server transforms the XML presentation into a form that is understood by the OLAP server.

## 6.3 Loading Data into an OLAP Server

OLAP servers can load data in different ways but most OLAP servers can read at least textual data where each column represents a dimension or a measure value and each row represents one "measure event". The columns are usually separated by spaces or tabulators. This kind of formulation can also be produced from our XML data. However, when operating with large data sets, it is not meaningful to store large temporal files. Actually, these files can become huge, since data in tabular format is often totally unnormalised and may even contain mostly null values. Therefore, it is more desirable to use methods that do not require intermediate files, for example the APIs of the OLAP systems.

# 7. CONCLUSIONS AND FUTURE WORK

We have presented a method to help in analysing large distributed heterogenous data warehouses. The method helps the user to construct an OLAP cube from distributed data for his/her analysis requirements. The method applies the XML language and we have given an XML presentation for OLAP cubes and OLAP cube schemata. An OLAP cube in XML form can be easily transformed to a form that OLAP servers can read. Finally, the actual data analysis is done by using an OLAP server product.

The next logical phase is to study how the aggregations and selections in the cube collection process can be distributed, that is, to perform the calculations where the data is stored.

# 8. ACKNOWLEDGEMENTS

We wish to thank Makiko Matsumoto of the ILO for consultation with our example application.

# 9. REFERENCES

[1] A. Ammoura, O. Zaiane, and R. Goebel. Towards a novel OLAP interface for distributed data warehouses. In Y. Kambayashi et al., editor, *Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Proceedings*, volume 2114 of *LNCS*, pages 174–185, 2001.

[2] T. Bray, J. Paoli, M.C. Sperberg-McQueen, and E. Maler. Extensible markup language XML 1.0. Technical report, W3C, 2000.

[3] Microsoft Corporation. XML for analysis specification, version 1.0. Technical report, 2001. Available on: `http://msdn.microsoft.com/library/default.asp?URL=/library/techart/XMLAnalysis.htm`

[4] M. Draoli, G. Mascari, and R. Puccinelli. *General Description of the DataGrid Project*, 2001. Available on: `http://web.datagrid.cnr.it/introdocs/DataGrid-11-NOT-0103-1_1-Project_Presentation.pdf`

[5] M. Gunderloy and T. Sneath. *SQL Server Developer's Guide to OLAP with Analysis Services*. SYBEX Inc, CA, USA, 2001.

[6] W. Hoschek and G. McCance. Grid enabled relational database middleware. In *Global Grid Forum, Frascati, Italy, 7-10 Oct. 2001*, 2001.

[7] M. Jensen, T. Moller, and T. Bach Pedersen. Specifying OLAP cubes on XML data. *Journal Of Intelligent Information Systems*, 17(2/3):255–280, 2001.

[8] Microsoft Corporation. *Microsoft OLE DB for OLAP Programmer's Reference*, 1998.

[9] T. Niemi, J. Nummenmaa, and P. Thanisch. Applying dependency theory to conceptual modelling. In O. Majer, editor, *Topics in Conceptual Analysis and Modeling*, pages 271–290. Czech Academy of Sciences' Publishing House Filosofia, 2000.

[10] T. Niemi, J. Nummenmaa, and P. Thanisch. Constructing OLAP cubes based on queries. In J. Hammer, editor, *DOLAP 2001, ACM Fourth International Workshop on Data Warehousing and OLAP*, pages 9–11. ACM, 2001.

[11] Spitfire. Project Spitfire. Available on: http://hep-proj-spitfire.web.cern.ch/hep-proj-spitfire/share/spitfire/doc/index.html, 2001.