Geospatial Information Extraction: Querying or Quarrying?

Agnès Voisard[*]   Marcus Jürgens[†]

TR-98-010

April 1998

Abstract

We focus here on the access to multiple, distributed, heterogeneous and autonomous information sources storing geospatial data and we study alternatives to integrate them. Common solutions to data integration in the database area nowadays are the data warehouse approach and the wrapper/mediator approach. None of them is really satisfactory to handle a large range of geospatial applications. In this paper we present a novel hybrid approach to data integration based on the two popular paradigms. We believe that such architectures will be of major importance in the geospatial applications of the near future.

[*]Author's permanent address: Institute of Computer Science, Freie Universität Berlin, Takustr. 9, D-14195 Berlin, Germany. E-mail: `voisard@inf.fu-berlin.de`

[†]Author's permanent address: Institute of Statistics and Econometrics, Freie Universität Berlin, Garystr. 21, D-14195 Berlin, Germany. E-mail: `juergens@inf.fu-berlin.de`

1 Introduction

Interoperability within Geographic Information Systems (GIS) is concerned with both data and operations. We focus here on the data aspect, more specifically on the access to multiple, distributed, heterogeneous and autonomous information sources and on their integration. For this purpose, we need to take into account the special requirements of applications dealing with geospatial data, such as:

- the vast amounts of data considered;
- the existence of complex and highly-structured data;
- the co-existence of many different geographic formats;
- the increasing trend towards reuse of geographic data all over the world;
- the distributed nature of geographic data.

In addition, temporal aspects play an important role in many geospatial applications. For instance, in geomarketing applications, it is useful to keep many versions of data for analysis over time. Applications such as traffic control or intelligent transportation require fast access to highly dynamic data.

GIS are concerned with many tasks, such as input and store, query and analyze, display and select geographic information. In this paper we concentrate on the data management aspect. The participating information sources may vary widely, from legacy systems to geospatial repositories. The three main actions we focus on are the following: (1) collecting data from the information sources, (2) organizing data in a geospatial store or keeping track of data organization in different systems and (3) querying data. In the sequel we refer to these actions as 3-step data handling.

In the database area, common approaches to data integration are as follows [Wid97]: In the *eager approach*, data is collected in a data warehouse, which allows direct querying and analysis. The *query-driven* or *lazy approach*, on the other hand, hinges on the concepts of a mediator together with wrappers built on top of the participating repositories. In practice, it turns out that a combination of both approaches is often more appropriate. In this paper, we present a hybrid alternative to geospatial data integration. Geospatial data sets are either integrated through wrappers and mediators or part of a data warehouse. In this context, a warehouse plays not only the role of a participating system but also the role of an efficient storage for further reuse. This storage can be related to data themselves but also to application sessions. For instance, results of queries likely to be posed in the near future are stored in the warehouse for efficiency reasons.

This paper is organized as follows. We first present the two common database approaches to data integration and compare them for different types of geospatial applications. We then present our hybrid solution, which we illustrate with a flood control application. We give a list of qualitative parameters for tailoring such general frameworks to particular application requirements. We conclude with the main research issues that need to be addressed in order to design such systems efficiently.

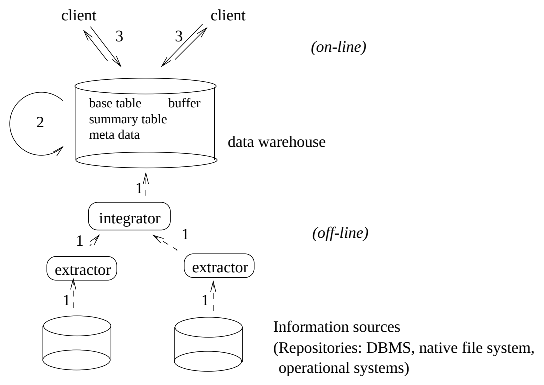

Figure 1: Eager approach to data integration.

2 Common approaches to data integration

This section starts with a description of the two common database approaches to data integration, i.e., the eager approach and the lazy approach. We then give examples drawn from geospatial applications in order to illustrate their use, before studying the main differences between the two solutions.

2.1 Eager approach (data warehouse)

In the *eager* or *in-advance approach*, data is integrated in advance in a data warehouse, which is defined as a subject-oriented, integrated, non-volatile and time-variant collection of data [Inm96]. It allows the multidimensional storage of data. A common integrator component extracts data sets from different sources and integrates them before sending them to the warehouse. Figure 1 illustrates this approach.

Following the 3-step data handling mentioned in the introduction, we now describe the major tasks necessary to run a data warehouse, namely data warehousing, warehouse organization and query processing.

Data warehousing (1). The process of collecting the data is known as *data warehousing*. New data is added to the data warehouse from different sources, which generally handle data in different formats. Metadata describes how data has to be transformed between formats and how it is integrated. Before data is added to the warehouse, consistency checks are performed and a mechanism guarantees that all data is loaded exactly once. The load time becomes a critical issue when the number of sources or the size of sources increases. Attention should be paid to keep the load time as short as possible.

Regarding the moment when data is fetched to the warehouse, two options are possible. With the *push* mechanism the source decides when to send the data to the warehouse. In the *pull* mechanism the warehouse asks explicitly for the data at a certain time. A push or a pull is either event- or time-driven. To reduce the amount of data transferred to the warehouse, only the changed data could be shipped. Sending the delta between two data sets saves significant amount of time and space in applications with small changes from time to time. This turns out to be useful in applications handling large maps, especially in the case of raster maps with changes in a few cells.

Warehouse organization (2). The resulting warehouse database is somewhat special as there is usually no delete nor update operation possible. The problem of constant growth can be solved either by moving the data that is not used to less expensive media or by summarizing it further. The role of data warehouses goes beyond data storage. Raw data is (re)-structured, and aggregated in such systems and stored in *base tables.* In addition there are aggregates held in summary tables. Aggregation rules are defined in the metadata. Further aggregation is useful because most common queries analyze subsets of data. Users are sometimes not interested in particular objects but in aggregated values, such as the population of a state computed from a table gathering the population of all its counties. If these results have to be calculated *on the fly*, i.e., when the query is processed in the system, the response time can be intolerably long. Results of aggregation are calculated in advance during times when the system is not used for analysis (e.g. during nights or weekends). They are stored in *summary tables* [MQM97].

The more data is aggregated, the faster can queries be processed, but the higher are update and storage costs. [WGL$^+$96] presents algorithms for selection of summary tables and indices. Aggregation is appropriate when clients require specific predictable portions of information. It is noteworthy that, even if this mechanism provides high performance when answering queries, the warehouse does not necessarily show neither the most recent state of information sources nor all details, because summarized data and not all raw data is stored there. However in many applications it makes sense to get a rough result fast rather than a precise value after long processing time.

Query processing (3). Users typically pose *ad-hoc* queries against the warehouse and expect answers quickly. To ensure that typical queries will be answered fast, queries are logged and users profiles are generated from the log-files. These profiles are used to adapt the organization of the warehouse to users' needs and to generate indices for faster access and better performance. Because profiles change over time, the reorganization of the warehouse might be necessary on a regular basis.

During the data warehousing and warehouse organization phases the warehouse is *off-line* and no queries can be processed. It is *on-line* only when no changes are made. This architecture is called a *read-mostly environment.* The clear distinction between on-line and off-line modes makes transaction monitors superfluous while being in on-line mode. This decreases response time.

2.2 Lazy approach (mediator)

In the *lazy* or *on-demand approach*, when a query is issued, the appropriate information sources are selected and a mediator generates the subqueries or commands that will be addressed to the information sources via their wrapper (Figure 2). A wrapper converts data objects of specific data sources into objects of a common information model (Step 1). A catalog keeps metainformation related to the information sources (Step 2). Data conversion works both ways. Queries from

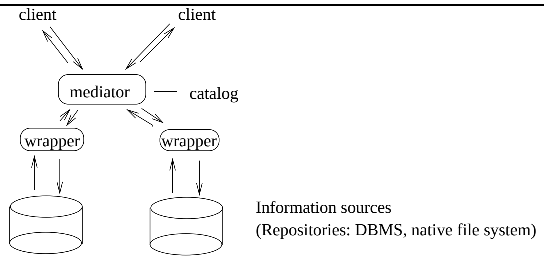

Figure 2: Lazy approach to data integration.

the mediator are transformed to queries understood by the sources and data from the sources is converted to a format processable by the mediator [GMHI+95] (Step 3).

In such systems, the results are collected from the information sources and the mediator performs translation, filtering and merging of data before returning the answer to the client. A mediator is a software module that refines in some way information from one or more sources. The wrapper does the technical transformation (e.g., unit transformation) whereas the mediators task concerns the semantical integration of data (e.g, generalization). The great advantages of this architecture are low storage costs and flexibility for integration of new sources. A disadvantage is that time analysis is only possible on data provided by the data sources.

One may wonder about the differences between a mediator/wrapper architecture and a federated DBMS. In the first architecture, the sources are often semistructured, and no schemas are available. Source integration occurs at access time. The traditional method of first integrating the schemas and then the data does not apply in this context. Because the format of data sources often changes, one important research issue is the development of tools to support the efficient development of wrappers and mediators.

2.3 Summary: Warehouse vs. mediator approach

Following our 3-step view of data handling, Table 1 sums up the functionalities of each approach.

Using a warehouse in geospatial application

A major interest in warehousing is to avoid converting large amounts of data in and out by having information at hand. This is an important issue in geographic information systems where map transformations are usually complex and time-consuming. From time to time, snapshots are taken from the data origins and loaded to the warehouse (Cf. Section 2.1).

Clearly, this approach brings new potentials to data handling, with the fact that data can be further processed, e.g., annotated, modified, structured and aggregated. Annotating data is particularly useful in geographic applications: Sometimes, old data cannot be used because information on its origin or on its accuracy is missing. Moreover, historical information can be associated with

	Data Warehouse (eager approach)	**Mediator (lazy approach)**
Data Collecting	Load/Clean/Consistency Check/ Aggregation Rules for Base Data	Wrapper
Data Organization	Summary Tables/Base Tables/ Aggregation Rules for Summary Data	Catalog
Query Processing	Warehouse Querying	Distributed Query Processing

Table 1: 3-step data handling in eager and lazy approach

the data. Techniques borrowed from related disciplines, such as statistical databases (e.g., aggregation or data cube metaphor), versioning, data mining or OLAP (On-Line Analytical Processing) can greatly improve data handling in this context.

Data warehouses are often used for *ad-hoc* queries and what-if analysis. What-if analysis calculates different scenarios for the future. This functionality is well supported by data warehouse architectures. It allows to calculate new data from the already stored data. This predicted data can be labeled differently from the data loaded from the real world. Such analysis would be difficult with the pure mediator approach. Geospatial applications dealing with analysis over time gain from using a data warehouse. In such systems the life-expectancy of data is 5-10 years (compared to 60-90 days in operational systems [Inm96]). In some applications such as the ones handled by decision support systems (DSS), expected response time is usually relaxed and the order is minutes or hours. Hence the analogy with "quarrying data".

Using the wrapper/mediator approach in geospatial application

In contrast, the mediator approach is in particular appropriate when data and data sources change rapidly. It is obviously well-suited to electronic marketplaces, in which prices for goods or services are assumed to alter constantly. However, the delay induced by significant processing needed for data transformation and shipping may make its use inefficient. Moreover, if the same queries are issued multiple times, numerous accesses to data are unnecessarily expensive. This approach is appropriate for near-real time applications, such as intelligent transportation systems or environmental problems monitoring. Note that it cannot be used if ad hoc querying is not feasible. This approach is inefficient for applications where static data from different sources has to be expensively combined many times in the same way (for instance, joined for a map overlay).

With each object in the warehouse is associated a validity time, whereas in operational systems the data is true at access time. This is one fundamental distinction between OLAP and OLTP (On-Line Transaction Processing), where the snapshot view of data in data sources makes data analyses impossible.

3 Hybrid approach

Given these two general approaches to data integration, one may wonder about the most appropriate solution for geospatial data handling. Part of the answer is that it depends on the *type of application* and on the *type of data*. We place ourselves in a context where data is coming from extremely heterogeneous sources, from a semantical and physical view point. For instance, it can be available freely on the Internet but also purchased from agencies (e.g., governmental institutions). Moreover, data can be either static in the sense that it is unlikely to change over time (e.g., state contour) or dynamic (e.g., traffic flow, population, weather, stock market, currency exchange rates, entertainment programs or sales data). Note that there are many degrees in dynamicity. A challenge is to integrate these different types of data within one coherent framework. Hence in many geospatial applications a somehow hybrid approach is needed.

This section starts with the basic concepts of such architectures. We then illustrate their use through an example drawn from an application that predicts water floods in a given geographic space. We conclude with a list of qualitative parameters to be considered when designing such flexible frameworks. This list is based on both the type of applications and on the types of data in the sources.

3.1 Basic concepts

In hybrid approaches, part of the data is collected, processed and integrated in advance in the warehouse, while other data is fetched when users pose queries against the overall system. The data catalog knows where to find appropriate data (the size of this registry is negligible compared to the size of the data involved). Figure 3 depicts a hybrid solution to data integration. This architecture is a generalization of the two architectures described in sections 2.1 and 2.2, respectively. If everything is loaded from the data sources S_1 ... S_n to the geodata warehouse, every query can be answered by using data from the warehouse. This approach is the pure first architecture described in Section 2.1. If the data warehouse is left empty, it cannot support answering queries. All data necessary for answering queries has to be shipped from the sources, wrapped and processed by the mediator. This is the pure second architecture as explained in Section 2.2.

To our knowledge, even though they are more and more used in practice, such architectures have not been studied in detail by the scientific community. The problem is obviously more complicated than considering a data warehouse as a regular information source. The warehouse is a special kind of data storage. In contrast to the other information sources, which are fixed and cannot be changed by the system, the warehouse is freely configurable by the system itself. The fact that the system can precompute and store regularly the required data can improve the performance considerably. In addition, other data related to applications and sessions can be stored for further use, which opens new horizons to many geospatial applications.

Many degrees of hybridization should be considered from the pure data warehouse approach to the pure mediator approach, and a major design issue is to determine which parts of each should

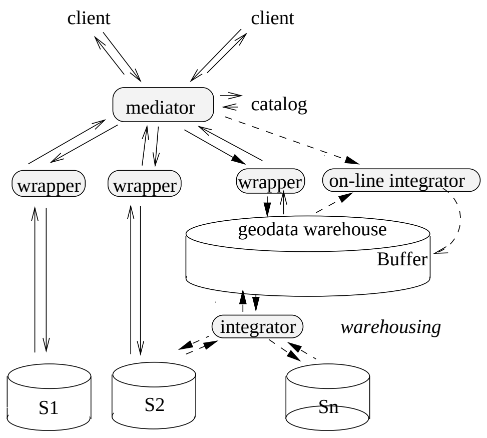

Figure 3: Hybrid approach to data integration.

be considered. A first step is to consider the sources where data may come from. Once the sources with data stored in the data warehouse are determined, the question is whether raw data should be stored or if it is, for space and time reasons, more sensible to store (pre)aggregated data. This is a common issue in pure data warehousing. There are many possibilities regarding data aggregation. First of all, data of only one source can be aggregated, such as the population of counties in one state aggregated in the population of a state. Besides, because of the multiple repositories, data from different sources can be joined and generate aggregated data. For example, the population of cities joined with sales data from other sources to get sales data for cities with more than 100.000 inhabitants. Once data is collected in the data warehouse, the later plays the role of a participating system, hence the wrapper depicted in Figure 3 (right-hand side).

In this approach, the data warehouse can be used for more than storing data coming from different sources, which we refer to as application-driven contents. The selection of the data to be stored is done when the system is installed. In practice, such a data warehouse serves many purposes. Following are the three types of data it hosts:

- Application-driven contents (100 % manual work).
  A warehouse is useful for archiving data related to a whole application field. In order to use it in the long run data is stored in one part of the warehouse. This is useful in geomarketing applications where data, being socio-cultural behaviors or statistical data, is considered over long periods of time. A mechanism enforces the storage of data (even some dynamic data) in order to keep it and make studies over years for a specific application. These application-dependent data sets are selected by users or system administrators, who decide in particular which tables have to be stored in the warehouse together with the update frequency.

- Usage-driven contents (programmed selection of data).
  As mentioned in Section 2.1. (query processing in data warehouses), it is useful to monitor the users queries and to generate user profiles. This is done during the data organization phase. This treatment can be applied to metadata as well, i.e., to save user sessions (query

graphs) over time. This metadata holds data about typical user behaviors and can be also exploited for analysis. It can also be referred to as session-driven contents.

- Data-driven contents (0 % manual work).
  First, one could think of abusing the data warehouse as a kind of cache. A cache is a small fast memory which stores the most recent accessed values from a large slow storage. It is based on the assumption that the user will require some of the most recent used data again. He/she can access the data much faster by using the cache than accessing the data from the large slow memory.
  Second, one could consider the integration of data *as it comes to the application* and not necessarily in advance; after being processed, it is stored for further use within the same application (e.g., retrieval and analysis). Collecting data in this context is more than simply merging data sets. This is the role of the "on-line integrator" displayed on top of the geodata warehouse in Figure 3. One of its tasks is to check that data is not stored many times.

3.2 Running example

Let us consider an application for predicting floods of major European rivers, and suppose that we concentrate on rivers that flow through more than one country, such as the Rhine. The overall system (the geospatial data manager) gets information from various repositories located in different countries. Some of these sources are static (e.g., topographic maps) while others are dynamic (e.g., water levels at some stations). To predict floods, we assume the existence of a model that runs on the following data sets: The topography of the landscape (altitude), the water network, the present water levels and speed of flow at specific reading points and finally the present and predicted precipitation. All these sources are overlaid and the simulation model returns a set of points that are likely to be flooded.

In the sequel, we start with a more complete description of the data sources. We then describe the major processing steps as well as the new relations to be created. Operations to be performed are given in a pseudo-code whose syntax is supposed to be intuitive. Database queries and schema definitions are given in our SQL-like language.

Supply of data by different data sources
As depicted in Figure 4, the data sets come from six data sources. A static map with altitude information is stored in R-alt. The static map R-water contains the water network where in particular the positions of all rivers and other waters bodies are stored. Two dynamic sources (containing the maps R-flood1, R-flood2) make available the water levels and the speed of flow of selected rivers for different regions, with updates every hour. R-prec-a keeps the actual precipitation and R-prec-p contains the predicted weather map.

We assume that these maps are stored in relations whose schema is given below. Regarding attribute types, we assume the existence of basic types and of geospatial types, such as point, line and region [Güt88, SV89]. Timestamp is a combination of types time and date.

- R-alt (Location: point, Altitude: real)
- R-water (Rivername: string, Area: region)

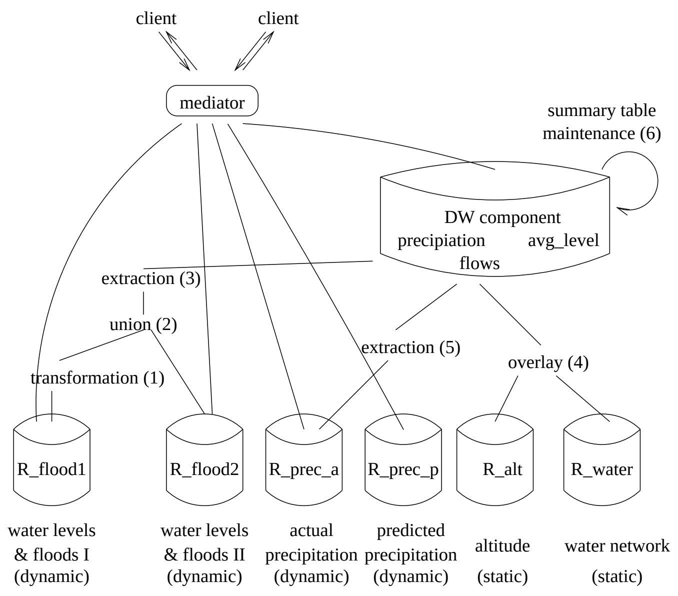

Figure 4: Example application.

- R-flood1 (Location: point, Level: integer, Speed: real) [level in cm, speed in km/h]
- R-flood2 (Location: point, Level: real, Speed: real) [level in m, speed in m/s]
- R-prec-a (Location: point, Precipitation: real)
- R-prec-p (Location: point, Precipitation: real)

We assume the existence of an operation `transform` (Relation-name: string, Attribute-name: string, Old-unit: string, New-unit: string), which, for all tuples of relation Relation-name, applies a unit transformation from Old-unit into New-unit for the value of attribute-name.

Table creation

In the data warehouse, two types of relations are considered: Base tables and summary tables. Base tables store the most detailed information handled in the warehouse. The summary tables are generated from the base data. The base tables are Topography (on which the model will be run), Flow and Precipitation. They are defined as follows:

`create table` Topography (Rivername: string, Location: point, Altitude: real, Rivername: string, Area: region);
`create table` Precipitation (Location: point, Time: timestamp, Precipitation: real);
`create table` Flow (Location: point, Time: timestamp, Level: real, Speed: real);

Sequence of operations

The pseudo-code corresponding to the first sequence of operations is given below.

Step 1: Data transformation

```
R-flood1' :=transform ( R-flood1, Level, "cm", "m");
R-flood1" := transform ( R-flood1', Speed, "km/h", "m/s")
```

Step 2: Union of the two dynamic sources R-flood1 and R-flood2

```
R-flood := union(R-flood1", R-flood2);
```

Step 3: Data extraction from R-flood

The following query is triggered every hour and extracts data from R-flood. The value of attribute Time *corresponds to the access time.*

```
insert into Flow (Location, Time, Level)
select Location, Access time , Level
from R-flood;
```

Step 4: Data extraction from the precipitation prec-a

The following query is triggered every 12 hours and extracts the present precipitation from the source prec-a.

```
insert into Precipitation (Location, Time, Precipitation)
select Location, Access time , Precipitation
from prec-a;
```

Because data about flows and precipitations are needed for analysis in the system, relations Flow and Precipitation are stored in the warehouse.

For the above `insert` operations the warehouse has to be *off-line*. To avoid switching the system on and off, new data could be stored in a buffer first.

Step 5: Map overlay

```
map :=overlay (R-alt, R-water);
```

Maps R-alt and R-water are assumed to be static. Therefore it is efficient to do the `overlay` only once and to store the resulting map in the warehouse.

Step 6: Maintenance of summary tables

In addition, tables with aggregated values over the base tables are stored in the warehouse. These summary tables can be defined by views, e.g.:

```
create view  avg-level(Location: point, Year: integer, Month: integer, Day: integer, Level: real)
as
select Location, Year (Time), Month (Time), Day (Time), avg(Level)
from prec-a
group by Year, Month, Day
union
select Location, Year (Time), Month (Time), NULL, avg(Level)
from prec-a
```

```
group by Year, Month
union
select Location, Year (Time), NULL, NULL, avg(Level)
from prec-a
group by Year
union
select Location, NULL, NULL, NULL, avg(Level)
from prec-a
```

Remark on aggregated values.
If the view above is materialized, it creates tuples with the average level of water at different locations for different time periods. This implies the existence of a function that computes the year, the month, and the day from a given timestamp. It should be mentioned that in this example, there exist `group by` operations over calculated values that are not supported by standard SQL. The extension of SQL with a cube operator to overcome this problem is discussed in [GCB+97]. One way to avoid using a `group by` over calculated values is to consider a star schema. In a star schema the dimensions are not directly stored in the fact tables but in dimension tables. The fact tables themselves contain foreign keys to entries in the dimension tables. Fact tables are placed in the middle of dimension tables (hence the name star schema). To get the desired data, the fact table and the the dimension table have to be joined and the `group by` is performed on the joined table. These operations can be defined in standard SQL although they are complex to write.

Instead of being aggregated over time, data could also be aggregated over space. Sets of locations (points) could be summarized to regions and, for example, the average precipitation per region could be calculated. It could also be aggregated over both time and space at each possible aggregation level. However, the type of data that can be summarized has to be carefully chosen. [LS97] defines a framework and a set of conditions for summarizability of data in statistical databases and OLAP.

3.3 Configuring parameters: A qualitative approach

Within this mixed solution, the criteria for using either mechanism are based on infrastructure, i.e., on the characteristics of the networks and the collection of data sources, as for instance:

- *Nature of data*, i.e., whether is it static or dynamic.
  Dynamic data is less likely to be integrated in the data warehouse because it increases the update cost. In our example, the altitude map and the weather map are static and are stored in the warehouse whereas the predicted precipitation is highly dynamic and is therefore not integrated.

- *Cost for transforming (integrating) the data* and *existence of wrappers and protocols.*
  High transformation costs can be reduced by transforming the data from the sources to a neutral format in the data warehouse only once. This is an important criterion in geographic applications where transformation of map formats is a costly operation.

- *Cost for accessing data.*
  Repeatedly paying high access cost can be circumvented by loading the data from the sources to the data warehouse, as above.

- *Size of the data sets.*
  The bigger the data set the more it is unlikely to be integrated in the data warehouse unless it can be easily compressed, e.g. for in the case of raster images.

- *Speed for data shipping and networks used.*
  The trouble caused by slow connections to data sources can be alleviated by storing the data in the data warehouse if possible.

- *Frequency of access to particular sources.*
  Some sources are needed more often than others. Sources rarely used should not be materialized in the warehouse.

- *Sources availability.*
  There is no need to integrate sources in a data warehouse if they are always available. But in case of low availability integration should be considered.

On the application side, examples of parameters to be taken into account are:

- *Possibility of predicting queries (e.g., the number of times that queries are issued).*
  If queries can be predicted and if they do not require dynamic (e.g., real-time) data, then it is advisable to precompute them and to store the results in the data warehouse.

- *Needs for data analysis (e.g., time).*
  If data is not stored in the warehouse, data analysis can be tedious and even sometimes impossible.

- *Possible joins over many sources.*
  Joins over different sources can be expensive and time can be saved by precomputing them (even though the results are large).

- *Granularity of the fetched information.*
  If data granularity is high, the results can be precomputed easily and stored in the warehouse.

- *Size of the query results.*
  Small answers in terms of size are more likely to be precomputed and stored in the data warehouse than large ones.

- *Required response time.*
  The shorter the required response time the greater the need for precomputing data.

- *Possible compromise on the accuracy of data.*
  If less accurate data can be used, data should be condensed to save space, which is sometimes a compromise in large geospatial applications.

Systems based on hybrid architectures should be tailored to accommodate the parameters above. A quantitative approach would consist in defining a cost model that takes these parameters into account.

4 Conclusion

The goal of this paper was to study solutions to the integration of geospatial data coming from multiple, distributed and autonomous sources. These sources are moreover heterogeneous, both physically and semantically, and their only common aspect is to store geospatial data. We assumed that the applications we deal with know the sources where the data come from, hence considering transparent access to these sources was out of the scope of this paper. However, if users do not know where data is coming from, one requires a mechanism to allow transparent access (e.g., that finds appropriate resources and dispatches queries). We presented the two popular database alternatives to data integration, namely the eager approach (data warehouse) and the lazy approach (mediator). We showed that the concept of a data warehouse goes far beyond a simple storage for terabytes of data. Data is not just copied from the sources to the data warehouse but is also integrated, cleaned, and aggregated. In the wrapper/mediator approach, on the other hand, data is collected on demand and not stored further, or only locally. However, both approaches have limits to handle various kinds of applications, as illustrated by many geospatial examples. We then presented a promising approach based on the two paradigms, and we illustrated its use through an example drawn from a flood control application. We finally gave a list of qualitative parameters for configuring a particular instance of such frameworks.

It is our belief that the future should bring adaptable, scalable frameworks based on the combination of the two common database approaches described above. However, even though these approaches are now well-understood by the community, lots of work remains to be done in the design of such hybrid systems. An important issue concerns the geospatial data warehouse organization itself, both at a logical and at a physical level. For instance, issues such as the adaptability of such systems to various applications and time-space trade-offs still need to be addressed. [Han97] shows a first step towards that direction with selective materialization, based on the fact that computing on the fly is expensive, saving all combinations has a huge space overhead and that rough approximations lead to accuracy problems.

Conceptual models to describe hierarchies in many dimensions (e.g., time and space) are also missing. These models encompass aggregation rules in addition to geospatial objects. Work from the statistical database community could be adapted to take the notion of space into account, as initiated by [SR95]. Another issue concerns the granularity and the partitioning of data. What level of detail has to be shown? How to break the data into separate physical units that are handled independently? Data can be partitioned by date, organizational units, geography, and clustered according to users' needs. Combinations should also be considered. Classical techniques (e.g., materialized views) are based on redundancy and cannot be used in the context of geographic information because of the huge amounts of data involved.

More related to software engineering issues is the design of the whole system. The ideal situation would be to have a list of quantitative parameters to tailor the system to particular application requirements and to adapt it automatically to users' needs. The input of the function that defines a suitable configuration would be a list of parameters together with possible compromises, as listed in Section 3. Its output would be a set of given configurations. This problem, which is related to cost models issues, is currently being addressed.

Acknowledgments. Support from the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316) is gratefully acknowledged. We also wish to thank Oliver Günther, Hans Lenz and Heinz Schweppe for stimulating discussions.

References

[GCB+97] H. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[GMHI+95] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Integrating and accessing heterogeneous information sources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering*, pages 61–64, 1995.

[Güt88] R. H. Güting. Geo-relational algebra: A model and query language for geometric database systems. In J. W. Schmidt, S. Ceri, and M. Missikoff, editors, *Advances in Database Technology. Proceedings of the International Conference on Extending Database Technology (EDBT'88)*, Lecture Notes in Computer Science No. 951, pages 506–527, Berlin/New York, 1988. Springer-Verlag.

[Han97] J. Han. Spatial data mining and spatial data warehousing. Tutorial Notes of the International Symposium on Spatial Databases (SSD'97), Berlin, Germany, 1997.

[Inm96] W. H. Inmon. *Building the Data Warehouse.* Wiley Computer Publishing, New York, 2nd edition, 1996.

[LS97] H. J. Lenz and A. Shoshani. Summarizability in OLAP and statistical data bases. In *Proceedings of 9th International Conference on Statistical and Scientific Database Management.* IEEE Computer Society Press, 1997.

[MQM97] Inderpal Singh Mumick, Dallan Quass, and Barinderpal Singh Mumick. Maintenance of data cubes and summary tables in a warehouse. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(2):100–110, May 1997.

[SR95] M. Scholl and P. Rigaux. Multi-scale partitions: Applications to spatial and statistical databases. In M. J. Egenhofer and J. Herring, editors, *Proceedings of the 4th International Symposium on Spatial Databases (SSD'95)*, Lecture Notes in Computer Science No. 951, pages 170–183, Berlin/New York, 1995. Springer-Verlag.

[SV89] M. Scholl and A. Voisard. Thematic map modeling. In A. Buchman, O. Günther, T.R. Smith, and Y.F. Yang, editors, *Design and Implementation of Large Spatial Databases, First Symposium SSD'89*, Lecture Notes in Computer Science No. 409, pages 167–192, Berlin/New York, 1989. Springer-Verlag.

[WGL$^+$96] J. Wiener, H. Gupta, W. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom. System prototype for warehouse view maintenance. In *Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications*, pages 26–33, 1996.

[Wid97] J. Widom. Research problems in data warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM'95)*, 1997.

Table of contents

List of Figures