

Abstraction and Decomposition in Interoperable GIS

Agnès Voisard* Heinz Schweppe

Institut für Informatik
Freie Universität Berlin
Takustr. 9
D-14195 Berlin

{voisard,schweppe}@inf.fu-berlin.de

Abstract

With the advent of distributed computing and the increasing trend towards the reuse of geographic data, a new generation of geographic information systems (GIS) is currently being specified. The key characteristics of these *interoperable GIS*¹ are modularity and extensibility, and they are composed of existing software systems such as database management systems, traditional GIS, statistics packages and simulation models. They can be defined in terms of generic frameworks which facilitate both information exchange between participating systems and the addition of new functionalities. Even though the idea of defining open GIS is not new, it is crucial that the steps necessary to design such a complex system be clearly decomposed. In this paper, we propose a layer decomposition for the design of an open GIS. Each layer corresponds to a different level of abstraction, starting with the application or user level down to the invocation of systems services. In addition, each such level can be specified by the same set of concepts: operation, data and session (ODS concepts). Orthogonally, the metadata needed for the interaction between levels is indispensable to achieve openness. The declarative style of specifying both user interaction and implementation on different levels of abstraction facilitates the incorporation of new software components. Therefore, the framework introduced in this paper supports the design and the evolution of interoperable GIS.

*Part of this work was carried out while this author was visiting the International Computer Science Institute, Berkeley, CA.

¹We use the terms *interoperable GIS* and *open GIS* interchangeably, however *Open GIS (OGIS)* is a trademark of the Open GIS Consortium (ODC).

1 Introduction

Geographic information systems (GIS) support applications that manipulate geographic data, such as urban planning, traffic control or natural resources management. Among the main issues in GIS design are the management of large amounts of data and the coexistence of two kinds of data: alphanumerical data (map descriptions) and spatial data (geometry and topology). In addition, spatial data typically has an extremely complex and variable structure, and it must be manipulated by specific operations. With a GIS being used in different applications, we cite among the possible queries of an enduser:

1. *Overlay the map of land use with the map of (administrative) counties in California.*
2. *Display today's weather forecast in the Bay Area.*
3. *Given the map of a neighborhood, find the best spot for opening a drugstore (based on a given set of optimality criteria).*

The intention of these examples is to illustrate the concepts of this paper. They are therefore kept simple on purpose. In particular, the methodology introduced is not restricted to operations on maps, although 'map' is a convenient metaphor.

It is now admitted that in the near future a GIS will be more than a static "monolithic" system designed for a given set of applications; it will be dynamic in the sense that independent and - to a large extent autonomous - systems have to be incorporated into the GIS in a modular fashion. The necessary network services are at hand. However, software architectures for application systems are not yet available.

Our goal is to contribute to the definition of this new GIS generation called *interoperable* or *open GIS*. We envisage an interoperable GIS as a system of communicating services which manipulate alphanumerical or spatial information (we prefer the term "service" since the term "process" is biased towards operating systems). As examples of subsystems or "modules" we can cite an expert system, a statistical analysis system for social and economic data or a simulation model for environmental studies in spatial decision support systems. We want to be able to consider a modular global system that allows the separated development and execution of tasks within such specialized systems.

One might wonder what the difference between open GIS and generic open distributed systems is, or, in other words, what makes geographic information handling special. Geographic applications require particular attention for the following reasons: (i) the existence of large amounts of data, which prohibit the use of some classical manipulation techniques; (ii) the co-existence of many different geographic formats; (iii) the increasing trend towards reuse of geographic data (see for example [Gov94]); (iv) the fact that distributed data and operations are equally important, (v) the complexity of geographic data structures and in many cases (vi) the interdependence of geographic entities. In addition, specific requirements for multimedia systems apply to this context, such as the integration of a specific set of functionalities, data visualization and modeling. Given these observations, our objective is to define a frame of reference for handling a wide range of geographic applications, which have only the existence of geographic data in common. Our framework enables the consideration of data from heterogeneous sources, which is extremely important in the context of geographic applications. It helps solve one of the major problems of GIS these days, namely a high level integration of data stored under different formats (such as the German ALK format, the French FEIV format, or the American SDTS format [St97]). So far the lack of widely accepted standards in geographic

formats implies that when a system needs to consider foreign geographic data, this data is integrated in an *ad hoc* manner based on the technology of the system that hosts it.

Interoperable GIS design and implementation is related to many general areas in computer science, including traditional GIS and database management of course, but also software engineering, artificial intelligence, distributed computing, network and hardware. Hence the task of designing and implementing open GIS goes beyond that for classical GIS. We believe that only the (challenging) successful integration of all these disciplines will lead to efficient open GIS.

On the GIS side, a significant work has been achieved by the Open GIS consortium for the past two years. According to [OGI96b], “the Open Geodata Interoperability Specification (OGIS) provides a framework for software developers to create software that enables their users to access and process geographic data from a variety of sources across a generic computing interface within an open information technology foundation”. The framework is technology-oriented and is geared towards implementations on top of various distributed computing platforms. It is articulated around (i) the Open Geodata Model [OGI96a], which is a centralized model for interoperable geoprocessing and (ii) the OGIS Services Model. It does not claim to be a standard but given the success of this enterprise it might become a *de facto* standard.

It would be too long to cite all work more or less related to our approach, but following are some of the most relevant research efforts with respect to the issues addressed in this paper. Prior to the initiative above, [AYA⁺92, AKC94] shows architectures of spatial decision support systems based on federated systems. [FK95] proposes the use of functional programming languages as specification and prototyping tools for open GIS components. [WWC92] presents the concept of megamodules to achieve communication among encapsulated software elements; megaprograms capture the functionality of services provided by megamodules.

An important step towards open geographic information systems are distributed maps built according to the browser / server paradigm of the World-Wide Web, i.e. HTTP. The MapGuide System [Aut96] is a recent product which even supports overlay of maps from different databases or zooming into attribute data. The lightweight client approach requires to perform all operations on the server site. Only operations on maps are supported. The application protocol between clients and server is proprietary. Thus a more general approach is desirable.

More associated with the database area, the data warehouse (DWH) metaphor has gained considerable attention in distributed database related applications [Wid95]. Basically, a data warehouse is supposed to provide a homogeneous view on heterogeneous data sources in a single repository. Data may be aggregated, transformed or raw. There are obvious similarities between open GIS and the DWH approach, as, for instance, distribution, the co-existence of different formats and many ways of combining or aggregating data. Therefore it is not surprising that there are several common technical issues. However, operations have not been considered explicitly. Functions may be applied to remote data, a strategy called function shipping, or data may be transferred to some host capable to perform the operations required. These different strategies have not been taken into consideration in DWH architectures. Furthermore, a data warehouse is still based upon the database-associated query / answer model. There is no explicit notion of application steps. In contrast, such an approach has been carefully studied in the context of generalized transactions [WR92]. The DWH architecture is biased towards consistent, fault tolerant and concurrent execution of distributed data related operations. However, in GIS, consistency problems arise less frequently than in transactional systems where updates happen very often. Complementary in a way to the DWH because it deals with distributed software (methods) is the MMM project [GMS⁺96], which allows the remote access of software modules on the WWW.

Our proposal, which is an extension of [VS94], is concerned with conceptual design issues. It is orthogonal to the above approaches with the exception of the ODC approach which does not focus, however, on the declarative specification of layers. The architecture presented in this paper decomposes a GIS into different layers. Each layer corresponds to a given level of abstraction. On the application layer next to the user interface, technical aspects like different formats or distribution may seem to be transparent. For instance, in order to overlay two maps, it may be necessary to extract them from independent GIS, convert their formats and then perform the overlay. But this is invisible at the application level. It depends of course on the specification, whether operations like format conversion are invisible for the end user. Semantic differences in data models and loss of information during conversion may be important, which is true for many applications. In this case, these conversions have to be explicitly specified by the user. In order to access data from different sites, low level invocation of systems and low level data transfer have to take place. Remote procedure calls or remote object invocation or simpler communication mechanisms like HTTP could be employed at this level.

It is no surprise that decomposition and abstraction help structure complex software systems. This approach has been successfully employed in the traditional software engineering process. In an open GIS this methodology is fundamental. Plugging a new service into an existing GIS in an *ad hoc* fashion is inappropriate. A systematic approach will define, for instance, specification, data formats and run time environments on different levels of abstractions. Furthermore, our approach makes the overall system extensible. It is unrealistic to assume that, e.g., all important formats of geographic data can be anticipated. But the incorporation of a new format, that is, data manipulation and conversion tools among others, should be as easy as the utilization of a powerful new statistical package. The architecture proposed in this paper addresses this kind of openness and extensibility.

This paper is organized as follows. Section 2 first gives a toy application example which serves as a reference throughout the paper. It then describes our abstraction decomposition of an open GIS as well as the three fundamental concepts used at every level. In Section 3, all layers, from the highest one to the lowest one in the decomposition hierarchy, is detailed through these concepts. Section 4 presents the various catalogues of metadata and shows the mechanism to map a layer onto the next layer by using high level functions and metavariables. Section 5 draws some conclusions and discusses our ongoing and future work.

2 The Multilayer Approach

There is an analogy between our layer decomposition and communication protocols like TCP/IP which are defined on different layers. The principle is to define the whole system at a certain level of abstraction in terms of general features; then this level is refined by the (lower) level which implements it, and so forth. The idea is to use standards or well-known mechanisms to implement each level. For instance, some features of an intermediate level can be implemented using a database system, while at lower levels they could be implemented with, e.g., CORBA [Cor97] and system calls.

A second interesting point is that each level of abstraction is described by the same set of concepts: *operations* and *data* considered at this level, and *sessions*, which expresses how operations relate to each other. We shall refer to this 3-component concept as the ODS model.

In this section, we first describe the example we use throughout the paper. Then we infor-

mally present the layers, which are detailed in Section 3. We end up with a description of the data, operations and session concepts.

2.1 Reference Example

The following example is borrowed from an environmental application. Suppose that for planning purposes one would like to study the impact of constructing a factory near a city, i.e., the pollution that it may cause. To do that, the town planner first overlays a land use map with a pollution map which shows the current state of the pollution in a given geographic context. This is carried out without knowledge of where (in which system) these maps are stored. The map overlay operation is invoked from the interface and the resulting map is displayed on the screen. On the latter, the user selects a point that seems appropriate to build a factory, but s/he would like to check it. This is achieved by invoking a given simulation module (a system). This module calculates the propagation in carbon dioxide starting at the point selected by the user. Then a new map of the critical areas is created and displayed on top of the previous map overlay. Figure 1 depicts the enduser steps.

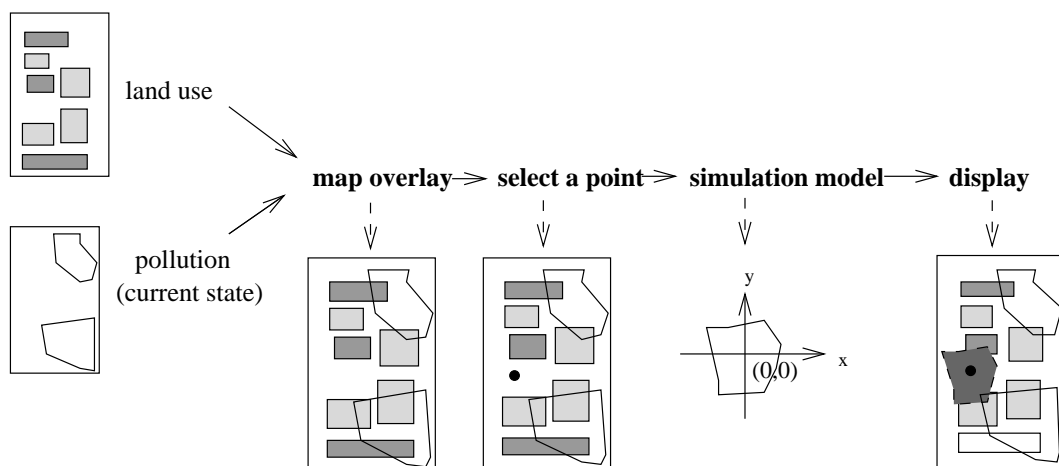


Figure 1: Example of enduser operations

An enduser might browse metadata and data before applying the operations on data needed for the specific task. The issues involved are similar to those envisaged in the above example. In the next section we introduce a service oriented layered architecture which contributes to solving these issues by a declarative specification of services.

2.2 The Four Layers

Our decomposition of the global open GIS design into levels of abstraction comes naturally after looking at distributed system architectures. More precisely, it is motivated by the following abstractions found in various distributed systems, and in particular in distributed databases:

- abstraction from low level communication to site-dependent services,
- distribution transparency: No location of services anymore,
- "view transparency": Data and / or operations may be combined,

- the notion of application: It utilizes services and it can be compared to the application layer of communication protocols.

These considerations lead us to the proposal of the following layers:

1. The *application level* is concerned with application-dependent enduser requests (such as a map overlay). These requests are forwarded to any of the subsystems (e.g., a database, a simulation model, a GIS), but in a transparent distributed system this is hidden to the enduser. Then requests are not addressed to any system in particular, but to the open system as a whole. Note that the application level is concerned with operations on data, but not with presentation which is the scope of this paper.
2. The *abstract services level* has a uniform view of the overall system. It is a platform for data and operations integration. Data either existing in different systems or generated by enduser requests can be stored in what we refer to as a virtual database, which is a logical unification of the underlying data sets.
3. The *concrete services level* has a view of precise operations which can be asked of each system. In particular, it is in charge of managing the distribution of tasks among systems and finding the appropriate “capabilities”.
4. The *system services level* deals with invocation of services to the specialized systems, such as a system call to a GIS in order to get a particular map or to perform a map overlay operation.

Two remarks are noteworthy. First, what happens below the last level cited above is beyond the scope of our proposal. For example, we do not discuss the pros and cons of synchronous RPC for database interaction. Second, one might wonder why a metalevel is not part of this organization. As a matter of fact, a metalevel is orthogonal to the levels described above. Metaconcepts such as a dictionary of operations addressable to the overall system, or, at a low level, the list of exact services that can be asked of each systems, are integrated at each level and are used to map a level onto the next one.

2.3 The Layer ODS Model

The overall system is described by the notion of *operation*, *data* and *session* (ODS concept) that it handles. While data and operations are similar to abstract data types, sessions relate the operations to be performed to each other: a kriging algorithm in a geological application can never be performed until all the required data has been supplied and the result cannot be presented before the algorithm succeeds, etc. Sessions keep track of these kinds of logical dependencies between operations. At each level the ODS concepts communicate (in an abstract sense) with concepts defined on the same or on adjacent levels. Below is a description of these three concepts.

2.3.1 Data

It corresponds to the objects of interest on each layer. Whereas these may be maps (vector or raster) on the screen at the application level, they are, for instance, collections of data in the DIGEST format or in the Spatial Data Transfer Standard (SDTS) at the base level [St97].

2.3.2 Operations

On each layer the operations to be performed are specified. For instance, a “map overlay” addressed to the whole system may, on a lower level, correspond to a service invocation to a GIS. In contrast to ADTs, operations need not be part of a data object’s specification. An

overlay is an operation on two data objects ($map \times map \rightarrow map$). Therefore it is unnatural to model a map as an ADT with an overlay operation. A further difference results from the requirements of openness: when defining a geographic object it is by no means clear that all operations to be performed on an object already exist. Someone might invent a tricky new algorithm on maps in the future. In an open GIS one must be able to incorporate it.

2.3.3 Session

The causal relationship between operations implementing higher level operations is expressed by a graph. On the application level it is typically a sequence of operations (e.g., *overlay the maps, then call a given simulation model, then ...*). On a lower level, the overlay is split into operations with a finer granularity: *fetch map A and map B, then apply overlay algorithm X*. At the system service level, if the system considered is a DBMS, a session corresponds to a standard database session. By grouping things together, sessions establish a scope and control structure.

A session is a logical relation among operations of the same level. Hence it includes a partial time-ordering aspect (i) to express sequences of instructions (*perform action X before action Y*) and (ii) to represent the fact that some operations can be performed in parallel (such as fetching arguments from different systems).

Many tools are of interest to specify sessions, such as data flows, Petri nets [Rei85] and tools borrowed from operations research and database systems transactions. A general and simple representation of session is a directed acyclic graph (DAG), whose nodes are operations or sets of operations of a given level. There is a one-way edge from op_i or $Op_i = \{op_i\}$ (“composed instruction”) to op_j or $Op_j = \{op_j\}$ if op_i (resp. Op_i) is supposed to be performed before op_j (resp. Op_j). In the sequel, we shall adopt the following notation. If $\langle \rangle_g$ denotes a session constructor, then a session at a given level n is represented as:

$$S_n = \langle OP_{ni} \rangle_g, \text{ where} \\ OP_{ni} = op_{ni} \mid \{op_{ni}\}, \text{ and } op_{ni} \text{ is an operation defined at level } n.$$

To express a given session instance, the list constructor $\langle \rangle$ is used to encompass a series of instructions: A session starts with \langle and ends with \rangle . An operation within a session is written *operation (list of argument)*. Operations are separated by “;”. To express the fact that many operations can be performed in parallel, we use the set constructor $\{ \}$. We use a simplified meta-syntax *à la* BNF with constructors $\{ \}$, $[]$, $\langle \rangle$ and $\langle \rangle_g$ as introduced above.

The (extended) Backus-Naur Form describing a session is:

```

<session> ::= '<' <operation-sequence> '>'
<operation-sequence> ::= operation-or-set-of-operations* ';'
operation-or-set-of-operations ::= operation | '{' operation-sequence '}'

```

Section 4 gives example of sessions corresponding to the reference example at each level.

An operation of a given level is translated into a session of the inferior level by a special module which is part of the kernel of the open GIS. This module can be seen as a particular contributing system, which is invariant even though the configuration changes. The necessary variables to accomplish this translation are metavariables of the overall system. However, for optimization reasons, there might be a difference between a session suggested by this module and the session really executed (for instance, some operations can be factorized or executed in

parallel after flattening sessions). Therefore we shall make a difference between a “suggested session” and a “real session”. Note also that a given operation can lead to many suggested sessions, which induces a high level graph with all possibilities. One of the graphs will be chosen and will lead to a real session.

In the sequel, the term session denotes a “suggested session” unless stated otherwise.

Keeping track of sessions is clearly fundamental to manage histories and to allow undo operation. Undo is a general operation on the *real session* DAG, which reads it backward.

3 Description of the Layers through the ODS model

To describe our architecture we follow a top-down approach. Starting with the application level, we define a model for each layer which encompasses the ODS concepts. We refer to all concepts introduced as *types*. Instanting these types for a particular environment (e.g., specific system configuration and particular actions) leads to *values*. In the pseudo-code given below, `/* ... */` denotes comments in a sequence of instructions.

3.1 Application Level

3.1.1 Data

Data corresponds to the objects manipulated by the enduser. This data is independent of any representation. What the user sees and interacts with are “maps” (or “themes”) or parts of them, as well as images (rasters) and standard data (e.g., text, numbers). To handle data at this level, many types of conceptual models exist which are either based on an entity viewpoint or on a space viewpoint of the data. We do not elaborate on this here. For information about such models the reader is reported to [HT96, Wor94]. In order to give the reader an insight into such a model, below is a simplistic entity-oriented conceptual model for (vector) maps. The main objects are of type `map` which are composed of `geographic-objects`. Each such object has two parts, an alphanumerical one (or description) and a spatial one. In addition to standard types (such as string, integers, etc.) to specify descriptions, basic spatial types such as points, lines, polygons are considered to express the object’s geometry. In addition, a geographic object can be atomic or complex (made of other objects).

```
type map : {geographic-object}
type geographic-object : [description, geometry]
                        | [description, {geographic-object}]
```

3.1.2 Operations

In transparent distributed systems the operations that endusers have in mind are independent of any system. Rather they belong to a set of operations one is likely to perform on maps and geographic objects. These are for instance: Map overlay, clipping, windowing. Although there exists a kernel of operations defined on maps, the set of operations at this level is application-dependent and can include more sophisticated operations such as the shortest path between two cities, etc. Users’ requests are addressed via the user interface by either a command language, or pop-up menus or graphical specifications. Describing a language for the invocation of the operations is out of the scope of the current proposal. An enduser operation corresponds to a series of operations defined at the inferior level. These are denoted `abstract-services` and are described in next subsection.

```
type user-operation : [operation-name, <argument>]
```

In addition to the application-dependent operations is a family of constant operations. For instance, it is clear that at this level, the user interface must offer a way to obtain maps, either by querying the overall system (*Give me the last map of the repartition of population in terms of age in California*), or possibly by offering a catalogue of the available maps. The query results will serve as input to enduser operations, whose result will be in turn assigned to entities at this level by giving them a logical name. Hence the most representative constant operations at this level are “pointing out” and “assignment” (denoted respectively `point-out` and `=` in the pseudo-code below). The main characteristics of this set of operations is that if the application changes, it remains unchanged.

3.1.3 Session

A (user) session is a graph of commands addressed by the user to the whole system (`user-operations`). At this level a session is no more than a list of operations together with parameters:

```
type session : < user-operation >g
```

The user session of the reference example is given by the following list of operations, where `land-use`, `pollution`, `newmap1`, `newmap2`, `newmap3` are the maps considered at the application (interface) levels, and `p` denotes a point. This is transparently handled by mouse clicks and popup menus if the operations are called from a graphical user interface. Note that where maps `land-use` and `pollution` are coming from and where the resulting maps are stored is not relevant here.

```
<                                     /* beginning of a session */
point-out (land-use);
point-out (pollution);
/* whatever the means are, the relevant maps are pointed out */
newmap1 = overlay (land-use, pollution);
/* map overlay invocation from the user interface
and assignment to ‘‘newmap1’’ */
p = click (newmap1);
/* select a particular point p on map newmap1 */
newmap2 = simulation-model (my-model-name, newmap1, p);
/* assuming the simulation model returns a map */
newmap3 = overlay (newmap2, newmap1)
>                                     /* end of a session */
```

Note however that assignment is transparent: For example, point `p` is probably only pointed out by the enduser and stored only locally.

At this level the notion of session is essential for managing histories and for allowing endusers to perform “undo” operations.

3.2 Abstract Services Level

3.2.1 Data

For a given user session this level sees all data from the different systems as a unified set of data. There is still no difference between systems, hence the repository where data may come from is

not relevant. (Geographic) data is represented according to a virtual data model not detailed here. Extensions to geographic types of [BNPS94] or [PAGM96] and of course of data warehouses [Wid95] (see below) are well-suited for this purpose. The data model at this level must be general enough to take many representations into account. It is fundamental that each object manipulated by the enduser be unique. Thus there is the notion of object identifier (surrogate) at this level. The set of data considered is a virtual collection of objects which may be distributed over systems, and it can be stored in a database, as shown on Figure 2. Only a minimum of information is stored at this level, and the data belongs to systems as part of materialized views.

Data management on the abstract services level has strong relationships to data warehouse management. The virtual database of the abstract services level could be viewed as a geo-data warehouse. We do not only abstract from distribution and heterogeneity at this level. The information hiding principle also suggests to make all implementation details transparent. Views may be materialized or answers may be generated on demand; materialized views may be refreshed periodically by the producer (the information source) or by means of active mechanisms. To provide for different strategies is essential in an open GIS setting: Maps are static in general and operations on maps are expensive. Therefore the results, say an overlay, should be materialized in most cases. On the other hand temperature data are valid only for a short period of time (given they are not averaged). An on-demand access would be appropriate in this case. It is obvious that users should not be bothered with these kinds of details, as long as they do not have an influence on their solution - which could happen with a refresh strategy. Other implementation aspects only influence performance, like the demand-driven query processing as opposed to materialization. Consequently they are not visible on the abstract services level.

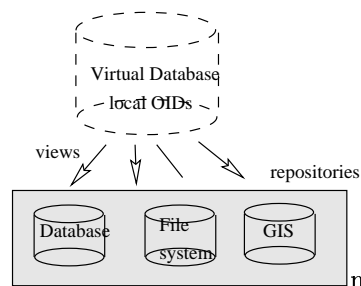


Figure 2: Virtual database at the abstract services level

3.2.2 Operations

At this level the term *operations* denotes the operations which can be directed to systems in general, i.e., without knowledge of the systems in which they are implemented. In other words, if two systems are able to perform a certain operation such as a map overlay, this is still transparent here. These operations are denoted **abstract-services**:

```
type abstract-service : [abstract-service-name, <argument>]
```

What is important is to see all the possible operations as sets of capabilities, and to decompose the user operations above into more elementary operations which can be handled at this

level. In particular, basic actions which can be performed in parallel are expressed at this level.

Similarly to the application level, on the abstract services level is also considered a family of operations that are not part of the underlying systems (like, e.g., map overlay is), but which apply nevertheless on data of the virtual data model. These are constants of the overall system. Take for example the general notion of fetching arguments (objects), which clearly belongs to this family. The term “fetching” may not be really appropriate: Arguments can already exist in the virtual database. We shall term the action of “getting” or “defining” an argument “logical define”, or `logdef` for short. It will in turn be decomposed into operations on the concrete service level.

Performing a “virtual union” of maps is also part of this collection of operations. The need for this operation occurs when a user works on a map which is made up of several other maps. As an example, take the merger of a map of France and a map of Germany. Suppose that an enduser needs to consider it for a spatial allocation operation (e.g., a study is made to build a supermarket near Saarbrücken, a city at the border between these two countries). The user may not know that the two maps may come from different repositories. What s/he sees on the screen is a “unified” view of these two maps. What is happening at the abstract services level is in fact a virtual union of these two maps, as expressed by the following session:

```
<
{of = logdef (france-map); og = logdef (germany-map)};
/* get the arguments from a system or from the virtual database */

map-res = union (of,og);
/* map-res: intermediate variable */

oum = logdef (map-res);
/* define ‘‘logically’’ the union of the two maps */
>,
```

where `union` is an operation either defined in a system such as a DBMS or which could be implemented with a view mechanism adapted to this case (which in particular takes the scale into account). In any case it is part of the invariant set of operations. Once this map is logically defined, enduser operations can be performed on map `oum`, such as “find the best spot for opening a supermarket” (based on a given set of optimality criteria).

3.2.3 Session

A session is a graph of abstract services. Above is an example of part of a session. In the reference example, map overlay corresponds to the following session:

```
<
{ol = logdef (land-use); op = logdef (pollution)};
/* the 2 instructions above can be executed in parallel */

map-res = overlay (ol,op);
/* overlay: abstract-service, map-res: intermediate variable */

omr = logdef (map-res)
>
```

3.3 Concrete Services Level

At this stage the peculiarity of each system is taken into account. There is an analogy between systems considered here and the megamodules defined in [WWC92], where “megamodules are internally homogeneous, independently maintained software systems.” Hence at this level we consider in particular the operations to address to systems.

3.3.1 Data

Because we know the interface to the underlying systems, at this level data is specified in the format of each system (although the the systems are still encapsulated). We can thus mention, say `map1`, as an argument of a map overlay and assume that it has – after conversion from a different representation – the proper format for the system which will perform the operation.

Encapsulation of information providers is another task for data warehouse systems. The only aspect in our approach related to encapsulation is format transformation. Providers of information or operations are expected to register their exports in the metadata catalogue. Thus issues related to building wrappers are not covered in our architecture.

3.3.2 Operations

These correspond to operations addressed to specialized systems. We denote them `concrete-services`. Each operation has the form:

```
type concrete-service : [SYSTEM-NAME:concrete-service-name, <argument>]
```

A given `concrete-service` is an operation to be performed by system `SYSTEM-NAME`. It is similar to an instruction in the megaprogramming language, and the concrete services layer could be implemented using that language. In addition, it provides useful features at this level, as for instance the primitive “EXAMINE” which causes a megamodule to provide a performance estimate. A `concrete-service` will be mapped in turn into a `system-invocation` graph.

In the set of constant operations at this level is the change of format for geodata, denoted `change-format` (`from`, `to`, `data`) in the pseudo-code. This can also be performed by the invariant module which provides the overall system with general services.

3.3.3 Session

A session is a DAG of `concrete services`. To get back to our reference example, suppose that the land use map is stored in a database which we refer to as “DB1”, that the pollution map is stored in a GIS “GIS2” and that the map overlay is performed in the GIS “GIS2”. The (concrete services) session is then:

```
<
{map1 = DB1: GET(ol); map2 = GIS2: GET(op)}
/* get the objects from their repository */

map1' = change-format (DB1, GIS2, ol)
/* prepare the right format for the first argument of the overlay */

map-res = GIS2:overlay (map1', map2)
/* invoke the GIS overlay operation and assign the result to
```

```
map-res */
>
```

3.4 System Services Level

This level is concerned with particular systems. Describing the possible interfaces to the systems or modules as well as the systems themselves is beyond the scope of our architecture. (Meta)data concerning the participating system is described in Section 4.

3.4.1 Data

Data is expressed according to private data structures using particular data models and formats. It serves as input of the systems. Note the difference with the previous layer where data were only *referred to* in the various underlying formats.

3.4.2 Operations

The operations are the ones addressable to a specific system. Once again we can make the distinction between (i) constant operations, namely opening and closing a system, and (ii) system-dependent operations. Defining the exact interaction with the system through system calls (e.g., *open DB*, and then *open cursor, ...*) is pointless here. The system call mechanism could be implemented using the dictionary proposed in [WWC92]. We propose a similar implementation by using the metadata that we describe in the next section.

3.4.3 Session

A session is a list of commands (queries) executed by a system. If a system involved is a DBMS, a session has the meaning used in the database domain. Because we consider encapsulated systems, what is happening within such a system session is not directly accessible to the structure we propose. However, some global tasks like query optimization can be forwarded to the systems.

4 The Role of Metadata

Defining only a layer decomposition is not enough: One has to know how to get from one stratum to the adjacent one in the hierarchy. This applies to the three ODS concepts. Two kinds of metadata coexist in a generic open GIS: Invariant metadata (i.e., application independent metadata), and “dynamic” metadata about both the underlying systems and the data itself. For a survey about the latter the reader can refer to [MSNRW91, GV97].

In this section we concentrate on the decomposition and abstraction of layers through meta-concepts. We start with the definition of global variables of the open GIS, which serve as arguments for the definition of the mapping functions. We then describe the use of these variables for switching from one level to the lower one.

4.1 Global variables

The following application-dependent metadata are instantiated with a given open GIS configuration:

- ◊ A `systems-operations` variable gives the set of operations (together with their arguments) that can be asked by the user to the whole system via the user interface. This variable is used exclusively at the user interface level, where all possible operations are encapsulated.

```
type systems-operations : {operation-name (<argument-type>)}
```

For a given configuration, it can be:

```
the-systems-operations = {overlay (map, map),
                          clipping (map, rectangle),
                          windowing (map, rectangle),
                          best-way (point, point, network-map)
                          simulation-model (model-name, point, map)}
```

- ◊ The `operation-translation` table gives for a user operation the corresponding graph of operations at the abstract services level:

```
type operation-translation : {[user-operation,
                               <abstract-service>g]}
```

- ◊ The `task-execution-plan` gives a corresponding graph of concrete services for all abstract services:

```
type task-execution-plan : {[abstract-service,
                              <concrete-service>g]}
```

- ◊ `dictionary` is a table which for all systems gives the set of operations that they are able to perform:

```
type dictionary : {[systemID, {service}]}
```

Note that this variable can be represented the other way around, by expressing for each service the set of systems able to perform it. At implementation time, this variable plays a fundamental role for optimization, in order to find the optimal resources given some global system description such as traffic congestion on the net. We call it `capabilities`:

```
type capabilities : {[service, {systemID}]}
```

- ◊ The `system-calls` variable gives the session of operations asked to a system:

```
type system-calls : {[system-name, <system-invocation>g]}
```

Besides the graph of system operations above we need to be able to set up a connection with the systems, hence to know the exact procedure for opening the communication with them. We shall call the variable `system-call-protocol`. Its study is beyond the scope of this paper.

- ◊ The `format-translation-table` gives the way of translating each format into another format (or the place where the appropriate procedure `proc` can be found):

```
type format-translation-table : {[from, to, proc]}
```

This variable differs from the other ones presented here because it will be used by a system which is not a participating system. Thus one could consider that it belongs to the static components of an open GIS.

All these variables are sets which can be extended with new elements for a new configuration of an open GIS. This point is crucial to achieve openness.

4.2 Decomposition Functions

There is an implementation / abstraction relationship between each two adjacent levels. Data, operations and sessions on level n are implemented by corresponding concepts on level $n + 1$. Implementation means decomposition in our case: A map viewed as a single entity on a higher level may be implemented as an overlay of two separate maps on a lower one. We therefore call the functions decomposition functions. Normally, there are many possible decompositions of one item. However, the metadata dictionary fixes exactly one function which decomposes data, operations and sessions onto the next lower level. On the other hand, the metadata dictionary also implements the abstraction function. It maps concrete objects of level $n + 1$ to the abstract objects on level n .

Decomposition functions are thus high level functions used to express the translation of concepts from a given level to a lower level. These are invariant metafunctions of an open GIS. For each such function, we adopt the following syntax:

```
function-name ( operation-name,  
                <argument>,  
                <metaconcept>,  
                operations-graph ),
```

where `operation-name` denotes an operation to be performed at the upper level (application, abstract services, or concrete services), `<argument>` its parameters, `<metaconcept>` the list of the above metaconcepts necessary for the transition, and `operations-graph` the session at the lower level, whose elements are in turn `operation-names` (of the lower level). Because we want to remain general we just give the concepts - the metainformation - we can use in the following mapping functions description.

4.2.1 From application level to abstract services level

The function is able to decompose the users' requests into tasks understandable by the overall system. We shall call it `parsing`. It uses the `operation-translation` table to fill out the abstract services graph. It is expressed by:

```
parsing ( user-operation,  
          <argument>,  
          operation-translation,  
          <abstract-service>g )
```

Defining an abstract services session means not only decomposing the operations but also managing the data by use of the `logdef` primitive. Hence the `{abstract-service}g` graph above is completed with `logdef` primitives to handle data at this level (from the virtual database).

4.2.2 From abstract services to concrete services

The function is a binding function. It can be expressed as a join between the abstract services session with the `task-execution-plan` variable, and is generally expressed as follows:

```
binding ( abstract-service-name,  
          <argument>,  
          task-execution-plan,  
          <concrete-service>g )
```

In order to define an optimal concrete services session some intelligence can be introduced with heuristics (i) to find the most appropriate order of tasks and (ii) to find data (<argument>) from the right repository.

4.2.3 From concrete services to system services

At this level one needs to know how to invoke the systems. This is done by using the `system-calls-dictionary` Data also has to be converted into the appropriate format. This is done by reading the `format-translation-table`. The general function is described as:

```
system-call ( concrete-service-name,
              <argument>,
              <system-calls-dictionary,
              format-translation-table,
              <system-invocation>g )
```

Figure 3 sums up the different layers as well as the mapping functions to the next layer. For each layer described on the vertical axis on the left-hand side of the figure, the main concepts are illustrated on the middle through the map overlay example. Metainformation necessary for the mapping functions appears on the right hand side of the figure.

4.3 Optimizing sessions by using metaconcepts

In Section 2.3.3 we made a difference between a “suggested session” and a “real session”. A suggested session is what comes naturally after reading the global variables. However, because many systems can be able to perform a given operation, there is not always a 1:1 mapping between a session at a given level and a session at the next level. On Figure 4, there is an explosion of possibilities when the knowledge about the systems comes into the picture, i.e., from the concrete services level to the system level. One branch of the tree (or one tree of the forest) will be chosen and will lead to a real session.

To tackle optimization issues, we have to make the distinction between compiled and interpreted environments. In the former, which is the case for traditional GIS, optimization is easy to consider because we have knowledge about all operations to be performed. Sessions can be flattened, and the sequence of operations they contain can eventually be optimized. In the latter (e.g., in real-time systems), a session is decomposed right away into a session at the inferior level and there is no control over the global set of sessions.

5 Conclusion

In the near future defining interoperable – i.e. open – GIS as sets of existing software systems will be a major challenge within the GIS community. Although some issues involved in open GIS implementation have already been addressed by many research groups (e.g., database systems, GIS, software engineering) nothing has been done, to our knowledge, to declaratively specify the steps necessary to the conceptual design of open GIS in a unified way. A remarkable exception is the OGIS architecture of ODC. It also emphasises a layered approach, but in a more pragmatic way without a unifying specification framework on different layers. In this paper we have proposed an abstraction decomposition for the design of an open GIS. This decomposition consists of four levels: The *application* level, the *abstract services* level, the *concrete services* level, and the *system services* level. Each level is characterized by the same set of concepts: *Operations*, *data* and *session* (ODS concepts). Metaconcepts are integrated at each level and

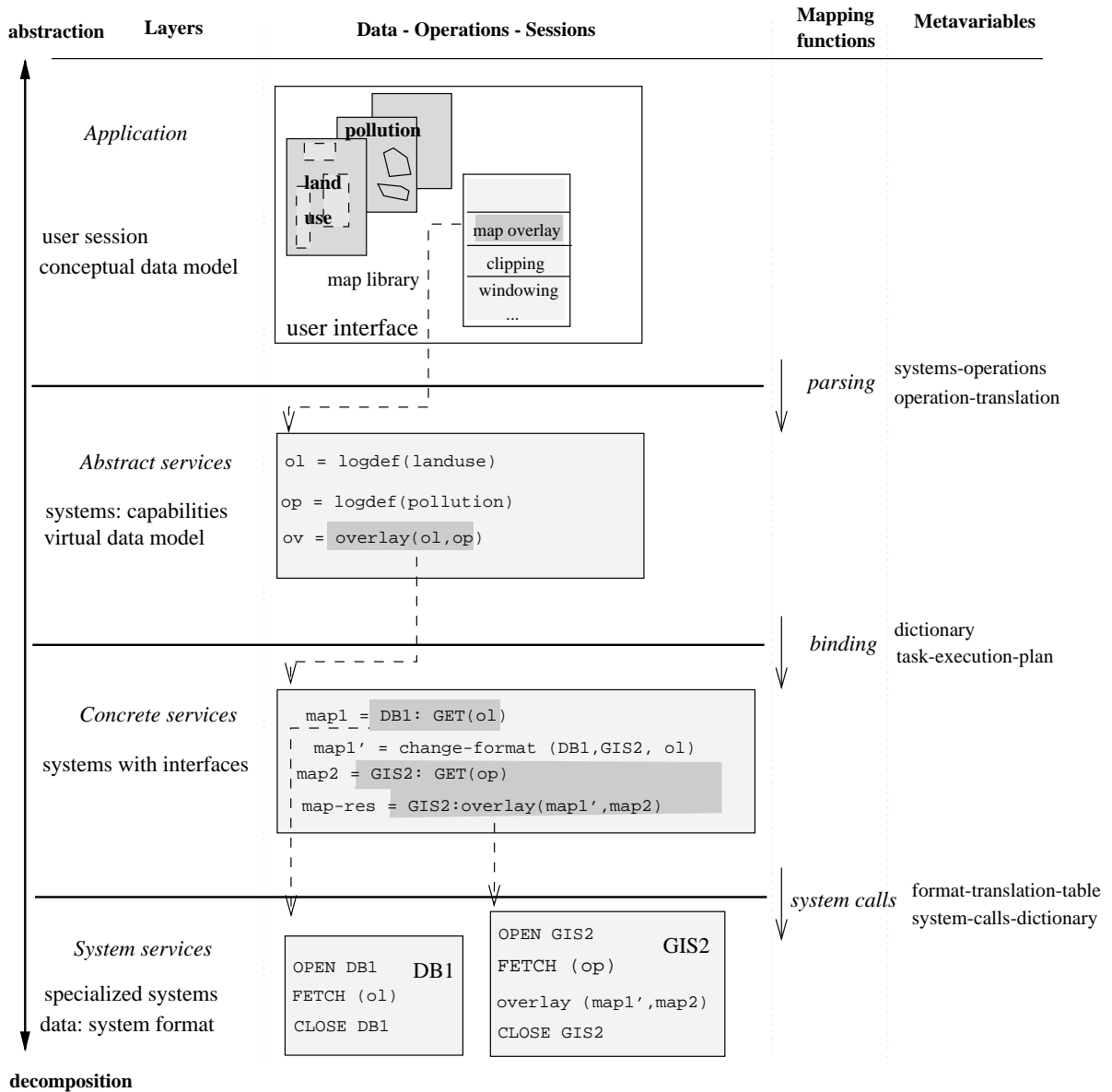


Figure 3: Our abstraction decomposition of an open GIS

they represent the key elements for mapping a layer onto the lower one. We insist on the fact that our framework hinges on extensibility, hence the role of these data is essential to achieve extensibility and openness of the overall system. Plugging a new system or a new format into the open GIS means updating the global variables. The participating system only have to export the relevant information they contain. Finally, we have described the method of switching to the next layer in the hierarchy by using these metaconcepts together with generic high-level functions.

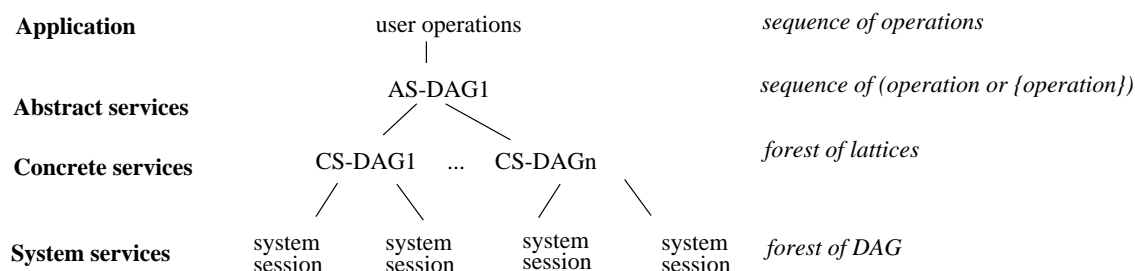


Figure 4: Session mapping: Decomposition tree

The methodology proposed here applies well to transparent distributed systems. Note however that the notion of transparency is not always applicable if a user wishes to refer to level $n+2$ or $n+3$ from level n . For instance, the action of fetching specific arguments does not fit in this framework if a user wants a map *from a particular system*. In this case, the notion of transparency does not apply with systems which serve as repositories (e.g., DBMS). The same reasoning applies if endusers address a given system to run a specific algorithm. Then the transparenting is broken (this phenomenon is similar to disencapsulation in object-oriented environments). We believe that designing a real transparent distributed system is a challenging task in the GIS area, as it implies the consideration of both the overall system and the ontologies of contributing systems.

Decomposing the system into levels of abstraction independently of the representation turns out to be very powerful at implementation time because it brings two levels of flexibility. Firstly, each level can be implemented by using well-known techniques and standards. For instance, the data set of the abstract services level can be manipulated through a DBMS (independently of any possible DBMS that could serve as a participating system). The implementation of the concrete services level can be carried out by using megaprogramming concepts [WWC92]. Choosing the right system to perform a given operation is implemented by introducing heuristics in the binding function. As another example, take a World-Wide Web geographic application over Internet and suppose that you need to browse a map library to find the most appropriate map for a given study. Such a mechanism is easily implementable in our framework by defining a browsing routine at the concrete services level. With the encapsulation of each level, implementing a level by using a given technique has no impact on the other levels.

Secondly, decomposition enables a clear distribution of tasks among all layers and a control on the overall system. The modularity of the whole architecture can be illustrated by query optimization (in our context we should rather call it “session optimization”). This is achieved at each level independently, from the highest level to the system level (although we do not have any control on it at the system level, optimization is forwarded to systems able to handle it). We believe that the kind of modularity offered by our framework is fundamental in the design of open GIS.

Our ongoing and future work consists in a refinement of many points of this framework and in prototyping. We are currently exploring the extension of data warehouses and materialized views to geodata for handling data at the abstract and concrete services level. In parallel, we are working on a specification language for describing the sessions at each level. Eventually, this

will be an extension of database transaction concepts, although classical database transactions usually refer to rather short actions. Our goal in the long-run is to design a prototype based on these foundations using standard tools.

References

- [AKC94] D. Abel, P. Kilby, and M. Cameron. A Federated Systems Approach to Design of Spatial Decision Support System. In *Spatial Data Handling*, pages 46–59, 1994.
- [Aut96] Autodesk. MapGuide Product Description, 1996. Available at <http://www.mapguide.com/mbr-main.htm>.
- [AYA+92] D. Abel, K. Yap, R. Ackland, M. Cameron, D. Smith, and G. Walker. Environmental Decision Support System Project: An Exploration of Alternative Architecture for Geographic Information Systems. *Intl. Journal on Geographical Information Systems (IJGIS)*, 6(3):193–204, 1992.
- [BNPS94] E. Bertino, M. Negri, G. Pelagatti, and L. Sabattella. Applications of Object-Oriented Technology to the Integration of Heterogeneous database Systems. *Distributed and Parallel Databases*, 2:343–370, 1994.
- [Cor97] *The Common Object Request Broker: Architecture and Specification*, 1997. Published by Object Management Group and X/Open, revision 2.1.
- [FK95] A. U. Frank and W. Kuhn. Specifying Open GIS with Functional Languages. In M. J. Egenhofer and J. R. Herring, editors, *Advances in Spatial Databases (SSD'95)*, pages 184–196. Lecture Notes in Computer Science No. 525, Springer-Verlag, Berlin, 1995.
- [GMS+96] O. Günther, M. Müller, P. Schmidt, H. Bhargava, and R. Krishnan. MMM: A WWW-Based Method Management System for Using Software Modules Remotely. Technical Report TR-96-044, International Computer Science Institute (ICSI), Berkeley, CA, October 1996.
- [Gov94] U.S. Government. Coordinating Geographic Data Acquisition and Access: The National Spatial Data Infrastructure. Office of the Press Secretary, The White House, USA, April 1994.
- [GV97] O. Günther and A. Voisard. Metadata in Geographic and Environmental Data Management. In W. Klas and A. Sheth, editors, *Managing Multimedia Data: Using Metadata to Integrate and Apply Digital Data*. McGraw Hill, 1997. To appear. Also available as ICSI Technical Report No. TR-96049.
- [HT96] T. Hadzilacos and N. Tryfona. Logical Data Modeling for Geographic Databases. *Int. Journal on Geographical Information Systems (IJGIS)*, 1996.
- [MSNRW91] D. Medykj-Scott, I. Newman, C. Ruggles, and D. Walker, editors. *Metadata in the Geosciences*. Loughborough, Great Britain, 1991.
- [OGI96a] OGIS Technical Committee and the Open GIS Consortium, Inc. The Open GIS Abstract Specification, 1996. Available at <http://www.ogis.org/public/abstract.html>.
- [OGI96b] OGIS Technical Committee and the Open GIS Consortium, Inc. The Open GIS Guide (Part I): Introduction to Interoperable Geoprocessing, 1996. Available at <http://www.ogis.org/guide/guide1.htm>.
- [PAGM96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In *Int. Conf. on Very Large Databases (VLDB)*, 1996.

- [Rei85] W. Reisig. *Petri Nets: An Introduction*. Springer-Verlag, Berlin, 1985.
- [St97] *Geographical Information Standards*, 1997. WWW page on standards prepared by the European Community. <http://www2.echo.lu/oii/en/gis.html>
- [VS94] A. Voisard and H. Schweppe. A Multilayer Approach to the Open GIS Design Problem. In *Proc. of the 2nd ACM-GIS workshop*, ACM Press, N. Pissinou and K. Makki (Eds.), New York, N. Y., 1994.
- [Wid95] J. Widom, editor. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, volume 18. IEEE Data Engineering, June 1995.
- [Wor94] M. Worboys. Object-Oriented Approaches to Geo-Referenced Information. *Int. Journal on Geographical Information Systems (IJGIS)*, 8(4), 1994.
- [WR92] H. Waechter and A. Reuter. The ConTract Model. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 7, pages 219–263. Morgan Kaufmann, San Mateo, CA, 1992.
- [WWC92] G. Wiederhold, P. Wegner, and S. Ceri. Toward Megaprogramming. *Communications of the ACM*, 35(11):89–99, 1992.