

Accessing Geographical Metafiles through a Database Storage System

Stephen Blott*

Andrej Vckovski†

Abstract

We describe a database storage extension for geographical metadata, discuss the retrieval requirements of such an extension, and describe the extension process itself.¹ Our aims in undertaking the work reported were twofold: on the one hand we wanted to better understand the basic requirements of a geographical metadata manager, and on the other we wanted to “stress-test” the storage model of our prototype storage system [19]. We discuss the following issues: What are the retrieval requirements of a geographical metadata manager? In what architectural contexts must such a manager operate? How can a database system be extended to meet both these classes of requirements? Characteristic of our approach is that such metadata remains primarily stored in files external to the database system, while indexing and query processing is carried out within the database system. We also report on our experiences in building such a prototype geographical metadata manager.

1 Geographical Metadata and Database Systems

The handling of metadata has gained importance in recent years, particularly within scientific and engineering disciplines. This importance is mostly related to the rapidly increasing amount of primary data being produced and exchanged within these various domains. In the field of geographical systems, the cost of data acquisition and datasets’ sheer volumes heightens this importance further, making data discovery a key challenge in the computerisation of the discipline.

1.1 Geographical Metadata and Standards

A widely-used definition describes metadata as *data about data*; this is sometimes generalised to *information which makes data useful* [13]. From the data

*Institute for Information Systems, ETH-Zentrum, CH-8092 Zürich, Switzerland; blott@inf.ethz.ch.

†Institute for Geographical Information Processing, University of Zürich, CH-8057 Zürich, Switzerland; vckovski@gis.geogr.unizh.ch.

¹The term *metadata* has technical meanings in both the fields of geographical and database systems. To avoid confusion, we use the term *geographical metadata* in discussion from a geographical point of view, and *metafiles* when discussing geographical metadata from a database point of view. Database schema information is not discussed.

producer's point of view, metadata documents a dataset. The US Federal Geographic Data Committee (FGDC) and other organisations have created standards to support the uniform assessment of geographical metadata [8, 7, 12]. These standards cover content and semantics of metadata, that is what information is documented, and also define its digital representation. We will use the FGDC's *Metadata Content Standard* as our primary reference for geographical metadata [8]. A pseudo extract from such a geographical metadata file is illustrated in Figure 1. Such metadata is used with varying levels of detail in the following tasks:

Quality control: Detailed metadata is essential for quality control within organisations which produce and maintain large spatial datasets. It documents the entire life-cycle of data including its collection, pre-processing and maintenance [14]. This kind of metadata is usually very detailed.

Data exchange: Due to the high cost of acquisition, data exchange plays a major role in spatial information processing. The geographical metadata accompanying a dataset is the key to successful exploitation of that data. To be of value to a large community, metadata's contents must adhere to some standard, and therefore be accurately interpretable by its final users. Missing or misinterpreted metadata is one of the major sources of avoidable errors and uncertainties when using external datasets. Less detail is required of such metadata; however large datasets are usually accompanied by correspondingly detailed metadata.

Data directories: The need to reuse expensive datasets motivates the establishment of data directories. These allow users to browse content descriptions of datasets without requiring access to the primary data itself. For example, they may be interested in locating data sources covering a specific geographic region. Directories might be in-house within large organisations, or describe the contents of digital libraries available from other sources. Such geographical metadata typically requires the lowest level of detail.

Details of the production process are not required, and in contrast to the cases above the dataset itself need not be present. That is, metadata has storage and querying requirements independently to the primary data itself.

The first two of the examples above do not require geographical metadata to be available as a separate entity, it can rather be seen as a part of the primary data. Geographical Information Systems (GIS) usually offer some means of storing short descriptions of a dataset, its list of themes or its geographic region. The whole range of metadata content, however, is not supported by current systems. While it is an important requirement for a GIS to offer comprehensive handling of the metadata, this alone will not address the problem of metadata management within data directories. In those systems, the primary data being described is usually not available, and the information system applied need not be a GIS. The metadata may stem from datasets managed by a variety of different systems, each having its own mechanism for metadata management. Due to their ubiquity, such data is often managed directly within operating-system files. Common tools for its management include simple paper forms, scripts and custom database systems.

1.2 Database Systems and Geographical Metadata

As part of our on-going research activities, we have developed a storage-management kernel named CONCERT [19, 3]. This kernel is designed to support object managers such as RDBMSs, OODBMSs, engineering systems and geographical systems. Such object managers extend the CONCERT kernel by embedding new types and their corresponding operations in the kernel; in a sense they *program* the kernel. The kernel's storage model is novel, and is based on the management of abstractly-defined objects. New types are added to the system as new implementations of a fixed set of abstract data types. To be sure that that model is sufficiently rich, we decided to "stress-test" it by *programming* it for the task of geographical metadata management. The variety of classes of information encompassed by such metadata, including spatial attributes, provides a very broad test of the flexibility of the storage model.

Databases supporting geographical applications must support the storage, querying and analysis of structured data, spatial data and its metadata. Current application of database technology, particularly with respect to metadata, is limited. There are a number of reasons for this. Firstly, the modelling and query-language concepts for metadata access remain poorly identified and supported; and secondly, the storage and query-processing strategies necessary are more complex than those supported by existing commercial systems. Though addressing the former is important (see, for example, [18]), only the latter is considered here.

One specific problem is the commitment associated with *one-giant-leap* solutions. Database systems typically require that objects are represented in the storage model of the system at hand, and operated upon only under the control of that system. This gives rise to a number of major problems in practice. Firstly, it implies that data must be translated from a data-exchange to a database-storage format. This implies the development of translation code, translation costs, and that pre-existing scripts and applications can no longer be used. The second problem is that mapping geographical metadata files to the relations or classes of a database system typically de-clusters the attributes of the file [22]. Different attributes will appear in different relations or classes, and the complexity of standards such as [8] is beyond that for which current commercial systems could provide the necessary (re-)clustering functionality. We conceive of our approach as being evolutionary rather than revolutionary: initially files are the primary storage representation, while over time we would like to support smooth migration to a database-system solution.

For both these reasons we decided to retain the file representations of geographical metadata files. In our approach, the primary representation is the file; however, secondary indexes are constructed within the database system, and query-processing across these indexes and the files themselves is managed from within the database system. For these purposes, it is necessary that the database system interpret the contents of those files, that is not treat them simply as BLOBS, and achieving this is one of the contribution of this paper. In some senses our problem is similar to those addressed by [1, 10, 5, 11], in which techniques for accessing (object-oriented) database objects through file-system interfaces (or *vice versa*) are described. Our contribution here differs from these, however, in a number of respects: firstly in that it elaborates a specific application-area problem, secondly in the complexity of the objects

<code>Data_set_scale</code>	1:250,000
<code>Identification_code</code>	23
<code>Data_set_description</code>	This dataset was derived from the 1:250,000-scale source by the U.S. Geological Survey for the Land-Use/Land-Cover program ...
<code>Theme_keyword</code>	hydrologic unit, river basin
<code>Data_set_G-polygon</code>	inside(49.10,-123.27 49.15,-116.27 49.16,-104.06 49.40,-95.12 48.12,-88.40 ...)
...	

Figure 1: Pseudo Extract from a Geographical Metadata File

described, and thirdly in its special treatment of spatial data.

Document Structure. The remainder of this paper is structured as follows. In the following section we discuss the retrieval (and hence storage-management) requirements of geographical metadata. We then, in Sections 3 and 4, map those requirements to a database storage model and show how a metadata manager based on such a model can be built in practice. Finally, Section 5 concludes.

2 Metadata Retrieval Requirements

Structure of Geographical Metadata. The FGDC standard [8] supports data producers by defining the information to include within a dataset's metadata. It defines a set of attributes and their meaning. Due to the scope of different datasets to be documented, a very large set of attributes was needed. These were grouped and structured hierarchically in a similar manner to other geographical metadata standards [16]. The FGDC standard exposes a nested, tree-like structure of metadata entries, where each leaf is a particular *data element* and the nodes are *compound elements*. The wide scope of the standard's application leads to many optional attributes. Further, non-standard, instance-specific attributes may extend the standard where necessary.

Each *data element* is of one of the basic data types **integer**, **real**, **text**, **date** and **time**. The *domain* of a data element describes valid data values that can be assigned to that element. For example, the domain may constrain a real-element describing latitude/longitude coordinates to the ranges $[-90, 90]$ and $[-180, 180]$. An important data-element type is **textual**, for which certain instances are constrained to contain only certain strings. An example of this is the **Metadata Security Classification** which is constrained to contain the values such as **Top secret** and **Secret**, etc.

Compound elements are defined through production rules, describing the relationship between the compound element and its components. These are

```

1. SELECT (summary-or-entry) WHERE attribute operator value
   SELECT (summary-or-entry) WHERE PROGRESS = "Complete"
   SELECT (summary-or-entry) WHERE BNDGCOORDS OVERLAPS
   (12.2,45.5,13.1,48.2)

2. SELECT (summary-or-entry) WHERE any OVERLAPS
   (12.2,45.5,13.1,48.2)
   SELECT (summary-or-entry) WHERE any LIKE "%soil-type%"

3. SELECT (summary-or-entry) WHERE exists attribute
   SELECT (summary-or-entry) WHERE exists THEME

```

Figure 2: Example SQL-like queries over Geographical Metadata

formed through record, list and union constructions over sub-components, which are themselves either data elements or nested compound elements. Production rules specify which of components are *mandatory*, *mandatory-if-applicable* or *optional*. A number of the compound elements can rather be regarded as indivisible data elements of compound structure. Examples of this include `Data_set_G-polygon`, which is implemented as a list of point-pairs, but logically represents a spatial polygon. For effective storage management and query processing, this spatial interpretation is the one required, and the representation is typically of no concern.

Retrieval Requirements of Geographical Metadata. The retrieval modes fall into two basic classes. If a query yields several applicable metadata entries, then a list of *summary information*² is returned. This provides an overview of each matching metadata entry. If the result is a uniquely identified metadata entry, then the whole entry is retrieved. A typical user interface would allow the user to select items from the list of summary information and then retrieve the detailed metadata.

The query types are similar to queries within library systems. One can consider queries over arbitrary specific parts of the hierarchical structures; such would be similar to queries over nested relations [21]. However, other views are also important for query processing. *Universal type queries* retrieve entries where *any* data element or compound element of a given particular type matches the query predicate. For example, a query may apply to *any* textual or *any* spatial attribute. The third category of queries retrieves entries for which a specific attribute exists. Queries for existence also include entries where the specified attributed is a *null-value* as defined in [8]. This accommodates missing information: if the attribute with missing information is *applicable* to the metadata entry, then the attribute is encoded using a data-type specific null-value. Missing information that is *not applicable* is not encoded and can be detected via queries for existence. This corresponds to the different types of null values discussed in [6] and [4]. Some examples of SQL-like metadata queries are given

²The summary information is basically the content of the compound element **Identification Information** (ASTM tag `IDENTINF0`).

in Figure 2.

All queries allow both individual data elements and compound elements to be specified. Queries for contents of compound elements must accommodate the semantics of the compound element's components. For example, compound elements defining a spatial object (e.g., `Data_set_G-polygon`) might be queried using spatial operations such as `enclosed-in` or `overlaps`. Type information is required, therefore, over specific compound elements in addition to simple data elements.

3 Developing a Storage Model for Geographical Metadata

Our starting point is FGDC [8], and our assumption is that we have some set of metafiles conforming to that geographical metadata standard. The first issue to address is that of how the contents of such files can be made known to a general-purpose database system such that they may be indexed, perhaps replicated and queried from within the database system.

3.1 An Abstract-Object Storage Model

Our solution is based on our prototype storage kernel CONCERT [19]. CONCERT provides exactly two base types (`SCALAR` and `UNKNOWN`), and exactly five constructed types. These are `RECORD` and `LIST`, `UNION`, `REFERENCE`, and `CONTINUUM`. Two novel aspects of CONCERT are the treatment of externally-defined types in terms of their *likeness* to built-in types, and the treatment of the `CONTINUUM` type as a basic, common model of arbitrary-dimensional extended objects. As both details are important in the current context, we sketch them below.

Likeness. CONCERT provides a number of built-in types, and built-in implementations of these. However, these may in certain contexts prove inadequate, for example to meet performance particular requirements or accommodate existing externally-defined representations. Metafiles are an example of the latter. In these cases, new implementations of the known types may be embedded in the system, and their semantics made known through their *likeness* to the known types. For example, the basic structure of a metafile is that of a sequence of attribute-value pairs; hence their structure is *like* that of the built-in `RECORD` type, though their implementation is externally-defined. With respect to Figure 1, example attribute names would be `Data_set_scale` and `Identification_code`, with types `scaleT` and `text-intT`, respectively. Based on such a *likeness*, any physical-design or query-processing strategy which is applicable to records, is applicable also to these externally-defined records.

The types `scaleT` and `text-intT` are then themselves also externally-defined types, defined to be *like* the built-in `SCALAR` type. They both have (different) textual representations, but they behave something *like* scalars. Any physical-design or query-processing strategy which applicable to scalars, is applicable also to these externally-defined scalars. Types declared through such *likenesses* are referred to as *abstract-object types*.

The CONCERT kernel performs storage management and query processing

to the level of selection and projection.³ To achieve this, the kernel must be able to manipulate objects in accordance with the *likenesses* declared. Hence, operations must be provided over abstract-object types which prove—in a sense—those *likenesses*. For example, over the metafiles with their *likeness* to **RECORD** types, an operation would be required to **project** the component attributes of the record. Over the attributes themselves, which in the examples above were *like* the built-in type **SCALAR**, comparison (<, =) operations are required. These then suffice, for example, for access structures and the query evaluator to interpret the contents of metafiles. For example, one of the basic access structures in the kernel is a B-tree, and this could be used to index metafiles over any attribute which is *like* the **SCALAR** type. The record operations allow the individual attribute to be extracted, and the comparison operations determine how they are managed within the B-tree.

Spatial Objects as Continua. The second novel aspect of CONCERT is its treatment of spatial data as a special basic type named **CONTINUUM**. Continua enable the special semantics of spatially-extended objects to be embedded within the kernel. As with the other types, new **CONTINUUM** types must provide implementations of particular operations. In this case, they are **partition**, **compose**, **is-empty**, **interval** and **overlaps**.⁴ When a *likeness* to a **CONTINUUM** type is declared, scalar types must be provided for each of its dimensions, as must implementations of the operations above over the new type. Take, for example, the **Data_set_G-polygonT** of the FGDC standard. Assume that the necessary **SCALAR** type **textual-float** is already declared. Attribute values of this new polygon type are declared to be *like* the **CONTINUUM** type in two dimensions of scalar **textual-float**. With the correspondingly-required operations, such externally-defined polygons can be indexed within access structures such as spatial grid-files and R-trees. This approach is a generalisation of that of our earlier prototype DASDBS [9, 20], and also has similarities with that of point-sets pursued in [17].

3.2 Describing Metafiles as Abstract-Object Types

Given metafiles' formats [8], and the storage model sketched above, we now describe those files' contents in terms of that model. This turned out to be relatively straight-forward.

Treatment of Errors. There are a number of reasons for which metafiles may be treated as erroneous: for example, they may be syntactically incorrect, or may include attribute values outwith the admitted ranges of those values. To function correctly, the storage system must be able to establish whether a given object is well-formed or not. One possibility is to treat this as a higher-level problem; however, we felt that it was possible to address it at a lower level and therefore we should. We considered two approaches, both based on **UNION**

³In this respect, the CONCERT kernel is at approximately the same architectural level as the RSS of System R [2], the Core of Starburst [15], or the DASDBS Kernel [20].

⁴The necessary operations are presented here from a logical point of view. In practice, particular care has been taken in defining the necessary operations such that: 1) trivial implementations are always possible and always logically correct; and 2) highly-specialised resource management is possible. Such details are particularly important, for example, if one considers the management of large raster-image data sets where excessive copying would present an unacceptable overhead.

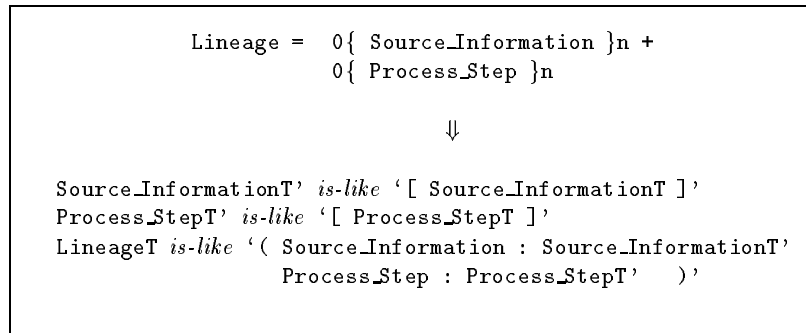


Figure 3: Mapping FGDC Types to CONCERT Type *Likenesses*

types. The first solution would be to encode in each base type an abstract **UNION** which is of one alternative if that attribute is well-formed, or of the other if it is not. We discarded this option as being of an unnecessarily fine granularity, with costs associated with almost every operation applied to leaf attributes. The second solution was to describe an entire file as being *like* a **UNION** between **correct-dataT** and **incorrect-dataT**. We chose this solution as it is simple and allows a dataset to be partitioned horizontally into well-formed and ill-formed partitions as new objects are inserted.

Metafile Base Types as Abstract-Object Types. A number of standard representations are used within FGDC. These are described easily in terms of their likenesses to the built-in base types. For example, calendar-dates, times-of-day, and degrees-of-latitude and -longitude are all described as being *like* the **SCALAR** type. Other values such as network addresses are described as being *like* the built-in **UNKNOWN** type, and hence are managed as unstructured objects. Since errors are managed at the top level, all constrained textual attributes are also described simply as being like the built-in type **SCALAR**. Spatial attributes are described in terms of their *likeness* to **CONTINUUM** types.

Constructed Abstract-Object Types. The basic constructions of metafiles are records, lists and unions, and these map straight-forwardly to the **RECORD** and **LIST** and **UNION** types of the storage model. For example, at the top level metafiles consist of a sequence of named attributes. These are described as abstract **RECORD** types. For attributes themselves, some additional types had to be inserted. An example of such a description is given in Figure 3. The top part is an extract from [8], and the bottom is the corresponding CONCERT types. The first step is to un-nest the set constructions of **Source_Information** and **Process_Step**, thereby generating two new types. The second is to map the metafile specification concepts of set and record to the storage-management concepts of **LIST** and **RECORD**, respectively.

3.3 Supporting Metafile Queries through Multiple Representations

Section 2 identified a number of special requirements of geographical metadata retrieval; these included *uniform type querying*, and the ability to retrieve objects on the basis of the existence of particular attributes. Both these aspects can be considered to be multiple-representation problems. These can be described in the CONCERT kernel through the use of RECORD-likenesses with computed attributes. Assume that an entire metafile's structure is described by the abstract-object type `MD-structureT`. Additional metafile representations (or views) are described by additional (computed) attributes.

```
MD-viewT is-like '( MD-structure : MD-structureT,
                   MD-strings   : [ stringT ],
                   MD-temporal  : [ timeT ],
                   MD-spatial   : [ Data_set.G-polygonT ∪ CoordinateT ],
                   MD-attributes: [ SCALAR ] )
```

The representation to query is selected by `project`-ing the appropriate attribute. Were one interested in *any* metafile containing the string “`soil-type`”, one would query the `MD-strings` attribute. Were one interested in any object defining the attribute `THEME`, one would query the `MD-attributes` attribute.

4 Extending the Database System

Operations must be provided over abstract-object types in order that the kernel be able to manipulate abstract objects in terms of their *likenesses*. Exactly which operations are required is dependent on the type. In the current context, two issues arise in this respect:

Number of Operations. Firstly, while only a few examples were given in the previous sections, several hundreds of attributes are present in practice. This then implies the need for a very large number of operations to implement the necessary abstractions. We concluded that generating all those operations by hand would be problematic and error-prone. Instead, we developed a tool to generate these automatically. This is in keeping with our context of programming the kernel on behalf of a higher-level data manager.

Operational Model. The operations concerned must extract information from the metafiles themselves; this must be done as efficiently as possible. We chose a general implementation strategy for those operations based on their being interfaced directly to the buffer manager.

4.1 Automatic Generation of Extraction Operations

Standard tools such as `lex` and `yacc` provide some help in parsing the contents of structured files. However, using these alone the specification of the operations remains complex; further, adding new operations to interpret instance-specific attributes would require further detailed `lex` and `yacc` programming. However, large parts of geographical metafiles are regularly structured; sufficiently so, we concluded, that it should be possible to generate a large percentage of the

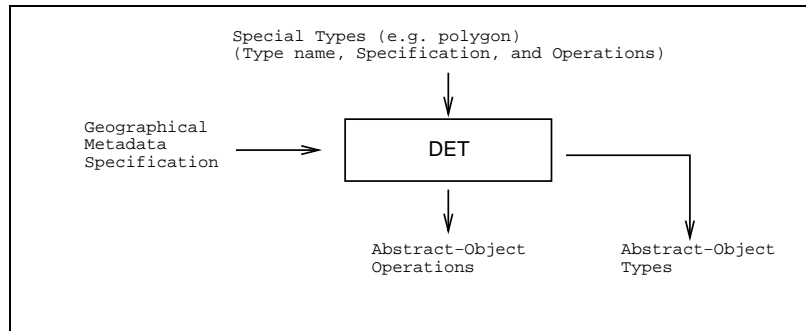


Figure 4: The Inputs/Outputs of the Database Extension Tool

```

Data_set_scale      = scaleT
Identification_code = integerT
Data_set_description = freeTextT
Theme_keyword       = 0{ constText Domain "river basin"
                    "hydrological unit" ... }n
Data_set_G-polygon  = Data_set_G-polygonT
...
  
```

Figure 5: Extract from a DET Metafile Specification

necessary operations automatically. We therefore decided to develop a tool which would take a specification of a metafile-format as input, and generate both the `CONCERT` types and operations as output. We refer to this tool as the *Database Extension Tool* (DET). However, not all metafile attributes are regularly structured. Some compound elements, such as `Data_set_G-polygonT`, have semantics beyond that of their structure alone. Therefore, the DET must also accept the specification of some special types as inputs. The functionality of the DET is illustrated in Figure 4. Its inputs are a specification of the metafile format and the special types for which automatic generation is inappropriate, and its outputs are the `CONCERT` types and their corresponding operations.

The notation of the metafile specification language is broadly similar to that used in the specification document [8]. It allows various attributes to be specified: that is, their name and their type. Attribute types include a number of base types, a textual type, constrained textual types, unions, nestings, and the special types. An example extract from such a specification is given in Figure 5, this corresponds to that metafile extract given earlier in Figure 1. The types `Data_set_G-polygonT` and `scaleT` are examples of the special types which are not built into the DET. The DET generates types as described in the previous section; for example, it accommodates potential syntactic errors

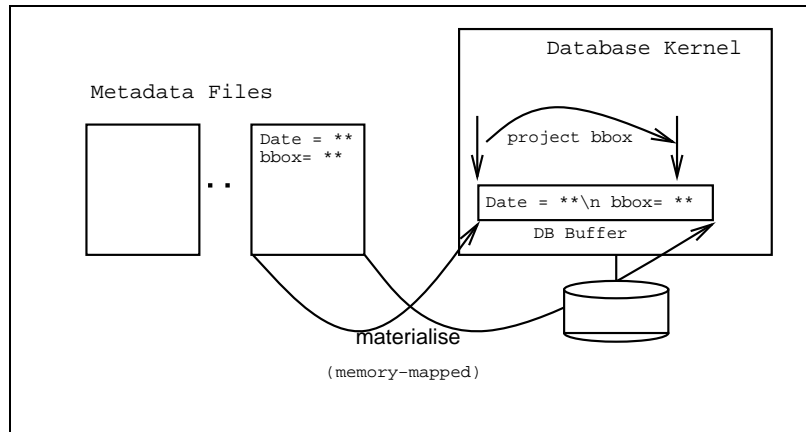


Figure 6: Materialising and Extracting Information from Metafiles

through the use of a **UNION** type, and replicates and aggregates all textual, temporal and spatial attributes and attribute names to the top level to support universal querying. The corresponding operations are generated automatically by the DET.

4.2 Operational Model

We had to decide on an operational model of what would actually happen when an operation over an externally-stored metafile was called. Clearly the cost of executing such operations should be minimised; however, other issues also arise. For example, multiple operations may be executed against the same metafile at the same time. This may result from a single query requiring multiple pieces of information from a file, from concurrent queries, or from parallelisation of a single query. In any of these cases, we would like that at most a single copy of the file be managed dynamically within the database, that copies of parts of that file only be made when necessary, and when access is no longer required, that the file be discarded and the resources it occupied be freed. These requirements are similar to those expected of a traditional database buffer.

The persistent representation of metafiles within the database system consists simply of the filename of the corresponding metafile. These are modelled within our storage manager as being *like* the built-in reference type. For any such reference, the operations **materialise** and **dematerialise** must be provided. In **CONCERT**, buffer management is based on direct (and uniform) access to database segments and files through the use of memory mapping [3]. Metafiles can therefore be accessed directly within the database buffer by *fixing* the appropriate file as if it were a database segment. By doing this, the **materialise** operation establishes addressability of metafiles within the virtual address space of the database system. Correspondingly, the **dematerialise** operation is implemented by *unfixing* the relevant buffer pages. Dynamic references, therefore, are simply virtual-memory pointers to metafiles' contents

within the database buffer. All remaining operations must interpret the contents of the metafiles within virtual memory. This is done by navigation within virtual memory, without files' parts having to be copied. This situation is illustrated in Figure 6. When the file is materialised, it is mapped into the database buffer and made addressable in the database system's address space. A pointer to the first byte of the file is the dynamic representation. Operations such as projection (**project bbox**, in the Figure) are implemented by searching the file in virtual memory for the corresponding attribute's name, and returning a pointer to the attribute's value.

Based on these techniques, the metafile operations interact directly with the lowest levels of the database system, and benefit from the resource-management services provided at that level. For example, concurrent access to a single file results in only a single copy being in the buffer. That copy remains addressable while materialised (fixed), and the buffer's replacement policy applies to determine when it is best discarded. This is only possible since the **materialise** and **dematerialise** operations are generated by the DET, and not by arbitrary application programmers. That the buffer manager provides uniform addressability over multi-page objects [3] implies that neither the operations generated by the DET nor those associated with the special types need be concerned with page boundaries within the buffer.

4.3 Prototype

We have developed a prototype of the geographical metadata manager described here. It supports all the basic functionality discussed above except that its operations are not yet interfaced directly to the database buffer in the way described (though their implementations are as described). A detailed description of this implementation is given in [24].

5 Summary and Prospects

We have described the extension of a database storage-management system to the management of geographical metadata. This extension is based on the assumption that data resides in externally-stored files conforming to existing standard representations [8], while indexing, replication and query processing are performed from within the database system. We described the requirements of such a metadata manager, showed how externally-defined metafiles can be described in the storage model of our CONCERT prototype, and described how the very-large number of operations demanded by such a database extension may be supported by automatic generation. Our choice of FGDC as our standard was based on our particular application-area specific experience. An obvious alternative would be SGML, and we believe that our prototype could be easily adapted to that representation. An important aspect of our approach is that the data remains stored in the externally-defined files of the operating system, while it is indexed and queried from within the database system. This is somewhat similar to the approach to document management taken in the Rufus project [23].

Our aims in undertaking this work were two-fold: firstly to better understand the requirements of a geographical metadata manager, and secondly to

stress-test the storage model of our CONCERT prototype. Despite the richness of the classes of metafile discussed, we always were able to describe the structures we required in our abstract-object storage model. Hence, we were able to support the functionality required of a metadata manager. One potential problem was that at several stages we were faced with a number of possibilities for the storage types to choose. This then implies that some form of data modelling is taking place at the storage-management layer, while such should really take place at a higher level. The other side of the same coin, however, is that we were able to describe multiple representations (or views) of the data directly within the storage system, thereby supporting the development of physical designs such as replication and indexing over those views. For example, we conceive of coupling our system to a textual indexing engine to provide improved support for textual querying.

A further consideration is how our approach extends to distributed environments. Clearly, some of the basic techniques applied (such those of memory management) are single-site in nature. However, one of the potential advantages of managing metadata within a database system is that the technology of distributed and federated database systems can be applied at a higher level. Coupling the access to externally-stored metafiles within a database system to technology for distributed access offers the potential for smooth migration to widely-available meta-databases.

In addition to *proof-of-concept*, the purpose of our prototype is to support improved querying of metafiles through the use of physical-design and query-processing support within a database system. We do not currently have sufficient data with which we can perform experimentation with such designs, nor do we have representative query classes and frequencies with which we could evaluate both those designs and our prototype as a whole. Were it to become available, such a benchmark suite would prove invaluable in the next phases of our project.

Acknowledgements

The authors would like to thank Thomas Etter who undertook the majority of the implementation work described, and also Gisbert Dröge, Lukas Relly, Hans-Jörg Schek and Andreas Wolf for their many helpful discussions and important contributions with respect to this and earlier related work.

References

- [1] S. Abitoboul, S. Cluet, and T. Milo. Querying and Updating the File. In *Proceedings of the 19th Conference on Very Large Database Systems*, volume 19, Dublin, Ireland, 1993.
- [2] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlain, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorrie, P. R. McJones, J. W. Mehl, G. R. Potzolu, I. L. Traiger, B. W. Wade, and V. Watson. System R: Relational Approach to Database Management. *ACM Transactions on Database Systems*, 1(2):97–141, 1976.

- [3] Stephen Blott, Helmut Kaufmann, Lukas Relly, and Hans-Jörg Schek. Buffering Long Externally-Defined Objects. In *Proceedings of the Sixth International Workshop on Persistent Object Systems (POS6)*, pages 40–53, Tarascon, France, September 1994.
- [4] K. Brassel, F. Bucher, E.-M. Stephan, and A. Vckovski. Completeness. In *Spatial Data Quality*. Elsevier Applied Science, London, 1995. in preparation.
- [5] Michael J. Carey, David J. DeWitt, Michael J. Fanklin, Nancy E. Hall, Mark L. McAuliffe, Jeffrey F. Naughton, Daniel T. Schuh, Marvin H. Solomon, C. K. Tan, Odysseas G. Tsatalos, Seth J. White, and Michael J. Zwilling. Shoring Up Persistent Applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, USA, 1994.
- [6] E. F. Codd. Missing information (applicable and inapplicable) in relational databases. *SIGMOD RECORD*, 15:53–78, 1986.
- [7] (US) Federal Geographic Committee. *ASTM Draft Specification*. U.S. Geological Survey, 590 National Centre, Reston, Virginia 22092, USA, 1994.
- [8] (US) Federal Geographic Committee. *Content Standard for Digital Geospatial Metadata*. U.S. Geological Survey, 590 National Centre, Reston, Virginia 22092, USA, (anonymous ftp: fgdc.er.usgs.gov), 1994.
- [9] Gisbert Dröge, Hans-Jörg Schek, and Andreas Wolf. Erweiterbarkeit in DASDBS. *Informatik Forschung und Entwicklung*, 5:162–176, 1990 (in German).
- [10] N. H. Gehani, H. V. Ragadish, and W. D. Roome. OdeFS: A File-System Interface to an Object-Oriented Database. In *Proceedings of the 20th International Conference on Very-Large Database Systems*, pages 249–260, Santiago, Chile, September 1994.
- [11] Illustra Information Technologies (Inc). *Illustra Users Guide*. 1111 Btoadway, Suite 2000, Oakland, CA 94607, Illustra Server Release 2.4.1 edition, March 1995.
- [12] Arbeitsgruppe Geographische Informationssysteme. SIK-GIS Empfehlungen 1992. Technical report, Schweizerische Informatikkonferenz, Bern, September 1992.
- [13] Francis P. Lenz, H. J. Bretherton and Paul T. Singley. Metadata: a user's view. In James C. French and Hans Hinterberger, editors, *Seventh International Working Conference on Scientific and Statistical Database Management*, pages 166–173. IEEE Computer Society Press, September 28–30 1994.
- [14] H. J. Lenz. The conceptual schema and external schemata of metadatabases. In James C. French and Hans Hinterberger, editors, *Seventh International Working Conference on Scientific and Statistical Database Management*, pages 160–165. IEEE Computer Society Press, September 28–30 1994.

- [15] B. Lindsay, J. McPherson, and H. Pirahesh. A Data Management Extension Architecture. In *Proceedings of the 1987 ACM SIGMOD Conference on Management of Data*, pages 220–226, San Francisco, CA, USA, May 1987. ACM SIGMOD.
- [16] National Aeronautics and Space Administration (NASA). *Directory Interchange Format Manual*, April 1993. Version 4.1.
- [17] Jack A. Orenstein and Frank A. Manola. PROBE Spatial Data Modelling and Query Processing in an Image Database Application. *IEEE Transactions on Software Engineering*, 14(5):611–629, 1988.
- [18] Ramesh Jain and Arun Hampapur. Metadata in Video Databases. *ACM SIGMOD Record*, 23(4):27–33, December 1994.
- [19] Lukas Relly and Stephen Blott. Ein Speichersystem für Abstrakte Objekte. In *Proceedings of the 1995 Conference Datenbanksysteme in Büro, Technik, und Wissenschaft (BTW95); (in German)*, pages 338–347, March 1995.
- [20] Hans-Jörg Schek, Heinz-Bernhard Paul, Marc H. Scholl, and Gerhard Weikum. The DASDBS Project: Objectives, Experiences, and Future Prospects. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):25–43, March 1990.
- [21] Hans-Jörg Schek and Marc Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2), 1986.
- [22] SEQUOIA 2000 Metadata Schema for Satellite Images. Metadata in Video Databases. *ACM SIGMOD Record*, 23(4):42–48, December 1994.
- [23] K. Shoens, A. Luniewski, P. Schwarz, J. Stamos, and J. Thomas. The Rufus System: Information Organisation for Semi-Structured Data. In *Proceedings of the 19th International Conference on Very-Large Database Systems*, pages 97–107, Dublin, Ireland, August 1993.
- [24] Thomas Etter. Geospatial Metadata: Standards and Storage-Management Considerations. (Diplomarbeit thesis, Institute for Information Systems, ETH Zürich, Switzerland; in German), February 1995.