An Open Abstract-Object Storage System

Stephen Blott Lukas Relly Hans-Jörg Schek
Institute for Information Systems, ETH-Zentrum, 8092-Zurich, Switzerland
E-mail: {blott,relly,schek}@inf.ethz.ch

Abstract

Database systems must become more open to retain their relevance as a technology of choice and necessity. Openness implies not only databases exporting their data, but also exporting their services. This is as true in classical application areas as in non-classical (GIS, multimedia, design, etc).

This paper addresses the problem of exporting storage-management services of indexing, replication and basic query processing. We describe an abstract-object storage model which provides the basic mechanism, '*likeness*', through which these services are applied uniformly to internally-stored, internally-defined data, and to externally-stored, externally-defined data. Managing external data requires the coupling of external operations to the database system. We discuss the interfaces and protocols required of these to achieve correct resource management and admit efficient realisation. Throughout, we demonstrate our solutions in the area of semi-structured file management; in our case, geospatial metadata files.

1 Introduction

Database systems must evolve from closed data vaults to open data services. Today's systems require all data to be owned by the DBMS. Data is accessed only through query-language and programming interfaces. Functionality not supported through these must be implemented at the application level. Much work on extensible database systems aims to extend the functionality of these interfaces.

We make here, however, also a complementary observation: *that the functionality of a database system is only available over objects owned by the database system.* This motivates us to consider how that functionality can be exported, and a database system provide *database services* over data of external repositories. For the future, we conceive even of database systems not necessarily owning data, but rather providing only these database services.

We believe that database systems must become more flexible, coexisting cohesively with other repositories. Database management systems should become brokers of information, coordinators of dependencies, and providers of database services. These services then become the tools of software engineers in developing (distributed) applications over heterogeneous (existing) components.

By database functionality we mean primarily the following key services: query processing, query optimisation, indexing and replication for improved query and update performance, and transactions for the management of concurrent usage and recovery. In the extreme, we envisage a database system exporting only these services, and managing only metadata about the repositories it serves: how to manipulate their objects, who is authorised to access them, and what dependencies exist among them.

We feel our vision is consistent with research and commercial directions in general. Object exchange environments such as Corba and Ole/Com [Obj95, Ber95] provide basic mechanisms for passing data and operations between repositories and applications. IBM's Garlic [CHP+95] provides access to data, the individual parts of which are distributed across a number of other repositories. TP-monitors and other middleware products [Obe94] provide coordination and transaction management without the data-management functionality of a fully-fledged database system. An earth-science database manages relationships between objects (satellite images) and schedules their processing, without itself owning the images [BS95].

In this paper we focus on the storage-management aspects of a 'data-less' DBMS. The functionality we consider is indexing, partitioning, replication, and basic query-processing. Figure 1 illustrates our approach in the case of the external repository being a file system. Key points are the following. **Forward Compatibility**: existing, external applications continue to access

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3.50

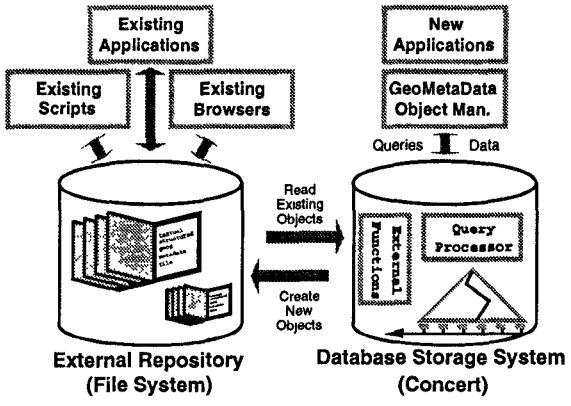

Figure 1: Overall Architectural Solution

existing, external data. **Querying External Data**: new applications exploit the 'value-added' functionality of the database system to query external data. For efficiency, this requires indexes and replication of key data parts within the database system. **External Physical Design**: physical-design strategies of the database system are applied to external data within their external repositories. This is illustrated by the newly-created small 'files' in Figure 1. We also consider using external services to augment the functionality of the database system, although this is not discussed further here [SW93].

The current approach extends our earlier DASDBS work [SPSW90]. We combine the techniques of complex objects and externally-defined types to develop a storage model for structured, abstract objects. This is embodied in our newer CONCERT prototype, a short report on which has been published previously in German [RB95]. Our work also has similarities with other ADT approaches, and comparison with that of Illustra is provided below [Sto86, Ill95].

Our contribution is the following:

- We introduce mechanisms for indexing and querying external data, and for external physical design over the data of external repositories.
- We introduce a simple mechanism, *likeness* to known types, which provides a more uniform treatment of ADTs in databases through a surprisingly small number of concepts.
- We discuss the impact on internal database architecture, and give a practical demonstration of our approach through an extended example.

We do not address here the important issue of transaction management, but refer rather to our related work on coordination through agents in CIM environments [NWM+94], and also on transaction management in layered systems [WS92]. We also do not address join queries.

This paper is structured as follows. The next section describes our extended example in some detail. Section 3 presents our abstract-object storage model, and Section 4 discusses the realisation of our example and the new functionality. Section 5 investigates the impact of our storage model on the internal architecture of CONCERT, our prototype system. Section 6 concludes.

2 Example: Geospatial Metadata Files

Our extended example is based on semi-structured files. Examples of such are electronic mail, network news, LaTeX and HTML; in our case we chose geospatial metadata files [FGD94]. While such semi-structured data is well-suited to database processing, there are only a few examples of this technology being applied or applicable. Rufus [SLS+93] addresses document management only, emphasising an object-oriented data model for heterogeneous document collections. SHORE [CDF+94] and OdeFS [GJR94] propose mechanisms for exporting object-oriented data through file-system interfaces only, but not services. The approach of Abiteboul *et al* [ACM93] is more closely related to our own. They also restrict themselves, however, to querying and updating files, addressing neither the more general context of other repositories, nor physical design.

An adapted extract of an FGDC metadata file is given in Figure 2. It is semi-structured data including textual representations of scalar and spatial values. It is of complex, nested structure. The composite `Citation_Information` occurs firstly for the data set at hand, and recurs for the `Lineage` of that data set; similarly, the composite `Browse_Graphic` is a list of components. Some composites, however, have a higher meaning; for example, the `Bounding_Coordinates` represent a single spatial object. In full generality, arbitrary polygons can be represented.

We developed a metadata extension for our storage system CONCERT. As illustrated in Figure 1, a prerequisite was forward compatibility; that is, that existing applications, scripts and browsers can be retained. This implies that the file-system interface and data representation must be retained. Our approach is like Abiteboul *et al* [ACM93] and Rufus [SLS+93], and unlike others [CDF+94, GJR94], in that the primary repository remains the external repository.

We show how structured querying mechanisms can be applied to such external data. A prerequisite to efficient processing is that indexes and replication of key data parts be maintained within the database system.

Example queries are illustrated in Figure 3. We use an extended SQL-like syntax for explanation purposes; the actual CONCERT internal syntax is different. The first example illustrates selection and projection; it re-

```
Identification_Information:
  Citation:
    Citation_Information:
      Originator: Schweitzer, Peter N.
      Publication_Date: 1993
      Title: Modern Average Global Sea-Surface Temperature
      Online_Linkage: http://geochange.er.usgs.gov/pub/magsst/magsst.html
  Description:
    Abstract:
      The data contained in this data set are derived from  NOAA Advanced
      High Resolution Radiometer Multichannel Sea Surface Temperature
      data (AVHRR MCSST), which are obtainable from the Distributed Active
      Archive Center at the Jet Propulsion Laboratory (JPL) . . .
  Spatial_Domain:
    Bounding_Coordinates:
      West_Bounding_Coordinate:  -180.0
      East_Bounding_Coordinate:   180.0
      North_Bounding_Coordinate:  72.0
      South_Bounding_Coordinate: -66.0
  Browse_Graphic:
    Browse_Graphic_File_Name: m_augna.gif
    Browse_Graphic_File_Type: GIF
  .
  .
  Browse_Graphic:
    Browse_Graphic_File_Name: m_augnae.gif
    Browse_Graphic_File_Type: GIF
Data_Quality_Information:
  Completeness_Report:
    Included in the data set is a table enumerating the days  for which
    sea-surface temperature data were available in the source material.
    In general, images were available every week during the time period
    from 811001 through 891231.
  Lineage:
    Source_Information:
      Source_Citation:
        Citation_Information:
          Originator: Jet Propulsion Laboratory
          Publication_Date: 1991
          Title:
            NOAA Advanced Very High Resolution Radiometer Multichannel
            Sea Surface Temperature data set produced by the University
            of Miami/Rosenstiel School of Marine and Atmospheric Science:
Spatial_Reference_Information:
  Horizontal_Coordinate_System_Definition:
    Geographic:
      Latitude_Resolution: 0.01757812
      Longitude_Resolution: 0.01757812
      Geographic_Coordinate_Units: Decimal degrees
```

Figure 2: An Adapted Extract from an FGDC Geospatial Metadata File

```
SELECT
 Identification_Information
  .Citation
   .Citation_Information
    .(Originator,Title)
FROM
  Metadata_Collection MC
WHERE
 MC
  .Identification_Information
   .Citation
    .Citation_Information
     .Publication_Date = '1993';
```

```
SELECT
 Identification_Information
  .Citation
   .Citation_Information
    .Online_Linkage
FROM
 Metadata_Collection MC
WHERE
 MC
  .All_Textual
   LIKE '%sea%temperature%';
```

```
SELECT
 Identification_Information
  .Citation
   .Citation_Information
    .Online_Linkage
FROM
 Metadata_Collection MC
WHERE
 MC
  .Identification_Information
   .Spatial_Domain
    .Bounding_Coordinates
     CONTAIN '(5.7, 56.5)';
```

Figure 3: Example Geospatial Metadata Queries in an SQL-like Syntax

trieves the originators and titles of all data sets published in 1993. The second illustrates the use of textual components for retrieval. A requirement for this data is textual search over all textual components; in Figure 2, over the `Title`, `Abstract` and `Completeness_Report` components, and the recurrent `Title` of the `Lineage`. `All_Textual` is a computed attribute aggregating these components. The third query illustrates a spatial selection, which might be well supported by a spatial index.

We show also how materialised views can be maintained by the database system in external repositories. For example, consider the materialised view containing the `Originator` and `Title` of data sets published in 1993 (Figure 3, left). We show how such a materialised view can be maintained in the external repository and in the external representation. That is, the view is accessible to existing applications, scripts and browsers. This is illustrated by the small 'files' in the external repository of Figure 1.

3 Approach: Abstract Objects

As a vehicle for the investigation of open storage systems, we have developed a prototype system named CONCERT. We now describe CONCERT's abstract-object model. CONCERT's key mechanism for coupling knowledge about external objects to the database system is '*likeness* to known types'.

CONCERT supports exactly six built-in, basic and constructed types: `UNKNOWN`, `SCALAR`, `RECORD`, `LIST`, `UNION` and `CONTINUUM`. A summary of the basic operations over these types is given in Figure 5.

3.1 Unknown Types

`UNKNOWN` is a binary, uninterpreted-object type. The only operations are those for copying. These are discussed in Section 5.

3.2 Scalar Types

Almost all database systems provide built-in scalar types including integers, floats, dates and times. We now show, as others have before [Sto86, WSSH88, Ill95], how new scalar types are accommodated.

Consider the value '`1993`', a textual representation of a scalar value. If `compare` and `hashable` operations are available over such objects, then existing access structures can be applied, as can basic query-optimisation and -evaluation techniques. We can express this another way. Given scalar operations over a new scalar type, its objects can be managed *like* those of the known type `SCALAR`. We declare this as given in Figure 4, (a). With this declaration and appropriate functions implementing the required operations, textual year values such as '`1993`' are managed exactly as internal scalars are. For example, they can form the basis of selections, or be the keys of hash-based or tree-based access structures. `TEXT_YEAR_COMPARE` and `TEXT_YEAR_HASH` are the names of externally-implemented functions; detailed discussion of which is the topic of Section 5.

3.3 Record Types

CONCERT provides a built-in record type for storing and manipulating structured objects. Our implementation is standard. The basic operations are given in Figure 5.

We now repeat the argumentation used above for scalars, this time for records. CONCERT knows how to manage record values, but itself provides only a single implementation. However, given record operations over a *new* record type, its objects can be managed *like* those of the known type `RECORD`. We illustrate this with our metadata example. At the top level, our file consists of the three components:

```
Identification_Information,
Data_Quality_Information,
Spatial_Reference_Information
```

These we consider to be the three components of an

```
create type TEXT_YEAR
  like SCALAR
  with (
   compare  = TEXT_YEAR_COMPARE
   hashable = TEXT_YEAR_HASH
   );
```

(a)

```
create type META_DATA
  like RECORD()
  with (
   extract = MD_EXTRACT
   create  = MD_CREATE
   project = MD_PROJECT
   compose = MD_COMPOSE
   );
```

(b)

```
create type META_DATA_FILE
  like RECORD(META_DATA)
  with (
   extract = MDF_EXTRACT
   create  = MDF_CREATE
   project = MDF_PROJECT
   compose = MDF_COMPOSE
   );
```

(c)

Figure 4: Declarations of New Types for Geospatial Metadata within CONCERT

abstract record. In turn, the first of these itself consists of the four sub-components:

```
Citation, Description, Spatial_Domain,
Browse_Graphics
```

These we also consider to be the four components of a (nested) abstract record. The full version has seven and fourteen components to these types, respectively.

Our approach, therefore, is to apply the standard techniques of record storage and processing to our metadata files by treating these as abstract, externally-defined records. In CONCERT, the operations required over record types are: **extract** and **create** to manipulate records' components, and **project** and **compose** to generate new records from old. We declare our new type as given in Figure 4, (b). We assume also similar RECORD-like types for all the composite components of metadata files.

3.4 List Types

CONCERT provides a list type for variable-sized collections of homogeneous objects. Together, records and lists provide a storage model as expressive as a nested-relational model [SPSW90, DKA+86]. We support operations for both element-at-a-time and list-at-a-time processing; see Figure 5.

Our argumentation is the same: our database system knows how to manage lists, and therefore knows how to manage abstract lists. The **Browse_Graphic** components of Figure 2 provide an example of this. Such lists consist of a variable number of uniformly-typed objects. Their objects are managed *like* those of the known type LIST. LIST-like types also allow the incorporation of external repositories with set-oriented data and interfaces.

3.5 Union Types

Union types support variants. Their operations allow determination, extraction and creation of variants.

The **Graphical_Coordinate_Units** component of Figure 2 illustrates an abstract union. A restricted number of values are valid for this component: '**Decimal degrees**', '**Decimal minutes**', '**Decimal seconds**', etc. These are managed *like* the objects of the known type **UNION**. The necessary operations are summarised in Figure 5. In an earlier CONCERT extension [BV95], unions were used to accommodate syntactically-incorrect files.

3.6 Continuum Types

Our treatment of extended objects—for example, polygons, raster images, 3D models and time intervals—is somewhat novel and deserves more detailed discussion. The types **RECORD** and **LIST**, while they are adequate for the representation of extended objects, are inadequate for expressing the semantically-important properties of those objects. For example, while a circle positioned in 2D space can be represented as a record of two coordinate values and a radius, record operations are inadequate for expressing semantically-important properties such as whether two circles intersect.

We identified a minimal set of spatial operations which suffice for many important storage and processing tasks for extended data. This led us to introduce a new abstraction for the management of such objects, which we name **CONTINUUM**. Continua provide a single abstraction for extended objects, independent of dimensionality. The required operations are given in Figure 5.

The abstraction is based on considering extended objects to be point sets, abstractly, in n-dimensional space. The **partition** operation decomposes objects into two sup-parts: those 'points' satisfying a predicate, and those not. The result is two new objects of the same type as the original. The **compose** operation is the inverse. The **bound-box** operation returns a predicate which is true of all the points in the region. If an object is entirely in one half of a partition, then the other half will be empty. This important case, which is necessary to trim search spaces, is detected through the **bound-box** returning the predicate '**false**'. The **overlaps** and **contains** operations are standard. The compound component **Bounding_Coordinates** is **CONTINUUM**-like in our example; more generally, these can represent arbitrary polygons.

Our predicate language allows intersection or containment in arbitrary-dimensional, axes-aligned boxes to be

```
T like UNKNOWN:
  (Copying operations only, see Section 5)
T like SCALAR:
  compare   : T, T              -> int /* strcmp */
  hashable  : T                 -> int
T like RECORD:
  extract   : T[T_1, ..., T_n],
              int_i             -> T_i
  create    : T_i               -> T[T_i]
  project   : T[T_1, ..., T_n],
              projection        -> T[T_j, ..., T_k]
  compose   : T[T_1, ..., T_j],
              T[T_j+1, ..., T_n] -> T[T_1, ..., T_n]
T like LIST:
  is-empty  : T[T']             -> int /* bool */
  query     : T[T'],
              predicate,
              projection        -> T[T'']
  head      : T[T']             -> T'
  tail      : T[T']             -> T[T']
  mk-empty  : (none)            -> T[T']
  cons      : T[T'], T'         -> T[T']
  update    : T[T'], int, T'    -> T[T']
T like UNION:
  which     : T[T_1, ..., T_n]  -> int_i
  extract   : T[T_1, ..., T_j]  -> T_i
  cons      : T_i, int_i        -> T[T_1, ..., T_j]
T like CONTINUUM:
  partition : T, predicate      -> T, T
  compose   : T, T              -> T
  bound-box : T                 -> predicate
  overlaps  : T, predicate      -> int /* bool */
  contains  : T, predicate      -> int /* bool */
```

Figure 5: Operations Required over in CONCERT Types

specified. Dimensionality of objects is encoded in the data structures of predicates. While axes alignment is a restriction, we find it is adequate for many spatial access structures and queries; for example, R-trees are supported through bounding boxes and containment [BKSS90], clipping grid files through those and also `partition` and `compose` [DS93]. The `CONTINUUM` abstraction has similarities with the point-set type of Probe [OM88] and others. It was developed, however, primarily as a generalisation of our earlier approach [DSW90, DS93]. The Illustra 2D Spatial DataBlade defines a similar interface to its 2D R-tree, though without object de- and re-composition operations [Ill94].

3.7 Summary

Our approach is to manage externally-stored, externally-defined objects in terms of their *likeness* to the known types of our database system. We have shown how, abstractly, the structure of metadata files is equivalent to a database type structure. We show, therefore, how the internal techniques of physical design and query processing can be applied to this external data. This requires externally-implemented functions to be available over external types, whenever internal functions would be used for internal types. We show below how these mechanisms allow *querying of external data*, and *external physical design*.

4 Realisation and New Functionality

This section describes how the necessary functions were realised in the case of our metadata example. It also illustrates by example the new functionality of our database extension.

4.1 Realisation with a Metadata 'Compiler'

In general, external functions may be realised through those at the interface of an external repository, or through an external library. In either case, well-defined and typically stable interfaces are available. In our metadata case, we had a 'compiler' for FGDC metadata available [Sch95], and this formed the basis of our metadata extension. Discussion of this illustrates difficulties and solutions when coupling external functions to database systems.

The metadata compiler is written in C, and the source code is available on-line [Sch95]. It provides functions for parsing a metadata file, building a syntax tree over that file, and also subsequently unparsing that syntax tree back into a file. The functions CONCERT requires were implemented with those provided by the 'compiler', and also by manipulation of and navigation within the syntax tree.

Two tasks had to be achieved for our metadata extension: firstly generating the necessary types, and secondly the necessary operations. The first was relatively straight-forward since internal tables of the metadata 'compiler' described the abstract structure of a metadata file. The functions we then implemented fell into three classes: those concerning the file system, those concerning composite structure, and those concerning attribute values.

4.1.1 File-System Functions

At the top level, we modelled a metadata file through the type `META_DATA_FILE`, Figure 4, (c). As it has only a single component, the important operations are `extract` and `create`. CONCERT's internal representation of metadata files is simply the file name, stored as a string. The `extract` operation over this new type loads the file itself into memory, and then calls a function of the metadata compiler to generate a syntax tree over that in-memory file. The leaves of the syntax tree reference the in-memory file.

This is illustrated in Figure 6. When evaluating the a predicate on `Publication_Date`, CONCERT manipulates objects object only through known ADT opera-

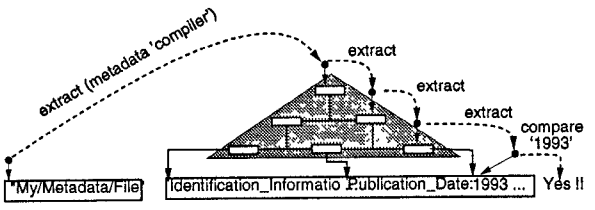

Figure 6: Evaluation of Predicates

tions (`extract` and `compare`, in this case). The implementation of the first `extract`ion function loads the data from the file, and generates the syntax tree. The `create` operation (when its result is flattened, see Section 5) is the reverse: a syntax tree is unparsed into a file.

4.1.2 Composite-Structure Functions

The operations on `RECORD`- and `LIST`-like composite types were implemented by manipulation of and navigation within the syntax tree. Because the metadata 'compiler' provided tables describing valid composites and their components, their implementation was surprisingly straight-forward.

An example is the `RECORD`-like `META_DATA` type described earlier (Figure 4, (b)). The `extract` operation was implemented by navigating one-level down to the appropriate child node in the syntax tree. The `project` operation was implemented by generating a new root to the syntax tree, whose children then reference the existing sub-trees being projected. The `create` and `project` operations were implemented by similar manipulation of the syntax tree.

These functions were actually implemented only once. The same implementations were re-used for the operations over other composite types. This was possible because internal tables of the metadata compiler described valid sub-components to each component.

4.1.3 Attribute-Values Functions

Except for `UNION`-like types, the functions for leaf-nodes were generated by hand. These included scalar types, text types and spatial types. Union types were generated from appropriate tables of variant values within the metadata compiler.

4.1.4 Discussion

We were lucky to have such well-structured, application-area specific code available to us. A more elaborate approach would have been required had this not been the case [BV95]. It is clear that only well-structured code and interfaces can be used as shown here. Commercial libraries and network services typically have such clearly-defined interfaces.

While we implemented `extract` on metadata files by loading external data into local memory, more generally this need not be the case. Alternatives include opening a connection to an external repository, or simply to an external operation service. The result may be a session identifier, and subsequent operations use this identifier to manipulate objects remotely.

4.2 New Functionality: Querying

We now illustrate how structured queries are supported against external data. Consider again the selection/projection query from Figure 3, left. For each metadata file, the evaluation of this query proceeds as follows: firstly the predicate is evaluated, and if that holds then the projection is evaluated. The predicate is evaluated by successively applying the `extract` functions of the five relevant `RECORD`-like types. For the first type, `META_DATA_FILE`, this loads the file and builds the syntax tree. For subsequent types this navigates within the syntax tree. The result is a date value, for which the `compare` operation of the `SCALAR`-like `TEXT_YEAR` determines the truth of the predicate. This is all shown in Figure 6.

The projection is evaluated by re-using the intermediate value of the predicate evaluation at the level of the `Citation_Information`, applying the two relevant `extract` operations, then the `compose` operation of the `Citation_Information` type, and finally the `create` operations of the enclosing types in inner-most to outer-most order. The unflattened result is a new syntax tree with references into the `Originator` and `Title` subtrees of the original syntax tree. This, for each object, is returned to higher-level software through a cursor for further processing.

Let us now consider the case that a physical design is applied to better support such queries. We consider the case that each of the `Originator`, `Publication_Date` and `Title` components of the `Citation_Information` are separately replicated as vertical partitions within CONCERT. Evaluation of the query need not, under these circumstances, visit the external repository at all. The predicate is evaluated by visiting the `Publication_Date` partition and applying the relevant `compare` operation to each entry in turn. The projection is evaluated by applying the `create` operations to the corresponding objects of the other two partitions, then the `compose` operation of the `Citation_Information` type, and finally the `create` operations of enclosing types, again in inner-most to outer-most order.

We now compare this with the services of the Illustra database system [Ill95]. Illustra admits new ADTs by attaching new functions to new types. We assume appropriate functions, similar to the `extract` functions described above, have been added to Illustra. In this case, predicate evaluation would proceed very much as described above. While some differences exist in

the details of resource management, these we discuss subsequently.

The result of an Illustra query is always an Illustra record. There is no mechanism for generating new objects of external representations using the internal mechanisms of Illustra. This implies that the results of the projection considered above cannot be handled. Similarly, with respect to physical design, the vertical partition supporting predicate evaluation could be managed similarly in Illustra, however those supporting projection evaluation cannot.

4.3 New Functionality: Physical Design

We now describe how CONCERT maintains materialised views in external repositories. Consider again the result of the query of the previous section (Figure 3, left). If this, for the file in Figure 2, is flattened, then a *new metadata file* is generated, containing exactly the lines:

```
Identification_Information:
  Citation:
    Citation_Information:
      Originator: Schweitzer, Peter N.
      Title: Modern Average Global Sea-...
```

This is the external representation of the query at hand, in the external repository. Hence: *the result of a database query is accessible to existing applications, scripts and browsers, which themselves know nothing of the database's involvement.* This functionality can be used to maintain extracts of important data subsets for convenient browsing, or preparing metadata sets to be shipped in the standard format. CONCERT can also use such materialised views to process queries, if these are to hand and likely to require less work.

We know of no other attempt to develop such functionality; and we are somewhat uncertain of its applicability. However, we feel it offers many opportunities. Examples include: automatic replication of data between repositories, extending classical databases with, say, spatial or textual functionality which they otherwise lack, or automatically maintaining historical data for repositories without that functionality.

5 Internals: Managing Abstract Objects

The management of abstract objects has considerable impact on the design and internal protocols of our abstract-object manager. We describe now some of these issues and their solutions within CONCERT. Once again, we illustrate this with our metadata example.

The in-memory representation of all CONCERT objects is contiguous, consisting of a memory reference, a length, and a number of flags; we refer to such as a *memory object.* The flags and their role are discussed below. Their purpose is to allow CONCERT to control resource usage, allocation and deallocation in the context of externally-implemented functions.

A buffer manager supporting uniform addressability across page boundaries [BKRS94] provides a uniform memory-object model independent of whether the target is in normal virtual memory, or the database buffer; it retains, however, the advantages of a traditional buffer with respect to paging decisions and coordination with the recovery subsystem.

5.1 Side Effects and Auxiliary Resources

The first problem we address is that of managing auxiliary resources allocated as a side-effect of an external function's invocation, but unknown to CONCERT. The metadata compiler generates a syntax tree as the result of the `extract` operation over the `META_DATA_FILE` type. Only the root of the tree is known as a memory object to CONCERT, the body of the tree is unknown. In general, arbitrary resources may be attached to a memory object; for example, file descriptors, open connections, memory, temporary files or processes. While not all classes of side effects can be accommodated, CONCERT provides a protocol for the timely deallocation of such auxiliary resources.

CONCERT knows when memory objects are created and deleted. Auxiliary resources can only be allocated through a function invocation; they must be deallocated when the corresponding memory object is deleted. To ensure resources are deallocated, all new types must provide a special operation (`deleteAuxiliary`) which is called immediately prior to a memory object's deletion. In many cases, such as that of `TEXT_YEAR`, no action is required. However, in cases such as that of `META_DATA_FILE`, an entire syntax tree must be deallocated. A similar approach was adopted in [DSW90].

CONCERT is informed that auxiliary resources are associated with an object by a flag (`HasAuxiliary`) associated with new memory objects.

The solution to this problem in Illustra is to provide new implementations of standard system calls, such as those for memory and file management (for example, `mi_alloc` instead of `malloc`, etc). Illustra guarantees correct resource management only if these Illustra functions are used. This implies that other auxiliary resources must be deallocated prior to a function invocation's completion, and cannot persist between invocations. This, in turn, rules out the implementation technique described above, and also the retention is session identifiers between invocations.

The Illustra memory manager provides mechanisms whereby memory persists until the end of either the current function's invocation, or the enclosing Illustra-SQL statement. The latter would be correct but unnecessarily conservative for the case above. The CONCERT mechanism provides more control over when resources are deallocated.

5.2 Query Evaluation and Shallow Copying

We now further address the allocation and deallocation of resources during query processing. Having generated the syntax tree, the evaluation of the query in Figure 3, left, proceeds as follows. The `extract` operations for the nested `RECORD`-like types receive pointers to the syntax tree as their inputs, and generate new syntax trees as their outputs. Clearly, generating entirely new syntax trees is redundant, as the necessary tree already exists as a sub-part of the original tree. Therefore, these operation return references into the syntax trees of their arguments. This situation is illustrated in Figure 6.

Shallow copying requires that an invocation's argument must be retained until that invocation's result is no longer required. This is the case both if the result is a reference to auxiliary data reachable from the argument, or if the result is a reference into the argument memory object itself.[1] The former case is illustrated in Figure 6; the latter arises, for example, with the built-in record implementation and also in our previous metadata prototype [BV95].

CONCERT is informed that an argument must be preserved by a flag (`ReferencesOriginal`) associated with memory objects.

5.3 Moving Objects Around, Deep Copying

There are circumstances in which memory objects must not contain references to auxiliary resources, nor to sub-parts of other memory objects. This arises, for example, when an object or object part is to be moved to persistent storage, or when results are to be delivered in a client-server environment. Under these circumstances, CONCERT must be able to ensure a new memory object is flat, and interpretable as a byte sequences.

This requirement is contrary to both the need to accommodate auxiliary resources, and the desire to admit shallow copying. Our solution is to allow the caller of a function to specify requirements of the results. In particular, the caller specifies the `HasAuxiliary` and `ReferencesOriginal` flags according to their requirements, and the function resets these flags according to how it behaved. The protocol is that a function may swap a `true` flag to `false`, but not the other way around. For example, the caller may specify that auxiliary resources may be allocated, but no such may be required by the function. In this case, the function's implementation swaps the relevant flag. However, if the caller specifies that no auxiliary resources may be allocated, then the function's implementation is not at liberty to swap the flag.

This places a basic minimum requirement on the object's managed by CONCERT: they must be able to generate flat results. This does not imply that they must always do so; but must when forced to do so.

[1] This latter case is explicitly disallowed in Illustra.

5.4 Caller Allocates Space

The final issue we address is that of efficiently moving data around when abstract objects' sizes may be unknown. We abandon our metadata example, and consider instead the problem of raster-data management which better illustrates the problem at hand. Assume a tile of a raster image is to be moved to CONCERT's persistent storage. We consider a raster image to be a `CONTINUUM`-like object, and hence have a `partition` operation for achieving this. However, under the protocol described thus far, the procedure would be the following: first we extract the tile, we then know its length and can allocate space (in the buffer, say), and finally we copy the object to its target location. This results in one unnecessary copy and one unnecessary scan of a potentially very-large object.

However, for many raster representations it is straightforward to establish the size of such a function's results before applying the function. In principle, the tile can be copied directly from the original to the target.

To overcome this problem, CONCERT implements a protocol based on *caller allocates space*. The intuition is simple: the caller of an operation knows what is required of the results, and therefore should be the one managing the space of those results. The approach is that copy operations are done in two phases. Firstly the necessary size is established; this information allows the caller, who knows the requirements, to allocate appropriate space. And secondly the copy is performed directly form the original to the target.

A further example of such functionality is that of gathering query results prior to shipping those results to a client for further processing. The results should be gathered in a form appropriate for the network interface being used. Typically, this means that they must be gathered in contiguous virtual memory. The protocol of 'caller allocates space' allows an object buffer to gather partial query results for a number of objects before shipping them in a single step to a client. This can be done by moving objects directly from the database buffer to a transfer buffer in a single step.

6 Conclusions and On-Going Work

Our vision is of database systems exporting not only their data, but also their services. Databases systems should become brokers of information, coordinators of dependencies, and providers of database services. These are the goals of the COSMOS project investigating openness, cooperation and database services at various architectural levels: workflows [BDS+93], multidatabases [SWS91], and coordination for CIM environments [NWM+94].

In this paper we have focused on open storage-management services: indexing, partitioning, replication, and basic query-processing. We introduced the

simple mechanism of *likeness* to known types, and showed how a surprisingly small number of concepts can provide the basis for applying database services to external repositories. We showed both how queries are processed against external data, and how external materialised views can be maintained within external repositories. Furthermore, we showed how forward compatibility allows existing applications, scripts and browsers to be retained.

We discussed also the impact of abstract-object management on the internal architecture of our prototype system CONCERT. External objects require the coupling of external functions to the database system. In the context of these, we have developed protocols for resource management, and also the 'caller allocates space' protocol for efficiently moving potentially-large objects around.

We have on-going work in the areas of transaction management and external access structures. We are also applying the ideas described here in the context of multimedia and geospatial cooperative projects, including in particular raster data management, image indexing, and continuous media. These problem domains are archetypical of those for which open database solutions are attractive.

Acknowledgements

We are very grateful to those who have helped in the development of this work; these include Gustavo Alonso, Gisbert Dröge, Thomas Etter, Armin Fessler, Christof Hasse, Andrej Vckovski and Andreas Wolf.

References

[ACM93] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proceedings of the 19th Conference on Very Large Database Systems*, Dublin, Ireland, 1993.

[BDS+93] Y. Breitbart, A. Deacon, H.-J. Schek, A. Shet, and G. Weikum. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. *ACM SIGMOD Record*, 22(3):23–30, September 1993.

[Ber95] Philip A. Bernstein. Repository internals. VLDB95 Tutorial Notes, September 1995.

[BKRS94] Stephen Blott, Helmut Kaufmann, Lukas Relly, and Hans-Jörg Schek. Buffering long externally-defined objects. In *Proceedings of the International Workshop on Persistent Object Systems*, pages 40–53, Tarascon, France, September 1994.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, USA, 1990.

[BS95] Paul Brown and Michael Stonebraker. BigSur: A system for the management of earth science data. In *Proceedings of the International Conference on Very Large Databases*, pages 720–728, Zurich, Switzerland, 1995.

[BV95] Stephen Blott and Andrej Vckovski. Extending a database storage system for geographical metadata. In *Proceedings of the Fourth International Symposium on Advances in Spatial Database Systems (SSD95)*, number 951 in Lecture Notes in Computer Science, pages 117–131, Portland, Maine, August 1995. Springer Verlag.

[CDF+94] Michael J. Carey, David J. DeWitt, Michael J. Fanklin, Nancy E. Hall, Mark L. McAuliffe, Jeffre 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.

[CHP+95] M.J. Carey, L.M. Haas, P.M.Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams, and E.L. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proceedings of the Workshop Research Issues in Data Engineering–Distributed Object Management (RIDE-DOM95)*, Taipei, Taiwan, March 1995.

[DKA+86] P. Dadam, K. Kuspert, F. Anderson, H. Blankel, R. Erbe, J. Guenauer, V. Lum, P. Pistor, and G. Walch. A DBMS prototype to support extended NF^2 relations: An integrated view on flat tables and hierarchies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 356–367, 1986.

[DS93] Gisbert Dröge and Hans-Jörg Schek. Query-adaptive data space partitioning using variable-sized storage clusters. In *Advances in Spatial Databases: Proceedings of the 3rd International Symposium, SSD '93*, pages 337–356, Singapore, June 1993.

[DSW90] Gisbert Dröge, Hans-Jörg Schek, and Andreas Wolf. Erweiterbarkeit in DASDBS. *Informatik Forschung und Entwicklung*, 5:162–176, 1990. In German.

[FGD94] *Content Standard for Digital Geospatial Metadata.* USGS/FGDC, Federal Geographic Committee, U.S. Geological Survey, 590 National Centre, Reston, Virginia 22092, USA, 1994. Anonymous ftp: `fgdc.er.usgs.gov`.

[GJR94] N. H. Gehani, H. V. Jagadish, and W. D. Roome. OdeFS: A file-system interface to an object-oriented database. In *Proceedings of the International Conference on Very Large*

Databases, pages 249–260, Santiago, Chile, September 1994.

[Ill94] Illustra Information Technologies (Inc), 1111 Broadway, Suite 2000, Oakland, CA 94607. *2D Spatial DataBlade Guide*, October 1994. Illustra 2D Spatial DataBlade Release 1.3.

[Ill95] Illustra Information Technologies (Inc), 1111 Broadway, Suite 2000, Oakland, CA 94607. *Application Programming Interface Guide*, March 1995. Illustra Server Release 2.4.1.

[NWM+94] M. C. Norrie, M. Wunderli, R. Montau, U. Leonhardt, W. Schaad, and H-J. Schek. Coordination approaches for CIM. In *Proceedings of the European Workshop on Integrated Manufacturing Systems Engineering*, pages 223–232, Grenoble, France, December 1994.

[Obe94] Special issue on TP-monitors and distributed transaction management. In Ron Obermarck, editor, *Bulletin of the Technical Committee on Data Engineering, IEEE*, volume 17, March 1994.

[Obj95] Object Management Group. *CORBA: The Common Object Request Broker: Architecture and Specification*, July 1995. Release 2.0.

[OM88] 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.

[RB95] Lukas Relly and Stephen Blott. Ein Speichersystem für abstrakte Objekte. In *Proceedings of the 1995 Conference Datenbanksysteme in Büro, Technik, und Wissenshaft (BTW95)*, pages 338–347, March 1995. In German, short paper.

[Sch95] Peter Schweitzer. *A Compiler for Formal Metadata*. U.S. Department of the Interior, U.S. Geological Survey, Geologic Division, Mail Stop 955, National Center, Reston, VA22092, USA, 1995. As of March 1996, available electronically at:
`http://geochange.er.usgs.gov/pub/...`
`.../tools/metadata/compiler/README.html`.

[SLS+93] K. Shoens, A. Luniewski, P. Schwarz, J. Stamos, and J. Thomas. The Rufus system: Information organisation for semi-structured data. In *Proceedings of the International Conference on Very Large Databases*, pages 97–107, Dublin, Ireland, August 1993.

[SPSW90] H-J. Schek, H-B. Paul, M.H. Scholl, and G. Weikum. The DASDBS project: Objectives, experiences, and future prospects. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):25–43, March 1990.

[Sto86] Michael Stonebraker. Inclusion of new types in relational database systems. In *Proceedings of the International Conference on Data Engineering*, Los Angeles, CA, February 1986. IEEE Computer Society Press.

[SW93] Hans-J. Schek and Andreas Wolf. From extensible databases to interoperability between multiple databases and GIS applications. In *Proceedings of the 3rd International Symposium on Advances in Spatial Databases (SSD93)*, pages 207–238, Singapore, June 1993.

[SWS91] H.-J. Schek, G. Weikum, and W. Schaad. A multi-level transaction approach to federated transaction management. In *Proceedings of the International Workshop on Interoperability in Multi-database Systems*, Kyoto, 1991.

[WS92] Gerhard Weikum and Hans-Jörg Schek. Concepts and applications of multilevel transactions and open nested transactions. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 13. Morgan Kaufmann, 1992.

[WSSH88] P.F. Wilms, P.M. Schwarz, H.-J. Schek, and L.M. Haas. Incorporating data types in an extensible database architecture. In *Proceedings of the 3rd International Conference on Data and Knowledge Bases*, Jerusalem, June 1988.