In: Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Special Issue on Interoperability, pages 43-51, 21(3), September 1998.

Exporting Database Functionality — The CONCERT Way

Lukas Relly Heiko Schuldt Hans-J. Schek
Database Research Group, Institute of Information Systems, ETH Zürich, Switzerland

Abstract

In order to extend database technology beyond traditional applications a new paradigm called "exporting database functionality" as a radical departure from traditional thinking has been proposed in research and development. Traditionally, all data is loaded into and owned by the database, whereas according to the new paradigm data may reside outside the database in external repositories or archives. Nevertheless, database functionality, such as query processing, indexing, and transaction management, is provided. In this paper we report on the CONCERT *project that exports physical database design for advanced applications, and we discuss the consequences for transaction management, that becomes an important coordination task in* CONCERT.

1 Introduction

Todays Database Management Systems (DBMS) make the implicit assumption that their services are provided only to data stored inside the database. All data has to be imported into and being "owned" by the DBMS in a format determined by the DBMS. *Traditional database applications* such as banking usually meet this assumption. These applications are well supported by the DBMS data model, its query and data manipulation language and its transaction management. *Advanced applications* such as GIS, CAD, PPC, or document management systems however differ in many respects from traditional database applications. Individual operations in these applications are much more complex and not easily expressible in existing query languages. Powerful specialized systems, tools and algorithms exist for a large variety of tasks in every field of advanced applications requiring data to be available in proprietary or data exchange formats.

Because of the increasing importance of advanced applications, DBMS developers have implemented better support in their systems for a broader range of applications. Binary Large Objects provide a kind of low-level access to data and allow individual data objects to become almost unlimited in size. Instead of storing large data objects in BLOB's, some newer systems such as ORACLE (Version 8) and Informix (Dynamic Server with Universal Data Option) provide the BLOB interface also to regular operating system files. Because the large objects in any of these two options are uninterpreted, database functionality for this kind of data is only very limited. In order to better support advanced applications, the standardization effort of SQL3 specifies, among others, new data types and new type constructors. Most recently, SQL3 and object-orientation have fostered the development of generic extensions called datablades [10], cartridges [14], and extenders [9]. They are based on the concept of abstract data types and often come with specialized indexing.

Although they provide better support for advanced applications, however, except for the file system case, they all have the same fundamental deficiencies: Firstly, it is the DBMS together with its added extensions that prescribes the data structure and data format of the data to be managed. The consequence is that all complex specialized application systems and tools must be rewritten using the data structures enforced by the DBMS, or at least complex transformations must take place to map the DBMS representation into the application representation. Secondly, the DBMS owns the data. All data has to be physically stored inside the DBMS requiring to possibly load gigabytes of data into the database store.

These observations led to a radical departure from traditional thinking as it is expressed in [24]. In the COSMOS project at ETH [19, 20, 21, 13, 23], we focus on *exporting database functionality* by making it available to advanced applications instead of requiring the applications to be brought to the DBMS. Figure 1(a) shows the traditional DBMS that offers its functionality to clients whose data are under complete control of the DBMS. As opposed to this, figure 1(b) shows the new paradigm we have followed: The DBMS changes its role and becomes a coordinator (DBCoord) of many local specialized component systems. DBCoord provides database access to external data sources stored in the component systems. It provides tools for creating indexes over external data as well as for replications of external data. Exporting database access to external data sources does not exclude clients from accessing this data directly. In contrary, we explicitly want to deal with given specialized application systems and we do not require existing applications to be rewritten. We additionally want to provide DB functionality for (new) applications that need it. DBCoord ensures that changes of external data are properly reflected in related derived indexes and replications, more generally, that dependencies between component systems are transactionally maintained. The DBCoord's task is to coordinate possibly heterogeneous, possibly distributed, possibly autonomous subsystems rather than to store and to own data.

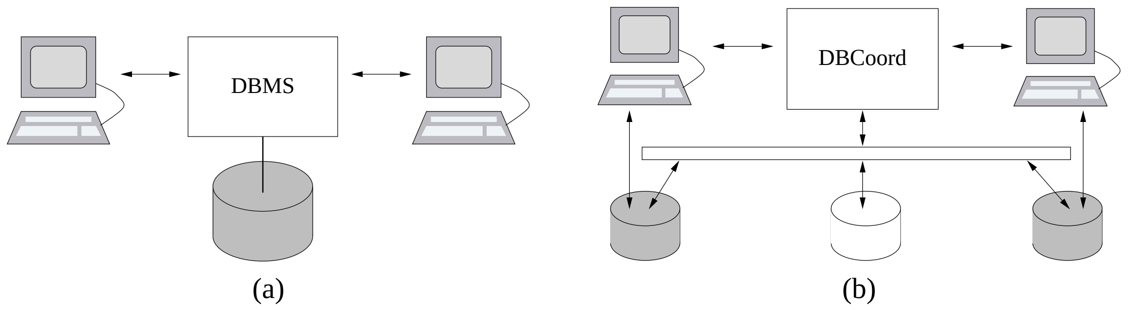

Figure 1: DBMS owning the data vs. DBMS providing database functionality

In this paper, we report on our research project COSMOS in general and the prototype DBMS CONCERT in particular [4, 16]. CONCERT provides implementations as "proof of concept". Here we discuss in more detail the aspect of physical database design and its consequences to transaction management. The paper is organized as follows: Section 2 introduces a sample application and focuses on the key concepts required for physical database design to efficiently provide access to external data. Then, in section 3, we discuss the implications to transaction management.

2 Exporting Physical Design: The CONCERT Approach

Throughout the paper, we will use a sample application — the management of geospatial images provided by a satellite — on which we show, how the concepts of providing database functionality for external data can be applied. A satellite is supposed to periodically generate geospatial images together with descriptive information (as, for instance, date and time the image has been made, the position of the satellite, etc). This information is stored in huge tape archives. Additionally, after an image has been taken, meta data describing this image is stored in a proprietary file format [7]. This meta data contains the descriptive information provided by the satellite and additional descriptions provided by a user. Furthermore, a preview of the geospatial image is materialized. The data objects to be managed therefore consist of the images in the tape archive and the meta data files (figure 2).

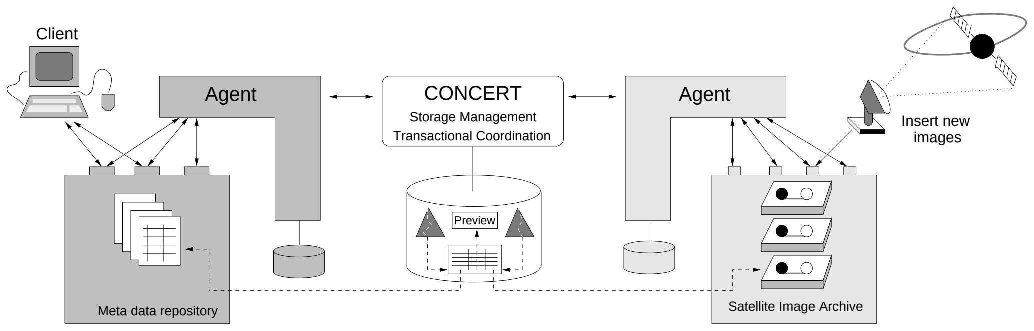

Figure 2: Management of satellite images and meta data stored at two different external repositories

In order to provide efficient access to the satellite image database, among others, the following physical design decisions can be considered.

- The relationship between the meta data and the image data has to be maintained. They should be viewed as the two sides of the same coin, although they are stored in independent systems.
- Index support for meta data attributes such as date and time of data capture should be available.
- Information retrieval techniques might be required to efficiently find images based on their description.
- Spatial indexing is required to access image data for certain regions.
- Sophisticated caching strategies for satellite images loaded off the tape archive have to be used.

Blades, cartridges, or alike allow the DBMS to be extended by *application specific types* and *access methods*. While implementing new types is relatively easy, new access methods is not. The new access method has to cooperate with the various components of the DBMS, such as concurrency control, data allocation, query evaluation, and optimization. This requires substantial knowledge of the DBMS internals. In contrast, CONCERT offers a limited, built-in set of physical design mechanisms in form of generic, trusted DBMS code provided by the DBMS implementor. Physical design is performed through relating new types to the fundamental concepts of the built-in physical design mechanisms.

In [25], Stonebraker introduced the idea of a generic B-Tree that depends only on the existence of an ordering operation to index arbitrary data objects. Our CONCERT approach generalizes this idea by identifying *all relevant concepts of physical database design* and expressing them by the so called *concept typical* operations required to implement them over external data. The data objects are treated as abstract data types (ADT) in CONCERT, and physical database design is performed based on the operations of the ADT only. In order to implement search tree access methods, a generic search tree approach (similar to GiST [8]) can be used as it integrates nicely into the CONCERT framework.

CONCERT provides the DBMS kernel functionality such as storage management, low level transaction management, and basic query processing. It's role is comparable to the one of RSS in System R. In order to enable good concurrency, CONCERT internally uses a multilevel transaction management scheme that is combined with ideas from [11, 6]. It uses a memory-mapped buffer on a multi-block page management [3] for efficient data access. It is beyond the scope of this paper to give full detail of the CONCERT approach. The following is a simplified sketch to give an idea. More detailed information can be found in [3, 4, 16].

Fundamental generic operations: the GENERIC concept In order to handle abstract data objects, to move them around in the DBMS and between remote repositories and to pass them between function invocations, the following three fundamental generic operations are required. In order to allocate space, the object's linearized **size** is required. The **copy** operation actually moves the object from one place to the other possibly performing necessary format transformations. During the copy operation, auxiliary resources (memory, file handles, network connections) might be required that are freed using the **free_aux** operation. These three operations are required for all CONCERT objects in addition to the concept typical operations that are discussed below.

The SCALAR concept The first specific concept is the concept of ordering, grouping, and clustering. It contains the generic B-Tree as a special case and is based on a comparison operation. The COMPARE operation of the SCALAR ADT has the following signature. It returns 0 in case of equality, otherwise -1 or 1 depending on the resulting order.

`COMPARE` (SCALAR_object, SCALAR_object) $\rightarrow \{-1, 0, 1\}$

This operation is sufficient to build a generic B-Tree (using techniques as in [25]) or any other access structure relying on an ordering or comparison such as a partition index, one of the new methods of physical design in the ORACLE8 DBMS tailored for advanced applications. In addition, the SCALAR ADT provides an optional HASH operation that allows certain data object to be used in the context of hash indexes.

Because arbitrary functions are allowed to implement the compare operation, for example remote procedure calls (RPC) can be used to compare external data objects residing on different machines or repositories. Obviously, for efficiency reasons, RPC might not be the appropriate method for external data and the compare operation can decide to create local copies. Also note that CONCERT concepts do not make assumptions about type information. For a single user object of a given type, several different concepts can be declared in order to instantiate, for example, more than one B-Tree on different user attributes.

The RECORD concept A second concept of physical database design is the one concerned with components or parts of objects. It is used implicitly in ordinary indexes that usually are built over an attribute (i.e., a component) of a data object. More important, the RECORD concept is used explicitly in vertical partitions. The relevant concept typical operation of the RECORD concept is the one identifying components of objects. It has the following signature:

`SUB_OBJECT` (RECORD_object, component) $\rightarrow$ object_part

extracting the object part identified by the parameter "component". The resulting object_part is a generic CONCERT object, that might be specified further by other concepts. The RECORD concept is especially important for objects with large, but infrequently used parts when vertical partitioning is used. Examples are objects that contain multimedia data such as images, audio, or video data. Often, only the descriptive attributes of such data objects are accessed. Therefore they are stored separately from the large multimedia components.

The LIST concept Information retrieval and its physical design techniques such as inverted file indexes and signature methods is a good example of the third concept. It represents a characterization of objects in order to efficiently answer queries looking for objects containing certain features. The relevant operations are the ones providing access to individual features of an object as follows:

`OPEN` (LIST_object) $\rightarrow$ cursor
`FETCH` (cursor) $\rightarrow$ feature
`CLOSE` (cursor)

In the simplest case, the resulting feature will be further specified using the SCALAR concept in order to use a B-Tree or alike for the inverted file index. The LIST concept is not limited to traditional information retrieval. It can support any kind of query asking for objects containing certain features such as, for instance, frequency spectra in audio signals or features in digital images. In these cases, the corresponding concept typical operations are the feature extraction or frequency analyzing operations.

The SPATIAL concept The last concept is the one concerned with spatially extended objects. They appear in many different contexts. Time intervals are spatially extended objects in a single dimension, spatial objects such as line segments and polygons in the context of GIS systems usually appear in a 2D space, CAD objects are similar, but in 3D space, and some applications are concerned with multidimensional data objects. The concept typical operations of the SPATIAL concept are:

`OVERLAPS` (SPATIAL_object, SPATIAL_object) $\rightarrow$ boolean
`SPLIT` (SPATIAL_object) $\rightarrow$ { SPATIAL_object }
`COMPOSE` ({ SPATIAL_object }) $\rightarrow$ SPATIAL_object
`APPROX` ({ SPATIAL_object }) $\rightarrow$ SPATIAL_object

The predicate OVERLAPS checks for a nonempty intersection of two spatially extended objects, SPLIT partitions an object into a set of partial objects, COMPOSE recombines objects that have been split and APPROX implements an approximation such as a minimal bounding box over a set of objects. It is up to the developer of the SPATIAL concept to make these concept typical operations meaningful to the application and its types. This means that the n-dimensional range query $\Box(q, \text{object})$ appears equivalent to the same query over the composition of the overlapping parts,

$$\Box(q, \text{object}) \equiv \Box(q, \text{COMPOSE}(\{o | o \in \text{SPLIT}(\text{object}) \wedge \text{OVERLAPS}(o, q)\}))$$

and for arbitrary query objects q, if they overlap any object $o \in obj$, they also overlap the corresponding approximation

$$\forall q (\exists o \in obj (\text{OVERLAPS}(o, q)) \Rightarrow \text{OVERLAPS}(\text{APPROX}(obj), q))$$

While the first three concepts are only used for data objects, the SPATIAL concept is used also for objects representing the data space in index nodes of a spatial index such as the space covered by an R-Tree node. An index node is split, for example, performing a SPLIT operation on the spatial object associated with the index node. The resulting objects define the data space of the new index nodes. The data objects are then distributed according to the OVERLAPS operation. Depending on the index strategy, the data object might be split into a set of smaller objects first. If data objects are inserted into a node, the corresponding data space can be calculated using the APPROX operation on the objects in the node. On retrieval, data objects that have been split need to be recombined using the COMPOSE operation.

Physical Design using CONCERT **Concepts** Using the CONCERT concepts, it is possible to get an integrated view of the satellite image database. Each image together with its meta data is associated with the RECORD concept containing components such as *image_data, preview,* and *description*. Access to the different components is performed through the concept typical operation SUB_OBJECT. For example, in order to compute the preview image for a satellite image, the operation

image_sub_object (image, preview) $\rightarrow$ preview_image

runs the corresponding program accessing the tape archive, reading the satellite image and feeding it through a preview computation process. The required index support for meta data can be achieved by implementing the concept typical operation COMPARE for each attribute that an index is to be built over, such as

image_date_compare (image1, image2) → $\{-1, 0, 1\}$
image_title_compare (image1, image2) → $\{-1, 0, 1\}$

In order to apply information retrieval techniques on the descriptive information, the LIST concept is used. Its concept typical operation FETCH extracts the index terms from the description accessing the meta data repository in the following way

image_desc_open (image) → image_desc_cursor
image_desc_fetch (image_desc_cursor) → image_index_term

These index terms can then be stored in an inverted file. The SPATIAL concept allows CONCERT to build a spatial index over the images in the database. Accessing the image data requires loading it from the tape archive to secondary storage. Using the SUB_OBJECT operation,

image_sub_object (image, image_data) → raster_image

the loader program can be made known to the database enabling it to view the load operation as a materialization of a computed attribute. Therefore, standard database design decisions for materialized views can be used for the caching of the loaded image.

It is important to notice that only those concept typical operations required for the actual physical design have to be implemented. Whenever a new physical design method is to be used, the corresponding operations are implemented and registered in CONCERT. This allows for incremental improvement of the physical design as required by the applications.

3 Exporting Transaction Management

In our sample scenario, dependencies between external meta data files and index structures managed within the CONCERT DBMS exist as well as dependencies between the single parts of data objects stored in the tape archive and in the meta data files. Although data is manipulated by applications not being aware of these dependencies, *coordination* must reestablish overall consistency when it has been violated by local operations on external repositories. As data is no longer completely under the responsibility of the DBMS, this coordination can no longer be achieved by the local DBMSs transaction management. We therefore additionally have to export transaction management to keep track of dependencies. In what follows, we will concentrate on the coordination architecture, the processes that have to be executed by the CONCERT coordinator, and the transactional execution guarantees that have to be provided for these processes.

Coordination Architecture To enforce dependencies between subsystems, the CONCERT system has to act as global coordinator. The subsystems to be coordinated may be different heterogeneous and distributed resource managers or, in the case data is only accessible and interpretable via specialized services or applications, the application systems itself [13, 23]. Although, for coordination purposes, transactional properties of subsystems are required, we do not expect all subsystems to be DBMSs. However, in order to provide database functionality, a *database coordination agent* is placed on top of each subsystem [23]. Thus, from the point of view of the CONCERT coordinator, the subsystem together with its agent can be considered as DBMS. The functionality to be provided by the subsystems and their agents includes the atomicity of service invocations, the compliance with orderings of service invocations and either the compensation of already committed services or their deferment by a two phase commit protocol. When a subsystem is not able to directly provide this functionality, its agent has to implement it. Coordination agents are similar to 2PC agents proposed for federated database management systems [26] but provide considerably more functions in cases where the subsystem does not provide DBMS functionality.

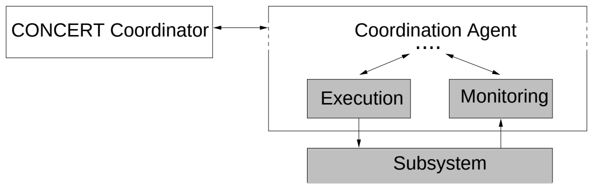

Figure 3: Subsystem specific part of generic coordination agent

Although certain parts of a coordination agent are tailored to the subsystem it belongs to, a generic structure of a coordination agent can be identified. Figure 3 shows the subsystem specific parts of such a generic coordination agent. When operations on external applications or repositories have to be invoked by the CONCERT coordinator, agents have to rely on existing services. Hence, the *execution module* of a coordination agent exploits, for instance, the API provided by a subsystem or performs a RPC call to a resource manager to be coordinated. All coordination agents together therefore overcome the inherent heterogeneity of the subsystems to be coordinated. As data within subsystems may be manipulated locally by local service invocations, information about these local operations has to be made available by the *monitoring module* of the respective coordination agent. Therefore, aside of heterogeneity and distribution, also autonomy of the systems to be coordinated is an important issue as both local transactions and global (coordination) transactions have to be considered [2, 12, 18]. Monitoring can, for instance, be performed on data level given the agent to be a DBMS itself by defining a database link and exploiting trigger mechanisms at the subsystem DBMS. However, in cases the subsystem is not based on a DBMS or if its data model is unknown, then monitoring has to be performed at application level, e.g., by plugging in trigger mechanisms at the application level. Often, systems like SAP R/3 [17] or Pro/Engineer [15] allow to pass control to the agent before some application function is executed.

Figure 2 shows the subsystems of our sample scenario, how they are provided with coordination agents and how the CONCERT coordinator communicates with external repositories and applications via these agents.

In addition to local transaction management in the subsystems and within CONCERT, we must add an additional transaction layer — the transactional CONCERT coordinator — keeping track of dependencies between external data. It should have become clear by now that the CONCERT coordinator provides functionality of an upper layer transaction manager coordinating subsystems while the agents and the CONCERT system provide local transaction management. Local transactions are intensively used for single operations. The agents, in turn, enrich a subsystem by a transaction manager at the application level enabling deferring or compensating single operations.

Coordination Processes in CONCERT A coordination process is a sequence of operations to be executed on subsystems [22]. For each coordination process, an execution guarantee is provided in the sense that at least one of several execution alternatives terminates successfully (generalized atomicity). The usual abort is treated as a special alternative. More generally, undoing everything is often not desired or even not supported by the subsystem and its agent. We will not present formal details (see [22]) but explain it using our example.

Considering our GIS application, coordination relies on appropriate coordination agents for both subsystems, the tape management and the meta data management. Each time a new satellite image is inserted into the tape archive locally, its agent notifies the CONCERT coordinator. In CONCERT, a record is then created representing the new image consisting in this state only of the *image_data* component (a reference to the image in the tape archive). Additionally, an index can be built over the date, the image has been taken by exploiting the concept typical operation *image_date_compare(image1, image2)*. Furthermore, a preview image is computed by *image_sub_object(image, preview)*, materialized in the CONCERT DBMS, and the image record is updated. The CONCERT coordinator then invokes the service *provide_meta_data* on the agent of the meta data repository. This agent requests a user for descriptive information of the new satellite image. After this information has been inserted in the meta data repository, the CONCERT coordinator is notified by the agent and the image record is updated (reference to the associated meta data description). Exploiting the concept typical operation *image_title_compare(image1, image2)*, an index is built over the title of the image provided by the user. Then, the index terms are extracted by the concept typical operation *image_desc_fetch(image_desc_cursor)* in order to store them in an inverted file. This coordination process is depicted in figure 4.

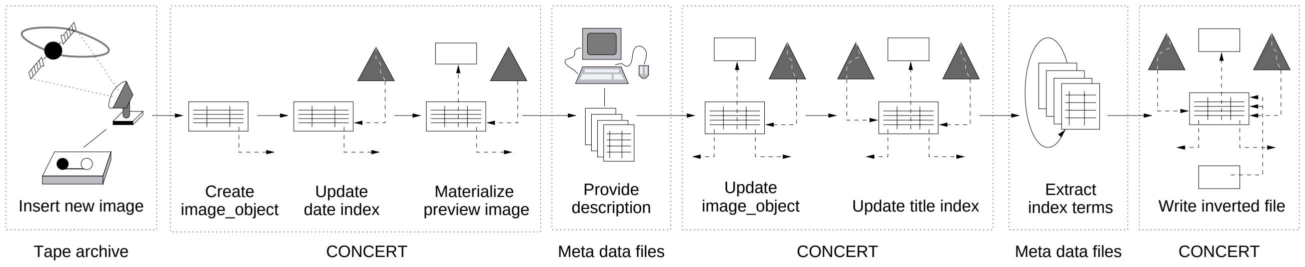

Figure 4: Coordination process to be executed when a new image is inserted into the tape archive

The CONCERT coordinator has to ensure that the execution of each coordination process leads to a system-wide consistent state even in case of failures and concurrency. With the generalized notion of atomicity for processes, at least one of the alternative executions (not depicted in figure 4) of our sample process has to terminate successfully. When, for instance, the date, the satellite image has been taken cannot be determined automatically, it is not desirable to undo all previous operations, e.g., delete the satellite image. It is more appropriate to alternatively ask a user for this information and continue executing the coordination process in order to ensure correct and guaranteed termination without undoing previous work. To enforce execution guarantees for processes, the global transaction management of the CONCERT coordinator relies on the local transaction management provided by the coordination agents. They have, for instance, to guarantee atomicity and isolation of local operations. Considering the local transaction within the meta data repository extracting index terms, the agent has to ensure that the sequence of *image_desc_fetch(image_desc_cursor)* operations is either executed successfully (thus returns all index terms) or fails and it has further to ensure that no changes of the meta data description are performed as long as the *image_desc_cursor* is open.

4 Conclusions

Exporting database functionality relaxes the assumption of traditional DBMSs to own all data to be managed and allows to offer this functionality to data that resides outside the DBMS. In the CONCERT project, we have identified four fundamental concepts of physical database design (SCALAR, LIST, RECORD, and SPATIAL) together with their concept typical operations. Physical design can be performed in an elegant way, based only on these concepts, for arbitrary data. By providing physical database design for external data, dependencies arise between data inside and data outside the CONCERT DBMS. We have shown, how the necessary coordination can take place in order to synchronize different data repositories with the help of subsystem-specific coordination agents. We showed, using a sample application, how both database services, physical design and transaction management, are provided by the CONCERT prototype system and how these services cooperate in order to contribute to a coherent whole. The aspects of physical database design and its consequences arising for transaction management, as we have discussed them, prove the feasibility of our approach.

In our current and future work, we generalize the ideas of transaction management to be exported in order to support coordination in two directions: Firstly, in the composite systems theory, we investigate arbitrary composite systems where — although scheduling takes place at different levels and schedulers can be connected in arbitrary ways — correctness of the whole system has to be provided [1]. Secondly, we decouple transactional properties in order to assign execution guarantees more flexibly to (sub)processes by exploiting the notion of spheres [5].

References

[1] G. Alonso, S. Blott, A. Fessler, and H.-J. Schek. Correctness and Parallelism in Composite Systems. In *Proceedings of the ACM SIGMOD/PODS Conference on Management of Data*, May 1997.

[2] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. *The VLDB Journal*, 2(1):181-239, Oct. 1992.

[3] S. Blott, H. Kaufmann, L. Relly, and H.-J. Schek. Buffering long externally-defined objects. In *Proc. of the Sixth Int. Workshop on Persistent Object Systems (POS6)*, pages 40–53, Tarascon, France, Sept. 1994.

[4] S. Blott, L. Relly, and H.-J. Schek. An open abstract-object storage system. In *Proceedings of the 1996 ACM SIGMOD Conference on Management of Data*, June 1996.

[5] C. T. Davies. Data Processing Spheres of Control. *IBM Systems Journal*, 17(2):179–198, 1978.

[6] G. Evangelidis. *The hB$^{\Pi}$-Tree: A Concurrent and Recoverable Multi-Attribute Index Structure*. PhD thesis, Northeastern University, June 1994.

[7] Content standard for digital geospatial metadata, June 8, 1994. Federal Geographic Data Committee (USGS/FGDC), U.S. Geological Survey, USA. `http://www.fgdc.gov`.

[8] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the 21st Int. Conf. on Very Large Databases*, pages 562–573, Sept. 1995.

[9] IBM Corporation. `http://eyp.stllab.ibm.com/t3/`

[10] Informix Corporation. `http://www.informix.com/informix/bussol/iusdb/databld/datablade.htm`

[11] P. L. Lehman and S. B. Yao. Efficient locking for concurrent operations on b-trees. *ACM Transactions on Database Systems*, 6(4):650–670, Dec. 1981.

[12] S. Mehrotra, H. Korth, and A. Silberschatz. Concurrency control in hierarchical multidatabase systems. *The VLDB Journal*, 6(2):152-172, May 1997.

[13] M. Norrie and M. Wunderli. Tool Agents in Coordinated Information Systems. *Information Systems*, 22(2):59-77, June 1997.

[14] Oracle Corporation. `http://www.oracle.com/st/cartridges/`

[15] Parametric Technology Corporation. `http://www.ptc.com/products/mech/proe/`

[16] L. Relly, H.-J. Schek, O. Henricsson, and S. Nebiker. Physical database design for raster images in concert. In *5th International Symposium on Spatial Databases (SSD'97)*, Berlin, Germany, July 1997.

[17] SAP AG, Walldorf, Germany. `http://www.sap.com`

[18] W. Schaad. *Transactions in heterogeneous federated database systems (in German)*. PhD thesis, Swiss Federal Institute of Technology Zürich, 1996.

[19] H.-J. Schek, M. H. Scholl, and G. Weikum. From the KERNEL to the COSMOS: The database research group at ETH Zurich. TR 136, Information Systems, ETH Zürich, July 1990.

[20] H.-J. Schek and A. Wolf. Cooperation between autonomous operation services and object DBMS in a heterogeneous environment. In *IFIP, DS-5, Semantics of Interoperable Database Systems*, Australia, 1992.

[21] H.-J. Schek and A. Wolf. From extensible databases to interoperability between multiple databases and GIS applications. In *Advances in Spatial Databases: SSD'93*, LNCS. June 1993.

[22] H. Schuldt, G. Alonso, and H.-J. Schek. A unified theory of concurrency control and recovery for transactional processes. Technical Report, Swiss Federal Institute of Technology Zürich, 1998.

[23] H. Schuldt, H.-J. Schek, and M. Tresch. Coordination in CIM: Bringing Database Functionality to Application Systems. In *Proceedings of the 5th European Concurrent Engineering Conference*, Erlangen, Germany, April 1998.

[24] A. Silberschatz, S. Zdonik, et.al. Strategic directions in database systems – breaking out of the box. *ACM Computing Surveys*, 28(4):764–778, Dec. 1996.

[25] M. Stonebraker. Inclusion of new types in relational database systems. In *Proceedings of the International Conference on Data Engineering*, pages 262–269, Los Angeles, CA, Feb. 1986.

[26] A. Wolski and J. Veijalainen. 2PC Agent Method: Achieving Serializability in Presence of Failures in a Heterogeneous MDBMS. In *Proc. of the IEEE PARBASE'90 Conference*, pages 321–330, Miami, March 1990.