Architecture and Implementation of the Darmstadt Database Kernel System

H -B Paul, H -J Schek, M H Scholl, G Weikum, U Deppisch

Technical University of Darmstadt
Computer Science Department
Alexanderstr 24, D-6100 Darmstadt, West Germany

Abstract

The multi-layered architecture of the **DArmStadt Data Base System** (DASDBS) for advanced applications is introduced DASDBS is conceived as a family of application-specific database systems on top of a common database kernel system The main design problem considered here is What features are common enough to be integrated into the kernel and what features are rather application-specific? Kernel features must be simple enough to be efficiently implemented and to serve a broad class of clients, yet powerful enough to form a convenient basis for application-oriented layers Our kernel provides mechanisms to efficiently store hierarchically structured complex objects, and offers operations which are set-oriented and can be processed in a single scan through the objects To achieve high concurrency in a layered system, a multi-level transaction methodology is applied First experiences with our current implementation and some lessons we have learned from it are reported

1 Advanced Applications: How to Meet the Requirements

During the last years, research and development in the database management system (DBMS) area has been oriented more and more towards so-called "advanced applications" This is a generic term for e g the integration of textual data into a DBMS, for supporting engineering or office applications by a DBMS, for the development of land information systems based on a DBMS, or for the integration of geometrical or graphical data into a DBMS The following summarizes the requirements which are seen for a database system supporting these advanced applications

- *Object orientation* "Molecular" [BB84] or "complex" [LKMP85] objects must be supported with the characteristics that objects as a whole or only parts or subobjects can be manipulated by the database system
- *Extensibility* The database system must be extensible to application-specific needs (often called abstract data type support [SRG83])
- *Knowledge representation* Approaching the complexity of real-world objects by more refined models is not possible without integration of knowledge representation schemes from AI (e g [BS85]) into databases
- *Set orientation* Sets of complex objects are often needed rather than single data items for algorithms in application programs
- *Database cooperation* In an environment of servers and workstations database systems should be able to cooperate on different levels of the system structure to enhance overall efficiency [DO87]
- *Modularity* Suitable components of the database system must be replaceable, depending on the demands of the application

While the first three requirements center around the area of usability and convenience of a DBMS, the last three clearly are performance oriented How can a system be built such that the higher level notions of objects can be supported efficiently? Performance is the main issue for a DBMS to be useful for advanced applications

Facing these requirements, several approaches to new DBMS architectures have been proposed We distinguish between three classes (cf [SL83]) **Special-purpose systems** for a specific (class of) application(s)—this solution allows fine-tuning but is not general enough to be further considered throughout this paper—, **full-scale** next generation DBMSs—trying to identify a kind of interface that is suitable for all or at least most of the new applications—and **database kernel** architectures, or **extensible** DBMSs

We believe that requirements from the various application areas differ too much as to be able to identify *one* interface that is appropriate for all of them Rather we pursue an architecture that builds application-specific front-ends on top of a common DBMS kernel Such a "kernel architecture" has been proposed and discussed in [SL83] and has further been developed in [HR85, DKM85, DOPSSW85, Ca86] Already conventional database systems like System/R [As76] have a two-layered architecture—RDS (Relational Data System) on top of RSS (Research Storage System)—, where the lower level can be considered a kernel Extending this layered architecture directly leads to the kernel architecture for new DBMSs

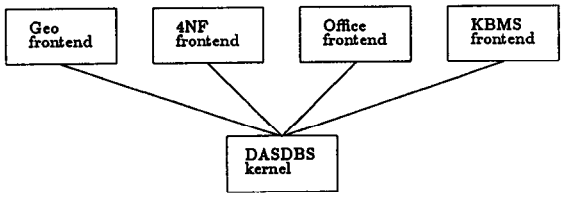

Figure 1 Architecture of the DASDBS family

The crucial question for such a kernel architecture is, how does the kernel look like? What are the common features required in all applications that have to be built into the kernel? On the other hand, in order to keep the kernel small enough to allow effective use in the design of new front-ends, can these features be implemented efficiently without introducing too much complexity into the kernel? From a slightly different point of view, these questions can be rephrased as How can we build an extension of the operating system (data management and process control) interface that is more suitable for the development of next generation applications?

The **DA**rm**S**tadt **D**ata **B**ase **S**ystem (DASDBS) project is an attempt to find concepts and solutions to some of these problems and to prove the solutions by integrating them into a prototype system As shown in figure 1, DASDBS is a database system *family* Based upon a single common kernel DBMS, different application-specific layers support various application classes Sample application areas shown include geo-sciences (the Geo-Kernel), office document filing and retrieval as well as standard relational applications and knowledge representation techniques like frames- or molecules-oriented models To obtain a certain application-oriented DBMS the kernel is linked to the corresponding front-end

For one application class the DASDBS overall architecture can be represented as shown in figure 2 Three main interfaces are distinguished the Application-specific Object Manager (AOM) operates on sets of hierarchically structured "Complex Records" that are manipulated by the Complex Record Manager (CRM) as primitive "complex objects" at the kernel interface The Stable Memory Manager (SMM) is a set-of-pages-oriented layer, it includes buffering and supports classical DBMS transaction management

In this paper we report on the decisions made during the project which started in 1983 and will be completed in early 1987 We concentrate on the question which features were built into the *kernel* system and which were not First we present general criteria used to answer this question in chapter 2 and show in detail our solutions concerning the points of storage objects, basic operations, and transaction management In chapter 3 we review our decisions and compare them to some more recent kernel and full-scale architectures for advanced database applications Finally, experiences with our current implementation are reported

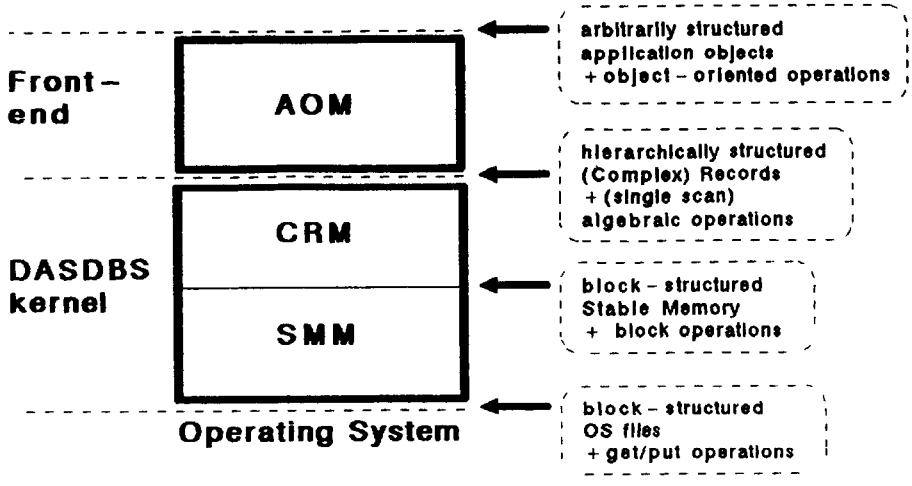

Figure 2 DASDBS as a 3-level interface system

2 The DASDBS Kernel

2 1 Kernel Features General Criteria

The main problem for the design of the kernel is how powerful (or general) to build the kernel On the one hand, common features have to be enclosed to avoid repeatedly implementing them in the application layers and thus giving effective support for the development of new members of the family On the other hand, care must be taken to avoid building a monster instead of a streamlined kernel As the overall intention was gaining performance for future DBMS-based applications, we had to find a carefully balanced compromise between increasing convenience and retaining efficiency of the implementation We used the following criteria as general guidelines for separating database kernel features from application specific features

- the kernel should meet most of the common requirements which are needed by advanced applications
- the kernel must support only those features which can be implemented efficiently *only within* the kernel, features that can be implemented with no or little decrease of performance by repeatedly using kernel functions should become non-kernel features

The first criterion concerns the semantics of the objects and operations of each layer in the system Our decision has been guided by the idea of an extensible multi-level software system where each layer offers certain well-defined services to the next higher level One essential design decision related to this has been to build—possibly application-specific—access path techniques on top of the kernel rather than to include e g a B^+-tree mechanism in the kernel A variety of access path techniques is used in the different advanced applications Therefore, access paths are considered application-specific and not included in our kernel This point will be discussed in more detail in section 3 1

The second criterion is the major one, because it concerns the performance aspect The essential guideline for the decision whether some operation on database objects should be implemented within or on top of the kernel is If the implementation on top is as efficient as an implementation within the kernel, build it on top! Otherwise, if repeated calls of the kernel result in a drastical loss of performance, it should become a kernel operation

Our choice of kernel features according to the above guideline has been very much inspired by the Research Storage System (RSS) of System/R [As76], which may be viewed as a data model independent kernel system *Similar* to the design decisions of RSS [Ch81], only single scan operations are supported at our kernel interface This means that non-single scan operations like the relational join are not within the kernel *In contrast* to RSS, however, our kernel offers set-oriented operations instead of a one-record-at-a-time interface, i e each operation retrieves or manipulates a set of Complex Records Set-orientation is one of the major attempts to improve performance of non-standard applications, because often large amounts of data are processed when complex objects have to be extracted from the database into the application program, as opposed to traditional applications manipulating small objects like bank accounts

2.2 Kernel Objects· How Complex are They?

In many publications it is shown that the flat relational model is not appropriate for advanced database applications e g [HL82, BB84] The requirements call for "complex objects" Without discussing the details of the various approaches, a characteristical feature is that objects are composed of (sets of) other (sub-)objects which, in turn, may be "aggregated" from more primitive objects In the notion of [BK86, AH84], a complex object is generated by the repeated use of the *tuple* and *set* constructors

Looking back again at System/R, flat records (RSS tuples) consisting of fixed and variable length bytestrings enriched by the "link fields" were a reasonable choice to support the (flat) relational data model but possibly also the CODASYL network model Now, faced with a similar question in view of the different notions of complex objects, we decided to restrict the kernel data structure to the common basis, namely (hierarchically) nested records consisting of bytestring fields Note that this includes link fields (sets of "pointers") or similar data as regular field entries

So, at the kernel interface we introduce "*Complex Records*" which are the internal implementation of tuples in the relational model with relation-valued attributes (non-first-normal-form, also called NF^2 relational model [SP82, SS86]) The hierarchical structure of such NF^2 tuples or Complex Records is the following a tuple has atomic-valued attributes as in the flat relational model and *relation-valued* attributes, i e again (sub-)relations Hence, this data structure is in fact obtained by an alternating sequence of tuple and set constructors (see above) Note that this model is the most restricted extension of the flat model the only data type is a relation, there are no lists, vectors, or other data structures At the kernel level they are left out intentionally and we assume these structures to be imposed onto the byte fields at a higher layer With the same argument, data types in general are not known to the kernel We will return to this question in the discussion of ADT support by the kernel in section 3 5

Another rather practical consideration is important we wanted to implement exactly those structures and operations in the kernel, that can easily and efficiently be mapped to the underlying linearized bytestrings in the page-structured database As hierarchical structures are the only ones that can be linearized without redundancy (details concerning this topic are contained in [SPS87]), we think Complex Records together with the operations presented below form an appropriate interface

2 3 Kernel Operations· Powerful but yet Efficient

The key concepts for all kernel operations are *set-orientation* and the *single scan property* The first one is concerned with the way results are presented to the next higher layer, i e one CRM-call fetches a set of Complex Records, or inserts or updates a set of Complex Records This is a highly essential feature because of the following reasons

- Complex objects of the application-specific layers may be very large and must often be handled as a whole Though we aim at a direct mapping of application objects to database objects, often sets of Complex Records have to be retrieved to establish application objects
- Each Complex Record consists of sets of complex subrecords which in turn consist of subrecords and so on So, even retrieving one single Complex Record as a whole would result in repeated calls of a "one-subrecord-at-a-time" interface similar to the use of conventional database systems This, however, has proven to be the main bottleneck when advanced applications are built upon usual DBMSs [GS82, HHLM86]
- Having set-oriented Complex Record operations and appropriate storage techniques for Complex Records (cf 2 4), set-orientation can be forwarded to the page server (SMM) interface The page server in turn can utilize corresponding features of the underlying operating system

Whereas the first criterion leads to more processing power in the kernel, the second one, viz the *single pass property*, gives the upper bound for the complexity of kernel operations Processing of all kernel operations should require access to all involved tuples only *once* The opposite are multi pass operations which can only be implemented by referencing parts of the involved relations several times

2 3 1 Retrieval· (Nested) Projection and Selection

We give an intuitive explanation of the single pass operations of the nested relational algebra [Sche85, SS86] that is available at the kernel interface Formal characterizations can be found in [Scho86, SPS87] Roughly, selections and projections are the single pass operations Rather powerful extensions were added to the projection in the case of relation-valued attributes In the classical projection of the flat relational model the attributes to be fetched are listed Now we can decide whether *all* of the subtuples should be retained or only a *subset* (identified by an inner selection) Similarly, all or only some of the (sub-) attributes may be retrieved (listed in an inner projection) Thus, within a projection list, selections and projections can be applied to subrelations In other words on each of the hierarchical levels of a nested relation it is allowed to specify by a selection and/or a projection which tuples or attributes respectively should be included in the result Obviously, projections, even nested ones, are single pass processible However, certain restrictions have to be imposed on the selection criteria To avoid comparison operations with join complexity in selection formulae, the only allowed term involving relation-valued attributes is of the form "attr $= \emptyset$" or "attr $\neq \emptyset$", where $\emptyset$ denotes the empty set

2 3 2 Retrieval Using Addresses

A virtual "attribute" is appended to every tuple stored in the database its *address* This attribute may be included with any query result and can be used to build access paths on top of the kernel in the application-specific layers Subsequently, selections on this "attribute" can be used to formulate direct access queries at the kernel interface Sets of addresses can be given along with any query to restrict the range of the query to those tuples whose addresses are contained in the set

2 3.3 Updates

The update operations were designed according to the same criteria It is possible to insert, delete or replace a set of complete Complex Records or subrecords including all subrelations, or to update atomic attributes All of these operations—except insertion of complete Complex Records—specify the corresponding (sub-)records by their addresses

Though the update operations are powerful, some operations must be realized by several kernel calls Because there is only the datatype bytestring for atomic attributes, no arithmetic operations, such as "increase the salary of all database specialists by 10%" are offered It is also not possible to move or to copy all subrecords of a subrelation from one record to another in one operation This operation does not reflect the hierarchical structure and has to be implemented on top of the kernel

2 4 Storage Clusters· Mapping Complex Records to Pages

To process the operations of the kernel efficiently, an appropriate storage structure for Complex Records has been implemented [DPS86] A Complex Record is stored clustered in as few pages as possible, forming its address space In DASDBS physical clustering is expressed in terms of Complex Records resulting from physical database design Therefore, we use the notion of a storage cluster for the above-mentioned address space Each Complex Record is prefixed by a directory header which contains the list of page numbers of the address space and data describing the record's internal structure The body of the Complex Record contains its data packed in atomic segments All record-internal pointers are relative within the address space Consequently, external (sub-) record addresses are stable w r t movements within the cluster and of the cluster as a whole Therefore, a Complex Record can easily be moved to another place with no maintenance cost except in the page list This efficiently supports dynamic reorganization and data transfer in server-workstation environments The main advantage of this storage concept in connection with the directory header is the performance gain through the set-oriented page server A kernel operation is performed in two basic steps first the root page containing the page list and probably the whole structural part of the directory header is requested, and the set of additional pages is determined which is necessary to execute the operation These pages are then requested in the second page server call Thus we can avoid to fetch the whole page set of a Complex Record if only parts of it are required, e g in the case of projections

2 5 Page Server and Set-Oriented I/Os

Because storage clusters of Complex Records typically span pages, Complex Record operations rarely need *single* page accesses Therefore the page server was designed with a (page-) set-oriented interface The advantage is that requests for large amounts of data may be passed through to a possibly set-oriented I/O-interface of the operating system to optimize disk accesses The set of pages of a storage cluster requested in one call of the page server may often be expected to be consecutive, due to our insertion strategy a set of pages as contiguous as possible is requested from the free place administration This knowledge can be exploited by the I/O routines provided that the contiguity property is preserved in the mapping of pages to disk blocks with high probability In fact, due to the stability of record-internal pointers it is rather inexpensive to move pages in order to maintain this contiguity We performed some experiments with chained I/Os, i e special channel programs to fetch a set of blocks with only one SIO ("Start I/O") instruction For large objects (about 128kB) this method is about 35 times faster than simply repeating I/O requests one per each 512 byte block Details of our performance evaluation are contained in [WNP86]

2.6 Main Memory Representation of Complex Records

Sets of Complex Records as the result of queries or as input to update operations have to be transfered between the kernel and the next layer, the application front-end The problem of how to represent these structures within a programming language had to be solved in order to avoid a bottleneck by introducing a one-record-at-a-time interface at this level (the "database portals" approach of [SR84, Ro86] aimed at the same problem) The structure that we needed would have been (in PASCAL terms) an arbitrary combination of **set of** and **record** constructors Unfortunately, the type constructors offered by usual programming languages (including PASCAL and C) do not allow this kind of data structure So, we decided to implement an ADT "object buffer" that is part of the kernel interface Each object buffer may contain sets of Complex Records of the same type Object buffers are private, i e they are not shared between transactions The *Retrieve* operation of the kernel generates an object buffer, and *Insert* expects a buffer containing the new records If the result of a retrieval operation does not fit into the object buffer, subsequent *Next* operations are used to continue However, Complex Records (of the resulting schema) must not be cut If a resulting CR is too big, the calling program has to reserve a larger buffer There are operations to navigate or browse through an object buffer as well as to read, insert, update, and delete atomic attributes, and to generate and destroy an object buffer Furthermore, as an object buffer can be regarded as a temporary relation, it should be possible to apply the kernel operations also to these temporary relations However, this is not (yet) supported in our current implementation (see chapter 4)

2.7 Transaction Management

In accordance with the multi-level interface architecture of DASDBS, we have developed a multi-level transaction methodology that can easily be integrated into the overall framework First ideas for this concept have been motivated by the "open nested transaction" approach [Tr83] of System/R, which uses long tuple locks to serialize application transactions and short page locks to serialize single RSS-(subtrans)actions As it has been recognized that access paths may become hot spots in advanced information

systems (cf [DLPS85]), we particularly considered the impact of indices on concurrency control To release locks on index pages as early as possible and to reduce contention on index entries (i e pairs of keys and pointer lists), we apply a 3-level transaction approach within DASDBS Locks on (all kinds of) pages are released at the end of each kernel operation, (a simple form of predicate) locks on Complex Records (including index entries in particular) at the end of each object-level (subtrans)action, and only "semantic" locks on the application-specific layer are held until the entire transaction commits [WS84]

The general rule behind this strategy is that whenever two non-conflicting operations of a particular layer spawn conflicting actions on the next lower level, this low-level conflict actually is some sort of a pseudo conflict that must be taken care of only for the duration of the issuing operations So, layer-specific conflict relations based on the semantics of operations lead to a potential increase of concurrency (cf [BBGLS83, MGG86]) On the other hand, recovery is more costly due to this abstraction, as it requires compensation of committed subtransactions to undo an operation For this purpose, log records containing sufficient information to construct inverse operations are written More details on the issue of multi-level transaction management, including preliminary performance results, have been published in [We86a, We86b, SW86]

3 Discussion and Comparison with other Architectures

Apparently comparisons to RSS have inspired our design decisions As RSS was not designed as a kernel for *advanced* applications, the following differences and extensions characterize our DASDBS kernel While RSS supports flat tuples and (flat) projection and selection, our kernel supports *nested* (complex) tuples with nested projection and selection While RSS has a single-tuple-at-a-time interface, our kernel transfers sets of Complex Records from sets of pages to main memory Our kernel does not maintain indices, whereas RSS does As a consequence, our kernel offers retrieval of Complex Records and subrecords with given sets of addresses The two systems are similar in that both include buffering, both are responsible for recovery, and both have a two-level concurrency control scheme Also data types are supported in neither system

In the following, we will discuss specific questions and try to compare our approach with some other projects we became aware of during our implementation

3 1 Access Paths and the Kernel

In contrast to generally required features like efficient storage management of Complex Records, we regard access paths as a rather application-dependent issue and therefore decided not to include any access path technique in the DASDBS kernel For example, access path techniques like grid file or R-tree [NHS84, Gu84] have been devised for geographical or graphical applications, others like signature techniques [FC84, De86] aim at office applications and other textual data A variety of different access path techniques are used in other advanced applications Building these different techniques into a DBMSs kernel seems not to be desirable for the lack of generality

This decision has often been criticized, as for example, B^+-trees admittedly serve a very broad class of applications, and could also be used as a basic access mechanism to build e g a spatial index This direction is pursued in the PROBE project [OM86] Nevertheless, we are convinced that without sometimes resisting such tempting ideas to include yet another feature, the concept of a streamlined kernel system would have failed from the beginning On the other hand, to justify our design we have to show that building access paths on top of the kernel does not result in a considerable decrease of performance

As the kernel consequently treats index entries in the very same way as primary data records, index pages and data pages cannot be distinguished within the kernel Therefore, an objection could be that it is not possible to handle index pages in a special way w r t concurrency control To manage all kinds of data in a uniform way, however, is exactly what we have tried to achieve with our kernel architecture Thanks to the concept of multi-level transactions, locks on index pages are released early anyway, such that there is actually no need for special index locking protocols

Another suspected drawback of building access paths on top of a record-oriented (kernel) interface is that this might cause additional page accesses Fortunately, we can easily disprove this objection by giving more details on how a B^+-tree, for example, is mapped to kernel objects One possible approach is to store (key , pointer list)-pairs of leaf nodes as variable length Complex Records, and to realize each inner node of the tree as a fixed length record which has the size of one page Traversing the tree, e g in order to determine candidate addresses to a given search condition, then requires one kernel call for each level of the tree As each of these "address selection" calls (cf section 2 3 2) still can be processed with only one page access, this particular kind of transitive closure query could not be implemented more efficient within the kernel either Similar considerations hold for more advanced access path techniques such as spatial indices

3.2 Why Hierarchies, Not Flat Records?

In spite of the considerations given before one might still ask, why we did not map hierarchically organized data to its atomic segments (like in IMS) first and then transform this intermediate level to pages Such an approach has been taken in the AIM project [Da86], with the advantage that the mapping of atomic segments to pages is much simpler However, additional clustering strategies must be invented to guarantee that the various segments are not dispersed over many pages As yet another solution we might have thought of an implementation as proposed in the EXODUS project [Ca86] There, an EXODUS kernel object is a byte string with arbitrary length and there is efficient support for arbitrary from-byte/to-byte extractions by a B-tree like organization [CDRS86] In fact, our approach is similar it could be regarded as a mapping of a Complex Record to a long byte container in the sense of an EXODUS object with a good clustering effect However, in contrast to the EXODUS organization of byte strings, our approach reflects the hierarchically organized record structure Our solution is a compromise between fast direct access to a specific atomic segment and fast access to large parts of a complex object Note that the definition of a Complex Record determines also the clustering strategy at the same time

3.3 Why Hierarchies, Not General Relationships?

Hierarchically structured Complex Records directly allow representation of entities and 1 n-relationships, but no n m-, non-binary and recursive relationships Complex Records—and so each 1 n-relationship implemented as one Complex Record—are mapped to the linear storage medium without redundancy This is, for principal reasons, not possible in the case of more general relationships [SPS87] Consider n m-relationships clustering related objects and supporting both hierarchical views equally well is impossible without introducing redundancy Therefore, "references" are usually introduced to solve the redundancy problem (foreign keys as part of the primary key in the 'relationship relations' of a flat relational representation)

Management of "references" between objects is not considered a kernel feature, nor is control of redundant information Thus, within our framework, hierarchical objects as the largest subclass of objects not requiring these techniques are the only ones appropriate for the kernel Anything beyond this scope is proposed to be built on top as an application-specific extension

As an example, let us examine the standard (4NF) relational layer (cf fig 1) which maps a classical relational interface to the kernel Within the corresponding front-end, a physical database designer (whether human or a component of the front-end) has to determine the kernel structures used to store the 4NF data In this case there are the following possibilities for mapping an n m-relationship from the conceptual flat level to the internal kernel level (omitting the symmetrical ones)—also see figure 3

(a) Three relations $S(\underline{A}, B)$, $T(\underline{C}, D)$, $R(\underline{A, C}, E)$, as on the conceptual level (trivial mapping)

(b) Denormalization of (Joining) R and S
$SR(\underline{A}, B, \underline{C}, E)$, $T(\underline{C}, D)$

(c) Clustering R to S $S'(\underline{A}, B, R'(C, E))$, $T(\underline{C}, D)$

(d) Clustering the corresponding addresses of R records to S and/or T $S'(\underline{A}, B, AddrR(Addr))$,
$T'(\underline{C}, D, AddrR(Addr))$, $R(\underline{A, C}, E)$

(e) Clustering R to S and the corresponding addresses of R records to T $S'(\underline{A}, B, R'(C, E))$,
$T'(\underline{C}, D, AddrS(Addr, AddrR'(Addr')))$

Case (a) is the only one in the flat relational model with no redundancy in relation R besides the key attributes A and C A join between R and S must be performed by fetching all R and all S records, sorting them along A and then merging the results In case (b) there is nothing to do because the relation SR stores the join between R and S but the S-tuples are redundantly stored with each matching R-tuple In case (c) there is no redundancy at all and the join is also already materialized On the other side this solution is asymmetric, because a join between T and R would require a join between T and the subrelation R' of each record of S'

In the other two cases we introduced record addresses as attribute values (to implement the "references") The special meaning of these values is only known within the 4NF relational layer but not to the kernel In case (d) a join between R and S is processed like the following fetch all S' records, take the corresponding addresses of R records, then fetch all R records the addresses of which are given The latter (second) kernel call is very fast because direct access to records with given addresses is efficient

Details about clustering and physical database design in the 4NF relational layer can be found in [SS83] and [SPS87] In fact, the optimization possibility due to the 4NF–NF^2 mapping [Scho86] was one of the main motivations for a "denormalized" [SS80] kernel Similar design considerations were made for the geographical kernel DBMS [SWa86], the office filing service [PSSW87], and molecular objects [SW86]

3 4 Why Single Scan Operations?

While we argued in favor of powerful operations at the kernel interface in section 2 3, restrictions to the full expres-

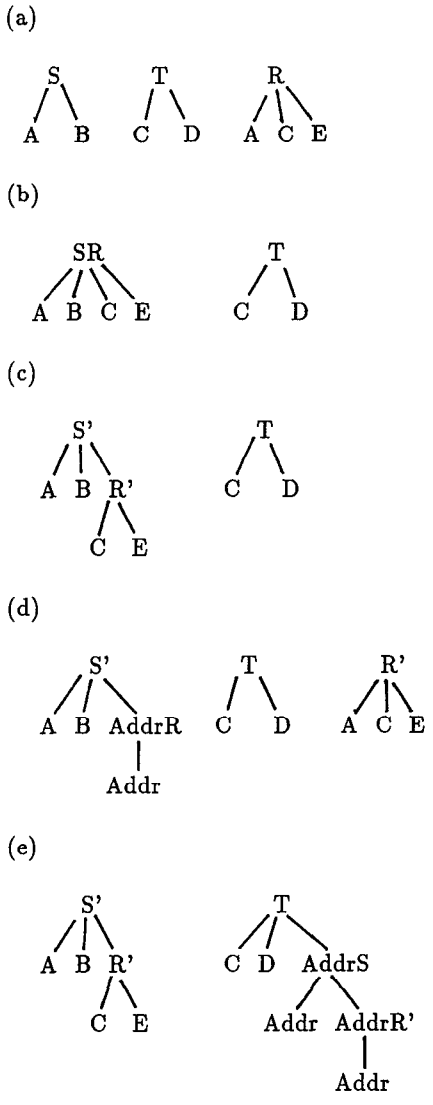

Figure 3 Representation of n m–relationships

sive power of a nested relational algebra [SS86] have to be imposed in order to retain efficiency The distinction between kernel and non-kernel operations led to the notion of "single pass processible" queries The classical example for multi pass operations is the relational join As there are no linear algorithms for the evaluation of such operations, execution of two nested scans ("nested loop" algorithm), for example, can be implemented on top of the kernel without loss of efficiency The same is true for more sophisticated algorithms

A very similar distinction was applied in the design of System/R [As76, Ch81] The RSS component—as the kernel—offers *single table* queries, whereas joins are executed within RDS—the front-end We intentionally do not use the term "single table" query, because there is a difference between single table and single scan in the nested relational model As attribute values may be relations in turn, selections comparing the values of two relation valued attributes are not single scan processible though they are single table queries (Equality or inclusion tests between two sets have the same complexity as the join!) If we carefully consider the *extended* operations of SQL for flat relations, we also find single table but not single scan processible operations, namely those introduced by the "GROUP BY" clause (involving sorting)

Nevertheless, the retrieval operations of the kernel are sufficiently powerful for the purpose of building a flat relational front-end on top of the kernel Our investigations have proven [Scho86] that these single pass operations are exactly those resulting from select-project-join queries at the relational interface when the joins have already been materialized in the hierarchical structure (cf section 3 3) A similar result was obtained in [AB84] for the VERSO model [Ab86] There the so-called "VERSO superselection" was reported to cover this important class of relational queries VERSO as a database machine project was oriented towards fast "on-the-fly" evaluation of queries by hardware filters [BRS82], operating on linearly ordered hierarchical structures This figure nicely illustrates our notion of the single scan property In fact, VERSO's superselection is very similar to the single pass processible subset of our nested relational algebra However, in contrast to our kernel the VERSO machine can compute the join of two relations whenever they are already sorted (in this case, join *is* single scan processible!)

3 5 Data Types, ADTs and the Kernel

For atomic attributes, the kernel has no knowledge about data types like integer, real or even more specific data types like surfaces, concepts, polygons or office folders, except for one data type, namely (arbitrary long) byte strings It is obvious that the kernel should not support datatypes like polygons, lines or office documents, because such data types enriched by corresponding operations would be application-specific Moreover, the DASDBS kernel neither offers any "standard" data types of conventional programming languages as integer, real or PASCAL-like records (Cartesian products) As appropriate operations for such types are considered to be efficiently implementable on top of the kernel, we decided to permit only one simple data type, namely byte strings

However, this statement does not hold in any case For instance, comparing real numbers in a selection formula is definitely less efficient on top of the kernel, because all tu-

ples have to be transfered to the higher layer, as opposed to filtering inside the kernel, which would only deliver matching tuples In this situation we applied the usual trick, namely integers and real numbers are coded in such a representation that the comparison can be performed on the bytestrings But this solution cannot be generalized for more refined and user-defined datatypes, we rather have to support ADTs ADT operations have to be supplied by the implementor of the application-specific front-end for those basic operations that should be performed within the kernel An example is a clip–operation on geometrical objects [SWa86]

General agreement on the necessity of the ADT support seems to exist Several recent projects are aiming at this topic, e g POSTGRES [St86] and STARBURST [Schw86] However, it is an open question what kind of architecture is best suited Should ADTs be mapped to extended NF^2 relations as in the R^2D^2 project [KW87] using the AIM prototype [Da86]? Should and could any storage structure or index technique for an ADT be embedded completely into a modular database system as STARBURST, or should a few primitive ADT operations on byte string fields be embedded into the kernel like in our approach? We do not know yet A variety of other systems including AIM [Da86] aiming at CAD applications, PROBE [MD86] for knowledge representation, DAMASCUS [DKM85] for software engineering environments, and PRIMA [Hä86] as a kind of extended ERM database system ("Molecule-Atom Data Model" [Mi87]) follow a full-scale DBMS approach instead of such extensible architectures

4 First Experiences with the Current Implementation

A first implementation of the kernel in PASCAL is completed This chapter indicates some experiences with our current code and some lessons we have learned from it After having committed ourselves on the external storage structures of Complex Records (see 2 4), we considered several alternatives w r t structures for the main memory representation of Complex Records (cf 2 6) Of course, hierarchically structured records could be represented as trees using (PASCAL-) pointers in main memory The advantage of this solution is that operations can directly be implemented using programming language constructs The disadvantages are such a tree can not be directly transfered between different address spaces, e g a database process and an application process Furthermore, parts of the records are scattered on the heap A second solution is to implement each record as a long bytestring containing offsets pointing to subrelations Each bytestring of arbitrary length is broken into several fixed length bytesubstrings After the implementation of this alternative, we realized that these fixed length bytestrings could be treated as pages, and that the main memory structures actually were very close to the external ones Therefore, we decided to pursue a third alternative, namely using identical structures for the main memory representation and on disk Besides further streamlining the kernel, this allows to apply kernel operations also to intermediate results contained in object buffers (cf 2 6)

Some important data structures might well require special treatment within the kernel Examples are sorted subrelations and subrelations with fixed-length subtuples A special implementation of the latter would allow, e g to apply an overlay technique to interpret them as usual arrays of the programming language (FORTRAN matrices in engineering applications) Currently, long fields are bounded by the addressing capacity of a storage cluster (256 disk blocks) To overcome this restriction, special treatment will be implemented [DPS86] following the EXODUS idea [CDRS86]

Our original idea about logging operations of various layers was to map higher level log records just as any other kind of data onto objects of the kernel interface, viz Complex Records Thus the Stable Memory Manager would be the only module performing I/Os A simulation study, however, clearly indicated that this would have resulted in a disastrous number of I/Os, as writing a higher level log record in turn invokes logging of the layers below So, the ongoing implementation of multi-level logging issues log-I/Os directly For each level, an "append-only" log file is managed containing UNDO information and EOT records to mark committed subtransactions Considerations how to reduce the I/O bottleneck can be found in [We86b], among other features, we think about using a group commit mechanism [De84]

Coding of the Stable Memory Manager and the object buffer is completed, as well as the Complex Record operations for inserting, deleting complete records and the retrieval operations As for transaction management, page level recovery is running as part of the SMM, and a parameterized lock manager suitable for the page layer and the kernel layer of our multi-level concurrency control approach is completed, but not yet integrated with the rest of the system Recently, a first version of the kernel system became available that implements most of the described features Now we are testing and evaluating this prototype

Besides the implementation of the kernel, we also have spent efforts in implementing several of the front-ends shown in figure 1 The geo-kernel DBMS is currently devel-

oped for geometric and cartographic objects [SWa86] The design of the 4NF relational front-end has been completed [SS83, DOPSSW85, Scho86, SPS87] The Office Filing Service project is going to implement the "Filing and Retrieval Service" for office documents according to an ECMA and ISO proposal Within this project a reimplementation of the kernel system in C has been started avoiding the above mentioned design errors and hopefully not introducing new ones [PSSW87]

Acknowledgements

First ideas of a new database system came up during the years 1978–82 while one of the authors (H -J Schek) was involved in the AIM project at the IBM Heidelberg Scientific Centre The architectural considerations have been enspired by early attempts to extend the RSS component of System/R for text management and retrieval

References

[Ab86] Abiteboul, S , et al *VERSO, a DBMS Based on Non-1NF Relations*, TR 253, INRIA, 1986

[AB84] Abiteboul, S , Bidoit, N *Non First Normal Form Relations to Represent Hierarchically Organized Data*, ACM PODS, Waterloo, 1984

[AH84] Abiteboul, S , Hull, R *IFO A Formal Semantic Database Model*, ACM PODS, Waterloo, 1984

[As76] Astrahan et al *System R Relational Approach to Database Management*, ACM TODS (**1**), 1976

[BB84] Batory, D S , Buchmann, A P *Molecular Objects, Abstract Data Types, and Data Models A Framework*, VLDB, Singapore, 1984

[BBGLS83] Beeri, C , Bernstein, P A , Goodman, N , Lai, M -Y , Shasha, D E *A Concurrency Control Theory for Nested Transactions*, ACM Symp Princ Distr Comp , 1983

[BK86] Bancilhon, F , Khoshafian, S *A Calculus for Complex Objects*, ACM PODS, Cambridge, 1986

[BP85] Blaser, A , Pistor, P (eds) *Data Base Systems for Office, Engineering and Scientific Applications (in German)*, IFB 94, Springer, 1985

[BRS82] Bancilhon, F , Richard, P , Scholl, M *On Line Processing of Compacted Relations*, VLDB, Mexico, 1982

[BS85] Brachman, R J , Schmolze, J G *An Overview of the KL-ONE Knowledge Representation System*, Cognitive Science, Vol 9, 1985

[Ca86] Carey, M J , et al *The Architecture of the EXODUS Extensible DBMS*, in [DD86]

[CDRS86] Carey, M J , De Witt D J , Richardson, J E , Skekita, E J *Object and File Management in the EXODUS Extensible Database System*, VLDB, Kyoto, 1986

[Ch81] Chamberlin, D D , et al *History and Evaluation of System/R*, CACM, 1981

[Da86] Dadam, P , Kuespert, K , Andersen, F , Blanken, H , Erbe, R , Guenauer, J , Lum, V , Pistor, P , Walch, G *A DBMS Prototype to Support Extended NF^2 Relations An Integrated View on Flat Tables and Hierarchies*, ACM SIGMOD, Washington, 1986

[De84] DeWitt, D J , et al *Implementation Techniques for Main Memory Database Systems*, ACM SIGMOD, Boston, 1984

[De86] Deppisch, U *S-Tree A Dynamic Balanced Signature Index for Office Retrieval*, ACM Conf Res Dev in Inf Retrieval, Pisa, 1986

[DD86] Dittrich, K , Dayal, U (eds) *Proc Int Workshop on Object-Oriented Database Systems*, Pacific Grove, 1986

[DKM85] Dittrich, K R , Kotz, A M , Muelle, J A *A Multilevel Approach to Design Database Systems and its Basic Mechanisms*, Proc IEEE COMPINT, Montreal, 1985

[DLPS85] Dadam, P , Lum, V , Pradel, U , Schlageter, G *Selective Deferred Index Maintenance & Concurrency Control in Integrated Information Systems*, VLDB, Stockholm, 1985

[DO87] Deppisch, U , Obermeit, V *Tight Database Cooperation in a Server-Workstation Environment*, submitted for publication

[DOPSSW85] Deppisch, U , Obermeit, V , Paul, H -B , Schek, H -J , Scholl, M H , Weikum, G *The Storage Component of a Database Kernel System*, TR DVSI–1985–T1, TU Darmstadt, 1985, partly in [BP85]

[DPS86] Deppisch, U , Paul, H -B , Schek, H -J *A Storage System for Complex Objects*, in [DD86]

[FC84] Faloutsos, C , Christodoulakis, S *Signature Files An Access Method for Documents and its Analytical Performance Evaluation*, ACM TOOIS (**2**), 1984

[GS82] Guttman, A , Stonebraker, M *Using a Relational Database Management System for Computer Aided Design Data*, IEEE Database Engineering (**5 2**), 1982

[Gu84] Guttman, A *R-Trees A Dynamic Index Structure for Spatial Searching*, ACM SIGMOD, Boston, 1984

[Ha86] Harder, T *New Approaches to Object Processing in Engineering Databases*, in

[DD86]

[HHLM86] Härder, T , Hubel, C , Langenfeld, S , Mitschang, B *KUNICAD - A Database System Supported Geometrical Modeling Tool for CAD Applications* (in German), to appear in Informatik Forschung und Entwicklung

[HL82] R L Haskin, R Lorie *On Extending the Functions of a Relational Database System*, ACM SIGMOD, Orlando, 1982

[HR85] Härder, T , Reuter, A *Architecture of Database Systems for Non-Standard Applications* (in German), in [BP85]

[KW87] Kemper, A , Wallrath, M *Concepts for the Integration of Abstract Datatypes into R^2D^2* (in German), to appear in Proc GI Conf Data Base Systems for Office, Engineering and Scientific Applications, Darmstadt, 1987

[LKMP85] Lorie, R , Kim, W , McNabb, D , Plouffe, W , Meier, A *Supporting Complex Objects in a Relational System for Engineering Databases*, in Kim, W , Reiner, D S , Batory, D S (eds) Query Processing in Database Systems, Springer, 1985

[MD86] Manola, F , Dayal, U *PDM An Object-Oriented Data Model*, in [DD86]

[MGG86] Moss, J E B , Griffeth, N D , Graham, M H *Abstraction in Recovery Management*, ACM SIGMOD, Washington, 1986

[Mi87] Mitschang, B *The Molecule-Atom Data Model* (in German), to appear in Proc GI Conf Data Base Systems for Office, Engineering and Scientific Applications, Darmstadt, 1987

[NHS84] Nievergelt, J , Hinterberger, H , Sevcik, K C *The Grid-File An Adaptable, Symmetric Multikey File Structure*, ACM TODS (9), 1984

[OM86] Orenstein, J A , Manola, F A *Spatial Data Modeling and Query Processing in PROBE*, TR CCA-86-05, CCA, Cambridge, 1986

[PSSW87] Paul, H -B , Schek, H -J , Soder, A , Weikum, G *Supporting the Office Filing Service by a Database Kernel System* (in German), to appear in Proc GI Conf Data Base Systems for Office, Engineering and Scientific Applications, Darmstadt, 1987

[Ro86] Rowe, L A *A Shared Object Hierarchy*, in [DD86]

[Sche85] Schek, H -J *Towards a Basic Relational NF^2 Algebra Processor*, Conf Found Data Org (FODO), Kyoto, 1985

[Scho86] Scholl, M H *Theoretical Foundation of Algebraic Optimization Utilizing Unnormalized Relations*, Int Conf on Database Theory, Rome, 1986

[Schw86] Schwarz, P , et al *Extensibility in the Staburst Database System*, in [DD86]

[SL83] Schek, H -J , Lum, V (Chairmen) *Complex Data Objects Text, Voice, Images Can DBMS Manage Them?*, Panel, VLDB, Florence, 1983

[SP82] Schek, H -J , Pistor, P *Data Structures for Integrated Data Base Management and Information Retrieval Systems*, VLDB, Mexico, 1982

[SPS87] Scholl, M H , Paul, H -B , Schek, H -J *Supporting Flat Relations by a Nested Relational Kernel*, submitted for publication

[SR84] Stonebraker, M , Rowe, L A *Database Portals A New Application Program Interface*, VLDB, Singapore, 1984

[SRG83] Stonebraker, M , Rubenstein, B , Guttman, A *Application of Abstract Data Types and Abstract Indices to CAD Databases*, IEEE Conf Engineering Design Applications, Database Week, 1983

[SS80] Schkolnik, M , Sorenson, P *Denormalization A Performance Oriented Database Design Technique*, Proc AICA Conf , Bologna, Italy, 1980

[SS83] Schek, H -J , Scholl, M H *The NF^2 Relational Algebra for a Uniform Manipulation of the External, Conceptual, and Internal Data Structures* (in German), in J W Schmidt (ed), IFB 72, Springer, 1983

[SS86] Schek, H -J , Scholl, M H *The Relational Model with Relation-Valued Attributes*, Information Systems (**11 2**), 1986

[St86] Stonebraker, M *Object Management in POSTGRES Using Procedures*, in [DD86]

[SW86] Schek, H -J , Weikum, G *DASDBS Concepts and Architecture of a Database System for Advanced Applications*, TR DVSI-1986-T1, Techn Univ Darmstadt, German Version to appear in Informatik Forschung und Entwicklung

[SWa86] Schek, H -J , Waterfeld, W *A Database Kernel System for Geoscientific Applications*, Conf on Spatial Data Handling, Seattle, 1986

[Tr83] Traiger, I L *Trends in Systems Aspects of Database Management*, ICOD-2, Cambridge (UK), 1983

[We86a] Weikum, G *A Theoretical Foundation of Multi-Level Concurrency Control*, ACM PODS, 1986

[We86b] Weikum, G *Pros and Cons of Operating Systems Transactions for Data Base Sys-*

tems, ACM/IEEE FJCC, Dallas, 1986

[WNP86] Weikum, G , Neumann, B , Paul, H -B *Concepts and Implementation of a Page-Set-Oriented Interface for the Efficient Access to Complex Objects* (in German), to appear in Proc GI Conf Data Base Systems for Office, Engineering and Scientific Applications, Darmstadt, 1987

[WS84] Weikum, G , Schek, H -J *Architectural Issues of Transaction Management in Layered Systems*, VLDB, Singapore, 1984