An Object-Oriented Database System for Engineering Applications

Alfons Kemper
Peter C Lockemann
Mechtild Wallrath

Institut für Informatik II
Universität Karlsruhe
7500 Karlsruhe, West Germany

Abstract

One of the most promising approaches to database support of engineering applications is the concept of object-oriented database management Object-orientation is usually approached from either a behavioral or structural viewpoint The former emphasizes the application-specific manipulation of technical objects while hiding their structural details whereas the latter concentrates on the structural aspects and their efficient implementation The thesis of the paper is that the two viewpoints may enter into a fruitful symbiosis where a behaviorally object-oriented system is implemented on top of a structurally object-oriented database system, thereby combining ease of use by the engineer with high database system performance The thesis will be demonstrated in the paper by a user-friendly interface based on user-definable abstract datatypes and its implementation using a prototype for the non-first-normal-form (NF^2) relational model, and will be supported by an engineering example application from off-line robot programming

1. Introduction

An accepted opinion in the field — albeit often for different reasons — is that the traditional database management systems are insufficient for technical applications For example, in the relational data model technical objects usually have to be decomposed onto different relations For one, this makes the model difficult to use by the database user, i e the engineer, in order to retrieve and manipulate the data For another, performance promises to be poor because of the large number of join operations needed to recompose a technical object

Object-oriented database systems have been proposed by many authors as a new concept for supporting technical applications In the database area in particular, two kinds of object-orientation should be distinguished [Ditt86a] the *structural* and the *behavioral* object-orientation The behavioral approach has a strictly application-oriented flavor What an object is, is largely determined by what a user perceives to be an entity that — at least at times — is to be manipulated as a whole In such an abstract view, data manipulation by necessity is object-type specific Take as examples a geometric object that is to be rotated in space or attached to another such object, or an image that is to be searched for the occurrence of a particular pictorial pattern or overlayed with another image The behavioral approach to databases has its origins in programming languages, particularly the notion of abstract data type Lately, considerable work in this area has been reported in the database literature [Gold83, Cope 84, Maie85, Atwo85, Zdon85, Zdon86a, Zdon86b, Zani86]

By contrast, the structural approach has originated from database technology and is essentially motivated on technical grounds The central notion here is that of "complex object" [Lori83] or of "molecule" [Bato84, Bato85] reflecting the fact that objects in the engineering world are composed of parts that may among themselves undergo a variety of other relationships Typical approaches are based on hierarchical extensions to the relational model such as

The work described in this paper was done within the R^2D^2 (A Relational Robotics Database System with Extensible Datatypes) project R^2D^2 is a cooperation project among the IBM Scientific Center Heidelberg and the University of Karlsruhe, Fakultät für Informatik

XSQL [Hask82] or the NF^2 data model [Sche82, Lum85, Dada86], and extensions to the entity-relationship model [Zani83, Glin86, Ditt86b] Although the structural view appeals to the engineering user who composes technical objects from parts, the important issue is to preserve the object structure across the implementation levels all the way to peripheral storage in order to gain in performance

The thesis of this paper is that the two approaches do not contradict but rather supplement one another They may enter into a fruitful symbiosis such that one takes an existing structurally object-oriented database management system, or explicitly develops one, and augments it by a behaviorally object-oriented user interface Whereas the user views the database from an abstract level where the object structure becomes visible only indirectly via the object operators, the object structure is explicitly established when mapping the behavioral level to the structural level, so that all optimization techniques available on the latter level may be utilized

Such a two-level approach appears to have three distinct advantages

1 Engineering users deal with databases only indirectly via programs such as various design tools Implementation of such tools takes place on the level of engineering objects and does not require any longer a detailed knowledge of database technology
2 Implementation of the engineering object types may also be shifted from the database expert to the engineering applications specialist, since mapping the object structures becomes a fairly routine matter
3 Database system performance can still be controlled in a manner geared to engineering applications

Nonetheless, the approach has so far not been exercised Behavioral approaches have tended to emphasize language design or database access from within existing language environments [Atwo85, Cope84, Maie85] All structural approaches tend to emphasize appropriate additions to existing data models or the development of new data models, with data manipulation restricted to the classical operations of navigation or set formation, insertion, update, and deletion [Zani83, Hask82, Sche82, Dada86, Ditt86b] A few attempts have been made to integrate abstract data types within database system interfaces, these, however, have restricted themselves to non-object-oriented database systems [Ston83a, Osbo86] Thereby they have to cope with a rather unnatural mapping of objects that exist at the user interface onto the internal structures Thus externally existing structural objects lose their structure in the internal representation This is also true for systems offering the concept of long field [Lori82, Lori83, Ditt86b]

In this paper we present the database system R^{2}D^2 (A **R**elational **R**obotics **D**atabase System with Extensible **D**atatypes) R^{2}D^2 constitutes a symbiotic approach to object-oriented database systems This is achieved by integrating the concept of abstract data types in the data definition and data manipulation language of a structurally object-oriented DBMS Thereby the database user can define his own data types and operations which correspond to his application-specific objects and object manipulations The behavioral object-orientation in R^{2}D^2 will be demonstrated on a specific engineering application, offline robot programming The structural aspects are given by choosing a particular object-oriented data model, the NF^2 (non-first-normal form) model The NF^2 data model allows nested relations whereby hierarchical relationships among subobjects can be modelled The internal representation of an ADT corresponds in R^{2}D^2 to a (possibly) nested NF^2-relation Thereby it is guaranteed that the objects defined at the user level are also internally treated as clustered objects This improves the performance of retrieving an object as a whole over systems where such an object is segmented over different (flat) relations After introducing both levels, we demonstrate how a quick integration of the two levels can be achieved, that offers modest gains in performance as compared to a basis with flat relations We conclude by discussing how a tighter integration could be obtained, what efforts have to go into such a solution, and what performance gains could be expected

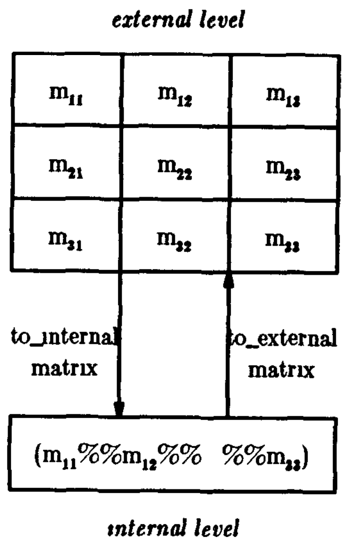
2. The Structural Basis

2 1 An Example Flat Relations

One critique regarding structurally object-oriented DBMS as a basis is that such systems are not yet commercially available, and that in any case they lack both the elegance and the solid theoretical foundation of the pure relational model This is an argument that has to be taken seriously We start, therefore, by surveying some related work that uses flat relations, and by discussing their drawbacks with regard to efficiency of object implementation and manipulation

The best-known such work is ADT-INGRES ADT-INGRES was proposed by Stonebraker [Ston83a] and implemented as an experimental prototype on top of the existing DBMS INGRES [Ston76] by Fogg [Fogg82] ADT-INGRES* provides a novel way of specifying new datatypes and corresponding operators in a database management system For example, in the context of engineering applications a commonly used data object is a 3x3 matrix To introduce it into ADT-INGRES, the user could specify the new datatype *matrix* together with possible operations, like matrix multiplication

Since ADT-INGRES was built on top of the flat relational system INGRES, the internal representation of abstract datatypes can not preserve the object structure that exists at the abstract level All abstract datatypes have to be mapped internally onto a single INGRES domain, in general a character string Schematically this would look for the example *matrix* as follows



external level
m_{11} m_{12} m_{13}
m_{21} m_{22} m_{23}
m_{31} m_{32} m_{33}
to_internal matrix
to_external matrix
$(m_{11}\%\%m_{12}\%\% \ \%\%m_{33})$
internal level

The labels *to_internal_matrix* and *to_external_matrix* correspond to functions that convert the external format of a matrix, i e what the user sees, to the internal representation and vice versa These functions have to be supplied by the user and have to be written in the programming language C [Ritc 78] Since all abstract datatypes have to be mapped entirely onto one and the same domain, the abstract structure of a data object gets completely lost In our example, the simple structured object *matrix* has to be decomposed into its 9 elements which have to be mapped into a character string with appropriate special symbols(%%) delimiting two elements Therefore the functions to convert from the external to the internal representation and vice versa can become quite complex to implement Also operations on ADTs, like matrix multiplication, have to be specified in C and have to be implemented on the unstructured internal representation of a data object Thus the ADT-INGRES user has to be familiar with two quite different systems (i) the database language QUEL and (ii) the programming language C The mapping itself could be considered low-level, unnatural and quite tedious

* The analysis of ADT-INGRES is – except for minor details – also valid for RAD [Osbo86]

Also, in ADT-INGRES it is not possible to implement new abstract datatypes in terms of previously defined ADTs For example, one could not use a previously defined datatype *vector* with the operations vector addition and multiplication in order to specify the ADT *matrix* and its operations Again, this stems from the fact that the external object structure is not forwarded to the internal level Therefore stepwise refinement in the implementation of new datatypes is not supported in ADT-INGRES

POSTGRES [Ston86], a redesign of INGRES, incorporates the ideas of ADT-INGRES and "Quel as a Datatype" to deal with complex objects "Quel as a Datatype" [Ston83b] was proposed by Stonebraker et al as an extension to the database management system INGRES [Held 76] It allows attributes of relations to be of type QUEL That is, the attribute consists of a QUEL query which retrieves as a result tuples from one or more different relations The purpose of this extension is to provide a very general referencing mechanism The database designer could define new objects, such as vectors, cubes, arrays, etc , in separate relations and access them from the parent relation via an attribute of type QUEL A problem with this concept seems to be that the representation is split up into very small partitions This might lead to inherently inefficient data manipulation processes, unless we can manage to cluster data appropriately Furthermore, the system does not provide a facility to define new operations on complex objects

2 2 The NF² Data Model

We conclude that a structurally object-oriented DBMS certainly is an alternative worthwhile to be explored in more depth We decided to use a DBMS whose development – although as a lab product – had progressed sufficiently well to provide a stable basis for our work Our choice fell on the AIM-P implementation of the NF² data model [Dada86] with the extended SQL language interface HDBL [Pist86] The NF² data model as introduced in [Sche82] provides, in addition to the usual built-in atomic data types, the composite data types ***tuple, list and relation*** The concept of the NF² tuple is analogous to the conventional relational model A list is an ordered set of elements, whereby the elements can be of atomic or again of composite type The elements of a list can be accessed by their position Attributes can be of type relation which, again, can have attributes of type relation Thus we can have arbitrarily deep nesting of relations

We illustrate NF² relations using a simplified schema to model industrial robots (for further details see [Kemp86]) The schema definition for a relation ROBOTS, in a syntax according to [Pist86], is shown in Figure 1 below

In the schema of Figure 1 we model a robot as an NF² object Of course this is by no means a complete representation model of a robot, rather it is intended to emphasize the database aspects of such a system Schematically the relation ROBOT is shown in Figure 2 It consists of the atomic attribute ID which denotes the name and type of the robot In addition, the relation has the two attributes ARMS and GRIPPERS The attribute GRIPPERS is of type relation since there are several possible grippers that can be connected to a particular robot (relations are denoted in AIM syntax by the pair of brackets {[ ]})

```
create ROBOTS {
  [ ID  string(20),
    ARMS  {
     [ ID  string(20),
       AXES  <              /* ordered list of axes */
         [  KINEMATICS
             [ DH_MATRIX   /* 4x4 Denavit-Hartenberg
                              Matrix */
                 < 4 FIX [COLUMN  integer,
                          VECTOR  <4 FIX real>] >,
               JOINT-ANGLE
                 [ MAX  integer,
                   MIN  integer ]
             ],
           DYNAMICS
             [ MASS  real,        /* mass of the axis */
               ACCEL  real ]   /* max acceleration */
           ] >          /* end of axis description */
     ] },
   GRIPPERS  {
     [ ID  string(20),
       FUNCTION  string(20)
     ] }          /* end of endeffector description */
  ] }
end
```

Figure 1 Definition of the NF²-Relation ROBOTS

In the following we will concentrate on the attribute ARMS The arm of a robot consists of a given number of axes Therefore, we model the axes of an arm as a list (denoted by the brackets < >) Since lists have an implied order we can model the relationship between predecessor- and successor-axis in the way that AXES[i] is the predecessor-axis of AXES[i+1] Each axis, i e each tuple (denoted by [ ]) of the list AXES, consists of the attributes KINEMATICS and DYNAMICS

The kinematics of an axis is typically modelled in relation to the predecessor-axis, i e , the axis that is closer to the robot basis The relative position of an axis is stored in the so called Denavit-Hartenberg (DH-) matrix This is a 4x4 matrix which describes the position of the joints relative to each other in the coordinate system of the basis This matrix has to be stored for each joint

As an example of an NF² subobject let us take a closer look at the Denavit-Hartenberg matrix

```
DH_MATRIX   /* 4x4 Denavit-Hartenberg matrix */
   <4 FIX [COLUMN  integer,
           VECTOR  <4 FIX real>]>
```

This matrix is here defined in column-format, that is, the DH matrix consists of a list of four elements, denoted by the bracketing <4 FIX > The elements of this list are tuples which themselves consist of two attributes, the atomic attribute COLUMN of type integer and the attribute VECTOR, which itself is a list of 4 floating-point numbers

Thus a data-filled list of such a DH matrix would look as follows

```
                  DH_MATRIX
  COLUMN                        VECTOR
 ---------------------------------------------
    1                     <1 0,2 5,3 5,0 0>
    2                     <                 >
    3                     <                 >
    4                     <                 >
```

In the following we present a few data manipulation constructs of the NF² data model Let us first demonstrate the insertion of data into a nested NF² structure In this case we insert the list <0,1,0,0> into the second column of the DH matrix that describes the second axis of the "left" arm of the robot "Artoo Detoo"

{ROBOTS}									
ID	{ARMS}							{GRIPPERS}	
	ID	<AXES>						ID	FUNCTION
		KINEMATICS				DYNAMICS			
		<DH_MATRIX>		JOINT-ANGLE		MASS	ACCEL		
		COLUMN	<VECTOR>	MAX	MIN				
Artoo Detoo	left	1	<1,0,0,0>	-180	180	50	1 0	#200	gripper
		2	<0,0,-1,0>						
		3	<0,1,0,0>					#150	welding machine
		4	<0,0,100,1>						
		1	<1,0,0,0>	-250	60	37 26	2 0		
		2	<0,1,0,0>						
		3	<0,0,1,0>						
		4	<70,0,20,1>						
		1	<0,1,0,0>	-80	250	10 4	6 0		
		2	<0,0,1,0>						
		3	<1,0,0,0>						
		4	<0,40,-10,1>						

Figure 2: The NF² Relation ROBOTS

```
insert
   [ COLUMN  2,
     VECTOR  <0,1,0,0> ]
into AR AXES[2] KINEMATICS DH_MATRIX[2]
from AR in R ARMS,
     R  in ROBOTS
where R ID = "Artoo Detoo" and AR ID = "left"
```

This example demonstrates the additional complexity of the operations on the nested NF² structures over conventional flat relations The nesting of the relations is reflected precisely in the operation, that is, the programmer has to specify where the subobject is located within the nested structure For the other two column vectors of the DH matrix one should use an analogous insert operation

Data retrieval in AIM is based on the *select from where* clause of the SQL database language [SQL/DS] Contrary to SQL, in AIM these constructs can be nested according to the nesting of the NF² objects over which the query is formulated The following example should demonstrate this

```
select [ R ID,
     select [ AR ID,
              AR AXES[2] KINEMATICS DH_MATRIX
            ]
     from AR in R ARM
     where AR ID = "left"
       ]
from R in ROBOT
where R ID = "Artoo Detoo"
```

In the above query we retrieve the robot identifier, the arm identifier and the DH matrix of the 2nd axis of the "left" arm of the "Artoo Detoo" robot Again, we observe a nesting of the query corresponding to the traversal within the hierarchical information structure of a robot model

The example should have made clear that expressing the complex, albeit hierarchical topology of a robot is a straightforward affair with NF² relations On the other hand, updating such a relation or retrieving elements from it is by no means trivial but requires an intimate knowledge of the schema and of HDBL Clearly then, our first expectation mentioned in the introduction — ease of use by the engineering user — is not met by a structurally object-oriented data model (similar observations hold for other such models, e g , CERM [Ditt86b])

3. Behavioral Object-Orientation in R²D²

3 1 *User-Defined Datatypes*

The problems with the complexity of nested queries can be avoided by using abstract datatypes, as we demonstrate below An ADT can be mapped onto a nested NF² relation, but the database user need not be aware of the internal representation if he only wants to use the pre-defined ADT operations

Language concepts for defining abstract datatypes have been integrated in programming languages for a long time Abstract datatypes are used for objects that are frequently used and whose internal representation should be hidden from the user Here we want to demonstrate the abstract datatype facility of R^2D^2 from a user's perspective We show how the datatypes and operations that are essential in robot programming can be integrated into the NF^2 model, thereby making the datamodel easier to use for storing engineering data

One advantage of this approach over others such as the ones mentioned in ch 2 1 is that both the internal representation of an abstract object and the object operations are specified in the same language, the former in a slightly extended data definition language, the latter in an extended SQL-like language In addition, R^2D^2 supports stepwise refinement of ADTs in the way that one can utilize previously defined ADTs to specify a new abstract datatype

The general approach to abstract datatype definition is as follows The NF^2 datamodel has, like most other models, only a limited set of built-in basic datatypes, such as character, numeric, boolean, etc Of these basic datatypes one can create structured objects, such as lists, tuples and relations, with nesting of structured objects allowed, e g a list could have elements that are lists In R^2D^2 we allow the user in addition to define his own datatypes These can be much more complex than the basic NF^2 types In R^2D^2 a user-defined datatype can be any "structured" NF^2 object The following syntax is used to define new datatypes

```
create ADT <identifier>
is <nf2_type>
with <operations>
end <identifier>
```

The with clause contains the definition of operations for the abstract datatype These will be discussed in the next section The expression <nf2_type> denotes any NF^2 object Since we allow an abstract datatype to be any NF^2 object it is, in particular, possible to define abstract datatypes that are nested NF^2 objects Furthermore, user-defined datatypes can be used like any built-in datatype, i e attributes can be of a type that has been previously defined by the user as an ADT

Let us now demonstrate how we can support in R^2D^2 the datatypes that are needed in robotics applications as described above For this purpose we will first define the ADTs vector and matrix

```
create_ADT vector
is < 4 FIX  real >          /* exactly length 4 */
with                        /* operations on vector */
end vector


create_ADT matrix
is < 4 FIX vector >   /* exactly 4x4 (a list of a list)
with                  /* operations on matrix */
end matrix
```

In the definition of these datatypes we use the built-in NF^2 data structure *list* which is denoted by the pair of brackets "< >" The expression '4 FIX' denotes that the list has exactly 4 elements The NF^2 model allows the user to access elements of a list by their position in the list, e g vector[i] returns the ith component of the list vector Another built-in function on lists is INDL which returns the index range of a list In our example INDL(V) returns 1 4, where V is of type vector

The definition of the ADT *matrix* neatly demonstrates how user-defined datatypes can be used like built-in datatypes *matrix* consists of a list of vectors, where *vector* is an ADT consisting of a list of 4 numeric values Therefore, the nesting within the datatype *matrix* is actually hidden from the user who might not be aware of the internal implementation of a matrix

In addition to these basic types that are needed to implement robotics applications one also needs to use more complex objects As mentioned above, a robot axis is described by its dynamics and its kinematics The position of an axis is described by a 4x4 matrix, i e we can use the ADT *matrix* in order to define an ADT *axis*

```
create ADT axis is
   [ ID  string(20),
     KINEMATICS
       [ DH_MATRIX  matrix],
     DYNAMICS
       [ MASS   real,
         ACCEL  real ]
   ]
with                      /* operations on axis */
end axis
```

3.2 *Definition of New Operations*

So far we dwelt only on the structural aspects of the ADT concept For a behavioral orientation, the definition of an ADT should include user-defined operations In R^2D^2 these operations are defined in an SQL-like language An operation can either be specified within the ADT definition clause like in the examples below, or can explicitly be added to an existing ADT at some later time Rather than presenting the formal syntax of this language let us demonstrate it from a user's view by implementing some example operations on the ADTs ***vector*** and ***matrix***

```
create ADT vector
is < 4 FIX real >
with
operation "+_V" (V,W vector) returns vector
     return
        select V[i]+W[i]
        from i in INDL(V)
     end "+_V",


operation "*_V" (V,W vector) returns real
     return
        SUM (select V[i]*W[i]
             from i in INDL(V) )
     end "*_V"
end vector
```

In the last operation "$*_V$" we use the built-in function SUM which takes as an argument a list of numeric values and returns their sum We also note that we allow for infix operators, which are denoted by enclosing the operator into quotation marks

```
create ADT matrix
is < 4 FIX vector >
with
operation"*_V,M" (V vector,M matrix) returns vector
     return
        select V *_V M[i]
        from i in INDL(M),
     end "*_V,M",

operation row (k integer,M matrix) returns vector
     return
        select M[i][j]
        from i in INDL(M),j in INDL(M[i])
        where j=k
     end row,


operation transpose (M matrix) returns matrix
     return
        select row(i,M)
        from i in INDL(M),
     end transpose,


operation "*_M" (M,N matrix) returns matrix
     return
        select
             select transpose(M)[i] *_V,M N[j]
             from i in INDL(M)
        from j in INDL(N)
     end "*"
end matrix
```

The last operation (query) obviously returns as a result a list of a list The inner select statement loops through all column vectors of transpose(M), i e the rows of the matrix M, and multiplies them with the j^th column of N, where j is fixed The outer select loops through all j in INDL(N), that is, it loops through all column vectors of N The result of this query, therefore, is the multiplication of the two matrices M and N

In the section 3 1 we defined the ADT ***axis*** For this ADT we now want to define an operation ***rotate$_z$*** which rotates an axis around its z-coordinate axis by an angle alpha Rotation of axes is described by multiplying the axis position relative to the robot basis as expressed by the DH-matrix with a special 4x4 rotation matrix The latter is created by an operation *rot_matrix$_z$* that should be part of the ADT ***matrix***

```
operation matrix_rot_z (alpha  real) returns matrix
  return

end matrix_rot_z
```

Accordingly then, rotation may now be introduced into the ADT ***axis*** in the form of an additional operation ***rotate$_z$***

```
operation rotate_z (A  axis, alpha  real) returns axis
   return
      [ ID  A ID,
        KINEMATICS
           [ DH_MATRIX  matrix_rot_z(alpha) *_m
                        A KINEMATICS DH_MATRIX ],
        DYNAMICS  A DYNAMICS
      ]
end rotate_z
```

The abstract data types and operations that were introduced here can be used in the data manipulation constructs like any built-in data type and operation Thus, a query that is based on a complex data object becomes much easier to formulate for the user because he is not required to know the internal representation of the ADT if all he wants is to apply the pre-defined operations We demonstrate this on an example where we want to retrieve a specified axis after rotating it by 30 degrees around the z-coordinate axis

```
   select rotate_z (AR AXES[2],30)
   from AR in R ARMS,
        R  in ROBOTS
   where R ID = 'Artoo Detoo' and AR ID = 'left'
```

4. Implementation of R^2D^2

4 1 *Implementation Aspects*

Chapter 3 should have substantiated our claims that R^2D^2 provides an easy-to-use object-oriented interface to the engineering user, and also to the engineering application specialist who wishes to declare new datatypes We now turn to our third objective maintaining the advantages that structurally object-oriented DBMS offer in terms of performance We start by sketching a "quick and dirty" solution that leaves AIM-P largely untouched We used this approach to implement an operational prototype in relatively short time in order to facilitate first tests by engineering applications Therefore, we decided to implement R^2D^2 as a preprocessor to the database system AIM-P that was developed at the IBM Scientific Center Heidelberg [Dada86] The basic architecture of AIM-P is shown in the lower half of Figure 3 [Lum85] The Buffer Manager and the Segment Manager are fairly conventional The same holds for the Catalog Manager except that the system catalogs are themselves nested NF^2 relations The Subtuple Manager is responsible for storing the smallest accessible

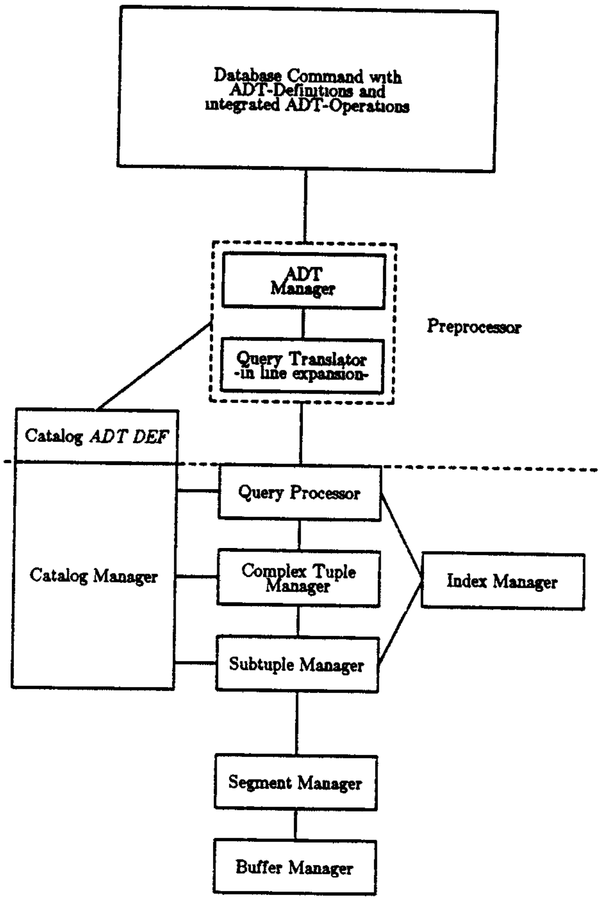


Figure 3: Preprocessor Architecture of R^2D^2

data elements of the database, i e the data subtuples Furthermore, this module provides mechanisms for version and access control The task of the Complex Tuple Manager is the management of complex tuples that can consist of many subtuples In order to gain performance this module attempts to store a complex tuple in a physical cluster The Index Manager of AIM-P is considerably more complex than in conventional database management systems because the addresses used in access paths have to be hierarchical It is not sufficient to just maintain the address of the referenced subtuple but one also has to keep track of the parent tuples of this subtuple The Query Processor parses and translates interactive HDBL programs In addition to the interactive user interface AIM-P also provides an application program interface (API) for the programming language Pascal

4 2 Preprocessor Implementation

With respect to abstract datatypes the R^{2}D^2 user can perform the following operations

1 definition of new ADTs
2 usage of a predefined ADT as an attribute type in a data definition statement
3 usage of an ADT operation in a data manipulation statement

The basic idea of the preprocessor implementation is to translate all ADT constructs in HDBL statements that can be processed by the AIM-P Query Processor Each ADT has an associated internal NF^2 representation that is constructed by the create ADT command, and maintained in a separate AIM-P catalog (see Figure 3) This catalog (ADT-DEF) is itself a nested NF^2 relation, defined as shown in Figure 4

Once an ADT construct is used in an HDBL data definition or data manipulation statement it is textually expanded using the internal representation This *inline expansion* [Ada83] may proceed across several levels depending on the depth of the ADT definition hierarchy, but is guaranteed to terminate because in an ADT definition we do not allow any recursion To demonstrate this idea let us look at a very simple data definition example in which we make use of the previously defined ADT vector Suppose we define a new ADT *cuboid*

```
create ADT-DEF {[
          IDENTIFIER  string(16),
          OWNER  string(16),
          REPRESENTATION  text(1024),
          OPERATIONS  {[
                         OP_NAME  string(16),
                         ARGUMENTS  < string(32) >,
                         RESULT  string(16),
                         STATEMENT  text(2048),
                         DESCR  text(1024)
                      ]}
          DESCR  text(1024),
          IMPORT  < string(16) >,
          EXPORT  integer,
          CONTROL  {[
                      USER  string(16),
                      USE  bool,
                      ACCESS  bool,
                      MODIFY  bool
                    ]}
      ]}
end
```

Figure 4: System Catalog ADT-DEF

```
create ADT cuboid is
  < 8 FIX [V_ID  string(2),  /* v1, v2,  , v8 */
           VERTEX  vector] >
with
operation in_origin(C cuboid) returns bool
  return
    exists (V in C)  V VERTEX = < 0,0,0,1 >
  end in_origin
end cuboid
```

Suppose further that we now use the new ADT in an HDBL data definition statement as follows

```
create Cuboid_Set                         /* a set of cuboids */
  { [ ID  string(16),
      REPRESENTATION  cuboid ] }
end
```

By expanding this statement the preprocessor translates the DDL command into a valid HDBL statement

```
create Cuboid_Set
  { [ID  string(16),
    REPRESENTATION          expansion of "cuboid"
        +-------------------------------------------------+
        |     < 8 FIX [ V_ID  string(2),                  |
        |               VERTEX      expansion  of 'vector'|
 cuboid |                                                 |
        |                       +-----------------------+ |
        |                       |  < 4 FIX real > ] >   | |
        |                       +-----------------------+ |
        +-------------------------------------------------+
    ] }
end
```

In a way this could be considered a "structural" expansion Expansion of operators is needed in case of data manipulation, as the following query demonstrates that retrieves all cuboids that have one vertex in the origin

```
select Q ID
from   Q in Cuboid_Set
where  in_origin(Q REPRESENTATION)
```

This query is translated into the following valid HDBL query

```
select Q ID
from   Q in Cuboid_Set
where                    expansion of 'in_origin'
+--------------------------------------------------------+
| exists (V in Q REPRESENTATION)  V VERTEX = <0,0,0,1>,  |
+--------------------------------------------------------+
```

As mentioned before, this expansion could go over several levels, e g if one expanded the operations for matrix multiplication

4.3 Tighter Coupling of R^2D^2

In the preceding subsection we described a first implementation of R^2D^2 This implementation basically consists of introducing another system layer to a structurally object-oriented DBMS The advantage of our approach – having the NF^2 model as a basis of the implementation – stems from the fact that hierarchically defined abstract datatypes are also stored in hierarchical database structures This clustering certainly enhances the system performance of retrieving such hierarchical complex objects from the database

In further experiments with the system we want to investigate the migration of the more basic datatypes and operations to a lower level of the database system In [Schw86] it is argued that operations on abstract datatypes should be passed down in the DBMS rather than passing the data up to the operations at the interface level It is conjectured that this will lead to performance gains since the operation can be carried out directly on the stored data

In our opinion this approach is only feasible for the implementation of the more basic datatypes In our application domain these datatypes would clearly be

- vectors with 3 and 4 elements
- matrices of dimension 3x3 and 4x4
- geometric primitives, like cuboid, pyramid, and cylinder representations

If any of the more application-specific abstract datatypes, e g robot axis, should be left on the interface level where they could be defined based on these pre-defined datatypes, or migrate to lower levels as well would presumably have to depend on the dynamic characteristics of the application Consequently, in the long run one should develop mechanisms for easy migration of ADTs previously defined on the user level

A further extension to the current R^2D^2 implementation concerns access support It is argued in the literature, e g [Ston83a, Maie86, Schw86] that it is necessary to extend the conventional access support mechanisms of current DBMSs The facility to define new operations on database objects not only increases the expressive power of the system but also requires better access support facilities to efficiently support retrieval operations using these operations We want to demonstrate this on a small example based on our previously defined relation Cuboid_Set For the ADT cuboid we could define the operation volume as

```
operation volume(C cuboid) returns real
    return
        /* computation of the volume */
end volume
```

This operation could then be used in a query as follows

```
select C ID
from C in Cuboid_Set
where volume(C REPRESENTATION)=100
```

In order to support such queries we designed a facility in R^2D^2 to define indexes over operations, that is one can define an index over a computed value For our example one would define

```
create index Cuboid_Volume
on Cuboid_Set
using volume(Cuboid_Set REPRESENTATION)
```

Now this access path has to be updated each time the relation Cuboid_Set is modified But by allowing the user only to access the ADT attribute REPRESENTATION via the pre-defined ADT operations, i e enforcement of information hiding, the system can control which modifications require an update of the index For example, a *rotation* of the cuboid would not change the volume, but if a *scale* operation is performed the index has to be updated

A final remark concerns user programming performance Like in classical programming, a thorough type checking facility would detect errors at a time where a timely and meaningful response to the user is still possible Due to the inline expansion, some of the type information becomes lost before it is needed Complete type checking requires changes in the AIM-P system catalog

5. Conclusions

In this paper we have shown how abstract datatypes integrated in the structurally object-oriented data model NF^2 can support engineering applications The presentation of our language proposal was mostly carried out from a user's view, that is, it concentrated on demonstrating how some example datatypes and operations could be specified in the extended SQL-language of R^2D^2

A first preprocessor implementation of R^2D^2 is currently operational which is based on the NF^2 prototype implementation AIM-P developed at the IBM Scientific Center Heidelberg Another project partner, the robotics group at the University of Karlsruhe, has finished a first example application in the area of robot simulation and is about to gain first experiences with the system If the tests prove successful it is planned to extend the new approach to other engineering applications on the campus

6. References

[Ada83] United States of America, Department of Defense Ada – Reference Manual, 1983

[Atwo85] T M Atwood An Object-Oriented DBMS for Design Support Applications, Proc IEEE Compint 1985, 299-307

[Bato84] D S Batory, A.P Buchmann Molecular Objects, Abstract Data Types and Data Models A Framework, Proc 10th Internat Conf on Very Large Data Bases, 1984, 172-184

[Bato85] D S Batory, W Kim Modeling Concepts for VLSI CAD Objets, ACM Trans on Database Systems, 10, (1985), 322-346

[Cope84] G Copeland, D Maier Making Smalltalk a Database System, Proc ACM SIGMOD Conf on Management of Data, 1984, 316-325

[Dada86] P Dadam, et al A DBMS Prototype to Support Extended NF^2-Relations An Integrated View on Flat Tables and Hierarchies Proc ACM SIGMOD Conf on Management of Data, 1986, 376-387

[Depp85] U Deppisch, J Guenauer, G Walch Speicherungsstrukturen und Addressierungstechniken für komplexe Objekte des NF^2-Relationenmodells Informatik-Fachberichte 94, 1985, 441-459 (in German)

[Ditt86a] K R Dittrich Object-Oriented Database Systems The Notion and the Issues, Proceedings International Workshop on Object-Oriented Database Systems, Pacific Grove, Ca, Sept 1986, 2-6

[Ditt86b] K R Dittrich, W Gotthard, P C Lockemann Complex Entities for Engineering Applications, Proc 5th ER Conf, North-Holland, 1986

[Fogg82] Fogg, D Implementation of Domain Abstraction in the Relational Database System INGRES, Masters Report, EECS Dept, Univ of Calif, Berkeley, Ca, Sept 1982

[Glin85] M Glinz, H Huser, J Ludewig SEED – A Database System for Software Engineering Environments, Informatik-Fachberichte 94, Springer 1985, 121-126

[Gold83] A Goldberg, D Robson Smalltalk-80 The Language and its Implementation, Addision-Wesley, 1983

[Gutt82] Guttman, A, M Stonebraker Using a Relational Database Management System for Computer Aided Design Data, Database Engineering, June 1982

[Hask82] R L Haskin, R A Lorie On Extending the Functions of a Relational Database System, Proc ACM SIGMOD Conf 1982, 207-212

[Jaes82] G Jaeschke, H-J Schek Remarks on the Algebra of Non First Normal Form Relations, Proc ACM SIGACT-SIGMOD Symp on Principles of Data Base Systems, Los Angeles, Cal, 1982, 124-138

[Kemp86] A. Kemper, M Wallrath, P C Lockemann Ein Datenbanksystem für Robotikanwendungen Robotersysteme 2(1986) 177-186 (in German) Also in English to appear in Proceedings NATO Intl Advanced Workshop on Languages for Sensor-Based Control in Robotics, NATO ASI Series, Springer, 1987

[Lori82] R Lorie Issues in Databases for Design Applications, in File Structures and Databases for CAD, J Encarnacao and F L Krause (ed), North Holland, 1982

[Lori83] R Lorie, W Plouffe Complex Objects and Their Use in Design Transactions, Proc ACM-SIGMOD Conf on Management of Data, 1983, 115-121

[Lum85] V Lum et al Design of an Integrated DBMS to Support Advanced Applications Proc Int Conf on Foundations of Data Organization, 1985, 21-31

[Maie85] D Maier, A Otis, A Purdy Object-Oriented Database Development at Servio Logic, IEEE Database Engineering 8(1985), No 4, 58-65

[Maie86] D Maier, J Stein Indexing in an Object-Oriented DBMS, Proc Intl Workshop on Object-Oriented Database Systems, Pacific Grove, Ca , Sept 1986, 171-183

[Osbo86] S L Osborne, T E Heaven The Design of a Relational Database System with Abstract Data Types for Domains, ACM Trans on Database Systems, Vol 11, No 3, Sept 1986, 357-373

[Pist86] P Pistor, F Andersen Designing a Generalized NF^2 Data Model with an SQL-type Language Interface Proc 12th Int Conf on Very Large Databases, 1986, 278-285

[Ritc78] Ritchie The C Programming Language, Prentice Hall, Englewood Cliffs, N J , 1978

[SQL/DS] IBM SQL/Data System, Concepts and Facilities, IBM Corporation, GH 24-5013, Jan 1981

[Sche82] H J Schek, P Pistor Data Structures for an Integrated Data Base Management and Information Retrieval System, Proc VLDB Conf , Mexico City, 1982

[Schw86] P Schwarz et al Extensibility in the Starburst Database System, Proc Intl Workshop on Object-Oriented Database Systems, Pacific Grove, Ca , Sept 1986, 85-92

[Ston76] M Stonebraker et al The Design and Implementation of INGRES, ACM Trans on Database Systems, Vol 2, No 3, Sept 1976

[Ston83a] M Stonebraker, B Rubenstein, A. Guttman Application of Abstract Data Types and Abstract Indices to CAD Databases Proc Database Week – Engineering Design Applications IEEE Comp Soc 1983

[Ston83b] M Stonebraker, E Anderson, E Hanson, B Rubenstein QUEL as a Datatype Mem No UCB/ERL M83/73 Univ of Calif at Berkeley 1983

[Ston86] Stonebraker, M and Rowe, L The design of POSTGRES, Proc 1986 ACM-SIGMOD Conference on Management of Data, Washington, D C , May 1986, 340-355

[Zani83] C Zaniolo The Database Language GEM, Proc 1983 ACM-SIGMOD Conference on Management of Data, San Jose, Ca , May 1983, 207-218

[Zani86] C Zaniolo, et al Object-Oriented Database Systems and Knowledge Systems, In L Kerschberg(ed) Proc 1st Internat Workshop an Expert Database Systems, Benj Cummings Publ Co 1986, 49-64

[Zdon85] S Zdonik Object Management Systems for Design Environments, IEEE Database Engineering 8 (1985), No 4, 23-30

[Zdon86a] S B Zdonik, P Wegner Language and Methodology for Object-Oriented Database Environments, Proc 19th Ann Hawaii Internat Conf on System Sci 1986, 378-387

[Zdon86b] S B Zdonik Version Management in an Object-Oriented Database, Proc IFIP WG 2 4 Internat Workshop on Adv Progr Environments 1986, 397-416