PANDA:
AN Object-Oriented PASCAL
Network Database Management System

Andrew U. Frank

Report No. 57

ABSTRACT

Object-oriented software design methods are very important for Computer Aided Engineering systems. The PANDA database we have built is suitable for such an environment, which is based on an extended network concept and strongly influenced by the functional data model [Shipman 1981]. Database objects follow an object semantic (as opposed to a value semantic), and are defined in separate modules without restrictions on their internal structure. All object types are considered specializations of a generic type for which the database operations are defined. Thus, all objects inherit database operations such as access by key, access through association, etc. Database operations must not use any information about the internal structure of the objects, and employ access operations defined in the object modules to retrieve values associated with an object. The system is implemented and in use for research and instruction at a few locations.

1 INTRODUCTION

In many areas of application, especially where geometric properties of real objects are considered, modelling methods that can combine simple objects to form more complex ones are necessary, and must be merged with database management systems and graphics packages. A suitable database management system (DBMS) must support this approach by object-oriented access and manipulation tools [Battory 1985]. Standard, non-object-oriented databases offer a limited selection of data types, which are usually not sufficient for engineering applications. This paper describes a method of dealing with the incongruities between an object-oriented modular software design, based on abstract data types, and a generic database management system.

The applications of particular interest to us are Cartography and CAD/CAM, The problems we encountered in our practical work are, however, general, and are found in a similar form in other areas. We are convinced that a formal and general treatment will provide us with more insight and lead us to better solutions than will an ad hoc 'fudging' applicable to one situation only.

This paper describes the concepts we used in the design of the object-oriented PANDA DBMS. First, we will explain the term 'object-oriented' as used in this paper. As background, we will briefly introduce the programming environment we used to implement large software systems. We will then explain what perceive to be the clash between object-oriented design and database functionality, and will go on to show how this can be overcome. Finally, we will present a data model suitable for an object-oriented approach, as implemented in our object-oriented database PANDA [Frank 1982], a CODASYL-like network database. The data model and the DBMS are currently used at several sites for research as well as for graduate and undergraduate instruction.

2 OUR NOTION OF THE OBJECT

We will use the word 'object' to describe a single occurrence (instantiation) of data describing something having some individuality, some observable behaviour, and upon which some

operations can be carried out. The terms 'sort', 'type', 'ADT', or 'module' will be used to refer to types of objects.

In our view, object-oriented programming is a concept independent of a message-passing control structure. Working with PASCAL, we are restricted to the use of subroutine calls (with strong typing). Nevertheless, message-passing together with concurrency, as found in Thoth [Cheriton 1982], should be noted as an interesting concept.

2.1 Abstract Data Types Or Multi-Sorted Algebras

Our notion of the object is influenced by the work done to date on formal specification methods using abstract data types. An abstract data type (ADT) or multi-sorted algebra [Goguen 1978] [Zilles 1980] is a mathematical structure which defines the behaviour of objects of certain types by describing a theory, stating what sorts (types) of objects are dealt with and what operations are applicable to them, and establishing a system of axioms which determine what the effects of the operations are. This is a formal method of fully defining the semantics of a sort and its operations. It remains the designers problem to assure that these sorts and operations are meaningful in real world terms.

2.2 Construction Of Types

2.2.1 Classification/Instantiation - Given an ensemble of objects, classification groups together all objects which respond to the same operations.

One level of classification is built into programming languages, as we use modules to describe the behaviour of classes of objects (types) and then create single instances of this behaviour.

2.2.2 Generalization/Specialization - Classes of objects which have some operations in common can be grouped into more general classes (sometimes called superclasses [Goldberg 1983]). From this, it follows that all subclasses of a superclass inherit all the operations of the superclass. We recognize only inheritance (i.e. all operations are inherited) and require a strict hierarchy between classes and superclasses. This is not only an implementation problem, but also a reflection of our as yet insufficient understanding of other inheritance schemes.

2.2.3 Aggregation/Part-of - Aggregation constructs a higher-level object from several objects (or values). We differentiate between aggregates with a fixed number of parts and those in which a varying number of parts of the same type is combined.

2.2.3.1 Aggregates With A Fixed Number Of Parts - With aggregation comes an operation into combine the parts to the aggregate (usually called 'make') and operations to get the individual parts from the aggregate.

2.2.3.2 Aggregates With A Varying Number Of Parts - This type of aggregation requires an operation to add a part to the aggregate, as well as a 'for each'-operator which permits the application of an operation to all parts [Backus 1978]

2.3 Object Semantics Vs. Value Semantics

All objects rely on the existence of an operation to create an object of their type, an assignment operation, and an operation to compare two objects for equality. Other operations may be available, but this seems to be the minimal set necessary.

Programming languages offer two different semantics for the create/assignment operation and the equality operations on variables [MacLennan 1982].

- value semantics: the 'assignment' operation creates a new object of the same type with the same value; two objects are equal if they have the same values;
- object semantics: each time it is used, the 'create' operation creates a new object, different from any object previously created; an 'identity' function tests whether two references point to the same object.

The difference between value and object semantics are easily detectable:

```
var a,b : sometype
...
begin
   create (a);
   ...
   b := a;
   change (a, x);    (* some changes which affect the values of a *)
   if a = b then   writeln ('object semantics')
           else    writeln ('value semantics');
end;
```

It may be noteworthy that ADA modules (packages) export assignment and equality [ADA 1983], but their semantics depend on the implementation of the package. It is therefore possible that a change in the implementation (value type to dynamic type) without a corresponding change of the interface may influence the noticeable behaviour of a package, and thus may cause errors in code using this module.

For CAD/CAM applicationa, a third form of semantics seems worth studying: version semantics. In version semantics, 'new version' creates something which is neither a new object nor a simple reference to the old object.

We will associate our notion of an 'object' with those things which use object semantics for assignment and equality. Similary, we will use the term 'value' to refer to those things which use value semantics for assignment and equality.

2.4 Persistence Of Objects

We will assume that objects are stored in a database and are therefore persistent, meaning that their lifespan does not depend

on the running of a program. While it is possible and sometimes useful to create things with object semantics which are not persistent, this topic will not be discussed here.

3 IMPLEMENTATION OF ABSTRACT DATA TYPES (MODULES)

A module implements an abstract data type, either of object or value type. Such a module exports the type definitions (including all used constants) and the procedures and functions which are defined for this ADT.

Every module exports,as a minimum, an operation to create or assign objects of this type and an operation to test for the equality or identity of two objects. The semantics of these operations are either 'value' or 'object' semantics, respectively.

For input/output operations, we include an operation to parse an object of this type from an input string and to convert an object to a string for output. We are presently studying better methods of dealing with I/O, especially the proposal made in [Shaw 1986]. Experience has shown that I/O operations are our major design problem, and as such we feel they are closely related to the discussion of object-oriented programming.

We exclude the treatment of errors and exceptional states, although theoretical solutions are known [Parnas 1982] [Goguen 1978].

4 PROGRAMMING ENVIRONMENT

A programming environment must support the software engineering method used, and should support only that one. When we built our system in 1983, not many languages supported modularization based on abstract data types. We decided to use PASCAL as a base language, because we were familiar with it and becaouse PASCAL compilers were available for most hardware.

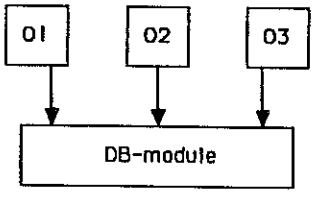
A precompiler [Frank 1983] provides an import/export facility for modules [Wirth 1983] (or a package concept [ADA 1983]). The same precompiler makes the changes necessary to produce PASCAL source code suitable for different compilers; in consequence, we have one set of source code which runs on such vastly different hardware as IBM 370 (under VM/CMS), DECsystem-10 (TOPS-10) and VAX (VMS).

Any module using an ADT from another module imports the lower module, which makes the complete type definition (including all type definitions of even lower modules) and all the operations on this type available at this higher level. We do not provide selective imports (as do ADA or MODULA-2), and under PASCAL rules the internals of the imported type are accessible to a malicious programmer. Our coding rules prohibit the use of any internal details of an imported ADT. We call exported access functions to get at the values in an ADT, in lieu of a record selector.

Unfortunately, the PASCAL host language does not allow coding of generic modules, so we have to produce these from a common template using a text editor. We would have been pleased if a more advanced language had been available, and are still considering a change. However, interesting new developments such as Clascal [Casseres 1983], CLU [Liskov 1981], etc. are not readly available, and ADA seems not to address the issues in a straightforward way.

5 THE CLASH BETWEEN DATABASE AND OBJECT-ORIENTATION

The object-oriented view considers the modules which define the objects to be upper-level using a common database (data storage) module to provide generic operations (figure 1a). This is somewhat analogous to using the built-in value types to construct other types (figure 1b). This analogy, however, is not correct, as the operations in the database need access to operations provided by the object modules, too. Further, the relations between the objects are not necessarily hierarchical: it will not suffice to provide operations in module B which result in objects of type A (and so on in a hierarchy) figure 1c, but we will need operations on objects from type A that yield objects of type B. This cannot be modelled in systems with a linear order (i.e. definitions appear before use), and conflicts not only with our way of writing specifications bu also with the requirements of our programming environment.



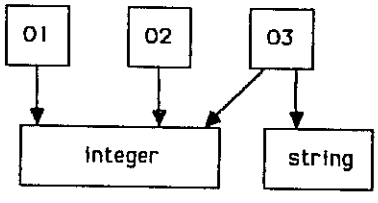


Figure 1b

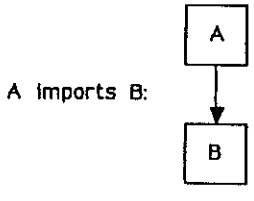


Figure 1a

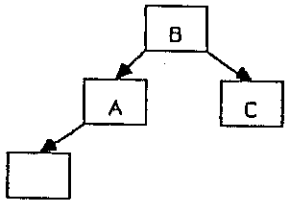


Figure 1c

Example: edge - node relation; given an edge, we need access to the two ending nodes. Likewise, given a node, we need access to all edges that are incident with it.

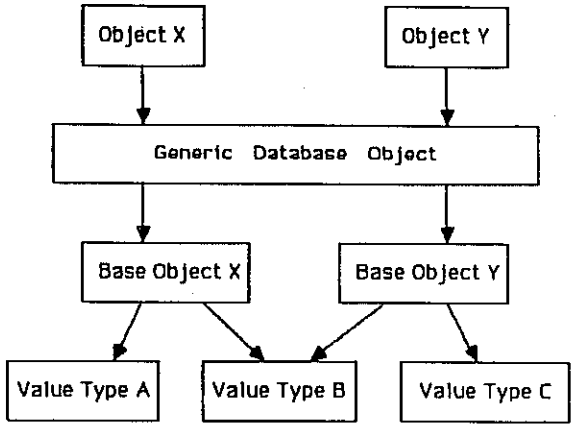


Figure 2

Our solution is shown in figure 2, where the division between basic object modules and upper-level object modules is clearly visible. This allows the DBMS to use the operations included in the basic object modules without resorting to circular import relations.

It is possible that once the problems are clearly understood and methods are available to define objects with non-hierarchical relations, a solution such as that shown in figure 1a will become possible. The following description may then be helpful in guiding the implementation of such a system.

6 DEFINITION OF OBJECT ATTRIBUTES

Objects provide access functions to their parts, a view proposed in functional database languages such as DAPLEX [Shipman 1981]. We separate these access functions into two groups, one yielding values (i.e. things with value semantics) and the other yielding objects (i.e. things with object semantics). The designer of the database schema has to decide what combination is most appropriate to model reality [Kent 1979]. Value and object semantics are distinct, and it is useful to force a designer to decide between the two.

The definition of an object consists of defining the value part (attributes) as a module ,and then indicating the object's relations to other objects.

An object-oriented database must support objects with value parts constructed from arbitrary data types (support for pointer types and dynamic data structures would require function to map from main memory representation to disk storage representation) and we describe such constructions before we introduce the data model proper. We want to stress that the definition of the base object data types is not part of the data model proper, as in our understanding the database must not 'see' any internal details of

the object data treated.

6.1 Built-In ADT

Every programming language provides some built-in data types and operations on them. We currently use PASCAL booleans, integers, reals, characters, and packed arrays of characters.

6.2 Quotient Algebras

Quotient algebras are based on restricted subsets of previously defined types (e.g. positive integers, angular measurement as a real modulus 2 pi).

6.3 Aggregates Of Value Modules

Object types are most often composed of a fixed number of value types. The construction in PASCAL is the record (or occasionally the array). In principle, aggregates of a variable number of values are possible, but these are easy to implement in PASCAL, and are therefore excluded from PANDA in its present form. A very general solution is the NF2 algebra [Schek 1984] which could be used to construct value modules.

Fixed-number aggregation of types a,b... into a new type t:
make : a x b ... -> t
a: t -> a
b: t -> a
equal: t x t -> boolean

axiom:
a (make (a1,b1,..)) = a1
b (make (a1,b1,..)) = b1
equal (t1, t2) = equal(a(t1),a(t2)) and equal (b(t1),b(t2)) and ...

6.4 Generalization Of Value Modules

In some cases, several different value types embody the same concept (semantics) in different forms, and therefore export the same operations (e.g. vectors in orthogonal (x-y) or polar (phi-r) coordinates, or straight lines and arcs of circles). It is possible to construct value types which make such differences invisible to the upper-level module.

The object values can be constructed from generalizations of value modules. It is up to the designer of the database schema to decide whether each of the specializations is an object type in its own right or whether only the generic object type is known to the database.

The implementation in PASCAL uses a record with variant part.

Generalization of two types p, q to form a new type t
p has operations p.a, p.b, etc.
q has operations q.a, q.c, etc.
make-from-p: p -> t
make-from-q: q -> t
type: t -> (p-type or q-type)

```
p : t -> p
q : t -> q

a (t) = if type(t)=p-type then p.a(t)
        if type(t)=q-type then q.a(t)
b (t) = if type(t)=p-type then p.b(t) else error
etc.
p (make-from-p(p)) = p
q (make-from-q(q)) = q
p (t) = if type(t)=p-type then p else error
q (t) = if type(t)=q-type then q else error
```

6.5 Special Operations For Objects To Support Database Operations

A number of specific operations is necessary to support special database operations (e.g. to test for the equality of two values, to compute an integer (as in hash computation), to convert to string for output etc). Modules are available for the most frequemtly used types such as integers and reals. For other special ADT's the database designer must construct the required operations.

7 DATA MODEL AND DATABASE OPERATIONS

By 'data model' we mean, as is customary in database studies, the tools available to the designer to structure his data.

The notion of a data model is less frequently used in programming language studies, its nearest equivalents being the constructors for data structures. Some similar concepts have been discussed in abstract data type literature [Lockeman 1979].

The explanation of a data model without the appropriate operations is not very helpful, and we want to escape the valid criticism in [Reiter 1984]. However, the formal definition of semantics for operations is rendered somewhat awkward due to the clash between the essentially functional description used for multisorted algebras (no side-effects) and the state-oriented view of a database [MacLennan 1982].

In an object-oriented design, the data model is somehow restricted, as it does not include the internal structure of objects (i.e. attributes with value semantics).

We require that the database schema be available to the database before data are stored. The following operations assume that object-types, access-path and association names are defined and return 'error' if an undefined value is encountered or if, for an association or access path, the objects are not of the correct type. The available metadata do not influence the semantics of association or access path, but permit us to select implementation methods which result in better performance (e.g. the establishment of auxiliary structures for access). At present we have not implemented operations to change the database schema once a database contains data. However, there is no reason why such operations could not be programmed.

7.1 Objects

The database deals with objects, which may contain some value types (aggregates of a fixed number of parts, generalizations of value types), but may not contain other objects (nor dynamic data structures using pointers).

The database provides special operations for these objects. Indeed, all objects are specializations of a generic object type. There is a special 'create' operation to make the generic database object from the base object values (cf. generalization) inherited by all object types, and a test for identity: modules that implement objects have object semantic. The assignment does not create a new object, but rather a pointer to the object; equality serves as a test for same-object. An equal-value function is often provided to test whether two objects have the same value.

```
create : value x object-type -> object
identical : object x object -> boolean
equal-value : object x object -> boolean
assign: object -> object
get-value : object -> value
get-type:  object -> object-type

axiom:
identical (o,o) = true
identical (create (v1, t1), create (v1, t1)) = false
identical (assign (o), o) = true
equal-value (create (v1),create (v1)) = true
equal-value (o1,o2) = if identical (o1,o2) then true
  -- but not: identical (o1, o2) = if equal-value (o1, o2) then true
get-value (create (v, t)) = v
get-type (create (v,t)) = t
```

7.2 Storage Of Objects

As a basic service, a database must provide an operation to store and retrieve a data element. As a low-level module in a database, we provide a storage and retrieval operation reminiscent somewhat of a random access file. In fact it can, but need not, be implemented using a random access file. A high-performance database will include buffer management and clustering of data which is frequently used together, but this is not part of the present conceptual discussion.

This low-level storage system requires a unique identifier for each object. This function may be implemented in conjunction with the storage module, but is logically separate.

```
id:    object -> id
axiom: object.identical (o1,o2) iff id (o1) = id (o2)
```

Objects are stored in an ADT 'db-storage', which is the explicit result of all storage modifying operations.

```
initial: . -> db-storage
store: object x db-storage -> db-storage
retrieve: db-storage x id -> object
```

```
modify: object x db-storage -> db-storage
delete: object x db-storage -> db-storage

axioms:
retrieve (store (o1,d1), id(o2)) =
             if o1 = o2 then o1
                        else retrieve (d1, id(o2))
modify (o, d) = if not retrieve (o,d) then error
  -- it is an error to modify an object which
  -- was not previously stored
retrieve (modify (o1,d1), id(o2)) = if o1 = o2 then o1
                        else retrieve (d1, id(o2))
retrieve (delete (o1,d1), id (o2) = if o1 = o2 then error
                        else retrieve (d1, id(o2))
```

7.3 Access Based On Unique Values

A database management system must also provide access to the objects based on values in them. The semantics of the method described requires that the values be unique: this is implemented using hashing techniques [Knuth 1973].

In order to avoid the database using information about the internal structure of an object (which would violate the principle of information hiding [Parnas 1978]), operations must be defined which calculate an integer value (which is then used to calculate the hash value for each object and access-path. In addition, a function must be provided which compares two object-values for equality with respect to the values used for an access path.

This access method works with single attribute values of any type, on a combination of attribute values, or on some value associated with the object in another way (sometimes called 'virtual attributes').

for each object type and access path the following must be defined:

```
to-integer: object-value x access-path -> integer
equal:      object-value x object-value x access-path -> boolean

axiom:
if equal (o1, o2, p1) then to-integer (o1,p1) = to-integer (o2, p1)
  -- however we do not demand that
  -- if to-integer (o1,p1) = to-integer (o2, p1)
  --                   then equal (o1,o2,p)
```

For the retrieval of an object, the database is provided with an object-value where the values to be used are filled. Since the database does not make any assumptions about the type used as a key for retrieval, the whole object type must be provided.

```
find: object-value x access-path x db-storage -> object

axiom:
find (o1, p, store (o2, d)) = if equal (get-value (o2), o1, p)
                    then o
                    else find (o1, p, d)
```

```
find (o1, p, modify (o2, d)) = if equal (get-value (o2), o1,p)
                    then o
                    else find (o1, p, d)
find (o1, p, delete (o2, d)) = if equal (get-value (o2), o1,p)
                    then error:not-found
                    else find (o1, p, d)
```

For storage and modification, we have to add:

```
store (o,d) = if find (get-value (o), p, d)
                        then error:not-unique-key
modify (o,d) = if find (get-value (o), p, d)
                        then error:not-unique-key
```

7.3.1 Associations (CODASYL: Set) - This construct associates a number of objects of one (or occasionally more than one type with one object of another type. An object can participate in several different types of associations, but only in one occurence of each at any given time. This construct models the typical 1 : n relation between objects. In the language of DAPLEX, an association is a single-valued function which returns an object with an inverse which is multi-valued [Shipman 1981].

We would like to be able to provide a distributive function as in FL [Backus 1978], but in a procedural, one-object-at-a-time-oriented language, a generic 'FOR EACH member-object IN association-type FROM owner-object' would be sufficient. Without extension of PASCAL we have to do with:

```
    initialize (ownerObject, memberObject);
    while nextobject (memberObject, association) do
        begin
(* operations on the single objects in the association *)
          action (memberObject);
          ...
        end;
```

```
operation:
insert: object x object x association x database -> database
all-members: object x association x database -> set-of-objects
  -- this is similar to the multi-valued functions in DAPLEX
find-owner:  object x association x database -> object
  -- this is the single-valued function in DAPLEX

axioms:
  -- for the formulation we make use of mathematical set operations:
contains (all-members (owner, a, insert (owner, member, a,d)),
                member) = true
   -- an object wich is inserted is found in the association
intersection (all-members (owner1, a, db),
                all-members (owner2, a, db))= empty
   -- no object can take part in two associations of the same type
   -- an insertion that would violate this condition raises
   -- an error condition
find-owner (member, a, insert (owner, member, a, d)) = owner
```

There is an additional operation to find the object with some given values in an association. The method of interaction between object module and database is essentially the same as that used for access based on unique values.

The association can be used to model two semantically different situations: aggregation and classification. It is thus a lower-level construct. We are in the process of studying the semantics of higher-level operations which might be more appropriate as an interface.

7.3.1.1 Aggregation - Aggregations of eparts which are objects can be accomplished using the association. The 1 : n restriction is necessary for a proper aggregations of parts - it seems correct that an object can only be part of one (aggregate) object in a given context. An object may be composed of parts in different ways - e.g. a department consists of personnel, offices, projects etc.: conversely, objects may participate in different ways in aggregations - eg. a person is part of an office, of a family etc.

We also use aggregations for values which appear an indefinite number of times for an object. Under these circumstances, these values become objects. It may be appropriate to allow fields of varying length for such cases (preferably in the generality as discussed as NF2 relations [Schek 1984]), as the meaning of this structure is different, and a value should not be converted into an object simply because limitations in the data model demands it.

7.3.1.2 Classification - Associations can also be used to model a classification/instantiation relation. (e.g. a type of bolt - an individual bolt used in an assembly). Again the 1 : n restriction seems appropriate, as an object can only be of one type (in a given context).

7.3.1.3 Inheritance - In both cases, aggregation and classification, some 'inheritance' of properties occurs. The details of this inheritance however, are different. In classification, the instantiations inherit most properties from the class (e.g. the bolt inherits the properties of dimension, material, price, etc.). In aggregation, the parts have conferred upon them some properties inherent in the top assembly but the resultant aggregate also has some properties which are directly determined by its parts (e.g. the total weight of the assembly is the sum total of the weights of its parts).

We are currently studying such situations in order to detect more appropriate, more meaningful constructs, for which we can formally state the semantics of this distribution of attribute values among related objects. As our present model does not include any inheritance, there is no need for a differentiation between aggregation and classification. The object programmer has to formalize his understanding of the specific situation individually for each object type.

7.4 Object Modules

For each object, meaningful operations are initially designed and then coded in a top-level object module. This module exports all meaningful operations on objects of this type, e.g. finding a specific object given some value, storing a new object with some given values, etc. These modules import the operations the database offers and use them to implement their own operations.

Consistency constraints are included in the storage and modifying operations in these modules (except those constraints which are part of the data model, such as unique keys for access paths). Consistency constraints can thus be written using the full power of our programming language - which is generally needed for non-standard applications.

Consistency constraints in this model are not necessarily expressed as predicate over database values (e.g. constraints in DAPLEX [Shipman 1981]), but the set of all consistent states is defined by an enumeration of all possible states reachable through a finite chain of operations exported by the object modules.

The application-level programmer sees only the top level of operations applying to the objects. She is not concerned with the database operations at all. It has been the purpose of this paper to show how the subordinate modules are constructed.

While objects appear to the application-level programmer to be of individual types, they are specializations of a generic object type. Operations are then implemented using the generic operations on objects for creation and deletion, for association between objects and for access methods. The operations that deal with specific values are handled using the basic object modules.

8 EXTENSIONS

8.1 Simplification For Object Definitions

The operations necessary for the database can be provided in the modules which define the value types (e.g. a module for integer or string). The code in the object modules can then be produced automatically from a record description and the declaration of access path and Associations. Unfortunately, this induces a view which is different from that of an object-oriented design.

8.2 Access By Nearly Unique Keys

We often have attribute values which are natural candidates for access keys even though their value is not guaranteed to be unique, e.g. names of persons or towns. Searching on such a path delivers

- the matching object if there is only one
- the matching objects if there are more than one. the user has to select the desired one in a dialog. this requires output operations for objects accessible from the database.

8.3 Spatial Access

Many data in engineering applications (e.g. land information system) are space-related, making access based on location necessary [Frank 1984]. We have included special internal support, invisible to the programmer, which allows the retrieval of all objects which fall within a specified rectangle [Frank 1983a]

9 EXPERIENCES AND OUTLOOK

9.1 Implementation

The implementation of the basic database operations follows traditional lines. Storage space is divided into pages, and each record has a tuple identifier similar to SYSTEM R [Astrahan 1976]. Input and output of pages is buffered, and there is an additional level of buffering for records: both are managed using a least-recently-used strategy. Associations are stored as linked lists. Access paths for unique keys use a hash algorithm.

Our experience shows that such a simple implementation can be presented with an easily understood, clean interface. It seems a natural candidate for the implementation of an object-oriented database. Further, these simple methods provide fast response times for the complex retrievals of engineering databases.

9.2 Implemented Projects

We have implemented a number of databases using this methodology; a simple database can be up and running in a few days. The PANDA database was used to store the facts and rules for our PROLOG interpreter [Frank 1984a]. We did not encounter problems during schema design, and the performance is acceptable - in our non-optimized prototype, access to an object takes no more than several milliseconds.

The construction of a new database schema is a non-trivial task. The data model presented here has more semantic precision than does, for example, a purely relational one, and provides better guidelines. The coding of all object modules required for a project is a substantial effort, but similar code is necessary for any database. The guidelines for the construction of object modules, and the similarity between objects, do assist the programmer. Subsequent higher levels of the application program do not include direct references to the database, and are thus easier to code and truly independent of the database.

9.3 Input And Output Operations

The construction of higher-level tools to use the data in a database (e.g. query language, browser, form based input) depends on routines for the input or output of database objects. It is not sufficient to provide a single output routine, as the presentation of an object must vary with context (sometimes, in fact a graphical representation is desired).

Writing the code for these input and output operations is exteemely tedious, and we are investigating more general methods, especially along the lines described in [Shaw 1986].

10 CONCLUSIONS

Commercial (standard) databases are too restricted for complex, non-standard applications. The approach presented here is object-oriented in the sense that:

- operations on objects have an object semantic (as opposed to a value semantic) for assignment and equality
- objects can have an arbitrary internal structure (attribute values) which is not interpreted by the database
- objects can be related to each other (1:n association)
- objects can be accessed using values which uniquely identify them (without limitationson certain data types)

Our experience with the construction of databases for Computer Aided Engineering applications leads us to beleive that

- the object paradigm is powerful and appropriate and the differentiation between values and objects important for a data model. Efforts to introduce surrogates into the relation data model seem to be a step in the same direction [Plouffe 1984] [Meier 1983].
- the concept of consistency defined by all states reachable with a finite sequence of operations on objects meshes well with an object-oriented approach. It permits arbitrarily complex consistency constraints and makes the full power of the programming language available for their encoding.
- applications in this area have structures which are so complex that schema design and coding are major efforts. It is more important to have a system which allows proper modelling than a system which is simple to use.

We are presently experimenting with query language, and again we found that conventional languages (e.g. SQL) are not powerful enough (lacking recursion). We are therefore investigating a PROLOG-like language, which blends well with the presented data model, although it is not yet clear how PROLOG may best be used in an object-oriented style [Frank 1984b].

11 ACKNOWLEDGEMENTS

The work reported here evolved over a number of years and many a friend and colleague of mine helped me better towards a better understanding of the issues involved. Special thanks go to Andreas Gautschi, Prof. R. Conzett, and Beat Sievers (then all at the Swiss Federal Institute of Technology at Zurich, ETH) and to Larry Latour, Max Egenhofer and Doug Hudson (from the University of Maine). Linda Langley helped with the preparation of this paper.

12 REFERENCES

ADA Reference Manual for the ADA Programming Language, Springer-Verlag, New York, 1983

Astrahan, M.M. et al., System R: Relational Approach to Database Management, ACM TODS, Vol. 1, No. 1,1976

Backus, J., Can Programming be Liberated from the Von Neumann Style? A Functional Style and its Algebra of Programs, CACM, Vol. 21, No. 8, Aug. 1978,

Battory, D.S, Kim, W., Modelling Concepts for VLSI CAD Objects, ACM Transactions on Database Systems, Vol. 10, No. 3, Sept. 1985

Casseres, D., CLASCAL Reference Manual for the LISA, first draft, Apple, 1983

Cheriton, David R., The Thoth System: Multi-Process Structuring And Portability, North-Holland, New York, 1982

Frank, A., PANDA Pascal network database system, Proceedings ACM SIGSMALL, Colorado Springs CO, 1982

Frank A., A Precompiler for Transportable Modular PASCAL, internal documentation, University of Maine at Orono, Department of Civil Engineering, Surveying Engineering, Orono ME, 1983

Frank, A., Problems of Realizing LIS: Storage Methods for Space Related Data: The Field Tree, No. 71, Swiss Federal Institute of Technology, Zurich (Switzerland), 1983

Frank, A., Requirements for database systems suitable to manage large spatial databases, in: D. Marble, et al. (Eds), Proceedings of the International Symposium on Spatial Data Handling, Zurich (Switzerland), 1984

Frank, A., Extending a Database with Prolog, in: Kerschberg, L. (ed.), Proceedings of the First International Workshop on Expert Database Systems, Kiawah Island (SC), Oct. 1984

Frank, A., Combining a Network Database with Logic Programming, No. 45, University of Maine at Orono, Department of Civil Engineering, Surveying Engineering, Orono ME, 1984

Goguen, J.A., et al., An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types, in: Yeh, R. (Ed.) Current Trends in Programming Methodology, Prentice-Hall, Englewood Clifts, N.J., 1978

Goldberg, A., Robson, D., Smalltalk-80, Addison-Wesley, 1983

Kent, W., Data and Reality, North-Holland, New York, 1979

Knuth, D.E., The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading MA, 1973

Liskov, B., et al., Lecture NOtes in Computer Science, CLU Reference Manual, Springer-Verlag, New York, 1981

Lockemann, P.C., et al., Data Abstraction for Database Systems, ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979

MacLennan, B.J., Values and objects in programming languages, SIGPLAN Notices, Vol. 17, No. 12, Dec. 1982

Meier, A., Lorie, R., A surrogate concept for engineering databases, in: Schkolnick, M., Thanos, c., (Eds.), Proceedings Ninth International Conference on Very Large Databases, Florence (Italy), 1983

Parnas, D.L., Share, J.E., Language facilities for supporting the use of data abstraction in the development of software systems, Naval Reasearch Laboratory, Washington, 1978

Plouffe, W., et al., A Database System for Engineering Design, IEEE Database Engineering, Vol. 7, No. 2, June 1984

Reiter, R. Towards a logical reconstruction of relational database theory, in: Brodie, M.L., et al. (Eds), On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer

Verlag, New York, 1984

Schek, H.-J., Nested Transactions in a Combined IRS-DBMS Architechture, Proceedings of the 3rd Joint BCS and ACM Symposium on Research and Development in Information Retrieval, Cambridge University Press, 1984

Shaw, M., An Input-Output Model for Interactive Systems, Proceedings CHI'86, Human Factors in Computing Systems, Boston, 1986

Shipman, D.W., The Functional Data Model and the Data Language DAPLEX, ACM Transactions on Database Systems, Vol. 6, No. 1, March 1981

Wirth, N., Programming in MODULA-2, Springer-Verlag, New York, 1983

Zilles, S. N., An introduction to data algebras, in: Bjorerner, D., (Ed.), Abstract Software Specifiactions, Lecture Notes in Computer Science, Vol. 86, Springer Verlag, New York, 1980

PUBLICATIONS
Surveying Engineering Program
University of Maine

The following is a current list of publications by the Surveying Engineering Program at the University of Maine. Copies, unless specifically designated as No Longer in Print (NLP) or available from some other publisher or agency, are available from University of Maine, Surveying Engineering Program, 120 Boardman Hall, Orono, ME 04469. (A fee to cover the costs of processing, printing and mailing is indicated after each available publication.)

7. Land Information Systems for the Twenty-First Century, E. Epstein and W. Chatterton, Real Property, Probate and Trust Journal, American Bar Association, Vol. 15, No. 4, 890-900 (1980). ($5.)

9. Legal Studies for Students of Surveying Engineering, E. Epstein and J. McLaughlin, Proceedings 41st Annual Meeting, American Congress on Surveying and Mapping, Feb. 22-27, 1981, Washington, D.C. ($5.)

15. Storage Methods for Space Related Data: The FIELD TREE, A. Frank in: Spatial Algorithms for Processing Land Data with a Minicomputer, MacDonald Barr (Ed.). Lincoln Institute of Land Policy, 1983. ($15.)

17. Semantische, topologische und raumliche Datenstrukturen in Land-informations-systemen (Semantic, topological and spatial data structures in Land Information Systems) A. Frank and B. Studenman, FIG XVII Congress Sofia, June 1983. Paper 301.1. ($5.)

18. Adjustment Computations, A. Leick, 250 pages. ($25.)

19. Geometric Geodesy, 3D-Geodesy, Conformal Mapping, A. Leick, 420 pages. ($30.)

24. Macrometer Satellite Surveying, A. Leick, ASCE Journal of Surveying Engineering, August, 1984. ($5.)

27. Adjustments with Examples, A. Leick, 450 pages. ($30)

28. Geodetic Programs Library, A. Leick, about 400 pages. ($25.)

30. Macintosh: Rethinking Computer Education for Engineering Students, A. Frank, August, 1984. ($5.)

31. Surveying Engineering at the University of Maine (Towards a Center of Excellence), D. Tyler and E. Epstein, Proceedings MOLDS Session, ACSM Annual Meeting, Washington, March, 1984. ($5.)

32. Innovations in Land Data Systems, D. Tyler, Proceedings Association of State Flood Plain Managers, Annual Meeting, Portland, Maine, June 1984. ($5.)

33. Crustal Warping in Coastal Maine, D. Tyler, et. al., Geology, Vol 12, pp. 677-680, November 1984. ($5.)

34. St. Croix Region Crustal Strain Study, D. Tyler and A. Leick, Technical Report submitted to the Maine Geological Survey, June 1984. ($10.)

35. Applications of DBMS to Land Information Systems, A. Frank, in: Proceedings, Seventh International Conference on Very Large Databases,C. Zaniolo and C. Delobel (Eds.), Cannes (France), September, 1981. ($5.)

36. MAPQUERY: Database Query Language for Retrieval of Geometric Data and Their Graphical Representation, A. Frank, Computer Graphics, Vol. 16, No. 3, July 1982, p. 199 (Proceedings of SIGGRAPH '82, Boston). ($5.)

37. PANDA: A Pascal Network Data Base Management System, A. Frank, in: Proceedings of the Fifth Symposium on Small Systems,G.W. Gorsline (Ed.). (ACM SIGSMALL), Colorado Springs (Colorado), August 1982. ($5.)

38. Conceptual Framework for Land Information Systems - A First Approach, A. Frank, paper presented to the 1982 Meeting of Commission 3 of the FIG in Rome (Italy) in March 1982. ($5.)

39. Requirements for Database Systems Suitable to Manage Large Spatial Databases, A. Frank, in: Duane F. Marble, et. al., Proceedings of the International Symposium of Spatial Data Handling, August, 1984, Zurich, Switzerland. ($10.)

40. Extending a Network Database with Prolog, A. Frank in: First International Workshop on Expert Databases Systems, October, 1984, Kiawah Island, South Carolina. ($5.)

42. Canonical Geometric Representations, A. Frank. ($10.)

43. Computer Assisted Cartography - Graphics or Geometry, A. Frank, Journal of Surveying Engineering, American Society of Civil Engineers, Vol. 110, No. 2, August 1984, pp 159-168. ($5.)

44. Datenstrukturen von Messdaten, A. Frank and B. Studemann, paper presented at IX International Course for Engineering Surveying (Graz, Austria), September 1984. ($5.)

45. Combining a Network Database with Logic Programming, A. Frank. ($10.)

46. Montgomery County (Pennsylvania) GPS Survey, A. Leick and J. Collins, ASP/ACSM Annual Meeting, Washington, D.C., March 10-15, 1985. ($5.)

48. Application of GPS in a High Precision Engineering Survey Network, R. Ruland and A. Leick, First International Symposium on Precise Positioning with the Global Positioning System, Rockville, Maryland, April 15-19, 1985. ($5.)

49. Graphics Programming in Prolog, R. Michael White and Andrew U. Frank. ($5.)

50. Instrumentation Needs (GPS and Related Matters), Alfred Leick. Workshop on Fundamental Research Needs in Surveying, Mapping, and Land Information Systems, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, November 18-20, 1985. ($5.)

52. Expert Systems Applied to Problems in Geographic Information Systems, Vincent Robinson, Andrew U. Frank, Matthew Blaze, presented at the ASCE Specialty Conference on Integrated Geographic Information Systems: A Focal Point for Engineering Activities, February 3-5, 1986. ($10.)

53. Integrating Mechanisms for Storage and Retrieval of Data, Research Needs, Andrew U. Frank, challenge paper forWorkshop on Fundamental Research Needs in Surveying, Mapping, and Land Information Systems, November 17-19, 1985, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061. ($10.)

54. Formal Methods for Accurate Definitions of Some Fundamental Terms in Physical Geography; Andrew U. Frank, Bruce Palmer, Vincent B. Robinson. Invited paper at the Second International Symposium on Spatial Data Handling, July 5-10, 1986, at Seattle, WA. ($5.)

55. Cell Graphs: A Provable Correct Method for the Storage of Geometry; Andrew U. Frank and Werner Kuhn. Invited paper at the Second International Symposium on Spatial Data Handling, July 5-10, 1986 at Seattle, WA. ($10.)

56. LOBSTER: Combining Database Management and Artificial Intelligence Techniques to Manage Land Information; Andrew U. Frank. Invited paper No. 301.1 for the XVIII. International Congress of FIG, Toronto, Ontario, Canada, 1986. Commission 3, Land Information Systems. ($5.)

57. PANDA: An Object-Oriented PASCAL Network Database Management System; Andrew U. Frank. ($5.)

58. GPS Network Adjustment with Apriori Information and Orbital Determination Capabilities; Kamil Eren and Alfred Leick. Proceeding of the Fourth International Geodetic Symposium on Satellite Positioning, Austin, Texas, April 28-May 2, 1986. ($5.)

59. MAINEPAC (1): Processing GPS Carrier Phase Observations; Alfred Leick, University of Stuttgart, Federal Republic of Germany. ($10.)

60. Mathematical Models within Geodetic Frame; Alfred Leick, Journal of Surveying Engineering, Vol. 111, NO. 2, August, 1985, (paper No. 19952). ($5.)

61. Deformation Measurements Workshop, Guenter Wittman. ($5.)

63. Three Dimensional Conceptual Modeling of Subsurface Structures, Eric Carlson; Appropriate Conceptual Database schema Designs for Two Dimensional Spatial Structures, Max Egenhofer; both papers were presented at the Eight American Congress of Surveying and Mapping, March 29-April 4, 1987 in Baltimore, Maryland. ($10.)

64. Query Languages for Spatial Relations, David Pullar; Knowledge Representation Using First Order Predicate Calculas, Doug Hudson; both papers where presented at the Eight American Congress of Surveying and Mapping, March 29-April 4, 1987 in Baltimore, Maryland. ($10.)

65. Processing GPS Carrier Phase Observations for Station and/or Orbital Adjustments; Alfred Leick, presented at the Eight American Congress of Surveying and Mapping, March 29-April 4, 1987 in Baltimore, Maryland. ($5.)

66. Survey Checklist: A Practical Guide to Review and Analysis of Land Surveys, J. Richard White and Harlan J. Onsrud. ($15.)

67. PANDA: An Object-Oriented Database Based on User-Defined Abstract Data Types, Max J. Egenhofer and Andrew U. Frank. ($5.)

68. Combining a Least Squares Adjustment with a Direct-manipulation Human Interface, R. Michael White and Werner Kuhn; presented at the Annual Conference of the American Congress on Surveying and Mapping, Baltimore, Maryland on May 30, 1987. ($10.)

69. Centimeter-Level GPS Surveys in Seconds: Formulation and Analysis, Cheryl A. Quirion, presented at the Eighth American Congress on Surveying and Mapping, March 29-April 4, 1987 in Baltimore, Maryland. ($5.)

70. The Global Positioning System in Kinematic Mode: Formulation, Analysis, and Use; Cheryl A. Quirion. ($15.)

71. Monitoring Vertical Crustal Deformation in Eastern Maine Using GPS Derived Orthometric Heights; Steven R. Lambert. ($15.)

72. Maine Pac II, Geoid Undulation Determination Program, Program Documentation; Steven R. Lambert. ($20.)

73. Maine Pac III, Geoid Undulation Determination Program, User's Manual; Steven R. Lambert. ($10.)

74. Research Topics in GIS. Three papers presented at the IGIS Conference at Crystal City, November 1987. Object-Oriented Databases: Database Requirements for GIS; Andrew Frank and Max J. Egenhofer. Towards a Spatial Theory; Andrew Frank. Artificial Intelligence Tools for GIS; Andrew Frank, Douglas L. Hudson and Vincent B. Robinson. ($15.)

75. GIS Point Referencing by Satellite and Gravity; Alfred Leick. 1987. ($5.)

76. Positioning 2001; Alfred Leick, Surveying and Mapping, Vol. 47, No. 3, pp. 181-189 (Sept. 1987). ($5.)

77. First Steps in Modernizing Local Land Records, Harlan J. Onsrud, Surveying and Mapping, Vol. 45, No. 4, pp. 305-311 (December 1985);
Research for Validating Cadastral Data, Harlan J. Onsrud, Proceedings of the IGIS Conference, Crystal City, Washington, D.C. (November 1987); Technical Standards for Boundary Surveys: Developing a Model Law, Harlan J. Onsrud, Journal of Surveying Engineering, Vol. 113, No. 2, pp. 101-115.

78. The Education of Surveyors and Cartographers in an Information Age, Harlan J. Onsrud, Technical Papers of the XII Surveying Teachers Conference, Madison, Wisconsin (July 1987); Challenge to the Profession: A Formal Legal Education for Surveyors, Surveying and Mapping, Vol. 47, No. 1, pp. 31-36.

79. Data Structures to Organize Spatial Subdivisions, Ian Greasley (Graduate Student in Surveying Engineering) 1987. Paper presented at the ACSM-ASPRS 1988 convention at St. Louis, MO. ($5.)

80. The MainePac Series, Alfred Leick. Paper presented at ACSM-ASPRS Convention, St. Louis, Missouri, 1988. ($5.)

81. Maine Crustal Project: Final Report. Steven R. Lambert and David A. Tyler. Final Report to the Maine Geological Survey on Grant No. NRC-G-04-82-009. 1987. ($10.)

82. Designing a User Interface for a Spatial Information System. Max J. Egenhofer. Paper presented at the ACSM-ASPRS 1988 convention at St. Louis, MO. Partially funded by grants from NSF No. IST-8609123 and Digital Equipment Corporation, principal investigator Andrew U. Frank.
($5.)

83. Graphical Representation of Spatial Data: An Object-Oriented View. Max J. Egenhofer. Paper presented at the third International Conference on Engineering Graphics and Descriptive Geometry at Vienna (Austria). Project was partially funded by grants from NSF No. IST-8609123 and Digital Equipment Corporation, principal investigator Andrew U. Frank. ($5.)

84. A Precompiler For Modular, Transportable Pascal. Max J. Egenhofer and Andrew U. Frank. This paper has been submitted to SIGPLAN Notices. Project was partially funded by grants from NSF No. IST-8609123 and Digital Equipment Corporation. ($5.)

85. Integrating a Database Management System into an Object-Oriented Code Management System. Max J. Egenhofer and Andrew U. Frank. Project was partially funded by grants from NSF No. IST-8609123 and Digital Equipment Corporation. ($5.)

86. Requirements for Database Management Systems for Large Spatial Databases. Andrew U. Frank. Paper presented at the COGEDATA meeting in Dinkelsbuehl, F.R. Germany, December 2, 1986, program for digital mapping in geo-science.
($5)

87. Integrated Processing of GPS and Gravity Data. Günter W. Hein, Alfred Leick, and Steven Lambert. Paper presented at the GPS '88 conference on Engineering Applications of GPS Satellite Surveying Technology, May 11-14, 1988, Nashville, TN. ($5)

88. Multisensor Image Fusion Techniques in Remote Sensing. Manfred Ehlers. (May 1988).($5)

89. Maine Laws Pertaining to the Practice of Land Surveying. Compiled and Edited by: Harlan J. Onsrud and Kathry Fessenden. Based on laws in affect in 1987. (June 1988). ($12)

90. Multiple Inheritance and Genericity for the Integration of a Database Management System in an Object Oriented Approach. Andrew U. Frank. Paper was submitted to the Second International Symposium on Object Oriented Data Bases, Bad Stein am Ebernberg, Germany, September 27-29, 1988.($5)

91. Towards a Spatial Query Language: User Interface Considerations. Max J. Egenhofer and Andrew U. Frank. Paper presented at the Fourteenth International Conference on Very Large Data Bases at Long Beach, CA, August 29 - September 1, 1988. ($5)

92. Designing Object-Oriented Query Languages for GIS: Human Interface Aspects. Max J. Egenhofer and Andrew U. Frank. Paper presented at the Third International Symposium on Spatial Data Handling at Sydney, Australia, August 17-19, 1988. ($5)

93. Toward Formal Definitions of Topological Relations Among Spatial Objects. David V. Pullar and Max J. Egenhofer. Paper presented at the Third International Symposium on Spatial Data Handling at Sydney, Australia, August 17-19, 1988. ($5)