Meta Modeling for End User Computing

J.H. ter Bekke
Department of Information Systems
Delft University of Technology
P.O. Box 356, 2600 AJ Delft, The Netherlands
E-mail: j.h.terbekke@is.twi.tudelft.nl

Abstract

Inherent constraints play an important role in the semantic Xplain model. These constraints enable us to represent certain real world knowledge once for subsequent use in several applications. The Xplain DBMS has demonstrated that a broad class of applications can be developed by only defining structural properties of the data required by the applications. This results in a concentration of all software in the DBMS and makes often specific application software superfluous. Advantages of this semantic approach are amongst others: reliable data processing, simple and rapid application development and a significant reduction of maintenance costs. It makes databases also attainable for a broad, new class of end users.

Keywords: semantics, semantic data modeling, meta modeling, design tools, automatic application generation, end user computing, declarative knowledge.

1: Introduction

Inherent constraints are determined by the structural properties of a data model [7]. They have always played a role in databases. For example, the first database systems supported hierarchical structures (including inherent parent-child relationships) or network structures (including inherent owner-member relationships). However, implementations of these constraints resulted in complex navigational application programs. This was caused by the missing data independence. Inherent constraints were considered therefore as a disadvantage instead of an advantage of a data model.

The disadvantages of the implementations of inherent constraints in hierarchical and network databases resulted in data models in which inherent constraints played only a subordinate role. The original relational model for example, contains only one inherent constraint: the relationship between the attributes of a relation. However, the absence of other inherent constraints led to multiple specification and enforcement of semantic relationships through software. This is demonstrated in relational languages (as SQL) by the use of triggers and join-operations; these operations contain the software specification of the required connection.

This lack of inherent constraints in the relational model has resulted in new models containing more semantics. Object oriented approaches are examples. However, gradually it becomes more and more clear that, by modeling complex values, disadvantages of the original hierarchical and network structures are introduced again [8].

All foregoing approaches make implicit or explicit the assumption that applications require software to be developed (for example in 3GL, 4GL or SQL). A consequence of this assumption is that similar software is developed several times (namely in each application). It results in a resource devouring application maintenance. This assumption is not made in the Xplain approach. Starting point here is the development of a rich structure, understood and maintained by the DBMS. This semantic structure enables the DBMS to make existing software available for a broad class of applications. This is a confirmation of the advantages of a declarative knowledge representation approach [3]. Specific application software becomes out of date. Advantages of the approach are numerous:

- Applications use a predefined, consistent semantic data structure, understood and maintained by the DBMS. Applications can be developed very rapidly without programming efforts, if desired even dynamically at run time.
- Application correctness is determined during the development phase of an application with the help of the contents of the semantic data dictionary. Debugging of application software is not necessary.
- Meaning of an application is known to the DBMS.

This meaning is determined by the software for interpretation of the semantic data dictionary.

- Application development and maintenance are easier to perform (even at run time). The DBMS is continuously guarding application consistency.
- Changes to the semantic data structure are easier to make. Consequences for applications are determined by the DBMS.
- Application software is already available in the DBMS. Software is therefore developed only once. Development and maintenance costs are therefore reduced significantly.
- End users are not forced to follow the logic of an application program. End users may self determine their sequence of actions (with support of the DBMS). Applications therefore better agree with the intuition of the end user.
- Applications are not used for one single mode (search, modify, etc.); although the user's manipulation rights can be subjected to authorization rules. The collection of applications is significantly reduced and therefore better manageable.

This paper presents an overview of concepts required for meta modeling to reach this goal. Subsequently, some examples are given to illustrate applications which can be realized using the Xplain DBMS without the development of specific application software. At Delft University of Technology we have gained experience with this approach in various application domains. This experience has led to the conviction that this approach makes DBMSs suitable for a large application domain for a broad, new class of end users because it does not require knowledge of database languages.

2: Meta modeling in Xplain

Essential for a data model are the supported abstractions. The semantic data model Xplain is based on three abstractions: classification, aggregation and generalization. They are based on the fundamental *type-attribute relationship*. These abstractions are introduced informally in the following. The examples have been selected to illustrate their use for the purpose of this paper: meta modeling i.e. modeling of data dictionaries. An extensive treatment of these abstractions is given in [7]. The concepts may be considered as a further evolution of abstractions introduced by Smith and Smith [5, 6].

Classification

The abstraction leading to a property is called classification. The examples required for the recognition of a property (i.e. the instances) are purely meant as an illustration. The property is *not* being defined hereby. Properties are represented by rectangles in diagrams. The counterpart of classification is called *instantiation*. Fundamental to the semantic meta model is the notion of 'type' (see figure 1). Instances of this meta concept could be 'member', 'name ', 'copy' etc. occurring in a conceptual database model.

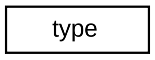


Figure 1: Classification

Aggregation

Aggregation is defined as the collection of a certain number of properties in a type, which in itself can be regarded as a new property (note the analogy with the mathematical set concept). A property from an aggregation is called an attribute of the type.

Aggregation allows view independence: we can discuss the obtained type (possibly as a property) without referring to the underlying attributes; it makes meta models extensible. By applying this principle repeatedly, a hierarchy of properties can be set up. An example for meta modeling is given in figure 2. Normally the hierarchy contains only aggregated types.

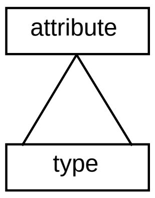


Figure 2: Aggregation hierarchy

Aggregation is indicated by a line connecting (the centers of) two facing rectangle sides, while the aggregate type is placed above its attributes. Of course, aggregation also has its counterpart: the description of a type as a set of certain attributes is called *decomposition*. An example of aggregation in a conceptual database model could be the following:

type member = name, address, zipcode, city.

In this definition, 'city' is considered as an attribute of 'member'; it represents a relationship between the 'member' type and the 'city' type. The first type is composite, the latter can be composite or base.

So, in the semantic meta model, attribute is considered as a relationship (a so-called membership relationship) between two types: one type is by definition composite (the type of which it is the attribute), the other related type can be composite or base. Besides that, attributes are of a certain kind; they specify a generalization or an aggregation abstraction. So we can consider the following type definitions in the meta model:

type attribute = composite_type, type, kind.
type type = name, representation.

These meta model definitions contain the relevant inherent constraints, for example: 'attribute *its* composite_type' defines a relationship between the 'attribute' and the underlying 'composite_type'. The prefix 'composite' in the attribute 'composite_type' is called a role.

Aggregation can be described using the verb *to have*. According to the above type definition, an attribute *has* a composite type, a type and a kind. Identifications are properties denoted by type names. This interpretation implies singular identifications; it precludes connection traps.

Generalization

The third type of abstraction, important to conceptual database models, is generalization; it is defined here as the recognition of similar attributes from various types and combining these in a new type (note the analogy with the intersection operation from mathematical set theory). We can equally discuss the new type without mentioning underlying attributes, and it can in itself again serve as a property (i.e. allows view independence). Generalization's counterpart (i.e. the union of attributes from different types) is called *specialization*.

An example of generalization occurs in meta modeling. For example: consider the properties 'composite_type', 'type' and 'kind'. The corresponding type is 'attribute' as defined earlier. Consider, in addition, 'composite_type', 'type', 'kind' and 'role', where the corresponding type might be 'role attribute'. The common attributes of the two types are 'composite_type', 'type' and 'kind'. This is the type 'attribute', which may be regarded as the generalization of 'attribute' and 'role attribute'. The union of attributes is found in the type 'role attribute'. This is considered as the specialization of the two. Generalizations can be represented in abstraction hierarchies, as we have seen in the case of aggregation. The above generalization is shown in figure 3.

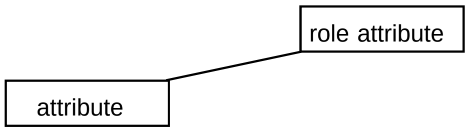


Figure 3: Generalization hierarchy

In abstraction hierarchies, generalizations are schematically represented by a line connecting facing corners of rectangles, the generalized types being placed below the specialized ones. The generalization, together with the attributes to be added to it, is described in the definition of the specialization. So the corresponding meta model definitions are:

type attribute = composite_type, type, kind.
type role attribute = [attribute], role.

Specializations are commonly associated with the verb *to be*. According to the above type definitions, a role attribute *is* an attribute with a role. The introduction of new identifications for specializations makes generalization hierarchies nontransitive [4] and extensible [7].

Inherent constraints

The following two inherent integrity constraints are recognized for the types in a conceptual database model (they are also relevant for the types in the meta model):

- *relatability*
  Each property in a type definition is related to one and only one equally named type, while every type may correspond to various attributes.
- *convertibility*
  Each type definition is unique: there are no type definitions carrying the same name or the same set of attributes.

It is important to realize that these two integrity constraints require neither additional specification nor declaration by procedures - they are inherent to the given type definitions proper. The DBMS contains software for enforcement and the DBMS is completely responsible for the integrity of the contents of the data dictionary and the corresponding database. User operations may ignore the aspect.

The two inherent constraints do not imply physical connections as known from implementations of hierarchical and network models. Types can be viewed as simple interrelated tables; the relationships being maintained by the DBMS itself and user operations have no influence on the integrity of these tables.

3: The contents of a semantic data dictionary

The contents of a semantic data dictionary (including all inherent constraints) is illustrated by using a simple example about a club of compact disk player owners in which members can borrow compact disks from each other (see also [8]). We assume relevancy of the registration of ownerships, reservations and borrowings. In the design occur: compact disk title *CD* (for reservations), compact disk *copy* (for borrowings), *member*, *owner*, *borrower*, *borrowing* and *reservation*. Therefore the conceptual database model looks as follows. For convenience, roles have been added to some of the attributes:

type CD	= title, performer.
type member	= name, address, zipcode, city.
type reservation	= CD, borrower_member, period.
type copy	= CD, owner_member.
type borrowing	= copy, borrower_member, from_date, to_date.

The corresponding abstraction hierarchy is given in figure 4.

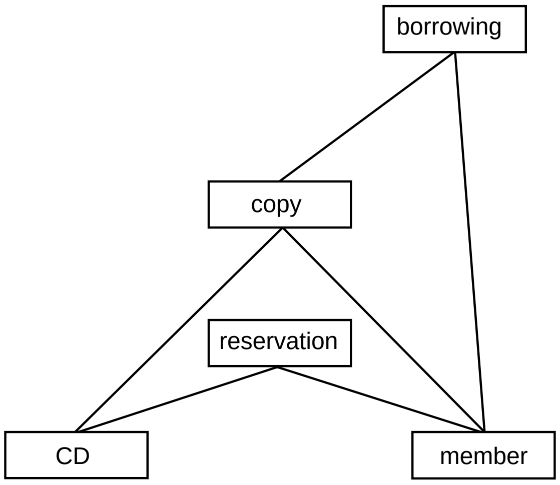


Figure 4: Conceptual database model

It is important to realize that meta data enables the DBMS to derive the attributes of a type and the semantic relationships between types. So, all necessary information can be derived during database use. As a result, the DBMS can support:

- query formulations without explicit joins;
- generic modification operations i.e. operations for all database types in which the aspect of semantic integrity is handled by the DBMS;
- application generation with complex updatable views.

4: Constraints in semantic systems

Essential for semantic query formulation is the use of the so-called *its*-construct. This construct is allowed because of the relationships expressed by the foregoing inherent constraints of relatability and convertibility. Related types (also related instances) are therefore uniquely present in a semantic database. The consequences of these inherent constraints can be perceived in operations on a semantic database. The advantages for application view development are illustrated below.

Application view development

Application views are developed in continuous interaction with the DBMS (i.e. using a WYSIWYG interface). To be able to present end users a facility in which only the location of fields in an application view has to be specified, an extension (not a modification) to the foregoing basic structure of the meta model is needed. Here the following concepts occur (see figure 5): panel (i.e. the application view), included type (conceptual type occurring in the application view), included attribute (i.e. the field), path element (to specify the path between the included attribute and the conceptual attributes, as in: copy *its* owner_member *its* city).

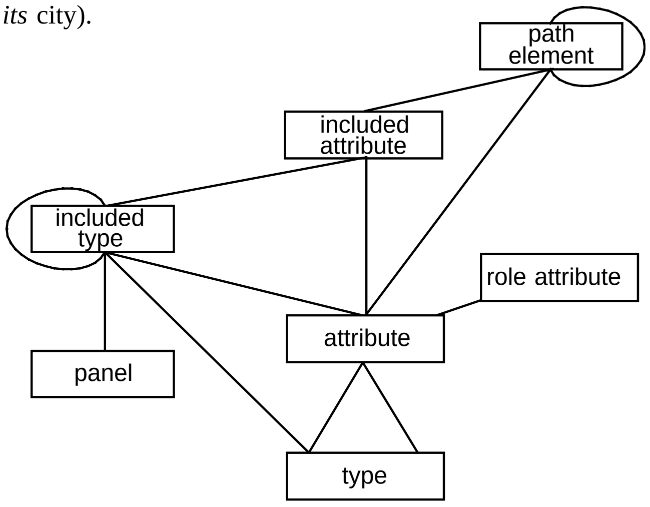
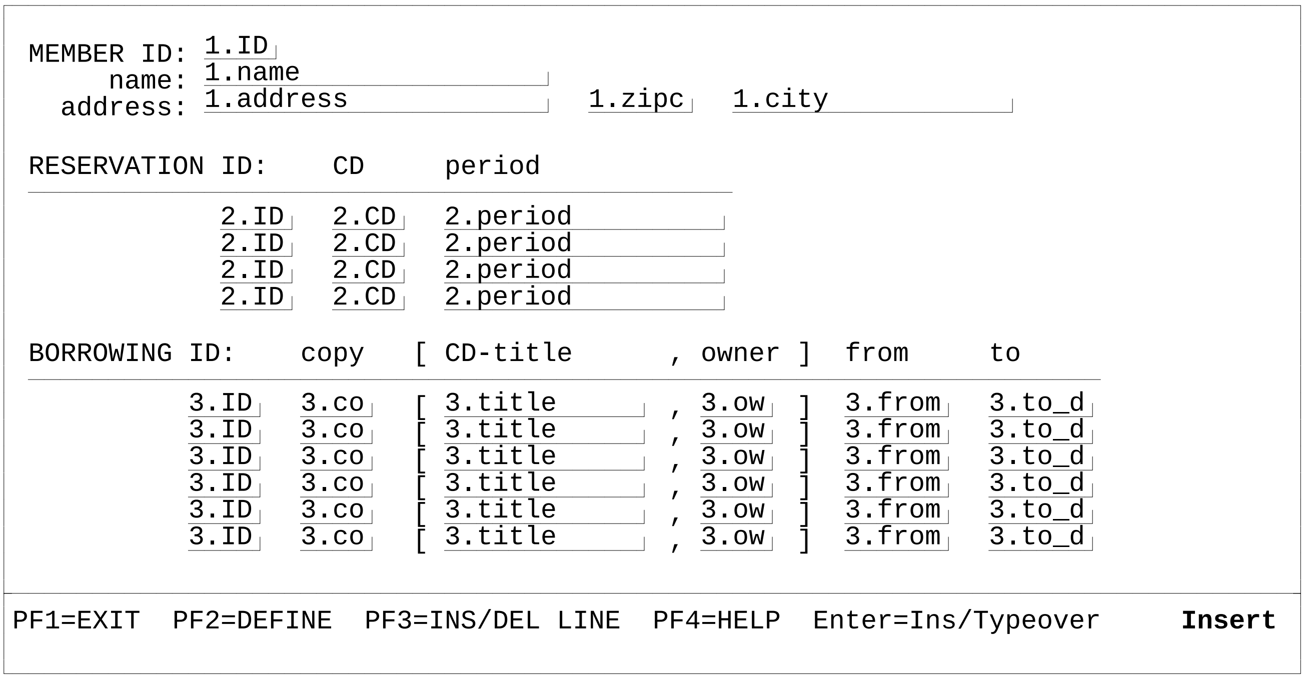


Figure 5: Abstraction hierarchy for view support

Following extensions are needed to support application views:

type panel	= panel name, text, creator.
type included type	= panel, type, parent_included type, coupling_attribute, start_row, rows_number, instance_number, level.
type included attribute	= included type, attribute, row_position, col_position, path.
type path element	= included attribute, attribute, next_path element.

Meta data enable the DBMS to support users in various stages:



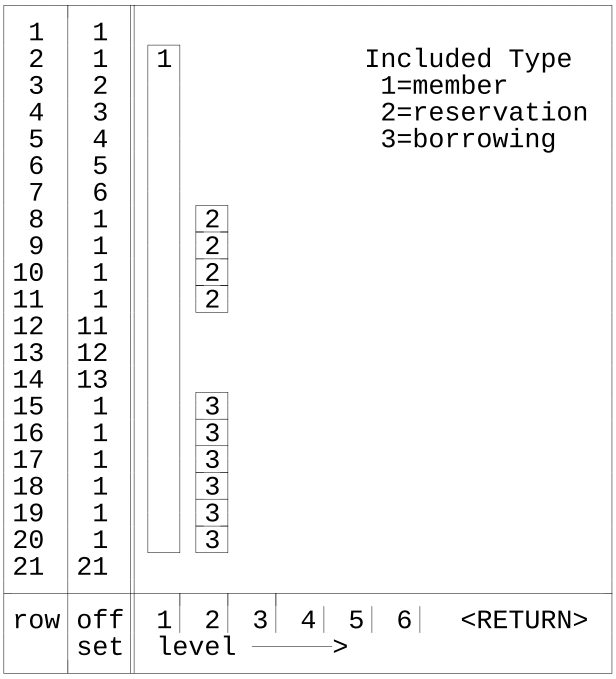
```
MEMBER ID: 1.ID
      name: 1.name
   address: 1.address                1.zipc   1.city

RESERVATION ID:    CD     period
           2.ID   2.CD   2.period
           2.ID   2.CD   2.period
           2.ID   2.CD   2.period
           2.ID   2.CD   2.period

BORROWING ID:    copy   [ CD-title      , owner ]  from     to
         3.ID   3.co   [ 3.title       , 3.ow  ]  3.from   3.to_d
         3.ID   3.co   [ 3.title       , 3.ow  ]  3.from   3.to_d
         3.ID   3.co   [ 3.title       , 3.ow  ]  3.from   3.to_d
         3.ID   3.co   [ 3.title       , 3.ow  ]  3.from   3.to_d
         3.ID   3.co   [ 3.title       , 3.ow  ]  3.from   3.to_d
         3.ID   3.co   [ 3.title       , 3.ow  ]  3.from   3.to_d

PF1=EXIT  PF2=DEFINE  PF3=INS/DEL LINE  PF4=HELP  Enter=Ins/Typeover    Insert
```

Figure 6: Development of an application view

- *during panel development*
  The relationships stored in the semantic data dictionary enable the DBMS to determine which relationships are useful to be incorporated in an application view. This makes it possible for the DBMS to offer only useful relationships.
- *during panel use*
  The DBMS is able to determine for a database which relationships can be inserted through user operations and which relationships can be deleted. The DBMS is also able to decide which modification operations are allowed in an application view (in other words: to determine the updatable application views).

In the application view missing chains (for example: copy *its* CD *its* title) can be added by the DBMS. The same is true for certain IDs (for example: memberID in reservation). Application views are defined using a WYSIWYG interface (as in figure 6). Definition of the application view consists of filling the semantic data dictionary of figure 5.

The application view above looks similar for the end user (see figure 8). The behavior (retrieval, insert, delete, update) of application views is completely determined by the contents of the semantic data dictionary. Operations are allowed if the user has the required authorization (according the authorization dictionary) and if the application view can be mapped unambiguously on the conceptual structure of the database (figure 7).

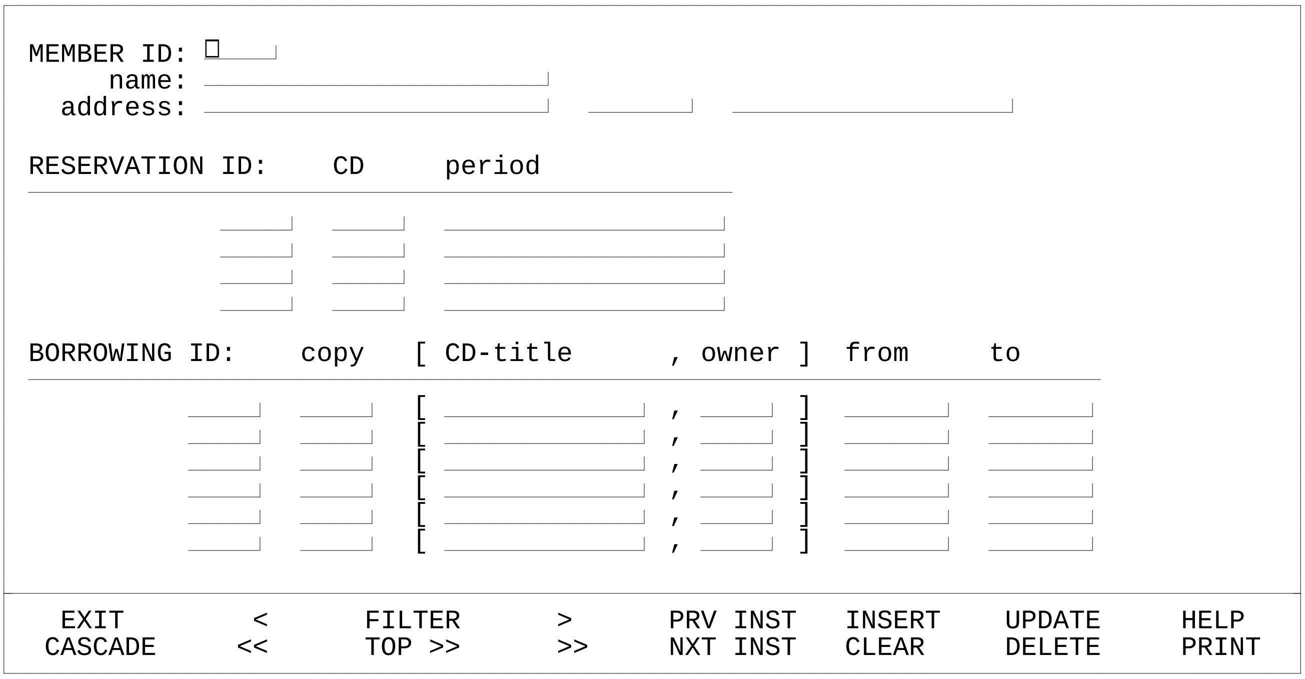


```
  1   1                         
  2   1   1                Included Type
  3   2                     1=member
  4   3                     2=reservation
  5   4                     3=borrowing
  6   5
  7   6
  8   1       2
  9   1       2
 10   1       2
 11   1       2
 12  11
 13  12
 14  13
 15   1       3
 16   1       3
 17   1       3
 18   1       3
 19   1       3
 20   1       3
 21  21

row  off   1  2  3  4  5  6   <RETURN>
     set   level ——>
```

Figure 7: Structure of an application view

Operations on application views

The panel user may use application views for different database operations depending on the user's authorization



```
MEMBER ID: □____
     name: ____________________
  address: ____________________   ______   ________________

RESERVATION ID:    CD     period
_____________________________________________

                 ____    ____    ________________
                 ____    ____    ________________
                 ____    ____    ________________
                 ____    ____    ________________

BORROWING ID:    copy   [ CD-title      , owner ]  from     to
____________________________________________________________________

               ____    ____   [ ____________ , ____ ]  ______   ______
               ____    ____   [ ____________ , ____ ]  ______   ______
               ____    ____   [ ____________ , ____ ]  ______   ______
               ____    ____   [ ____________ , ____ ]  ______   ______
               ____    ____   [ ____________ , ____ ]  ______   ______
               ____    ____   [ ____________ , ____ ]  ______   ______

  EXIT       <     FILTER      >     PRV INST   INSERT    UPDATE     HELP
 CASCADE     <<    TOP >>      >>    NXT INST   CLEAR     DELETE     PRINT
```

Figure 8: Using an application view

profile, the cursor position (denoted by □ in figure 8), the conceptual structure and the view structure. Search conditions containing AND/OR (so called: filters) can be placed on all fields, search processes can be cascaded/ restricted and inherent constraints are automatically enforced. Extensions to the given structure of the meta model can be used for additional behavior (for example: static constraints, dynamic constraints, etc.) and look and feel of application views (e.g.: highlighting, colors, lookup panels, etc.).

Conclusion

This paper has demonstrated the importance of two inherent constraints for use in a database environment. These constraints do not contain any implementation aspect and are close to the intuition of a database user. Both queries and application views benefit from these constraints. Knowledge represented by means of inherent constraints is available for several applications. The DBMS uses this knowledge for all applications. A broad class of applications can be developed without any special programming knowledge. The advantages have been demonstrated: reliable data processing, simple and rapid application development and a significant reduction in software maintenance costs. Databases become available for a broad, new class of end users.

The semantic approach has shown that only one fundamental relationship is essential for the derivation of information from a database, namely the type-attribute relationship. The use of this relationship has made it clear that other relationships (such as the n:m relationship in the entity-relationship approach and object oriented approaches) can be conceived as consequences of the type-attribute relationship.

Acknowledgments

I wish to thank all persons who contributed to the presented parts of the Xplain DBMS: Rien Bos as software manager; Joep Hompus and Huub Goossens who defined and implemented the basic structure of the semantic data dictionary; Jan Bron, Martin van Dinther, Ton van Wijk and Ronald Krijgsheld who contributed to the application generator of the Xplain DBMS. My colleagues Dolf van der Ende and Bert Bakker are acknowledged for frequent discussions on semantic data modeling and meta modeling.

References

[1] J.A. Bakker, A semantic approach to enforce correctness of data distribution schemes, *The Computer Journal* **37** (1994) 7, pp. 361-375.

[2] F.J. Corbato, On building systems that will fail, *Communications of the ACM* **34** (1991), pp. 73-81 and Interview with K.A. Frenkel pp. 83-90.

[3] Y. Freundlich, Knowledge bases and databases: converging technologies, diverging interests, *IEEE Computer*, **23** (1990) 11, pp. 51-57.

[4] J.P. Rosen, What orientation should Ada objects take?, *Communications of the ACM* **35** (1992) 11, pp. 71-76.

[5] J.M. Smith and D.C.P. Smith, Database abstractions: aggregation, *Communications of the ACM* **20** (1977) 6, pp. 405-413.

[6] J.M. Smith and D.C.P. Smith, Database abstractions: aggregation and generalization, *ACM Transactions on Database Systems* **2** (1977) 2, pp. 105-133.

[7] J.H. ter Bekke, *Semantic data modeling*, Prentice Hall, Hemel Hempstead (1992).

[8] J.H. ter Bekke, Complex values in databases, *Proceedings Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering*, Hong Kong (1994), pp. 449-455.