Advantages of a Compact Semantic Meta Model

J.H. ter Bekke
Department of Information Systems
Delft University of Technology
Zuidplantsoen 4, 2628 BZ Delft, The Netherlands
E-mail: j.h.terbekke@is.twi.tudelft.nl

Abstract

The Xplain data model is a powerful semantic model developed at TU Delft. The first goal in the research project was the development of an effective conceptual framework for databases. Meanwhile a complete DBMS has been developed which has been used to illustrate the advantages of semantic concepts. This paper presents an overview of some of the benefits of semantic modeling in practice and emphasize semantic meta modeling aspects as implemented in Xplain DBMS. The system consists of modules for data definition, data manipulation, application generation, report generation and authorization. It is an interpretative and completely interactive system: all modules operate on metadata in an active data dictionary which guards all user operations for integrity and consistency. Its performance is in complex situations mostly an order of magnitude faster than classical relational systems. A live demonstration of Xplain DBMS is part of the presentation.

1. Introduction

The Xplain data model [5, 6] supports the semantic abstractions: classification, aggregation, and generalization. The data model is, as contrasted with many other data models, based on only one fundamental notion: the type concept. This results in an extremely compact meta model for databases with several interesting advantages, to mention a few:

- **flexibility**
  A simple foundation results in flexibility. Because the interpretation of an object is not fixed also new interpretations of the same object are allowed. They can be added to existing ones. This is allowed during the design process but also while using existing databases. Restructuring is not a separate process including unloading and loading the database; it can be realized interactively on the fly.
- **inheritance**
  Structural aspects are defined inherently without redundancies. They lead to meta information. This implies that users have an intelligent system at their disposal: the software can guide users towards effective solutions. This has one important consequence: users don’t have to debug their applications because no application software is developed. User defined applications will always run properly while the systems will automatically guarantee the integrity and consistency of the database due to the metadata in the active data dictionary. Both structure and behavior can be inherited, not only methods as in OO approaches [7].
- **compactness**
  The required software to support the data model is extremely compact. This has been proven with Xplain DBMS, a user friendly system for databases. Its expressive and processing power goes in some respects far beyond relational but it requires less than 1.5 Mbytes object code. The system operates under Unix and Linux. A wide range of

computer hardware is therefore suitable for this data model.

- **efficiency**
  Practical experiences have shown that the system can handle complex database queries very efficiently. System's performance is independent of the record size, and is utmost linear dependent on the number of records. For complex queries on large databases (millions of records) this can result in a gigantic performance improvement compared with SQL-based systems. This improvement is standard and automatic; all users have these advantages. This makes the system suitable for a large class of interactive applications, even on the internet.

This paper will introduce the basic notion and abstractions in the Xplain data model. The consequences of this framework will be illustrated using the structure of the semantic data dictionary: the utmost compact meta model.

2. Design Decisions

Classical database systems are implemented using record structures consisting of more than one field, not so in Xplain DBMS. This approach can easily be explained by comparing the simplified structure of a relational dictionary (or catalog) with the semantic data dictionary. Figure 1 shows the two structures in terms of semantic abstractions. Figure 1a indicating a *1*:*n* relationship between relation and attribute, and figure 1b indicating that an attribute is related to two types.

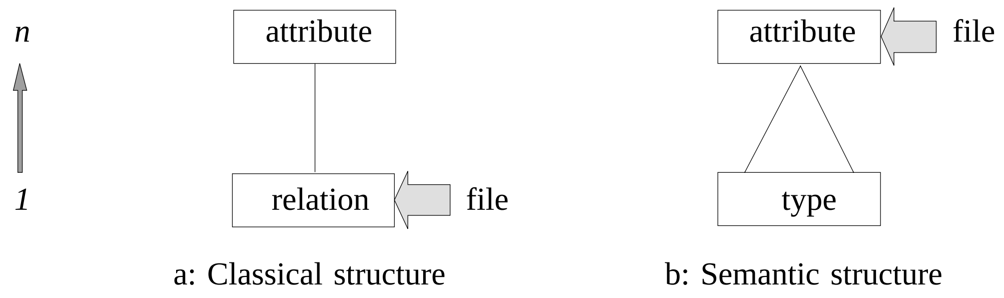

Figure 1: Simplified data dictionary structures

The big arrows in figure 1 denote places where metadata about physical data structures is stored. In classical relational systems one file for all attributes in a relation, whereas in Xplain DBMS a file for each attribute (a transposed file [1]). The latter is conform the meaning expressed in the data dictionary. It gives more flexibility (attributes/files may come and go), a better performance (not all data is loaded for a particular user operation) and it is also suitable for deviating data structures (e.g. multimedia). The approach lends itself for the use of B+trees [2] as a method for automatic balanced database access.

It is evident that the semantic structure allows hierarchies of types, while the classical structure only a flat structure of one level. This has enormous consequences for all user operations: in a relational system all connections (or joins) must be formulated explicitly, while in the semantic structure they can be derived resulting in automatic database navigation.

Another important aspect of the data manipulation language is found in the usage of metadata for intermediate results of query language statements. An example is given in figure 2. The solution in a typical SQL formulation (which is not unique) is compared with a typical Xplain formulation (which is unique) for the same query. Both have completely different complexity characteristics for both man and machine (see also [12]).

```
SELECT item#, description
FROM items i
WHERE stock <
        SELECT SUM (number)
        FROM sales s
        WHERE s.item# = i.item#
```

a: Typical SQL formulation

```
extend items with sold =
        total sales its number
        per items.

get items its description
        where stock < sold.
```

b: Typical Xplain formulation

Figure 2: Query formulations

SQL leads often to one single complex formulation. The processing complexity is in the order of the product of the relations involved, in this case: (#items) × (#sales). The Xplain formulation on the other hand results in a few simple statements, each with linear complexity. The total processing complexity is in this case therefore: (#sales) + (#items). Note also that both statements require only a subset of attributes.

It implies in practice a square order (or more) needed for the SQL formulation. It must be reduced by so-called optimizers. They can often result in enormous improvements of a problem, however they will not lead automatically to the extreme high performance of the semantic solution. Finally, an advantage of the semantic solution is caused by the orthogonality of concepts for data manipulation: the semantic solution to a problem is unique. It can therefore easily be expanded for more complex problems.

3. Semantic Concepts

The research on semantic modeling concepts started with the seminal papers on database abstractions [3, 4]. From these papers a new and unique approach for data modeling evolved. The approach is mainly characterized by its analogy with set theoretic concepts. The differences with other approaches are given in [5, 6, 7].

It is well known that mathematics is based on one single notion: the set concept. In the Xplain data model something similar has been constructed for databases. The data model is now completely defined by the type concept. Its definition is analogous to that of a set as occurring in the first axiom of mathematics.

Definition: A **type** is an aggregation of distinct properties. A property from the collection is called an attribute of the type.

This definition has some essential properties:

- It is remarkable that the definition does not refer to a data structure. So, not all types in Xplain have a corresponding data structure (see also figure 1). This in contrast with the relational data model where each relation corresponds with a database table.
- A type is defined by a membership relationship. However not set membership; the definition above does not refer to the instances of a type. Instances are irrelevant. This implies also ignorance of instances in type operations; only attributes can play a role there.
- The definition illustrates already different interpretations of a type: it can be considered as an aggregation but also as an attribute. Similar properties have set and element in mathematics: both being sets in different contexts. In practical situations this principle can often be extended. For example a type can (NOT must!) be

considered as: property, type, instance, generalization, aggregation, attribute, function, constraint, etc. Xplain DBMS supports this flexibility in all modules. New meaningful interpretations can be added at any time.

Data model specifications imply validity of certain integrity rules. Two inherent integrity rules are recognized for type definitions in a semantic data model:

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

It is important to realize that these two integrity rules require neither separate specification nor declaration by procedures - they are inherent to the type definitions in a semantic data model. Xplain DBMS is completely responsible for database integrity and user operations may ignore the aspect.

Similar to mathematics are operations on types (cf. union and intersection):

- the **specialization** of types A and B is the type consisting of the attributes in A or B.
- the **generalization** of types A and B is the type consisting of the attributes in both A and B.

This unique analogy between data definition concepts and fundamental mathematical concepts results in orthogonality. The same is true for data manipulation concepts because they depend on these concepts.

4. Meta Model

The Xplain meta model must be able to handle data models defined with foregoing concepts. The meta model is self-describing. Its kernel contains three interrelated types, as follows:

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

These meta model definitions contain the relevant inherent constraints:

- Both composite and base types occur as instances of the first type definition. A type does not contain data storage information.
- An attribute is defined as a relationship between two types, one being composite and the other being composite or base. Kind is related to the two abstractions supported in the data model, its domain includes the values aggregation and generalization (denoted by square brackets in type definitions). An attribute contains also data storage information (in Xplain DBMS only B+trees and transposed files are used).
- A role attribute is considered as a special attribute with a role, an example is attribute composite_type with role composite.

Aggregation and generalization abstractions are essential in semantic data models. The basic structure of the data dictionary confirms this. It consists of three interrelated types, with two relationships: between type and attribute and between attribute and role attribute. It is curious that the first concerns aggregation, the other generalization.

5. Global Architecture

Generally a 4GL development environment makes use of two different systems, each with their own metadata in repository and dictionary. There is a weak connection between the two collections of metadata. One system is only used for the management of application software (4GL) and another only for database functions (DBMS), see figure 3.

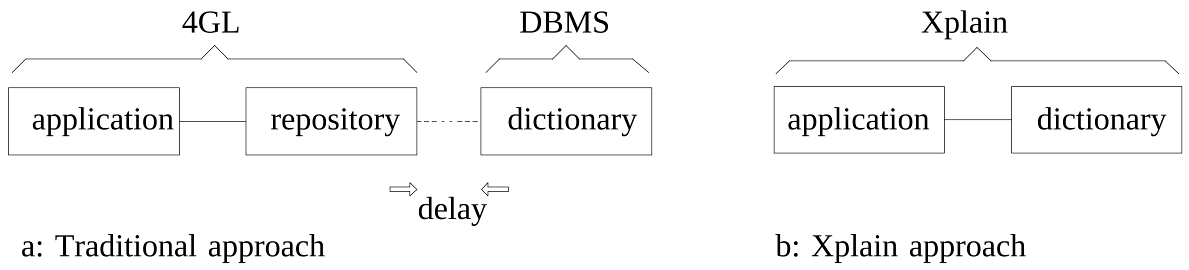

Figure 3: Alternative system approaches

The Xplain DBMS has both functions integrated into one system with an extended data dictionary. An advantage is the indissoluble connection between application functions and data management. The Xplain architecture can also allow a client-server architecture. In that case Xplain DBMS can be considered as an efficient engine for access and storage of data.

6. Practical Experiences

The benefits from a system which supports semantic abstractions are among others:

- The development of any data model (or query) is simple, straightforward without anomalies or exceptions. The development must start with simple parts. Other parts can be added later on. A data model that allows multiple interpretations facilitates this process. In that case it is absolutely irrelevant whether the first parts have been implemented in a database or whether they exist in the design stage.
- It is much easier to extend the class of solvable problems. Practical experiences have shown that problems can be solved which cannot be solved using other approaches. In education environments is shown that resulting semantic data models are much more reliable [11].
- Performance characteristics have shown gigantic differences with SQL-based systems [12]. It is also easy to reach that goal because the data language consists of orthogonal components. Queries are also easily adaptable.

Semantic data modeling concepts have been used extensively in a wide range of application areas, to mention some: hydrography, CAD frameworks [13, 14], language dictionaries, VLSI data management, printed circuits boards and version management [10].

6. Implementation

To be able to prove the promises of semantic modeling in practice, Xplain DBMS has been developed. It started as a simple research product, meanwhile it has been extended and improved. Now, it can be considered as an interactive, compact, super fast, robust (i.e.

"studentproof") and user-friendly system for databases. Complex applications can now run up an order (i.e. hundred times or much more) faster than SQL-based systems using standard hardware (e.g. Unix workstation or Linux PC), without extra efforts (e.g. by creating additional index structures).

End users see a system consisting of ± 50 menus with ± 400 options altogether. These amounts have resulted in semi-automatic, context-dependent, intra/inter menu transitions. System programmers see a system consisting of the following modules:

- **Kernel**
  including elementary data storage functions for transposed files.
- **B+tree management**
  enabling rapid search of data for integrity maintenance and user applications; a B+tree is a special indexing technique.
- **Main**
  containing menu software and the delegation of control to specialized modules.
- **DDL processor**
  processor for all definition and modification of conceptual models, including dynamic constraint handling.
- **DML processor**
  processor for all manipulation by query language statements.
- **Authorization manager**
  for the management of groups of database users with selective database rights.
- **Picklist generator**
  for dynamic, automatic generation of simple panels (i.e. updatable views).
- **Panel generator**
  enabling database users to define structured panels according their own needs. No application software is developed during definition. Panels can be used for all types of data manipulation and retrieval, allowing users unfamiliar with semantic concepts to use the power of the system (see also [8, 9]).
- **Report generator**
  enabling users to process reports from Xplain databases. It is also suitable for database import.
- **Data editor**
  simple editor for data of one database type.

7. Summary

The main characteristics of the system are:

- Efficient interactive system based on semantic data modeling concepts.
- Model driven: all interactive database operations are guarded by the system. Active data dictionary in all modules. No compilation of queries.
- System can be used without specialized knowledge of databases.
- Modifications caused by a changing reality are easy to perform. System guards database integrity and consistency.
- Simple, powerful manipulation language. Even the most complex queries can be formulated in few steps.
- Model driven, interactive panel generator that allows complex database applications without using a programming/query language.

- Query execution is extremely fast due to characteristics of the query language and application of modern file management techniques.

8. Further Research

Research on the application of semantic concepts could be continued with:

- GUI for definition and manipulation of databases using abstraction hierarchies.
- Introduction of libraries with predefined extensions, including static constraints.
- Realization of a client-server architecture and internet applications.

Acknowledgments

All graduate students who completed their master’s thesis in the Xplain project are acknowledged for their contribution to the system. Among them: Edwin Barbier, Jan Bron, Yan Cheung, Martin van Dinther, Huub Goossens, Martin Goossens, Joep Hompus, Rene Huygens, Harry Janson, Caspar Kruijf, Eddy Lasschuyt, Ed Pasma, Herman Polderman, Thijs van Bruchem, Paul van Vuren and Hans Wijntjes. I thank Bert Bakker for frequent discussions on semantic data modeling.

References

[1] D.S. Batory, On searching transposed files, ACM TODS, 2 (1977), 531-544.
[2] D.E. Knuth, The art of computer programming, Vol. 3: Sorting and searching, Addison Wesley (1973).
[3] J.M. Smith and D.C.P. Smith, Database abstractions: aggregation, CACM 20 (1977), 405-413.
[4] J.M. Smith and D.C.P. Smith, Database abstractions: aggregation and generalization, ACM TODS 2 (1977), 105-133.
[5] J.H. ter Bekke, Semantic data modeling in relational environments, PhD thesis Delft University of Technology (1991).
[6] J.H. ter Bekke, Semantic data modeling, Prentice Hall, Hemel Hempstead (1992).
[7] J.H. ter Bekke, Object databases: structure and behavior, Report 93-58, Delft University of Technology, Faculty of Techn. Math. and Computer Science (1993).
[8] J.H. ter Bekke, Complex values in databases, Proceedings International Conference on Data and Knowledge Systems for Manufacturing and Engineering (DKSME 1994), Hong Kong (1994).
[9] J.H. ter Bekke, Meta modeling for end user computing, Proceedings Workshop on Data and Expert Systems Applications (DEXA 1995), London (1995).
[10] J.H. ter Bekke, Semantic modeling of successive events applied to version management, Proceedings International Symposium on Cooperative Database Systems for Advanced Applications (CODAS ’96), Kyoto (1996).
[11] J.H. ter Bekke, Comparative study of four data modeling approaches, Proceedings EMMSAD, Barcelona (1997).
[12] J.H. ter Bekke, Can we rely on SQL?, Proceedings Workshop on Data and Expert Systems Applications DEXA 1997, Toulouse (1997).
[13] P. van den Hamer and M.A. Treffers, A data flow based architecture for CAD frameworks, Proceedings ICCAD (1990), 482-485.
[14] P. van der Wolf, Architecture of an open and efficient CAD framework, PhD thesis Delft University of Technology (1993).