Research Issues in Database Specification

Michael L. Brodie

Computer Corporation of America
575 Technology Square
Cambridge, MA 02139

Abstract

This paper summarizes discussions of a panel on "Type Specifications and Databases" at VLDB in Mexico City. Panel members are listed at the end of the paper.

Significant advances have been achieved in software engineering and programming language research in the development of specification techniques. There are important consequences for design, redesign, precision, and analysis of software. The importance of this work to database applications, and indeed data models and data languages, is now becoming apparent. However, specific database issues (e.g., constraints, complex data relationships, shared data, data independence) alter the specification problem as encountered in programming languages. The summary emphasizes the importance of (precise) specification in the database context and relates recent results in both programming languages and databases. It also lists outstanding theoretical problems and the relationship of advances in specification research to the development of semantic data models and high level languages for databases.

Various Kinds of Specifications

In the database field, specifications may be written for data models, data languages (DDLs, DMLs, query languages), DBMSs, and database applications (database schemas and data manipulation programs).

Potential Benefits of Specification for Databases

1. Means of communication between designers, implementors, and users (e.g., a specification can play the role of a contract).

2. Precise definitions of the semantics of data models, data languages, DBMSs, and database applications.

3. Means for reasoning about database models, languages, and applications.

4. A basis for constructing database theory and for precisely formulating database problems (e.g., incomplete information).

5. Means for comparing the expressive power of data models and languages.

6. Guidance for database software design, implementation and modification.

7. Underlying proof techniques for analysis and verification of static and dynamic properties of database applications (semantic integrity and transaction verification).

8. Means to define the satisfiability of an integrity constraint and to check the correctness of the integrity maintenance algorithms (semantic integrity validation).

9. Means for defining precisely the answer to a query and the effect (semantics) of a transaction and the ability to check the correctness of the associated algorithm.

10. Basis for defining inferencing capabilities.

With the above potential benefits, why are formal methods so limited in their application or practical usefulness? What is needed to make them more effective in practice?

Database Specific Problems for Specification

Database software has specific characteristics not generally encountered in the classes of applications considered in programming language research into specifications. Consequently, specification theory must be adapted to accommodate specific database properties. Examples of such properties are: data sharing, views, concurrency, complexity of integrity constraints, the importance of static as well as dynamic properties of objects, importance of queries as well as updates, persistence of data and the existence of systems of languages (e.g., definition, manipulation, and query).

Two particular problems are worth noting: structuring specifications and specifying partial functions. Database applications tend to be large, complex, and evolutionary in nature. For specifications to be useful for databases, one must be able to structure and to manipulate specifications, to compose higher level objects (specifications) from constituent objects (specifications), to modify existing objects, and to provide different views of existing objects.

Due to the importance of integrity constraints and exceptions, one must also be able to specify partial functions.

The properties of an object can be defined in terms of a data type: a given domain, some operations or functions over the domain, plus some partial functions used to define constants. Ideally, to define the functions over a given domain, one could define the ranges of the functions (hence the expected semantics of the associated type) and the values over which the type operators are not well defined, in order to deal with all the exceptions.

These two problems are being addressed by programming language researchers who are developing languages such as CLEAR (Burstall of Edinburgh and Gougen of UCLA), HOPE (Burstall and MacQeen of Bell Labs), and CIP-L (Baurer et al. of Munich), and by database researchers (Brodie of Computer Corporation of America and Ridjanovic of the University of Minnesota).

Specification = Development + Analysis

Database specification is not simply a definitional process. It should consist of both development and analysis. Currently, there are no formal languages or methods for database design and in particular for database specification. However, researchers are beginning to apply research results such as those for algebraic types to database problems. The following paragraphs discuss the need for development and analysis in the context of algebraic specifications.

Database specification starts with an informal requirements analysis. In a process of stepwise refinement the specification is complemented and made precise, yielding, for example, a conceptual schema. Then a joint development of data structures and algorithms leads to an internal scheme. Thus the development of a database specification is a formal activity consisting of a sequence of transformation steps. The formal correctness of these transformations can be uniformly defined by an algebraic notion of implementation that takes into account only the visible behaviour and not the representation.

It is not sufficient simply to write down a large number of axioms; at each level, semantic analysis may be required to ensure the soundness of the specification. This analysis gives a guideline and control for further development and feedback with the informal ideas in mind. Thus algebraic types offer a formal background to work with and to reason about database specifications.

The size and complexity of database applications requires a structured and flexible specification technique. A specification language may be needed to express operations on specifications (e.g., parameterization, instantiation, renaming, composition, and decomposition). Hierarchical algebraic types offer a uniform formal tool for a structured database design. Dosch and Wirsing are working on these aspects at the Technical University of Munich.

Purpose of Specification

Specifications are intended to be tools that ensure the accuracy of models, languages, systems, etc., with respect to some given perception (e.g., requirements, functionality, theoretical notions). One approach to achieving accuracy is through decomposition and abstraction. One attempts to decompose a complex system into individual properties which are then defined abstractly. The definitions need only be as detailed as the problem or analysis at hand requires. Hence, precision of specific details is an immediate goal for given aspects of a specification, whereas accuracy is the ultimate goal of an entire specification.

Formality and Precision

Formal description techniques should be used as tools that help to produce precise definitions. Formalism per se can cause problems for those who are not familiar with the notation. However, any formalism is better than no formalism. It is therefore important to understand the relationship between formalism (the tool) and precision (the immediate goal).

Something that is formal lends itself to mathematical and mechanical manipulation. "Formal" means expressed in a well defined language; there may be no useful content. Something is precise if it provides all the needed details accurately. Precise means that one understands all the details needed for the problem at hand. A common misconception is that formality implies precision. Precision does not require completeness; this is where the issue of abstraction detail comes in to play.

Something can be formal and imprecise or informal and precise; all combinations exist. (This comparison is owed to Chuck Rich at MIT.) Generally, descriptions that are both informal and imprecise are rejected as being sloppy. Many specification techniques are both formal and precise, since they are best for mathematical and computer manipulation. There is considerable potential for mechanisms that are informal yet precise, (e.g., natural language can be used precisely). Euclid's algorithms were written precisely in natural language. The final combination, formal and imprecise, is also interesting; much of AI deals with using formal means to express imprecisely defined problems.

Need for Precise Definition

The emphasis in using precise specifications should be on mental hygiene. Developing a precise specification can be extremely valuable in developing an understanding of the concepts at hand and their interactions. It is extremely important that precise specifications be given for any new data model, model extension, or data language. This is particularly important and problematic when dealing with complex, semantic data models and languages. If the semantics are not clear, the models or languages are hard to use and may not work. In the future, a new model, feature, or language should not be accepted without a precise specification that is accurate, complete, and unambiguously communicable to those who will use it. Precise specification of data models and languages is now a critical issue for database and programming language standardization. In the standards context, the classical data models have been defined precisely by Manola and Pirotte at Computer Corporation of America, and the relational data model has been defined precisely by the ANSI Relational Task Group (see SIGMOD Record 12, 4 July, 1982).

Choice of Formalism

No formalism is uniformly superior. Operational formalisms tend to be biased toward the discipline on which they are based; however, they can be used to give implementors good guidance. Although axiomatic and denotational formalisms are abstract and can provide representation free definitions, they can be very difficult to comprehend. Each formalism has its limitations. Some typical criticizisms of various approaches follow. It is difficult to specify (add) constraints using algebraic specifications. The axioms required for a data model specification tend to be formidable. Hundreds of axioms are required and intuition can be lost due to the amount of detail. A major disadvantage of first order logic (FOL), set theory, axiomatics, and many other formalisms is their inability to handle the dynamic properties of database applications. To specify behavioural properties of databases, special logics, such as dynamic logics, are being considered. A major challenge in this regard is not only the specification of dynamic properties but also its integration with the specification of static properties.

Another important aspect in the choice of formalisms is the availability of an associated specification methodology. This is particularly

important considering the size and complexity of a database application specification.

Complementary Specifications

Since many people have their own favorite ways of specifying objects, there is a need for a variety of specification languages. Also, since no specification language can be used effectively to specify all aspects of database applications (e.g., concurrency, sharing, indeterminacy) special languages will be required. The challenge here will be to ensure that the resulting specifications are consistent and complementary to the extent that they support adequate analysis and guidance for the complete system. One approach to these issues is to map all specifications onto a common formalism so that communication can be established between different views and problems of semantics and analysis can be addressed. FOL, set theory, function theory, and algebras have been proposed as candidates for the common language. In the case of FOL, special logics are being developed for the specific issues.

Some existing specification techniques are complementary in that they are used to address different levels of abstraction. Abstract specification techniques, such as those used for abstract data types, are employed in order to produce high level, representation-free models. Constructive specification techniques such as denotational semantics and the Vienna Development Method (VDM) are used to produce models that are more concrete (detailed) than abstract data types. Hence, an abstract data type specification can be followed by a constructive specification as a more detailed design. On different levels of abstraction the emphasis shifts from the behaviour to the representation (e.g., from axiomatic to operational specifications).

Need for Appropriate Formalisms

In programming language research, specification is generally considered independently of target implementation languages. By contrast, specification in databases is generally considered in terms of a data model and its associated languages. The motivation here may be that, whereas specification is important, the formalism and the required mathematical sophistication have been viewed as inappropriate for database designers and analysts. As a result, research on the specification of database applications has focused on less mathematically sophisticated formalisms such as graphics, interactive languages, and other "user friendly" interfaces. Such database design and specification research has been carried out by McLeod at the University of Southern California and by Brodie at Computer Corporation of America.

Application Dependent Specifications

Before formal specification techniques were proposed there was no alternative left to practitioners but simply to program the database applications using some DBMS. At this point, database objects were referred to using machine-oriented terminology (records, files, pointers, etc.).

Next came the invention of data models. The motivation was to provide a high level, considerably more formal specification tool, using some abstract representation (e.g., trees, networks, tables, sets), that was completely independent of the underlying physical structures.

Research on abstract data types in programming languages showed that types in general (and hence databases in particular) can be described by the definition of and the mutual relationships among some fixed set of operations defined on them. This opened the way to data model-independent specifications, a fact that has not been fully recognized by the research community. Data model independence is achieved when the types cease to be "relations," "entities," etc., but are chosen instead from the domain of discourse of particular applications. Instead of operations such as "insert employee tuple," application oriented operations such as "hire employee" are specified.

For objects arising in programming languages, a specification in terms of operations may suffice. What is a stack if not what is captured by the operations push, top, etc.? However, for databases it is fair to say that the information objects are more meaningful and stable than the operations that happen to be used to handle them. However, there is some debate about this issue. Some people believe that data is significantly characterized not by its static properties but by the available operations and their properties. There are distinct advantages of application oriented specifications compared with parametric (data model dependent) specifications of database applications.

Hence an operation-independent level of specification seems desirable, in terms of states and transitions (i.e., by enumerating the kinds of facts that will be stored and what situations (states) are reachable from others). This captures the intuitive notion that a database can still be the same type of object if new operations are added or existing ones are dropped or modified. Furtado (of Rio de Janierio) is now investigating data model-independent specifications, or, more precisely, application dependent specifications.

Multi-level Specifications

The three ways mentioned above for formally specifying a database should be used within a multi-level methodology (states/transitions -> operations -> abstract representation) starting with the initial informal characterization of "reality" and ending with some physical implementation. In this process, higher levels are more general and stable.

Database Systems Engineering

A precise specification of a software system can be viewed as an abstract implementation of the system. In this case, database development in the specification process corresponds to systems design. A major benefit of a formal systems specification is the ability to reason about the system at an abstract level. There is also the potential of analytical tools to be considered. Bjorner of the Danish Datamatiks Center is working on this issue.

Database Specification Desirata

1. Specifications based on the metatypes of a database model (e.g., aggregation, generalization, and association hierarchies).

2. Means for structuring specifications and defining partial functions. For example, specifications organized by generalization hierarchies that support property inheritance of both the structures and operations.

3. Methodologies for writing specifications, including techniques for designing, analyzing, and writing more structured specifications.

4. Means for specifying database specific properties: objects or states, incomplete information, integrity, recovery, exception handling, security, concurrency, messages, indeterminacy, data sharing, and interference as it occurs in database views.

5. Means for specifying interactions between types (e.g., decomposition and composition of types).

6. Means of integrating specifications within the same formalism in a convenient way (e.g., library of types with rules for them to forms new types).

7. Consistent and complementary specification techniques and means for using them together to specify a system completely, particularly to specify both static and dynamic properties in a consistent way.

8. Different specifications of an object at one level of detail or abstraction, each having different properties (e.g., views and multi-level specifications of an object at different levels of detail -- hierarchies of views).

9. Specification languages that are sufficiently precise and formalisms that are appropriate for the intended users (e.g., increased declarative versus procedural specifications, and higher level (closer to the problem domain) specification languages).

10. Effective proof and analysis techniques that are efficient for large complex database application specifications.

11. Specifications that support the integration of modelling abstractions such as abstract data types, database abstractions such as generalization hierarchies (property inheritance), views, control abstractions, etc.

12. An intelligent specification environment to provide increased automated support of the database specification process (e.g., support of: database abstraction and modelling concepts, specification methodologies, specification development and analysis, and moving effective methodological concepts into data models and languages).

13. Specification of parameterized queries that could use abstract data type-like operations.

14. Types for self-modifying and more intelligent objects (e.g, frames in artificial intelligence).

Acknowledgement

I would like to thank the members of the panel (listed below) for their participation and in particular Walter Dosch and Alain Pirotte for their comments on this paper.

Robert Balzer — Information Sciences Institute, 4676 Admiralty Way, Marina del Ray, CA 90291

Dines Bjorner — Scientific Director, Dansk Datamatik Center, Lundtafteuej 1C, DK-2800 Lyngby, Denmark

Michael L. Brodie — Computer Corporation of America, 545 Technology Square, Cambridge, MA 02139

Walter Dosch — Technical University of Munich, Institut fuer Informatik, Arcisstrasse 21, D-8000 Munich, West Germany

Antonio Furtado — Department of Informatica, Pontifica Universidade Catolica of Rio de Janeiro, Rua Marques de Sao Vincente, Rio de Janeiro, Brazil 22453

Alain Pirotte — Phillips Research Lab, 2 Ave. Van Becelaere, 1170 Brussels, Belgium