FUNCTIONAL COMPLETENESS IN OBJECT-ORIENTED DATABASES

Priti Mishra

Margaret Eich

Department of Computer Science & Engineering

Southern Methodist University

Dallas, TX 75275

(mishra, eich)@seas.smu.edu

ABSTRACT

A definition of completeness in the context of Object Oriented Databases (OODBs) is proposed in this paper. It takes into account the existence of various categories of functions in OODBs, each of which must be complete in itself. The functionality of an OODB can be divided into sets of related functions. For example, functions needed to perform all schema evolution operations or all version management operations belong in two distinct sets. Further, each set of functions must include all functions needed to perform all operations defined for that set. Thus, for an OODB to be functionally complete, it must support a certain number of sets (or categories) of functions and each such set must be complete in itself. The purpose of this paper is not to give a precise definition of the categories of functions but rather to define a framework within which such categories should be examined. This paper contains a working definition of functional completeness. We would welcome any feedback on our proposal.

1.0 Introduction

It is generally agreed that a formal OODB data model, in the sense of the relational data model, is not feasible at this point. At the same time, there is general consensus on the features which should be present in an OODB [MAIE89, ATKI89, KIM90, UCB90]. In relational databases (RDBs), "completeness" was used to measure the expressive power of query languages. The equation changes for OODBs because they aim to include more features for a wider set of application areas than relational databases. They are expected to include more areas of functionality than the set of relational algebra operations associated with RDBs. As a result of their diverse origins and the fact that many of them have been developed to solve specific problems, existing OODBs do not all support the same set of features or functions. For instance, some OODBs have more powerful modeling features such as support for multiple inheritance while OODBs aimed at CAD/CAM applications usually have strong support for versions. Again, the level of version support, including the kinds of versions and the operations which can be performed on them, varies between systems. For example, historical databases require a simpler versioning system than CAD/CAM databases. In view of this diversity, it is felt that a definition of functional completeness is needed for OODBs in order to provide a means of evaluating the power of individual OODBs and to provide a basis for comparing different systems.

Many definitions for completeness in the area of databases have been in the context of query languages. The intent has been to quantify the expressiveness of query languages. In the context of OODBs, completeness has a different interpretation. It is related more to the functionality provided by the OODB as a whole system rather than to the query language alone. Thus, completeness in the context of OODBs encompasses more areas of database operations than completeness in the context of RDBs. The purpose of this paper is not to precisely define the various categories of functions but to provide a framework for evaluating the functionality present in an given OODB or for comparing the functionality of OODBs.

The remainder of the paper is organized as follows. First, the basic concepts of completeness and operational equivalence are introduced. Next, we review object-oriented systems which have either studied the issue of completeness or used the concept of grouping OODB functions into categories. We build on these two concepts and present our definition of completeness in the next section. Next, the power of this definition is illustrated via examples.

2.0 Completeness versus operational equivalence

In order to evaluate the power of an OODB or to compare two OODBs we must have two items: a definition of functional completeness and a means of applying it to OODBs. In this section we review the development of completeness in relational

databases and the concept of operational equivalence. The former is used to develop a definition of functional completeness for OODBs and the latter is used to provide a way to use the definition in evaluating OODBs.

2.1 Completeness in Relational Databases

Completeness has been discussed in the context of query languages for relational databases. Initial analysis of the relational algebra by Codd [CODD72] lead to the following definition of *relational completeness.* The set of relational operators $\{\sigma, \pi, \cup, -, \times\}$ was considered complete because all other relational operations such as join, division, intersection, etc. can be expressed in terms of these operations [ELMA89]. Any query language which included these operations was considered to be relationally complete. However, as experience with RDBs added up, it was found that operations such as transitive closure, aggregate operations, and updates cannot be expressed in relational calculus [CHAN80, ELMA89, p. 159]. That is to say, operations other than the relational operations were found to be so useful as to be considered essential. It became clear that these operations should be a part of database query languages but they could not be generated using the relational operators. Relational completeness was therefore found to be an inadequate measure of the expressive power of a query language.

This lead to the following interpretation of completeness [CHAN80]. A set of reasonable and computable queries is defined. Then, the "completeness" of a query language depends on whether it can be used to formulate all the queries in this set. The validity of this method hinges on the definition of a reasonable base set of queries. There are two approaches to defining the base set of queries. One approach is to define an appropriately large set of computable queries. The other is to augment the set of operations offered by the relational algebra by operations such as the transitive closure as and when they are identified. In both cases, it is important that all computable queries be included in the set. (The term computable queries refers to valid queries which can be computed over a given database.)

The point to be noted in the development of this definition of completeness is that it has evolved from the restricted definition based on relational algebra to a more practical one which is based on the concept of computable queries. This makes this definition very flexible and powerful because it includes any and all computable queries. It must be remembered that the power of the definition depends on the contents of the set of computable queries. Therefore it is not as precise as Codd's interpretation of completeness.

2.2 Operational equivalence

One of the shortcomings of OODBs is that there is no formal mathematical data model. In the absence of such a model, designers have chosen their own definitions of what constitutes an

OODB. As a result, OODBs in existence today contain varying sets of features. Therefore, instead of merely defining completeness of OODBs it may be more useful to define *operational equivalence* of OODBs as well. This concept is examined in this section.

Let there be two database systems, S_1 and S_2, with operation sets F_1 and F_2, respectively:

$$F_1 = \{f_{11}, f_{12}, \ldots, f_{1n}\}$$

$$F_2 = \{f_{21}, f_{22}, \ldots, f_{2m}\}$$

where f_{ji} is the ith function in the operation set F_j and n and m are the number of operations in sets F_1 and F_2 rspectively.

If for every f_{1i} in set F_1, there exists a set of one or more operations in F_2 which, when executed in some fixed order, have the same effect as f_{1i} on any database state, then system S_2 is said to be operationally equivalent to S_1. However, the reverse is not necessarily true. It is quite possible that there are operations in F_2 which cannot be simulated by operations in F_1. If F_1 is found to be operationally equivalent to F_2 and vice versa, then S_1 and S_2 are said to be *equipollent*.

Operational equivalence has been defined in the context of the Multi-Lingual Database System (MLDS) [DEMU88]. MLDS consists of a kernel data model (KDM) and a kernel data language (KDL). It accepts queries written in a number of languages, e.g., SQL for relational databases, CODASYL-DML for network databases, Daplex data language for functional databases. MLDS supports data-language translation between these input languages and its own KDL. Therefore, for every operation in the input query language, the MLDS language is able to provide an equivalent operation or sequence of operations which produces the same effect. An example of equivalent hierarchical and kernel data language queries is given in [DEMU88]. In this sense, KDL is operationally equivalent to each of the languages which MLDS accepts. They, however, are not necessarily operationally equivalent to KDL.

The query languages SQL and QUEL can also be compared in a similar way [ELMA89]. For instance, SQL provides explicit set operations while they have to be simulated in QUEL via other operations [ELMA89]. On the other hand, QUEL allows the specification of a temporary file to receive the results of a query in the query statement itself while in SQL a separate CREATE statement is needed to achieve the same result [ELMA89]. By analyzing all operations in both languages we can determine whether they are equipollent. However, we do know that both languages are operationally equivalent to relational algebra. Existing data models and their DMLs are also compared in detail in [ELMA89, Chapter 12].

While MLDS uses operational equivalence as a means of integrating databases with different data models and data languages, we propose to use the concept to compare or equate different OODBs based on a basic set of OODB functions.

2.3 Conclusions

The concept of operational equivalence can be extended to the case of OODBs as follows. If the set of functions in a given OODB can be used to generate all the functions which are present in another OODB, it can be said that the first OODB is operationally equivalent to the second. The set of functions can be expanded if and when a function is desired which cannot be generated from the existing functions. The possibility for expansion must be left open because not enough is known about the usage of OODBs to give a final definition of the "core" set of functions. In order to distinguish it from other definitions of completeness, the term *functional completeness* is used in the context of OODBs. OODB functional completeness is more like query language completeness based on the set of computable queries and less like the definition based on a fixed set of relational operators. A formal definition of functional completeness follows.

The important feature here is that it provides a way of comparing the functionality (albeit only in the area of querying) of DBMSs regardless of the means by which these functions are implemented in individual databases. Thus, for two sets of operations to be equivalent, it must be possible to simulate the operations in each set via the operations in the other set. It has been used in some of the systems described below to show that the languages in question are at least comparable to relational algebra based languages.

3.0 Previous work on completeness

There is a diversity of opinions on the topic of OODB functionality. A variety of approaches have been taken to solve the problem of defining OODB completeness. This section does not cover all the work being done in the development of OODBs or database programming languages. Instead, the systems reviewed were chosen as representatives of their relationship to our concept of functional completeness of OODBs. Each of these systems have either tried to define completeness for a particular category of functions or justified the grouping together of related functions.

O²FDL: O²FDL is a database programming language based on the functional programming paradigm [MANN89, MANN90]. It is shown that a subset of operations in O²FDL is relationally complete according to Codd [MANN90]. The language also includes database functions such as aggregate, filter, group, combine and update. The authors mention that the language could be improved by including features such as schema evolution, versions, and views which have traditionally been database functions and not programming language features. While the programming language does include aggregates, etc. functions, it is acknowledged that features such as schema evolution, etc. have not yet been taken into consideration. This is in agreement with our view that the definition of completeness for OODBs needs to take into account categories of functions other than query functions alone.

RAD: This is an experimental DBMS which allows users to add abstract data types to a DBMS and define primitive as well as complex operations on these new data types [OSBO86]. Three types of operations which can be defined on a new data type have been described. It was shown that the query language based on this set of operations is complete in the sense of [CHAN80]. The significance of this work lies in the fact that RAD is an extensible database system and the query language is complete for an arbitrary set of data types.

ORION: This work considers a single category of functions, namely, the schema evolution functions [KIM88]. After defining an extensive list of schema evolution operations, a short list of core operations is extracted. It is shown that this set of functions is complete with regard to schema evolution. That is, all the schema change operations defined can be generated from the operations in the set of core operations. Thus, the determining completeness in the category of schema evolution for a given OODB is based on determining whether the schema evolution functions in the OODB are operationally equivalent to the core set extracted by Kim. The significance of this work lies in that it considers the completeness of functions related to a specific area, namely, schema evolution.

Maier: Maier promotes the viewpoint that there cannot be an object oriented data model for OODBs as there is for relational databases [MAIE89]. The alternative to having a data model is to define a loose collection of features by extracting a "common core shared by a group of similar DBMSs" [MAIE89]. This definition of a substitute for a data model is used below in defining completeness in the context of OODBs.

Database programming languages: Such programming langauges should be computationally complete. The ideal way to make a database programming language computationally complete is to integrate it seamlessly with a programming language [GYSS90, COPE84]. However, this approach suffers from the problem of impedance mismatch. The impedance mismatch problem has two parts: (i) the programming paradigm mismatch, and (ii) the data structure mismatch, based on the differences in programming paradigms and the data structures supported by the two languages [COPE84]. Since we are concerned with database functionality without restricting ourselves to any one technique for achieving it, this issue will not be considered further in this discussion.

Summary

From the above review, it can be seen that most discussions on completeness in DBMSs have involved database programming language aspects or query language aspects. The RAD related work takes the notion of completeness further into the area of completeness of a query language in the presence of extensibility. Kim's work covers the area of schema evolution. O^2FDL covers query language completeness. It is also seen that groups of functions other than query languages are also

being considered important in OODBs. Maier's view of OODBs leads to the conclusion that an OODB consists of a collection of modules. Each module supports a particular OODB feature and has its own set of functions. Based on these works we view OODBs as having associated with them several distinct groups of functions. In the next section, a proposal to combine the ideas of completeness and the creation of groups of functions into an integrated definition of functional completeness applicable to OODBs is presented.

4.0 Proposed definition of functional completeness

Completeness in the context of OODBs can be approached by first defining an appropriately large set of functions. Each function in the set must either support some database function or some object-orientation feature. (Keep in mind that so far people only agree that an OODB can be defined not by a data model but by a "set" of database and object-oriented features [ATKI89, KIM90, UCB90].)

The categories of functions which an OODB should have and the "core sets" of functions for each category are determined by the following procedure:

(i) Create a set of all functions which should be present in an OODB. A preliminary approach is to create a set F of functions, regardless of the database feature they support, which is the union of all the functions provided by current database systems.

$$F = \{f_1, f_2, \ldots, f_m\}$$

(ii) Divide these functions into categories. The functions in each category perform an operation related to a specific area of database activity. Thus, we have the following set C of categories:

$$C = \{C_1, C_2, \ldots, C_n\}$$

where C is the complete set of categories, n is the number of categories in the complete set, and each C_i is defined as follows:

$C_i = \{f \mid f \in F\}$ and all functions f in the set perform operations related to the same category.

At this point we cannot say whether or not all C_i are disjoint sets but since every function in F must be in at least one category, the following relation is always true:

$$\bigcup_{i=1}^{n} C_i = F$$

(iii) Examine each category of functions individually. In each category, remove any functions which can be generated by means of the remaining functions in the set. The result is the set of the core functions for that category.

Let a category C_1 of functions be defined as follows:

$$C_1 = \{f_1, f_3, f_4, f_5\}$$

If function f_1 can be expressed in terms of functions f_3 and f_4 and the remaining functions

are indivisible, then the set C_1'' corresponding to category C_1 is:

$$C_1'' = \{f_3, f_4, f_5\}$$

Then, C_1'' is the core set of functions for the category C_1 and C_1'' is operationally equivalent to C_1.

It must be emphasized that OODBs are an evolving concept and there is a definite possibility that new operations will be added to OODBs, resulting, possibly, in the addition of new categories of functions or new functions to individual categories. Therefore, the definition of completeness must take extensibility into account. A definition of completeness which takes the scope for change into account follows.

4.1 Two-level definition of functional completeness

An OODB must have a complete set of categories of functions and each category of functions should be complete in itself. An OODB is defined to be *functionally complete* if it provides certain categories of functions and each category is itself complete. In terms of the above notation, a functionally complete OODB must incorporate all the categories defined in the set C and each category must be operationally equivalent to the corresponding category C_i in the functionally complete OODB.

Therefore, there are two levels of completeness which together form the notion of functional completeness. Since an OODB is not based on a formal model and users are allowed to add both data types and functions, the definition of OODB functional completeness is not as precise as Codd's definition of relational completeness. With this in mind, functional completeness in OODBs can be defined as follows:

(1) **Category-completeness:** An OODB is said to be *category-complete* if it supports all categories of functions in the complete set of categories. The complete set of categories can be the union of all categories of functions provided by current OODBs. Let the set of categories in a given OODB be

$$C' = \{C_1', C_2', \ldots, C_n'\}$$

Then, this OODB is category-complete if

$$C' = C$$

where, C is the complete set of categories as defined at the beginning of the section.

(2) **Set-completeness:** The functions included in a category should be such that they can be used to generate any and all operations needed by an OODB in that category. This does not have to be a minimal set [KIM88]. If for a category of functions in an OODB, the set C_i' of functions defined for that category are operationally equivalent to the corresponding category C_i in the functionally complete OODB, then the given OODB is set-complete for that category.

(3) **Functional completeness:** Based on the two definitions above, an OODB is said to be functionally complete if it is both category-complete and set-complete for all the function categories defined in the complete set of categories.

That is,

if $C' = C$ and for $i = 1..n$, C_i' is operationally equivalent to C_i, then the given OODB is functionally complete.

4.2 Example

In this section, we clarify the meaning of the proposed two-level definition of OODB functional completeness by means of examples.

Example 1

Set-completeness: The application of the definition of set-completeness to the category of schema evolution functions is as follows. Let the set C_1 of core operations for performing schema evolution functions consist of the following operations taken from [KIM88] *:

(1) Add-attribute: adds an attribute to an existing class.

(2) Delete-attribute: deletes an attribute in an existing class.

(3) Set-attribute: sets the value of an attribute in an existing class.

(4) Add-edge: connect two class-defining objects in a schema.

(5) Delete-edge: delete relationship between two class-defining objects in a schema.

(6) Transpose-edge: change the order in which the superclasses of a class are listed. The order is significant in resolving conflicts due to multiple inheritance.

(7) Add-node: add a new class to a schema.

(8) Delete-node: delete an existing class from a schema.

(9) Rename: rename a class in a schema.

This list can be further reduced to a minimal set of operations. Since the minimal set obscures the basic operations in the category it is not shown here. The corresponding minimal set is given is [KIM88]. It has been shown that all schema evolution operations listed in [KIM88] can be generated from the operations listed above.

As mentioned earlier, not all OODBs have the same definition of schema evolution. Most systems have a subset, proper or otherwise, of the operations listed in [KIM88]. Strictly speaking, such systems are not functionally complete. Some additional operations related to schema evolution are [LI91]:

(1) Specialize: create a number of subclasses of a class based on the value of an attribute of the class, e.g., given a class *people*, create subclasses *adults* and *children*.

(2) Generalize: create a superclass based on the common attributes of two or more classes,

* A detailed taxonomy of operations appears in the same document

e.g., given subclasses *faculty* and *staff*, create a superclass *employees*.

(3) Merge: this is semantically the inverse of specialization. Two or more classes are merged into one class if all classes have the same set of attributes irrespective of the values of the attributes.

(4) Combine: combine classes which share the same set of data objects to reduce redundant classes. For instance, a new class *students* may be created to replace existing classes *undergraduates* and *graduates*. This operation involves major reorganization of the database because it involves moving all instance objects from the two original classes to the class created by combination.

The power of the operation set C_1 can be tested to see if these four operations can be generated from it. Let A, B, C, and D be classes such that B, C, and D are subclasses of A. All these classes are nodes in the DAG representation of a database schema. Then, the four operations listed above can be described in terms of the minimal set operations as follows:

Specialize:

Pre-condition: DAG contains node A with the attribute on which specialization is to be done.

Post-condition: Node A no longer has the specialization attribute. New nodes B, C, and D are positioned as subclasses of A in the DAG.

(1) Remove attribute on which the specialization is being done from node A (one Delete-attribute operation)

(2) Add classes B, C, D (sequence of Add-node operations)

Generalize:

Pre-condition: Classes B, C, and D exist in DAG.

Post-condition: Class A exists as superclass of classes B, C, and D.

(1) Add node A (Add-node operation)

(2) Delete common attributes from nodes B, C, and D (Delete-attribute operations)

Merge:

Pre-condition: Classes B, C, and D exist in DAG.

Post-condition: Class A is added to DAG and classes B, C, and D are removed from DAG.

(1) Add node A (Add-node operation)

(2) Remove nodes B, C, and D (sequence of Delete-node operations)

Combine:

Pre-condition: Classes B, C, and D exist in DAG.

Post-condition: Class A is added to DAG and classes B, C, and D are removed from DAG.

(1) Add node A (Add-node operation)

(2) Delete nodes B, C, and D (sequence of Delete-node operations)

Note: In the above steps, the operations Add-node and Delete-node include calls to the Add-edge and Delete-edge operations. The Add-node operation adds edges between the new node and its ancestor nodes. The Delete-node operation makes the descendent nodes of the deleted node the immediate descendents of its ancestor nodes.

Thus, it can be seen that the operations in Kim's set are all that are needed to generate these additional functions. This set will have to be augmented in cases where it cannot be used to generate a schema evolution function. To the best of our knowledge, the operations listed in [KIM88] and [LI91] cover all aspects of schema change. Therefore, an OODB can be said to be set-complete with respect to the schema evolution category if it can generate all nine functions listed above.

Example 2

Category-completeness: Based on a study of existing OODBs, the following categories of functions are identified as being essential components of OODBs. An OODB which supports all the categories would be category-complete.

(-) Schema evolution (see set-completeness example above)
(-) Version management
(-) View management
(-) Data reorganization in the event of schema changes and for views
(-) DML or query language
(-) Transaction processing operations
(-) Storage subsystem management

Therefore, the set C of all categories is:

$$C = \{C_1, C_2, \ldots, C_n\}$$

where,

C_1 = set of schema change operations,

C_2 = set of version management operations,

C_3 = set of view management operations, etc.

This list, while it cannot be considered exhaustive, can be justified by comparing the various manifestoes written regarding OODBs [ATKI89, KIM90, UCB90]. They agree on the major features even though they don't always agree on the relative importance of these features.

5.0 Use of definition of completeness

Two major uses of the two-level definition of functional completeness are foreseen.

(1) Definition of a language independent kernel of functions for OODBs: Work is in progress to extract functions which constitute the core set of functions needed in an OODB.
(2) Evaluating and comparing OODBs: One metric to evaluate and compare OODBs should be the functionality provided by the OODBs [BALL90]. The application environment determines which OODB is better; a functionally complete OODB which gives poor performance because of its complexity may not be

better than an incomplete OODB which gives good performance.

6.0 Summary

Since OODBs are in a state of evolution, it is expected that this definition of functional completeness will undergo modification as well. However, it is felt that it is flexible enough to absorb these changes. The set C of categories can be modified to include new categories as and when they are identified without affecting any other categories. The set of functions making up each categories can be enlarged also be enlarged if and when the need arises. Again, such a change would not affect any of the other categories. Just as the definition of completeness for relational query languages is only as good as the set of computable queries it is based on, this definition of functional completeness must be backed up by an acceptable set of categories and sets of core functions for each category.

REFERENCES

[ATKI89] M. Atkinson et al., "The object-oriented database manifesto", Int'l Conference on Deductive and Object Oriented Databases, December, 1989.

[BALL90] N. Ballou, *Evaluating Functionality and Performance in Object-Oriented Databases*, M.A. Thesis, University of Texas at Austin, 1990.

[CASA90] Eduardo Casais, "Managing class evolution in object-oriented systems," Technical Report, University of Geneva, 1990.

[CHAN80] A.K. Chandra and D. Harel, "Computable queries for relational databases," Journal of Computer and System Sciences, 21, 1980, pp. 156-178.

[CODD72] E.F. Codd, "Relational completeness of data base sublanguages," in *Data Base Systems*, R. Rustin (Ed.), Prentice-Hall, 1972.

[CODD86] E.F. Codd, "An evaluation scheme for database management systems that are claimed to be relational," IEEE Conference on Data Engineering, 1986, pp. 720-729.

[COPE84] G. Copeland and D. Maier, "Making Smalltalk a database system," SIGMOD, 1984.

[DEMU88] S.A. Demurjian and D.K. Hsiao, "Towards a better understanding of data models through the multilingual database system," IEEE Transactions on Software Engineering, 14(7), July, 1988, pp. 946-958.

[ELMA89] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*, Benjamin/Cummings, 1989.

[GYSS90] M. Gyssens, J. Paredaens, and D. Van Gucht, "A graph-oriented object model for database end-user interfaces," SIGMOD, 1990, pp. 24-33.

[KIM88] H.J. Kim, *Issues in Object-Oriented Schema Evolution*, Ph.D. Dissertation, University of Texas at Austin, 1988.

[KIM90] Won Kim, "Object-oriented databases: Definition and research directions", IEEE Transactions on Knowledge and Data Engineering, 2(3), September, 1990.

[LI91] Qing Li and Dennis McLeod, "Conceptual database evolution through learning," in *Object-oriented databases with applications to CASE, networks, and VLSI CAD*, Editors: R.Gupta and E.Horowitz, Prentice-Hall, 1991, pp. 62-74.

[MAIE89] D. Maier, "Why isn't there an object-oriented data model?" Tech. Rep. CS/E-89-002, Computer Science and Engineering Department, Oregon Graduate Institute, May, 1989.

[MANN89] M. Mannino, In Jun Choi, and Don Batory, "An overview of the object-oriented functional data language," IEEE Conference on Data Engineering, 1989, pp. 18-26.

[MANN90] M. Mannino, In Jun Choi, and Don Batory, "The object-oriented functional data language," IEEE Transactions on Software Engineering, 16(11), November, 1990, p. 1258-1272.

[OSBO86] Sylvia Osborn and T.E. Heaven, "The design of a relational database system with abstract data types for domains," ACM Transactions on Database Systems, 11(3), September, 1986, pp. 357-373.

[UCB90] Committee for Advanced DBMS Function, *Third-generation Database System Manifesto*, Memo No. UCB/ERL M90/28, Electronics Research Lab, University of California at Berkeley, April, 1990.