Semantic Heterogeneity as a Result of Domain Evolution

Vincent Ventrone, *The MITRE Corporation, Bedford, MA*
Sandra Heiler, *GTE Laboratories, Waltham, MA*

Abstract *We describe examples of problems of semantic heterogeneity in databases due to "domain evolution", as it occurs in both single- and multidatabase systems. These problems occur when the semantics of values of a particular domain change over time in ways that are not amenable to applying simple mappings between "old" and "new" values. The paper also proposes facilities and strategies for solving such problems.*

1 Introduction

In current database systems most of the data semantics reside in the applications rather than in the DBMS. Moreover, data semantics are often not represented directly in the application code, but rather in the assumptions which the application--or, more correctly, the programmer--makes about the data. This situation is tolerated in local database environments largely because the local applications work with a shared set of assumptions. However, serious problems are likely to occur during a database integration--or federation [ShLa90]--effort because sets of local assumptions clash and local applications do not have access to the semantics represented in the "foreign" applications. This is the semantic heterogeneity problem. When semantic information that is hidden in applications is made explicit and accessible through the database then the semantic problem becomes a much more tractable syntactic problem [DeMi89, BrHu90]. Syntactic heterogeneity problems can be solved by modifying data to enforce homogeneity, or they can be dealt with explicitly in the applications.

This paper discusses "domain evolution", a source of semantic heterogeneity which can occur in single- as well as multidatabase systems. The term refers to changes in the meanings of the real-world counterparts of domain values that may cause a domain to become an aggregate of semantically incompatible sub-domains. For example, a domain "location" that formerly contained room numbers may come to include building numbers as well. As with other types of schema change, old data may become unprocessible by some new applications, or new data unprocessible by old applications. Worse, the result may be subtle incompatibilities so that "old" and "new" values cannot be sensibly combined or compared, and results that span them cannot be correctly interpreted.

We argue that domain evolution can create problems of semantic heterogeneity *within* a database similar to those encountered in multidatabase systems and that similar solutions are required. We describe the problem in terms of examples. We then propose needed facilities and identify some evolving technologies which might be expected to provide components of solutions.

2 The Problem

The following paragraphs present examples of different forms of domain evolution:

1. *Heterogeneous Instances:* Over time, different occurrences of the same value in a domain extension may have different meanings. For example, organizations sometimes merge or split departments, representing a reshuffling of resources (employees, offices, etc.) for which "department" serves as a convenient shorthand. The effect is that certain applications, e.g., historical reporting, statistical analysis and very long "transactions" such as amortization and depreciation may require changes to deal with "old" and "new" occurrences. Consider depreciation on an asset with a useful life of 10 years purchased for department 'V78'. If three years later V78 is split into two departments (say, new V78 and V79) and its resources are partitioned, who gets charged for the depreciation? To allow an application to address this problem, the database must store additional information about departments and assets. For example, it could define a department as a group of *employees* with *resources* and an *organization*. If versions of this description were synchronized (e.g., time-stamped) with the data indexed by department, and if comparison operators were available, it would then become possible for an application to discover that the department domain had changed.

2. *Cardinality Changes:* Cardinality relationships between domains may also change over time. For example, a 1-to-n relationship between departments and projects may become m-to-n as a result of an organizational change (i.e., projects now span departments). After the change, updates must pass different constraints and applications programs must deal with sets of departments for a project. Certain canned operations--such as joining departments and projects--may no longer work.

3. *Granularity Changes:* Values may be added to a domain extension that represent a different granularity

---

The views expressed in this paper are those of the authors and do not reflect the policies or positions of the MITRE Corporation or GTE Laboratories.

from the existing population. For example, an attribute called "location" may originally have been set up to store room numbers, reflecting an organization convention. Later, perhaps due to the acquisition of a new division, the meaning of "location" is extended to include building numbers. The two types of domain value may look similar--building 'A99' versus room 'C110'--but they clearly represent a different granularity. Queries against "location" (e.g., "average number of phones per location") may return results that mix apples and oranges--without any indication that this is happening.

4. *Encoding Changes:* Database values often have encoded meanings. These may be relics of predecessor manual systems or they may creep into systems over time, possibly to store information that is not otherwise provided for in the existing system. For example, "location" may come to include values that have special meanings, such as "status" (i.e., 'X00' means "away for repairs" or 'X99' means "lost"). These are not locations in the original sense. As another example, consider a company working on projects under government contract that is barred from charging employee vacation time to those projects. The database may include a "dummy" project, number '0000', to provide a code for charging employee vacation time. Database queries and reports retrieving "all projects" will return the often misleading dummy vacation project. Indeed, by some measures such as "total labor hours", the vacation project may appear to be the largest--and most important!--in the company.

5. *Time and Unit Differences:* Database values that users wish to compare may be incompatible due to differences in time or units of measurement. Stored calculations in the same domain may, over time, be the products of different formulae. Units of measure may change so that values are similar but not identical, such as imperial gallons versus U.S. gallons. Currency units also change as a result of devaluations; figures stored before and after such changes will no longer be comparable. "Snapshot" values, such as inventories, trade balances and monetary reserves, may not be comparable if the times of measurement during the business cycle are different. Finally, a firm that formerly calculated revenues on a calendar-month basis may switch to a per-week basis--no simple aggregation of weekly figures will make them comparable to the old monthly figures for trend analysis, etc.

6. *Identifier Changes:* In response to changing needs, indexing strategies may change over time, leading to parallel and even overlapping identifier schemes. For example, an organization may originally identify items of property in its computerized systems using numbers traditionally maintained by the Property Department. A new coding scheme may appear later, perhaps using longer numbers with a different format (e.g., bar codes). Later still, a third identifier may appear--for example, to meet an application need for a unique, immutable identifier. The result of this evolution is a "property system" with two or more overlapping index schemes. Queries and applications retrieving new and old property items must use multiple indices. This may force the organization to maintain parallel identifiers until all of the "old" property has retired from the system.

7. *Field Recycling:* In many systems it is difficult or infeasible to alter certain characteristics of the database. Perhaps record sizes cannot be altered because of application or system software dependencies. Changing the names of fields may involve reloading the database or recompiling hundreds of software modules. The response in many cases to this inflexibility is to recycle an existing field so that the new use may have different semantics from the old one. For example, a company may switch from a hierarchical to a matrix organization. As a result, employees are assigned job titles, reflecting function, to replace the old job level codes. Instead of adding a job title field, the decision is made to recycle the job level field, using title abbreviations to make them syntactically consistent with the old scheme. Thus, in place of job level values like 'EX6' and 'NE9'--examples of "exempt" and "nonexempt" job levels--job title values such as 'PA9', for "programmer analyst", appear. Applications that key off the recycled field may produce incorrect results. Database retrievals that return old job codes and new job titles will mingle unlike values.

Each of these examples illustrates two types of problem: those that must be solved by the application--how should depreciation be assigned for equipment that is transferred among departments? what kind of summary data on numbers of telephones per location make sense?--and those that must be solved by restructuring the database to include more semantic information to support the applications--when did the equipment belong to V78? is the reference to a room or to a building? In a sense, each change in semantics produces a distinct version of the database and a corresponding version of the affected applications that reflects an understanding of the domain semantics at a particular point in time.

Domain evolution can introduce semantic heterogeneity within a database that further complicates the problem of developing multidatabase systems. When equivalent domains from multiple

databases are combined the resulting heterogeneity in their union is the *same problem* as in the single database case. The result is that the multidatabase system must now deal with semantic problems both among and *within* component databases.

3 Components of Solutions

We believe that the following facilities are needed to address semantic heterogeneity in either single- or multidatabase systems:

Representation of Semantics: *the ability to capture the semantics of the domain*
Domain values require metadata to describe their semantics. The descriptions may need to include, for example, time of measurement, accuracy, source, and derivation formula. Different aspects of values' semantics may best be served by different representations [McCa82]: text, program code, rules [SiMa91], constraint languages [UrDe88], tags or footnotes, as well as other data in the database that provide context for the data of interest (e.g., the interpretation of attribute "A" may depend, in part, on the values of attributes "B" and "C"). The choice of representation must balance expressive power against readability, since the user or application--or the database system itself--needs to be able to interpret the semantic description.

Semantic data models provide means of capturing some domain metadata, including entity associations, cardinality, existence dependencies, constraints and user-names. However, while these models are commonly used in database design, the information is not explicitly represented in the resulting database and so is not accessible to applications, queries or users. Likewise, data dictionaries often capture relevant domain descriptions. But given the current lack of integration between data dictionaries and databases the metadata is not usable by the database system or applications, nor is it accessible to the user in combination with the data.

Data/Metadata Synchronization: *the ability to associate appropriate semantic information with specific database values*
Synchronization is required between the domain members and the metadata. As the meaning of a domain changes over time, the associated metadata must also evolve and remain coupled to the data. For metadata stored in the database, "triggers" can provide a way to link attributes in order to return context to the user along with the requested data. Current work on database schema evolution and versioning provides some synchronization mechanisms, particularly for object-oriented databases [BKKK87, SkZd87]. However, schema versions address only type changes that apply to *all* members of the domain extension, beginning at some point in time. More flexible mechanisms, such as those provided by footnoting schemes, are needed to deal with metadata that apply to arbitrary subsets of domain values.

Metadata Comparison: *the ability to detect and express differences in domain semantics*
Applications and queries must be able to determine that a set of values includes heterogeneous members of a domain and to specify the nature of the incompatibility. Some representations for metadata will provide comparison operators. For example, [SiMa91] presents a rule-based representation that detects differences based on comparing rules, and some constraint languages allow constraint specifications to be compared. Textual representations, such as footnotes and program code, are more expressive than constraint languages or rules. However, they are often unsatisfactory for determining and representing semantic differences because they usually base comparisons on string matching. Even "tags" usually base equivalence on string matches and provide no way to represent differences.

Metadata Generation: *the ability to create semantic information for derived data*
Derived data also require domain information to describe the semantics of particular values or the results of particular computations. The required metadata representations are similar to those for stored data, but the metadata values must be *generated* during computation of the data values. For example, accuracy tags can be derived for results of computations over values which, themselves, are associated with accuracy specifications. Other values might be annotated with the time of computation. Units can be derived and associated with computed results. Metadata for derived information may be derived from metadata associated with the base values of the derivation. For example, the results of computations over values that have been annotated as "estimated" might, themselves, be annotated as "estimated".

Relevance Evaluation: *the ability to determine when particular semantic differences affect the results of a query or application*
Not all differences in semantics among members of a set of values are relevant to all queries or applications, depending on the nature of the query or the processing performed by the application. In general, it must be possible to determine the particular *types* of metadata differences to which a particular query or application is sensitive. For example, the fact that data retrieved for department V78 include values related to that department before

and after the division of the department will be relevant to computations of depreciation, but not to determining the number of departments.

For some queries or applications it may be possible to perform mappings to produce homogeneity within their intermediate results, similar to the way in which the "dynamic attributes" of [LiAb86] are produced. For example, a mapping for the split department, V78, could produce a homogeneous set of values for depreciation expense spanning the periods before and after the split by dividing the expense values for the post-split period in half.

4 Solution Strategies

In general, the solution to problems of semantic heterogeneity is to make semantic information explicit so that it can be read and interpreted by the code. This would replace problems of *semantic* heterogeneity by more tractable problems of *syntactic* heterogeneity. The code could then associate appropriate semantic information with data values rather than *assume* that all values have uniform semantics. For example, databases that store similar information in different units can be augmented with a "units" field; assignments of equipment to departments can be time-stamped; encoded values can be recognized syntactically and dealt with appropriately by the code (e.g., locations that start with 'X' should be omitted from "real" locations).

The goals of many multidatabase systems are to make heterogeneity among components *transparent* to users and applications and to avoid requiring changes to the code of the underlying systems. In contrast, the goal of facilities to deal with semantic heterogeneity *within* a domain is to make it possible for applications and users to determine when accessed values do not match their assumptions, and so avoid presenting misleading information or prevent program failure. The heterogeneity is transparent only to users and applications unaffected by it. Further, it is probably not possible to avoid requiring code changes--only to minimize them, since the nature of the heterogeneities cannot be anticipated. If it were possible to predict the new values that would result from evolution, the domain would probably have been defined to include them from the beginning.

The amount and complexity of required code change is, of course, affected by the chosen architecture of the system. If metadata storage and management have been embedded in the applications, then the latter must be modified as new values and different types of metadata are introduced. Alternatively, solutions that include a metadatabase that applications access will reduce the number and complexity of changes to the applications for a more modular result. The applications become metadata-independent, though they must still include knowledge of the metadatabase protocol. And in the multidatabase case it will still be necessary to integrate the (possibly heterogeneous) metadata to make "foreign" metadata accessible to local users and applications. (Schema integration strategies, such as the methods based on attribute relationships proposed by [ShGa89] and [LSE89], provide mechanisms to help achieve this.) Finally, if the metadata are encapsulated in *objects*, the changes will be transparent to users of those objects. However, the object classes may still require modification to deal with new values in evolving domains.

Capturing and interpreting metadata to detect domain changes and resulting internal heterogeneities also has performance implications. A precise strategy to reduce the performance penalty of testing for domain changes and heterogeneous values will depend upon the anticipated usage patterns of the database in question--as with other types of optimization. The situation may benefit from the use of alerters that indicate when domain changes have occurred and the code needs to be modified, or when retrieved values do not satisfy application assumptions so that users can take appropriate steps to deal with them.

5 Conclusions

Domain evolution introduces into single database systems problems of semantic heterogeneity that are similar to those that complicate multidatabase systems. We have argued that the most effective way to deal with these problems is to transform them, where possible, into syntactic problems by making the semantics explicit in the data and applications. The latter can be addressed by incorporating more descriptive data--metadata--in the database: constraints, cardinality relationships, units, derivation algorithms and formulae, confidence measures, and heuristics. These will not solve the semantic heterogeneity problem. But they provide applications developers and users with the means to address the problem. And the metadata approach stores semantics in the database, instead of allowing them to continue to reside in constantly-changing applications code and the assumptions of their developers.

Acknowledgements

We thank Sara Haradhvala, Michael Siegel, and especially Frank Manola for their helpful reviews and insightful comments. We also thank Barbara Klain for editorial assistance.

References

[BKKK87] Banerjee J., Kim W., Kim H-J, Korth H.F. "Semantics and Implementation of Schema Evolution in Object-Oriented Databases", *Proc. SIGMOD Conf.*, San Francisco, CA, 1987.

[BrHu90] Bright M.W., Hurson A.R. "Summary Schemas in Multidatabase Systems", Computer Engineering Technical Report TR-90-076, Penn State Univ., Univ. Park, PA, 1990.

[DeMi89] DeMichiel L.G. "Resolving Database Incompatibility", *IEEE Transactions on Knowledge & Data Engineering*, 1:4 (Dec. 1989).

[LiAb86] Litwin W., Abdellatif A. "Multidatabase Interoperability", *IEEE Computer* (Dec. 1986).

[LSE89] Larson J.A., Navathe S.B., Elmasri R. "A Theory of Attribute Equivalence in Databases with Application to Schema Integration", *IEEE Transactions on Software Engineering*, 15:4 (Apr. 1989).

[McCa82] McCarthy J.L. "Metadata Management for Large Statistical Databases", *Proc. 8th VLDB Conf.*, Mexico City, Mexico, 1982.

[ShGa89] Sheth A.P., Gala S.K. "Attribute Relationships: An Impediment in Automating Schema Integration", *Workshop on Heterogeneous Database Systems*, Chicago, IL, Dec. 11-13, 1989.

[ShLa90] Sheth A.P., Larson J.A. "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys*, 22:3 (Sept. 1990).

[SiMa91] Siegel M., Madnick S. "A Metadata Approach to Resolving Semantic Conflicts", *Proc. 17th VLDB Conference*, Barcelona, Spain, 1991.

[SkZd87] Skarra A.H., Zdonik S.B. "Type Evolution in an Object-Oriented Database", *Research in Object-Oriented Databases*, B. Shriver, P. Wegner, (eds.), Addison-Wesley, 1987.

[UrDe88] Urban S.D., Delcambre L.M.L. "Constraint Analysis: A Tool for Explaining the Semantics of Complex Objects", in *Lecture Notes in Computer Science #334*, G. Goos, J. Hartmanis, (eds.), Berlin, Germany, Springer-Verlag, 1988.