Facilitating Database Attribute Domain Evolution Using Mesodata

Denise de Vries and John F. Roddick

School of Informatics and Engineering
Flinders University of South Australia
PO Box 2100, Adelaide, South Australia 5001
{Denise.deVries,roddick}@infoeng.flinders.edu.au

Abstract. Database evolution can be considered a combination of schema evolution, in which the structure evolves with the addition and deletion of attributes and relations, together with domain evolution in which an attribute's specification, semantics and/or range of allowable values changes. We present a model in which mesodata – an additional domain definition layer containing domain structure and intelligence – is used to alleviate and in some cases obviate the need for data conversion or coercion. We present the nature and use of mesodata as it affects domain evolution, such as when a domain changes, when the semantics of a domain alter and when the attribute's specification is modified.

1 Introduction

The way we view and deal with information evolves. In paper-based manual systems this evolution did not present a great problem – we turned the page and ruled it up differently, renamed columns, used different terminology and proceeded to store our information. We could always review what had been stored historically by viewing the information exactly as it had been recorded. The static nature of this method means that notations that were recorded retained their semantics in context, that is the headings and layout of the form/paper imparted the structures and conventions as well as the values themselves. We, the human, translated and transformed the information when we retrieved it. It was simple. It was also so time consuming that much of what we now consider to be basic tasks, such as sorting, aggregating, summarising and reporting was infeasible.

The development of RDBMS and automated systems put the layout and *form* into the unchanging metadata and gave us *record once* systems. Database technology has provided the power to store and manipulate information in a variety of ways, however we still cannot reproduce the simplicity of dealing with information evolution as we used to. A major introduced problem that has not yet been completely solved is that of attribute domain evolution. For example, if one were searching for a particular value, time consuming though it was, subtle differences in data values were captured because the searcher understood the domain and therefore included or excluded records based on their own knowledge of the domain. For instance, a database query searching through historical medical records for an illness matching 'rubella' generally uses a string comparison only, thus the string 'german measles' would not be retrieved even though semantically it matched.

There have been many techniques developed to deal with database evolution but none can currently deal with all aspects of evolution and few of them deal specifically with the problem of attribute domain evolution. Middleware, using various approaches, has been used to alleviate evolution problems by translating, transforming or coercing data and metadata. In this work we use a new approach that introduces complex data structures with embedded intelligence that lie between metadata and data – *mesodata* [1]. Mesodata allows domains to be engineered so that attributes can be defined to possess additional intelligence and structure and thus reflect more accurately ontological considerations, including changes in the domain itself.

Attribute domain evolution refers to the evolution of the valid range of values that a database attribute (field) may store and the semantics they infer. For example, an integer field of 4 bytes can store values in the range of -2,147,483,648 to 2,147,483,647 whereas a float field of 4 bytes has a range of negative values from -3.402823E+38 to -1.401298E-45 and positive values from 1.401298E-45 to 3.402823E+38. The domain has changed even though the storage requirement has not altered. Domain evolution can be broadly categorised into three types;

Attribute Representation Change: expansion or contraction of field, for example, `CHAR(15)` to `CHAR(20)` or vice versa, change of base type: integer to float, numeric to character, character to enumerated list.

Domain Constraint Change: the possible range of values that may be recorded has changed without the metadata changing or the currently stored data changing, for example, the minima and/or maxima change. The new constraints may, or may not, be applied retrospectively.

Perception (Meaning) Change: the semantics of the data change, for example, Reference 116Q15 no longer is interpreted as 'Burbridge Road' and is now 'Sir Donald Bradman Drive', however, both interpretations are required for historical purposes.

Currently when a schema changes two events typically occur - the application is modified and recompiled to deal with the changes and the data is converted to the new format, either by *strict*, *lazy* or *no* conversion [2]. Lazy conversion performs data conversion only when data are accessed and they are still recorded with superseded formats (or values), no conversion is done if the data are not accessed. The advantage of this approach is that only the data that are used are converted and the whole database does not need to be locked or taken off-line to perform the conversion. The disadvantages are that a record of schema changes must be recorded and accessible and that every time data is accessed it must be checked to see if it conforms to the current schema. Until all data have been accessed there exist some that are invalid, incomplete or uncertain.

Strict conversion requires that as soon as there is a modification to the schema all data are converted to conform with the current definition. The advantage of this approach are that all data are consistent with the new schema. The disadvantage is that all applications interacting with the database must be stopped and the database locked while the conversion takes place. Depending upon the nature of the modifications this can take a long time. In addition, information is lost and changes cannot be reversed.

By adding a mesodata layer to the structure, some of the data would not need to be converted and the application itself may not need to change. Mesodata can store semantics as well as operators and operations in data structures other than base types.

2 Related Work

The primary goal for schema evolution in databases is to preserve the integrity of the data. Sjøberg [3] observed several reasons for schema change. These included that:

– people do not know in advance, or are not able to express, all the desired functionality of a large-scale application system. Only experience from using the system will enable the needs and requirements to be properly formulated.
– the application world is continually changing. A viable application system must be enhanced to accommodate these changes.
– often the scale of the task requires incremental design, construction and commissioning. This results in requirements to change the installed subsystems.

Sjøberg's case study, a health management system, revealed that schema changes were significant both during the six months of development and the twelve months after the system was operational. In the study, the changes covered the gamut of possibilities including each relation being changed, 139% increase in the number of relations, 274% increase in the number of fields, and 35% more additions than deletions. During the development phase (5 months) there were 65 changes (additions and deletions) to relations and 470 changes (additions and deletions) to attributes. During the operational phase (11 months) the corresponding numbers were 299 and 2324 respectively. In Sjøberg's study changes to the type/domain of an attribute was not captured in the statistics, however, in the last month of the study the changes to fields were 18 renamings, 4 changes of unique/non-nulls, 23 changes of length and 4 changes of representation. That is 31 changes at attribute level.

A consensus glossary [4] provides the definition that a database supports *schema evolution* if it allows modification to the schema without loss of extant data and that no support for previous schemas is required, whereas it supports *schema versioning* if it allows the querying of all data, both retrospectively and prospectively, through user-definable version interfaces. Roddick *et al.* [5] present a taxonomy of schema versioning issues with respect to the Entity-Relationship Model and the effects on the relational database model. The work discussed in this paper does not deal with all issues of schema evolution and versioning, but concentrates on the specifics of domain/attribute evolution. The pertinent evolutionary operations are:

– Expanding an attribute domain
– Restricting an attribute domain
– Changing the domain of an attribute

There has not been a great deal of research done in the field of evolving relational databases over the past few years, however work done in object-oriented database evolution, data warehousing, data integration and schema integration research has areas of relevance to the evolution of domains.

2.1 Information Capacity

Hull [6] and later Qian [7] provided formal approaches to evaluate the *information capacity* of schemata. The four relative information capacity measures between database structures as defined by Hull are, in progressively less restrictive order, calculus dominance, generic dominance, internal dominance and absolute dominance. These measures are used to evaluate the information capacity of two or more schemata by mathematically mapping between the schemata. An important point to note is that even when two schemas can be proved to have the same information capacity, it does not then follow that they are equivalent semantically. Qian's formalisation of Abstract Data Types (ADT) for schema transformations presents a slightly different notion of 'information preservation', which is strictly less restrictive than calculus dominance, strictly more restrictive than absolute dominance and incomparable to generic and internal dominance. These formal approaches are the foundation of later work into *schema equivalence* and *schema integration*.

Miller [8] describes *Equivalence* as the requirement that all data stored in one schema (S_1) can be accessed and updated through another schema (S_2),

– for queries the transformation function (f) must be total: $q(i_2) = q(f(i_1))$;
– to access all data f must be injective: i_1 must correspond to a unique i_2, a 1-1 cardinality;
– for updates f must be onto: $I(S_2)$;
– for equivalence ($S_1 \equiv S_2$) there exists a bijective function: $f : I(S_1) \rightarrow I(S_2)$;

and *Dominance* as $S_1 \preceq S_2$ allows all data stored under schema S_1 to be queried through S_2,

– to access all data f must be injective: 1-1 cardinality;
– every instance of S_1 can be transformed to an instance of S_2 without loss of information;
– S_2 may hold more information.

Miller *et al.* [9, 8, 10] point out that whilst equivalent information capacity is a required condition, it is not sufficient to guarantee a natural correspondence between schemas and in practice database administrators rely on their own intuition when defining *transformations* between schemas. Schemas, in practice, contain constraints that define which instances of a schema are meaningful in a certain context. Their research in this area shows that deciding information capacity equivalence and dominance of schemas is an undecidable problem. As a result, they developed tests to evaluate equivalence and dominance more restrictively. These tests utilise a set of *schema transformations* that declare that Schema S_1 is dominated by schema S_2 if and only if there is a sequence of transformations that converts S_1 to S_2. These transformations use Schema Intension Graphs (SIG) data models to aid in understanding the relative information capacity of schemas containing constraints. The authors have developed algorithms for deciding equivalence of schemas with constraints. The SIG model must be data-centric rather than type-centric in order to reason about constraints on collections of entities rather than the internal structure of a single entity. (This approach ignores the problem of data type changes and the conflicts type changes present.) They define the Schema Translation problem as follows:

> Given two schemas one needs to know with respect to information capacity if each instance of the first schema can be represented as an instance in the second schema and whether the translation can be reversed?

Table 1 identifies possible conflicts that can occur between semantically equivalent schemas with regard to attributes and values. Capacity and equivalence, therefore, is not sufficient, data integration must considered to take into account query and view requirements.

2.2 Schema Integration

Xu and Poulovassilis [11], when addressing the integration of deductive databases, considered both the extensional and intensional parts of the component databases for integration. The Common Data Model (CDM) uses a binary relational Entity-Relationship model with subtyping to integrate the extensional parts. The authors proposed a semi-automatic method which requires only the declaration of the relationships between schema constructs to perform the integration. For the purposes of their model, they defined a database as the quintuple:

```
        < Schema, Extensional Database, Intensional Database,
          Constraint Database, Procedural Database >
```

Each of these sets is integrated in turn and in that order. The schemas and the extensional databases are represented by directed graphs and from those graphs correspondences between nodes are declared. These mappings are then used to perform the integration into a CDM. The intensional database is integrated by integrating the rules from the component databases according to their denotational semantics employing a method of comparing the semantics of the rules.

Table 1. Schematic Conflicts between Attributes and Values

	Value	**Attribute**
Value	Domain conflicts between schema S_1 and schema S_2, such as expression conflicts, data unit conflicts, precision conflicts	The values in S_1 are used as attributes in S_2
Attribute		Different definitions for semantically equivalent attributes – 1-1 one attribute used to model the same information in each schema, such as naming conflict, integrity constraint conflict, data representation conflict. – 1-N and N-N different numbers of attributes used to model the same information in each schema. – The N-N conflict is a generalisation of the 1-N conflict

2.3 Schema Transformation

Transformation consists of the tasks schema conforming, schema merging and schema restructuring. McBrien *et al.* [12] present a formal framework again using the CDM for ER schema transformation in which they have defined a set of primitive transformations based on schema equivalence. This is achieved by formalising a database instance as a set of sets containing entity type names, subtypes, attribute names and associations. These are also integrated in order, however, in this work there is a distinction made between transformations which require knowledge of the instances in the database and those that do not.

Continuing this work [13] and combining schema integration and schema evolution activities, the authors propose using a Hypergraph Data Model (HDM) to build a global schema from heterogeneous source schemata and from this transformations may be used to translate queries between the global and source schemas. Schema transformations defined on the HDM are reversible. Every *add* transformation step is reversed by a *del* transformation with the same parameters, *renaming* (ren) transformations from $S_1 \rightarrow S_2$ are the reverse of $S_2 \rightarrow S_1$. Contract transformations map to void, queries and sub-queries over such constructs then translate to void. *Extend* transformations requires domain knowledge, either from a human expert or a domain ontology and cannot be automated. Higher level modelling only works with names, tables, relations but not at attribute data type level.

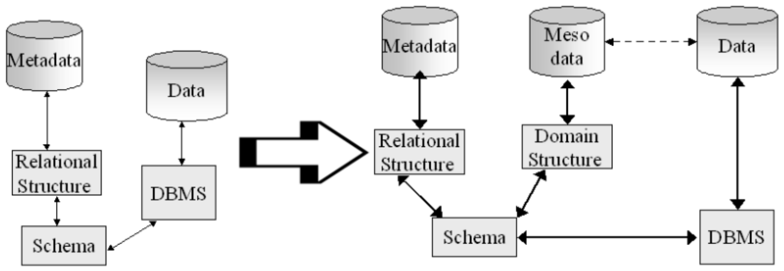

Fig. 1. Mesodata is an additional layer in RDBMS

Davidson *et al.* [14] recognising that information capacity preserving transformations do not necessarily preserve the semantics of databases developed the declarative language *WOL* (Well-founded Object Language) for expressing database transformations and constraints. They argue that approaches which allow a fixed set of well defined transformations to be applied in series (for most methodologies the outcome is dependent on the order in which the schemas are integrated - they are not associative) are inherently limited in the class of transformations that can be expressed and that while using a high-level language for transformations is necessary for general transformations, it is difficult to reason about, and prove, properties of transformations. This work tackles the difficulty of correctly transforming complex data structures (sets, records and variants) and recursive structures. Constraints on the source and target databases are crucial to notions of information preservation, but typically are not, or cannot, be expressed in the models of the underlying databases.

3 Model

In this research, we use the term 'intelligent domains' for our enhanced mesodata generated domains as they provide increased semantic content over the domains. Mesodata is an additional domain definition layer containing structure and operators.

A traditional relational database can be viewed as consisting of relations that are a subset of the Cartesian product of their attributes' domains [15].

$$R \subseteq (dom(A_i) \times dom(A_j) \times \ldots dom(A_n)) \qquad (1)$$

where R is the relation
A is an attribute
dom is the domain of the attribute A.

The mesodata layer extracts the domain to a separate level such that the Mesodata Domain (Mdom) is the domain of the mesodatatype of the basetype, for example a weighted graph of strings or a list of graphs of strings. Mdom is defined as:

```
Mdom :: dom(attribute) | dom(mesodata)
dom(mesodata):: dom(mesodatatype(dom(mesodata)) |
                 dom(mesodatatype(dom(attribute))
dom(mesodatatype) :: dom(wgraph)|dom(wdgraph)|
                     dom(list)|dom(clist)|...
                     any mesodata structure
dom(attribute):: dom(basetype)
basetype::  all valid database base types.
```

The redefinition of the relation R thus becomes

$$R \subseteq (Mdom_i \times Mdom_j \times \ldots \times Mdom_n) \tag{2}$$

that is, the cartesian product of the mesodata defined domains.

Table 2. Examples of domain structures suitable for adoption as mesodata types

Domain Structure	Operations (Extended SQL Op.)	Source Relation(s)
Unweighted Graph (GRAPH)	Adjacency (NEXTTO)	Binary relation (FROM, TO)
Weighted Graph (WGRAPH)	Adjacency, Proximity (CLOSETO)	Ternary relation (FROM, TO, WEIGHT)
Directed Graph (DGRAPH)	Adjacency	Binary relation (FROM, TO)
Directed Weighted Graph (DWGRAPH)	Adjacency, Proximity	Ternary relation (FROM, TO, WEIGHT)
Tree (TREE)	In subtree (DESCENDENT), Parent(PARENT), Ancestor(ANCESTOR), Child(CHILD), Sibling(SIBLING)	Binary Relation(PARENT, CHILD)
Weighted Tree (WTREE)	In subtree, Parent, Ancestor, Sibling, Proximity	Ternary Relation(PARENT, CHILD, WEIGHT)
List (LIST)	Next (NEXT), Previous (PREV), First(FIRST), Last(LAST), Between(BETWEEN)	Binary Relation(SEQUENCE, ITEM)
Circular List (CLIST)	Next, Previous, Between	Binary Relation(SEQUENCE, ITEM)
Set (SET)	In Set(INSET)	Unary Relation(ITEM)
Tri-State Logical	Maybe Equal (MAYBE)	None

Table 2 of mesodata types presents a few examples of domain structures with their intrinsic operations. It is important to differentiate between *mesodata* structures and Abstract Data Types (ADT). For instance, the specification of a graph as a mesodata type and the specification of a graph as an abstract data type (ADT). In the former, the attribute would take as its value an instance of a base type that exists *within a graph* while in the latter the type is a graph for which code must be included in the application. The mesodata type is not directly accessible through the attribute. The semantics of information (S) held in a database can be considered as a function of the data value. That is,

$$S = F(v) \tag{3}$$

where F is a mapping external to the database which maps the data value (such as 116Q15) to an understood concept (such as Burbridge Road). The introduction of a mesodata layer allows regularly used mappings to be accommodated in the database, ie.

$$S = F(M(v)) \tag{4}$$

where M is the mesodata layer mapping. Conceptually, it is then possible to have mappings of mappings

$$S = F(M_1(M_2(\ldots M_k(v)\ldots)) \tag{5}$$

where M_i are mesodata layer mappings. Therefore, $S = F(M_1(M_2(116Q15)))$ where $F(M_2(116Q15))$ = 'Burbridge Road' and $F(M_1$('Burbridge Road') = 'Sir Donald Bradman Drive'. For more information about using mesodata types in DBMS refer to [1].

4 Domain Evolution

Domain evolution can be broadly categorised into three types - *Attribute Representation Change*, *Domain Constraints Change* and *Domain Perception (meaning) Change* - that to date require a manual solution to their successful incorporation into a DBMS. Using a mesodata layer in the database can reduce these problems. We illustrate these through examples. The mesodata types selected for the examples are not proscriptive, just as the DBA judges which attribute data type to use, so too must the decision of which mesodata type to employ lie with the DBA.

4.1 Attribute Representation Change

Example: A character code is replaced by a number code. The specification CHAR(20) is altered to an INTEGER.

Current Typical Solution: Add new attribute of type INTEGER to relation, convert old data values to new values and store in new attribute, delete old attribute, rename attribute to old name, update application to handle different type.

Mesodata Solution: Use the mesodata type, LIST, that maps the existing CHAR(20) values to the new INTEGER values. The attribute in the relation remains unchanged as does the application, as the operators to access the changed attribute type are built into the mesodata type.
For example:
`old AppCode = 'widgetA'`
`new AppCode = 2131`
using the mesodata domain layer, we have, (see Eq 4),
`AppCode = Mdom('widgetA') = 2131`

Both code values **2131** and **widgetA** are accessible and valid. Information capacity holds as both *equivalence* and *dominance* requirements are met. It is recognised that not all attribute type changes can be handled using mesodata, for example from BLOB to INT, however there are many instances where the evolutionary process can be alleviated.

4.2 Domain Constraints Change

Example: 'Country of birth' is an attribute contained in a number of databases, the allowable values of which have changed significantly during the twentieth century. When a country name changes it may be a one-to-one change, such as *Rhodesia* to *Zimbabwe*, or one-to-many change, for example *Yugoslavia* to {*Bosnia Herzegovina, Croatia, Macedonia, Serbia, FYR Montenegro, Slovenia.*}

Current Typical Solution: Convert all old values and replace them with new values. This could be an ongoing task and it results in loss of information.

Mesodata Solution: Utilise the mesodata types WGRAPH or TREE to map old values to the new values. The domain of 'countries' includes *All* country names, current and superseded, which are then accessible to the DBMS with the extended SQL operators and original values are not lost.

4.3 Domain Perception (Meaning) Change

Example: A perception change may entail an absolute change where there is new interpretation or it may be the addition of synonyms. The days of the week stored numerically from 1 to 7 inclusive may interpret the value '1' as 'Monday', equally valid are the interpretations 'lunes', 'lundi', 'maandag', 'Montag', 'segunda-feira' and so forth.

Current Typical Solution: The application may be parameter driven to select a single preferred interpretation (such as language setting) or the users must learn the dominant term.

Mesodata Solution: A mesodata layer allows regularly used mappings to be accommodated in a database (see Eq. 5). Therefore we have
`1 = 'Monday' = 'lundi'` etc.

Mesodata helps to reduce potential systems changes to one of two simpler solutions

1. A change to the schema definition that requires no change to either the application or data.
2. A change to the mesodata reference relation with or without a change to the schema but again, without the need to change either the application or the data.

Schema integration and transformation is not required as the mesodata type has the operators and 'intelligence' to replace these tasks. Information capacity is not only maintained, as both requirements of equivalence and dominance are met, but also in many cases expanded as the cartesian product of the mesodata domains is greater than the original domain of the relation.

Our implementation of this model uses MySQL [16] software with wrappers to transform our extended SQL, as described in [1]. Space precludes a more detailed report of the implementation which will be the subject of a future paper.

5 Conclusion and Further Research

Though an attribute change in itself may not be a complex process it is not a trivial task. Database evolution and maintenance consists of many such simple steps as shown in [3] most of which also necessitate changes to application code and system down time. The mesodata layer, an additional domain definition layer containing domain structure and intelligence, provides the means to manage some aspects of attribute domain evolution. We have shown that its use when a domain changes, when the semantics of a domain alter or when the attribute's specification is modified can reduce or remove the necessity of schema conversion, schema integration, data conversion and application change as well as maintain or expand the schema's information capacity.

Work in this area is progressing, particularly in the use of ontology frameworks to describe and incorporate evolving domains into the mesodata layer.

References

1. De Vries, D., Rice, S., Roddick, J.F.: In support of mesodata in database management systems. In: 15th International Conference on Database and Expert Systems Applications DEXA 2004. Lecture Notes in Computer Science, Zaragoza, Spain (2004)
2. Ferrandina, F., Meyer, T., Zicari, R.: Implementing lazy database updates for an object database system. In: Twentieth International Conference on Very Large Databases, Santiago, Chile (1994) 261–272
3. Sjøberg, D.: Quantifying schema evolution. Information and Technology Software **35** (1993) 35 – 44
4. Jensen, C.S., Clifford, J., Elmasri, R., Gadia, S.K., Hayes, P., Jajodia, S., Dyreson, C., Grandi, F., Kafer, W., Kline, N., Lorentzos, N., Mitsopoulos, Y., Montanari, A., Nonen, D., Peressi, E., Pernici, B., Roddick, J.F., Sarda, N.L., Scalas, M.R., Segev, A., Snodgrass, R.T., Soo, M.D., Tansel, A., Tiberio, P., Wiederhold, G.: A consensus glossary of temporal database concepts - february 1998 version. In Etzion, O., Jajodia, S., Sripada, S., eds.: Temporal Databases - Research and Practice LNCS. Volume 1399. Springer-Verlag (1998) 367–405
5. Roddick, J.F., Craske, N.G., Richards, T.J.: A taxonomy for schema versioning based on the relational and entity relational models. In: Proc. Twelfth International Conference on Entity-Relationship Approach. (1993) 143–154
6. Hull, R.: Relative information capacity of simple relational database schemata. Society for Industrial and Applied Mathematics **15 No 3** (1986) 856 – 886
7. Qian, X.: Correct schema transformations. In Apers, P.M.G., Bouzeghoub, M., Gardarin, G., eds.: Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology. Volume 1057 of Lecture Notes in Computer Science., Avignon, France, Springer (1996) 114–128
8. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: Schema equivalence in heterogeneous systems: Bridging theory and practice. Information Systems **19** (1994) 3 31
9. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: The use of information capacity in schema integration and translation. In Agrawal, R., Baker, S., Bell, D., eds.: Nineteenth International Conference on Very Large Data Bases, VLDB'93, Dublin, Ireland, Morgan Kaufmann, Palo Alto, CA (1993) 120–133
10. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: Schema intension graphs: A formal model for the study of schema equivalence. Technical report, University of Wisconsin-Madison (1994)

11. Xu, L., Poulovassilis, A.: A method for integrating deductive databases. In: British National Conference on Databases. (1997) 215–231
12. McBrien, P., Poulovassilis, A.: A formal framework for er schema transformation. In: International Conference on Conceptual Modeling / the Entity Relationship Approach. (1997) 408–421
13. McBrien, P., Poulovassilis, A.: Schema evolution in heterogeneous database architectures, a schema transformation approach. In: CAiSE'02, Birkbeck College and Imperial College (2002)
14. Davidson, S., Buneman, P., Kosky, A.: Semantics of Database Transformations. Volume 1358. Springer-Verlag, Berlin (1998)
15. Elmasri, R., Navathe, S.B.: Fundamentals of database systems. 3rd edn. Addison-Wesley, Reading, Mass ; Menlo Park, Calif. (2000)
16. MySQL: Sql shareware software: documentation and source code (2003)