Defining and Implementing Domains with Multiple Types using Mesodata Modelling Techniques

Sally Rice[1,2], **John F. Roddick**[1] **and Denise de Vries**[1]

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

[2] School of Computer and Information Science
University of South Australia
Mawson Lakes, South Australia 5095
sally.rice@unisa.edu.au

Abstract

The integration of data from different sources often leads to the adoption of schemata that entail a loss of information in respect of one or more of the data sets being combined. The coercion of data to conform to the type of the unified attribute is one of the major reasons for this information loss. We argue that for maximal information retention it would be useful to be able to define attributes over domains capable of accommodating *multiple types*, that is, domains that potentially allow an attribute to take its values from more than one base type.

Mesodata is a concept that provides an intermediate conceptual layer between the definition of a relational structure and that of attribute definition to aid the specification of complex domain structures within the database. Mesodata modelling techniques involve the use of data types and operations for common data structures defined in the mesodata layer to facilitate accurate modelling of complex data domains, so that any commonality between similar domains used for different purposes can be exploited.

This paper shows how the mesodata concept can be extended to facilitate the creation of domains defined over multiple base types, and also allow the same set of base values to be used for domains with different semantics. Using an example domain containing values representing three different types of incomplete knowledge about the data item (coarse granularity, vague terms, or intervals) we show how operations and data structures for types already existing within the mesodata can simplify the task of developing a new *intelligent domain*.

Keywords: Mesodata, intelligent domains, multiply-typed domains, incomplete information, vagueness, coarse granularity, intervals, relational model, hierarchical domains, data integration.

1 Introduction

Integrating multiple sources of data is of vital importance to many enterprises. Within a single organization such integration can lead to better strategic planning and decision making. Integration across multiple organizations leads to more efficiency and quality of service due to better utilization of information and elimination of redundancy (Zeng 1999).

Integration of sources with different conceptual schemas is not a trivial process. The need to combine attributes defined on different types often leads to loss of information. For example, consider something as simple as a numerical attribute recorded in one source using integers and in another as fixed point values with two decimal places. These representations could be combined by converting the latter into integers, which clearly involves a loss of accuracy in the decimal data. But perhaps the most likely solution when combining these values is to convert the integers to numbers with two decimal places. This may not appear to involve any information loss, but reporting a value like 24 as 24.00 gives it the appearance of an accuracy that it does not have: in fact, we have lost information about the accuracy of the value. A much better solution is to allow the attribute to take values from both types. This concept we call *multiply-typed domains*.

A problem with the former SQL standard (commonly known as SQL-2) is its lack of built-in support for the creation of user-defined complex data types. By this we do not mean merely the ability to create simple domains using the `CREATE DOMAIN` command as it is defined in the SQL standard (see e.g. (Melton & Simon 2002)), but the ability to create domains with complex structure and semantics. This has led to commercial relational database management systems (RDBMS) adding this facility in an ad hoc way as a response to user demands, as, for example, Oracle has done with its extensive `CREATE TYPE` facility (Lorentz & Gregoire 2003). These extensions can be an awkward fit with the concepts of the relational model. Mesodata provides a method for users to implement domains of arbitrary complexity that fits well with the relational model, and there is no reason why the new features offered in SQL:1999, such as the ability to create complex user-defined types, can't be used in the implementation of these domains.

The concept of mesodata – a middle layer of domain definition sitting between the metadata and the data – allows the separation of the definition of the structure from the semantics of a domain. Commonly used data structures (such as lists, graphs and trees) and operations for the manipulation of these data structures are defined in the mesodata layer. These *mesodata types* can then be used to build specific attribute domains: a mesodata type is used to define the structural aspects of the domain, and the domain is then built by populating it with values, and by implementing any additional semantics through additional operations which can utilise the structural operations

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Tasmania, Australia. Conferences in Research and Practice in Information Technology, Vol. 53. Sven Hartmann, Markus Stumptner and Yasushi Kiyoki, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

of the mesodata type.

In the past, the incorporation of additional semantics in attribute domains has generally been accomplished by extending data models such as the relational model for particular types of data. This type of extension includes developments such as temporal databases (Snodgrass 1995), spatial databases (Schneider 1997, Egenhofer & Franzosa 1991) and probabilistic databases (Dey & Sarkar 1996). The advantages that the mesodata approach has over the development of a special-purpose extended data model for each case include making it easier to combine separate extensions into a single data model (as has been necessary for the development of spatio-temporal databases, for example (Abraham & Roddick 1999)), and the ability for reuse of the same data structures with different semantics. We believe that the potential for reuse facilitates the creation of intelligent domains.

Earlier papers on the mesodata concept have argued for the incorporation of mesodata in Database Management Systems, and have shown how domain evolution can be facilitated using mesodata (de Vries, Rice & Roddick 2004, de Vries & Roddick 2004). This paper further defines the mesodata concept by discussing how attributes can be defined over *multiply-typed domains* – domains capable of accommodating multiple types – and showing how the same set of base values can be used for domains with different semantics. We illustrate multiply-typed domains using an example with three types, two whose base values form hierarchies semantically, and one whose base values are numeric intervals. The example we present is of an attribute defined over a multiply-typed domain within a single relation. However, the methods discussed are relevant to the definition of a common schema to be used across different data sources.

The rest of this paper is organised in the following way. Section 2 presents a conceptual model for mesodata and some examples of basic mesodata types, Section 3 introduces multiply-typed domains through an example involving incomplete data, Section 4 discusses how we used these mesodata types to implement this domain as an exemplar of the use of mesodata techniques, and Section 5 provides a conclusion to this paper.

We have tried to be consistent in our use of the terms *domain* and *type* in this paper. In our usage we intend type to refer to the *format* of the data, and domain to the broader concept of *allowable values*. However sometimes we are constrained to use one or the other due to things outside of our control, such as SQL syntax.

2 Mesodata

Mesodata is a concept that facilitates the implementation of structurally and semantically-rich domains (*intelligent domains*). Key features include a special *mesodata layer* within which structural aspects are defined for common structures such as graphs and trees, and the ability to accommodate domain variability by mapping between different representations (for example, between names and three-byte RGB values for colours). An intelligent domain is built by matching a mesodata type with a *base type* and a *source relation* to hold the specific structural information for the domain. The base type could form a simple domain (such as `INTEGER` or `CHAR(12)`), or it could in turn be a domain based on a mesodata type, so we can define graphs of trees, for example. The important difference between mesodata techniques and object-oriented concepts, is that the former introduces the idea of storing complex *domain values* in the database. Object-oriented databases are concerned with complex *attribute values.*

This section describes a data model for mesodata. We first give a conceptual model and show how it can be incorporated into a relational database, then we present two mesodata types used in the development of our example intelligent domain.

2.1 Conceptual model

Figure 1 shows an entity-relationship model for mesodata, using UML notation as in e.g. (Connolly & Begg 2005).

Domain represents a multiply-typed domain for an *Attribute.* It is composed of one or more *Types*. *Type* is completely specialised into either a *Simple* type (such as `INTEGER` or `CHAR(12)`) or a *Complex* type (one built using mesodata types).

A *Complex* type is described by a *Mesodata* type, which has a structure *SRstructure* and a set of *Operations.* Structural details of the *Complex* type are stored in its source relation *SourceRel*, and its base values have a *Domain.* For example, for a graph whose nodes were strings of length 12, the graph's structure would be described in its source relation, and the base domain of its nodes would be `CHAR(12)`. Note that the base *Domain* can itself be *Complex* (to accommodate graphs of lists, for example), and a *SourceRel* may be used for more than one *Complex* type.

A *Mapping* shows how to convert a value from one *Type* of a multiply-typed domain to another. Each *Mapping* has a type *MapType* which can use a function or a lookup table (or a combination of the two) to convert the values.

The implementation of this conceptual model can be separated into four parts:

1. The entities *Mesodata*, *SRstructure* and *Operations* describing the mesodata types form the layer between the metadata and the data described earlier.

2. The entities *Domain*, *Type*, *Mapping* and *MapType* are implemented as tables belonging to a super-user (such as `SYS` in Oracle), which are protected from direct manipulation by the user.

3. User tables are created as normal, except that attributes with multiply-typed domains must specify their data type as well as their value.

4. There are some hidden tables automatically created when the complex types are defined, i.e. the source relations and look up tables for mappings. These tables should also not be directly manipulable by the user.

2.2 SQL extensions for multiply-typed domains

For illustrative purposes, we offer the following extensions to the syntax of SQL data definition commands. This consists of extensions to `CREATE DOMAIN` and `CREATE TABLE`, and a new command `CREATE MAPPING`.

To allow the same mapping to be used to map more than one type without redefinition, the definition of a mapping is divided into two parts:

- associating a mapping name with a look-up table and/or a mapping function, and

- associating a mapping name with a `FROM` type and a `TO` type.

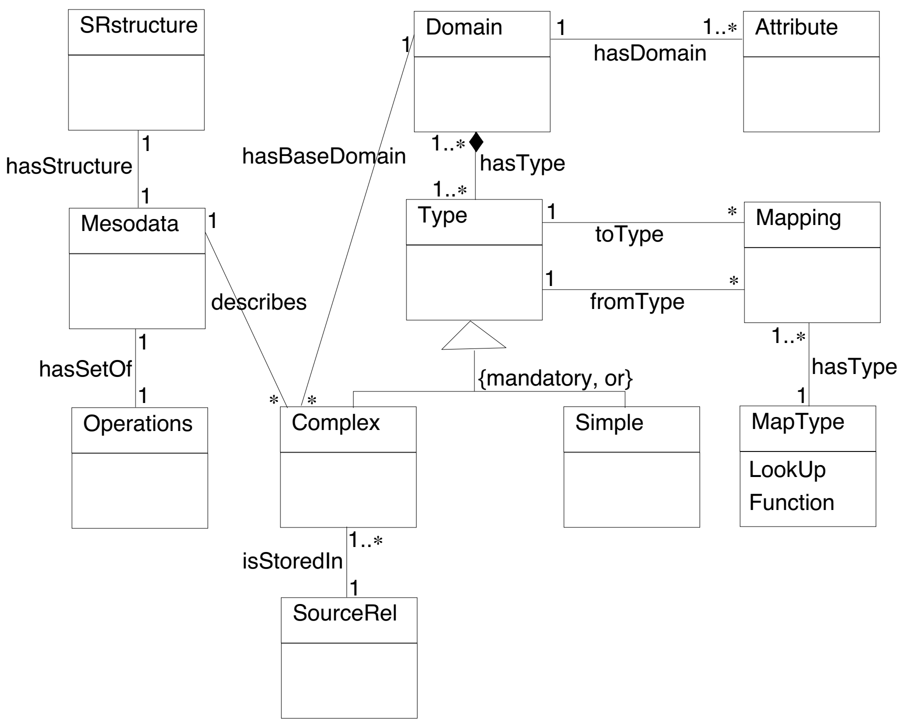

Figure 1: Conceptual model for mesodata

The CREATE MAPPING command associates a mapping name with a look-up table and/or a mapping function.

```
CREATE MAPPING mapping_name
   [LOOKUP mapping_rel]
   [FUNCTION function_name]
```

A MAPPING clause has been added to the extended CREATE DOMAIN command originally defined in (de Vries, Rice & Roddick 2004) to accommodate mesodata types. The MAPPING clause defines mappings from the base values associated with a particular CREATE DOMAIN command to another type. Because a type may be mapped in more than one way, there may be more than one MAPPING clause. The to_type can be either a simple or a complex type.

```
CREATE DOMAIN dom
   AS mesodatatype
   OF basedom
   (RETURNS returndom)
   OVER sourcerel {(attribute {,attribute})}
   {MAPPING mapping_name TO to_type}
   [EXCLUSIVE | NONEXCLUSIVE]
```

And finally, we create the multiply-typed domain by allowing a domain for an attribute in CREATE TABLE[1] command to be a set of possible types.

```
[attribute_domain | (type {, type})]
```

2.3 Hierarchy and interval mesodata types

For our example, we use hierarchy and interval types defined in the mesodata layer. We use the term hierarchy for the mesodata type rather than tree because we do not restrict nodes to having a single parent. We used the term *lattice* to describe our hierarchies in earlier work (Rice & Roddick 2000), but because they can include overlapping nodes, it is possible for two nodes not to have a unique least upper bound, so the term lattice is not general enough. Our structures are not as general as a DAG (directed acyclic graph), because they do have a defined *top* node ($\top$) and *bottom* node ($\bot$). The directedness of our hierarchy is also different from that of a DAG: you can traverse its edges in both directions, but there is a semantic difference between going towards $\bot$ (more specific) and towards $\top$ (more general).

The source relation for domains based on interval has the schema (DESCRIP, START, FINISH) with key DESCRIP. It is used to associate the end-points of the interval with the stored string used to represent it in the attribute, and is required when the RDBMS does not have a base interval type. START and FINISH can have any numeric or date/time type: the underlying logic is the same whether the intervals are numeric or temporal. Operations defined on intervals include the Allen relations (Allen 1983):

- $equals(i_1, i_2)$
- $before(i_1, i_2)$ / $after(i_2, i_1)$
- $starts(i_1, i_2)$ / $startedBy(i_2, i_1)$
- $during(i_1, i_2)$ / $contains(i_2, i_1)$
- $finishes(i_1, i_2)$ / $finishedBy(i_2, i_1)$
- $meets(i_1, i_2)$ / $metBy(i_2, i_1)$
- $overlaps(i_1, i_2)$ / $overlappedBy(i_2, i_1)$

In addition, we use $startOf(i_1)$ and $finishOf(i_1)$ to retrieve the end-points of interval i_1.

The source relation for domains based on hierarchy describes the structure of the hierarchy: it has the schema (CHILD, PARENT) which has a key consisting of both attributes because a CHILD can have more than one PARENT. Required operations include predicates $childOf$, $parentOf$, $descendentOf$, $ancestorOf$ and $inFamily$, and set-valued functions $allDescendents$, $allAncestors$ and $family$. Let SR be the source relation, then we define:

[1]The choice we have made of extending the CREATE DOMAIN command then allowing an attribute to take values from more than one of these domains is possibly unfortunate, as it implies that an attribute can be drawn from different domains, rather than that a domain can be formed from values of different types.

- $childOf(x,y)$ is *true* if $(x,y) \in SR$.
- $parentOf(x,y)$ is *true* if $(y,x) \in SR$.
- $descendentOf(x,y)$ is *true* if $childOf(x,y) \vee (\exists(z) \wedge childOf(x,z) \wedge descendentOf(z,y))$.
- $ancestorOf(x,y)$ is *true* if $parentOf(x,y) \vee (\exists(z) \wedge parentOf(x,z) \wedge ancestorOf(z,y))$.
- $inFamily(x,y)$ is *true* if $x = y \vee descendentOf(x,y) \vee ancestorOf(x,y)$.
- $allDescendents(x)$ is the set $\{y|descendentOf(x,y)\}$.
- $allAncestors(x)$ is the set $\{y|ancestorOf(x,y)\}$.
- $family(x)$ is the set $\{y|inFamily(x,y)\}$.

These operations are part of the mesodata type and do not need to be coded by the user.

3 Multiply-typed domains

As an example of a multiply-typed domain, we introduce a data model for incomplete data where that incompleteness can be of three different base types (i.e. values with variable granularity, values that are vague, or where an interval is used to represent a single value). The same queries can be posed over attribute values of any of these types: the difference between them lies in the data structures used for each type and the corresponding operations used to answer the queries. Our data model uses a third truth value *unknown* and *hierarchical domains* to cope with the partial knowledge. In other work (Rice & Roddick 2000) we discuss an earlier version of this data model.

In this section we give an overview of this intelligent domain through an example based on archaeological data. In addition to the multiple types, we show how attributes with different semantics can use the same base values. This requires an implementation that allows the same stored domain values to be used with multiple semantics.

3.1 The example domain

We use in our example two relations KILNS shown in Table 1 and POTS shown in Table 2 containing information about pots and pottery kilns in Roman Britain (Swan 1984). The date values given for the attributes `inOperation` in KILNS and `dateCreated` in POTS are expressed in three different forms: using the names of Emperors, as an interval of years, or using vague terms such as mid I (i.e. in the middle of the first century). They reflect the terms used by the archaeologists who conducted the initial research over a period of nearly 200 years. All three forms use values which explicitly or implicitly define a range of values, but the semantics of the domain for each attribute are different – whereas `inOperation` defines the interval during which the kiln is believed to have been in operation, `dateCreated` uses the same values to represent not an interval, but an unknown point of time somewhere in that interval. The sorts of questions that archaeologists would like to answer about the pottery industry using these data include:

- What kilns were operating during the reign of Nero?
- Were the Hardingstone 1 and Binsted 15 kilns operating concurrently?
- Which kilns could have manufactured this pot?

To answer questions like these for the data shown, it is necessary to be able to map between the different representations used, which could be done by translating all dates into numerical intervals on data entry. However, this can lead to loss of information: if Claudian is translated into $[41, 54]$, it loses the historical context of the original estimate, and if early I is changed to, say, $[1, 30]$ the vagueness inherent in the original form disappears.

3.2 Hierarchical Domains

By hierarchical domains we mean domains where there is a hierarchical structure between (at least two of) the elements of the domain. The examples $D_1 =$ {Claudio-Neronian, pre-Flavian, Flavian, Claudian, Neronian}, $D_2 =$ {[50,100], [30,60], [70,100], [50,60]} and $D_3 =$ {early–mid I, mid–late I, early I, mid I, late I} whose structures are shown in Figure 2 demonstrate this. Connections between nodes in the hierarchy represent a *containment relationship* – the lower of two connected nodes is (at least partially) contained within the upper node. We call the upper node of two connected nodes the *concept* and the lower node the *element*, following the usage introduced in (Roddick 1994). Any node in a hierarchy with both a child and a parent can be either a concept or an element, depending on which connection is being considered. An unlabelled top node $\top$ is shown for each hierarchy, which by definition completely contains every node in the hierarchy. Nodes further down the hierarchy are more specific. For simplicity, the bottom node of the hierarchy $\perp$ (which represents the empty set ϕ) is not shown. In the domains shown in Figure 2, the node mid I demonstrates the multiple-parent structure, because it belongs to (is contained in) both early-mid I and mid-late I.

There are three kinds of containment relationships shown in Figure 2 by the labels N, O and S. N is the 'normal' containment relationship where the element (lower node) is completely contained within the larger concept (upper node), O is the overlapping relationship where each node overlaps the other, and S is the relationship between two synonyms. For example, pre-Flavian and Claudio-Neronian are synonyms, and early-mid I and mid-late I are overlapping. The hierarchical structure has been retained in the presence of O and S containments by choosing one of the pair of concepts involved to be lower in the hierarchy than the other.[2] In both cases the choice is arbitrary, because each of the connected nodes contains the other, either partially (O containment) or completely (S containment). Although N, O and S containments can apply to sets in general, the domain elements in Figure 2 are all intervals, even if the bounds of the interval are not obvious (hierarchy (a)), or not precisely known (hierarchy (c)). Of course, the containment relationship between two numerically-expressed intervals such as $[50, 60]$ and $[50, 100]$ can be worked out directly from their end-points, without recourse to stored hierarchical information.

3.3 Queries

The development of a new type of intelligent domain usually entails extensions to query languages. In our example, we introduce some new syntax to SQL to cope with queries involving the attributes `dateCreated` and `inOperation`. For `inOperation`

[2]Note that the O containments shown are an extension of the usual meaning of hierarchy. It is perhaps better to describe O and S links as sibling links rather than parent-child ones. Containment relationships were added to this data model to reduce the complexity of algorithms for querying this data.

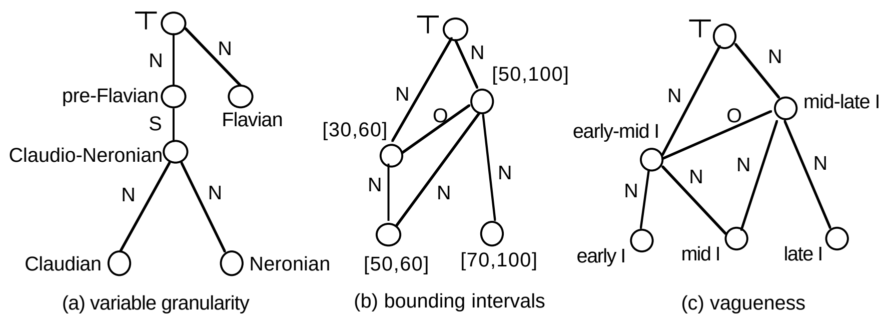

Figure 2: Domains showing connection types

Table 1: Relation KILNS

kiln	easting	northing	inOperation
Dates of varying granularity			
Hardingstone 1	476	257	pre-Flavian
Colchester 15	599	226	Claudio-Neronian
Chichester	486	105	Claudian
Stoke-on-Trent	387	343	Neronian
Biddlesden	464	240	Flavian
Dates as numerical intervals			
Oxford 5	455	206	$[50, 100]$
Binsted 15	477	141	$[30, 60]$
Hendon 1	517	194	$[50, 60]$
Dorchester 2	458	194	$[70, 100]$
Dates using vague terms			
Colchester 11	599	226	early–mid I
Little Houghton 5	481	260	mid–late I
Harrold	493	255	mid I
Kettering 1	489	278	late I

this syntax is based on Allen's logic for temporal intervals (Allen 1983). For example

```
inOperation CONTAINS 'Claudian'
```

The different semantics of `dateCreated` mean that conditions involving it are not comparing two intervals, however, but comparing a point *somewhere* in an interval with another interval. The Allen operations, as modified in (Vilain 1982) to compare a point with an interval, can be used here, but they need to be adapted to deal with uncertainty in the value of the point. Consider the condition

```
dateCreated DURING 'Claudian'
```

applied to a relation consisting of pots 1, 2 and 3 (see Table 2). This is *true* for pot 2, *false* for pot 3, and *unknown* for pot 1 (since Claudio-Neronian represents an interval that includes some dates that are Claudian and some that are not). On the other hand, there is no uncertainty about the condition

```
dateCreated DURING 'Claudio-Neronian'
```

which is *false* for pot 3, and *true* for the other pots. In hierarchies that only have N and S containment relationships, the condition `dateCreated DURING` v is *true* for all pots with a date that is a descendent of v in the hierarchy, but is *unknown* for all pots with a date that is an ancestor of v that is not synonymous with v where *unknown* is a third truth value between *true* and $false$ that indicates there is insufficient information to decide whether a condition evaluates to *true* or *false*. Truth tables for the operators $\wedge$ (and), $\vee$ (or) and $\neg$ (not) are shown in Table 3.. The situation is more complex in hierarchies which include O containments: the truth value of the condition is also *unknown* for all descendents of v in the hierarchy to which there is no path that does not include an O containment.

In order to accommodate the *unknown* truth value, we introduce the keyword `MAYBE` as shown in these queries.

```
SELECT * FROM pots
WHERE dateCreated DURING 'Claudian';

SELECT * FROM pots
WHERE MAYBE dateCreated DURING 'Claudian';

SELECT * FROM pots
WHERE MAYBE dateCreated
      NOT DURING 'Claudian';
```

`MAYBE` indicates we want to retrieve kilns for which the query condition is *true* or *unknown*. To answer these queries, we need to return all kilns where the condition `dateCreated DURING 'Claudian'` is *true*, *true* $\vee$ *unknown*, and *unknown* $\vee$ $false$ respectively.

Table 2: Relation POTS

potID	description	easting	northing	dateCreated
	Dates of varying granularity			
1	Belgic grey ware cooking pot	599	226	Claudio-Neronian
2	poppy-head beaker	486	105	Claudian
3	ring-necked flagon	464	240	Flavian
	Dates as numerical intervals			
4	Clapham shelly ware bowl	517	194	$[50, 60]$
5	large storage jar	458	194	$[70, 100]$
	Dates using vague terms			
6	narrow-mouthed bowl	493	255	mid I
7	lid-seated jar and lid	489	278	late I

Table 3: 3-valued logic truth tables

C	$\neg C$	$C_a \vee C_b$	t	u	f	$C_a \wedge C_b$	t	u	f
t	f	t	t	t	t	t	t	u	f
u	u	u	t	u	u	u	u	u	f
f	t	f	t	u	f	f	f	f	f

4 Implementing the domains

We have shown that, while the attributes `inOperation` and `dateCreated` both use the three domain hierarchies shown in Figure 2, their semantics differ. Different semantics for the same domain values can be accommodated by defining two domains on the same source relation, each with their own set of operations.

In this section, we develop different mesodata types for each of the three types of data for both these semantics, then create multiply-typed domains for each attribute using the mesodata types. It is a multi-layered approach, that allows re-use of the stored domain values.

The mesodata types we use in our example are:

`interval`: The `interval` type described above.

`typedHierarchy`: The `hierarchy` type described above extended to include an attribute specifying the containment relationship for each edge in the hierarchy.

`pointInterval`: The `interval` type where the interval is used to represent an unknown point somewhere within the interval.

`pointTypedHierarchy`: The `typedHierarchy` type where the nodes of the hierarchy represent points within an interval similar to the pointInterval type.

4.1 The extended mesodata types

First let us consider the source relations for the three hierarchies. The intervals shown in part (b) of Figure 2 can use the source relation schema described for the `interval` mesodata type in the previous section. For the other two hierarchies, we extend the schema for the source relation for `hierarchy` mesodata type to include the containment relationship between `CHILD` and `PARENT` to be `(CHILD, PARENT, CONTAINMENT)`.

Now consider the attribute `inOperation`. As discussed above, the attribute values are intervals, and query conditions use the Allen operations. For the interval hierarchy these need no adaptation (apart from perhaps interpreting `DURING` to also include `EQUALS`, `STARTS` and `FINISHES`). For the other two hierarchies though, these operations must be redefined using the structure of the hierarchy and the containment relationships of its edges. For example, consider the query condition `inOperation DURING` d. For the hierarchical domains, we can work out whether the query condition is *true* or $false$ for a value v using these definitions:

- `SYNONYMS`$(d, v) = true$ if v is a synonym of d
- `N_PATH`$(d, v) = true$ if there is a normal path from d to v
- `FULLDESCENDENT`(d, v) = `DESCENDENTOF`$(d, v) \wedge$ `N_PATH`(d, v)
- `DURING`(d, v) = `SYNONYMS`$(d, v) \vee$ `FULLDESCENDENT`(d, v)

By a normal path we mean a path that does not contain an O containment.

The attribute `dateCreated`, however, represents a point located somewhere within the value used for the attribute, and query conditions use operations comparing a point with an interval, the keyword `MAYBE`, and three-valued logic as discussed earlier. For example, consider the query condition `dateCreated DURING` d. For the interval hierarchy we can use these definitions to determine the truth value of this condition for a value v:

- *true* = `EQUALS`$(d, v) \vee$ `CONTAINS`$(d, v) \vee$ `STARTS`$(d, v) \vee$ `FINISHES`(d, v)
- *unknown* = `DURING`$(d, v) \vee$ `OVERLAPS`$(d, v) \vee$ `OVERLAPPEDBY`$(d, v) \vee$ `STARTEDBY`$(d, v) \vee$ `FINISHEDBY`(d, v)

Table 4: Source relations

Source Relation intervalsrel		
desc	start	finish
[50,60]	50	60
[70,100]	70	100
[30,60]	30	60
[50,100]	50	100
Source Relation emperorsrel		
child	parent	containment
Claudian	Claudio-Neronian	N
Neronian	Claudio-Neronian	N
Claudio-Neronian	pre-Flavian	S
Source Relation vaguerel		
child	parent	containment
early I	early-mid I	N
mid I	early-mid I	N
mid I	mid-late I	N
late I	mid-late I	N
early-mid I	mid-late I	O

- $false = \texttt{BEFORE}(d,v) \vee \texttt{AFTER}(d,v) \vee \texttt{MEETS}(d,v) \vee \texttt{METBY}(d,v)$

For the other two hierarchies, once again we need to consider domain structure and containments. We can determine the truth value of the condition for value v using these definitions:

- $true = \texttt{SYNONYMS}(d,v) \vee \texttt{FULLDESCENDENT}(d,v)$
- $unknown = \texttt{INFAMILY}(d,v) \wedge \neg(\texttt{SYNONYMS}(d,v) \vee \texttt{FULLDESCENDENT}(d,v))$
- $false = \neg\texttt{INFAMILY}(d,v)$

The operations not defined in this section are defined for the base mesodata types in Section 2. The difference in capitalisation is not meant to be significant.

4.2 Creating Mappings, Domains and Tables

To define the mappings between the source relations, we use the `CREATE MAPPING` command described earlier with a lookup table and a mapping function whose purposes are discussed below. The mappings required for our example are from Emperor names to intervals, and from vague terms to intervals.

```
CREATE MAPPING emperorMap
       LOOKUP emperorIntervals
       FUNCTION emperorToIntervals

CREATE MAPPING vagueMap
       LOOKUP vagueIntervals
       FUNCTION vagueToInterval
```

To create the domains for `inOperation` and `dateCreated`, we use the `CREATE DOMAIN` command described earlier. The `AS` clause defines the mesodata type to be used, the `OF` clause defines the base data type for the nodes in the hierarchy, and the `OVER` clause specifies the source relation to use.

```
CREATE DOMAIN inOpInterval
       AS interval OF CHAR(10)
       OVER intervalsrel

CREATE DOMAIN dateCrInterval
       AS pointInterval OF CHAR(10)
       OVER intervalsrel

CREATE DOMAIN inOpEmperor
       AS typedHierarchy OF CHAR(16)
       OVER emperorsrel
       MAPPING emperorMap TO inOpInterval

CREATE DOMAIN dateCrEmperor
       AS pointTypedHierarchy OF CHAR(16)
       OVER emperorsrel
       MAPPING emperorMap TO dateCrInterval

CREATE DOMAIN inOpVague
       AS typedHierarchy OF CHAR(12)
       OVER vaguerel
       MAPPING vagueMap TO inOpInterval

CREATE DOMAIN dateCrVague
       AS pointTypedHierarchy OF CHAR(12)
       OVER vaguerel
       MAPPING vagueMap TO dateCrInterval
```

We now use these domains to create tables KILNS and POTS. The syntax used for the domains of `inOperation` and `dateCreated` shows that values from each of the three domains listed can be used for that attribute.

```
CREATE TABLE kilns (
       kiln CHAR(20) PRIMARY KEY,
       easting INTEGER,
       northing INTEGER,
       inOperation (inOpInterval,
                    inOpEmperor,
                    inOpVague))

CREATE TABLE pots (
       potID INTEGER PRIMARY KEY,
       description CHAR(30),
       easting INTEGER,
       northing INTEGER,
       dateCreated (dateCrInterval,
                    dateCrEmperor,
                    dateCrVague))
```

Table 5: Mapping between Hierarchies and Intervals

Mapping Emperors to Intervals		
emperor	start	finish
Claudian	41	54
Neronian	54	68
Flavian	68	96
Claudio-Neronian	startOf(Claudian)	finishOf(Neronian)
pre-Flavian	startOf(Claudio-Neronian)	finishOf(Claudio-Neronian)

Mapping Vague Terms to Intervals				
vagueTerm	start	finish	startMin	finishMax
early I	1	33	1	startOf(mid I)
mid I	34	66	finishOf(early I)	startOf(late I)
late I	67	100	finishOf(mid I)	100
early-mid I	startOf(early I)	finishOf(mid I)		
mid-late I	startOf(mid I)	finishOf(late I)		

The source relations describing the hierarchical structure of the domains are shown in Table 4. It is not necessary to include any edges connecting nodes to $\top$, as these always have N containment.

Table 5 shows how to map the two hierarchical structures to intervals. This means providing start and finish values for the Emperor's reigns and the vague terms. In the case of the vague terms, minimum and maximum values have been given as well as default start and finish values to allow the default values to be varied if desired, but only in such a way that the relationships between the terms remain consistent. Wherever possible, startOf and finishOf operations are included in the table to reduce redundancy. The lookup table and mapping function defined in the `CREATE MAPPING` command are used to implement this mapping. The lookup table contains the numerical values in Table 5, and the mapping function is used to calculate the values using the *startOf* and *finishOf* operations.

4.3 Discussion of the Implementation

The implementation described in this paper comprises four distinct tasks:

Implementing the basic mesodata types. The mesodata approach is new, and the only mesodata types implemented previously are graph, weighted graph, and circular list, so the hierarchy and interval mesodata types must be implemented. This task would not normally be part of the development of a database using a mesodata type.

Implementing the adapted mesodata types. The typedHierarchy, pointInterval and pointTypedHierarchy implementations are based on those of interval and hierarchy. Once created, these are reusable mesodata types like any other, but the creation of a new intelligent domain may require adaptations of existing mesodata types like these.

Implementing the intelligent domain. This task involves the implementation of the mapping functions and different semantics for the query operations for the attributes inOperation and dateCreated.

Implementing the database This task incorporates the creation and population of the specific domains, mappings and relations.

These tasks are shown in decreasing order of likelihood of being required for a particular application.

The non-standard SQL syntax is handled by wrappers which perform any required mesodata operations and convert the various data definition and manipulation commands to standard SQL, as mesodata is still at the proof-of-concept stage.

In concept, a mesodata-style implementation of an intelligent domain is three-layered. The implemented algorithms are built using the operations defined for the mesodata types used in the domain, which in turn use the operations for the DBMS's base data types. In comparison, a direct implementation is two-layered: its special-purpose data structures and algorithms are built directly on top of the base data type, possibly enhancing its efficiency at the expense of re-use. As an experiment, our example domain is being implemented using both methods to see how they differ in ease of implementation, and efficiency of operation.

For illustrative purposes, consider the mesodata implementation of the query[3]

```
SELECT * FROM KILNS
WHERE inOperation DURING 'pre-Flavian'
```

for the `inOpEmperor` part of the multiply-typed `inOperation` domain. This requires identification of the tuples in the KILNS table with a value for `inOperation` which lies entirely within the pre-Flavian era. The algorithm `CALC_DURING` returns the set *DVduring* of domain values which satisfy this condition (in our example these are `'Claudian'`, `'Claudio-Neronian'` and `'pre-Flavian'`) using as input d (`'pre-Flavian'`, the domain value being matched) and *DVall* (the set of `inOpEmperor` domain values used in KILNS). `CALC_DURING` uses the `SYNONYMS` and `FULLDESCENDENT` operations described in Section 4.1.

Algorithm `CALC_DURING`(d, $DVall$)

```
BEGIN
  Initialise $DVduring$ to $\phi$
```

[3]It will be necessary to introduce syntax to determine which of the multiple types the value `'pre-Flavian'` belongs to, especially where there is more than one possible as is the case with our example.

```
    FOR (each $v \in DVall$)
      IF SYNONYMS$(d, v) \vee$ FULLDESCENDENT$(d, v)$
        Add $v$ to $DVduring$
      ENDIF
    ENDFOR
END
```

5 Conclusion and Further Research

Data integration often leads to compromise in the adoption of schemas that do not fit the data very well, in order to incorporate data from different sources into a single global schema. At the attribute level, this problem can be addressed by allowing attribute domains to accommodate multiple types. We have shown in this paper that the concept of mesodata can be used to define such domains.

We believe that the mesodata modelling methodology provides a handy tool for developing novel intelligent domains of all types. In particular, the ability to separate the domain values and structure from the semantics of attributes defined using the domain proved very useful for this data model, by providing a paradigm for thinking about the modelling process as well as enabling the reuse of the same set of complex values for attributes with different semantics.

Implementation of these ideas is already underway. The MySQL RDBMS (MySQL 2003) is being used for this purpose. We are implementing the same data model both with and without using mesodata techniques. Analysis of these algorithms so far shows no significant theoretical difference in complexity. Comparisons are being undertaken to determine whether there is a difference in practice.

6 Acknowledgements

The authors would like to thank the anonymous reviewers for their very helpful comments on our paper.

References

Abraham, T. & Roddick, J. F. (1999), ‘Survey of spatio-temporal databases’, *GeoInformatica* **3**(1), 61–99.

Allen, J. (1983), ‘Maintaining knowledge about temporal intervals’, *Communications of the ACM* **26**(11, November 1983), 832843.

Connolly, T. & Begg, C. (2005), *Database Systems: A Practical Approach to Design, Implementation and Management, 4th Edition*, Addison Wesley.

de Vries, D., Rice, S. & Roddick, J. F. (2004), In support of mesodata in database management systems, *in* ‘DEXA 2004’, Springer, Zaragoza, Spain.

de Vries, D. & Roddick, J. F. (2004), Facilitating database attribute domain evolution using mesodata, *in* F. Grandi, ed., ‘Third International Workshop on Evolution and Change in Data Management (ECDM2004)’, Lecture Notes in Computer Science, Springer, Shanghai.

Dey, D. & Sarkar, S. (1996), ‘A probabilistic relational model and algebra’, *ACM Transactions on Database Systems* **21**(3), 339369.

Egenhofer, M. J. & Franzosa, R. D. (1991), ‘Point-set topological spatial relations’, *International Journal for Geographical Information Systems* **5**(2), 161–174.

Lorentz, D. & Gregoire, J. (2003), *Oracle Database SQL Reference 10g Release 1 (10.1)*, Oracle Corporation.

Melton, J. & Simon, A. R. (2002), *SQL:1999 – Understanding Relational Language Components*, Morgan Kaufmann Publishers.

MySQL (2003), ‘SQL open source software’.

Rice, S. & Roddick, J. F. (2000), Lattice-structured domains, imperfect data and inductive queries, *in* M. Ibrahim, J. Kung & N. Revell, eds, ‘11th International Conference on Database and Expert Systems Applications, DEXA 2000’, Lecture Notes in Computer Science, Springer, London, pp. 664–674.

Roddick, J. (1994), A Model for Temporal Inductive Inference and Schema Evolution in Relational Database Systems, Doctor of philosophy, La Trobe University.

Schneider, M. (1997), *Spatial Data Types for Database Systems*, Vol. 1288 of *Lecture Notes in Computer Science*, Springer.

Snodgrass, R. T. (1995), *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers.

Swan, V. G. (1984), *The pottery kilns of Roman Britain*, Royal Commission for Historical Monuments.

Vilain, M. B. (1982), A system for reasoning about time, *in* ‘National Conference on Artificial Intelligence’, Pittsburg, PA, pp. 197–201.

Zeng, J. (1999), Research and practical experiences in the use of multiple data sources for enterprise-level planning and decision making: A literature review, Technical report, Center for Technology in Government, University at Albany / SUNY.