Conceptual Data Model with Structured Objects for Statistical Databases

Giuseppe De Giacomo

Dipartimento di Informatica e Sistemistica, Università di Roma
Via Salaria 113, 00198 Roma, Italy
degiacomo@dis.uniroma1.it

Paolo Naggar

Autorità per l'Informatica nella Pubblica Amministrazione (AIPA)
Via Po 14, 00198 Roma, Italy
paolo.naggar@aipa.it

Abstract

In this paper we present a conceptual data model, called $\mathcal{SDM}$, which is able to represent the relationships between elementary and statistical data at a conceptual level. $\mathcal{SDM}$ borrows elements from research both in Object Oriented Databases and in Knowledge Representation. In addition it has suitable mechanisms to form classes of individuals by classifying the instances of a target class according to some specified criteria. Notably, the class resulting from such a statistical aggregation can then be treated exactly as a class of elementary data. This ability fulfills the often perceived necessity, in modeling real domains, of treating statistical aggregates and elementary data in an homogeneous way.

1 Introduction

Typically statistical surveys are focused on directly obtaining some predefined statistical indexes (statistical variables). This setting comes from an organization of the work that promotes a new statistical survey each time the necessity of a new statistical product (a set of statistical indexes exhibiting certain phenomena) is sensed. However, this organization does not permit us to perceive possible correlations that exist at a conceptual level among the various pieces of information gathered by the different statistical surveys. The failure to perceive such correlations brings about costly anomalies in the process of acquisition and aggregation of elementary (non-statistical) data, as for example the repetition of more than one survey on a single unit of analysis.

Overcoming this is possible if a further element is introduced in the organization of the work: the statistical information system (related issues are discussed in [5, 19, 14, 16, 17, 12, 15, 3, 7, 8]). The statistical information system determines a decoupling between the phase where elementary data are gathered and the phase where aggregate data are used. Statistical surveys become the way a statistical information system is maintained, and thus they are set up independently of the contingent need of some statistical data. Instead, statistical data are obtained from a suitable view of the statistical information system without making reference to the statistical surveys that provided the elementary data to make the computation.

The statistical information system has to be provided with a specific language to represent its knowledge on the modeled reality, i.e. it needs a conceptual data model which is able to represent statistical aggregates – a conceptual statistical data model.

A conceptual statistical data model is a conceptual model that is able to represent the usual semantical relationships among elementary data. But in addition, it is able to represent the semantical relationships that exists between elementary data and the aggregate data derived from them by means of statistical classification.

Generally conceptual models bypass this kind of semantical relationships going directly to numerical data, which in fact should be attributes of the conceptual entities derived from the classification, entities that they are not able to represent explicitly. For example this happens when the Entity Relationship Model [9, 2] is used to model summary data [3]. Instead a conceptual statistical data model extends a conceptual model by allowing it to represent explicitly such statistical aggregates.

At the moment virtually all conceptual statistical data model proposals, e.g. [19, 16, 3, 7, 8], consider statistical aggregates as a new kind of objects. This induces a dichotomy in treating elementary data and data aggregated for statistical means, since for the former kind of data a rich variety of semantical relations is representable, while for the latter kind the only semantical relationship representable is the one that led to the aggregation. Our experience at AIPA has shown that such a dichotomy is often a strong limitation in the use of statistical data models for modeling a real domain.

In this paper we present a statistical data model, called $\mathcal{SDM}$, which, starting from elementary data, is able to recursively:

- represent in a conceptual schema the relationships among existing data;
- enlarge such conceptual schema by representing

explicitly the relationships between existing data and their aggregations for statistical purposes.

Such a statistical data model allows one to encapsulate information that was originally deeply hidden and spread throughout elementary data, in suitable summary data, while keeping track at the conceptual level of the derivation schema applied to make such information explicit.

$\mathcal{SDM}$ borrows elements from research both in Object Oriented Databases and in Knowledge Representation. In particular several notions developed in [10, 11, 6] were used as the base for our proposal. Prominent among these is the notion of "polymorphic object" introduced in [11, 6] – i.e. an object may be seen simultaneously as an individual object, as a tuple, and as a set.

$\mathcal{SDM}$ enhances the formalisms in [11, 6], by introducing suitable mechanisms to form classes that are statistical aggregations of other classes. These mechanisms where inspired by those found in [7, 8], which however induced the dichotomy mentioned above.

Specifically, $\mathcal{SDM}$ has a construct for aggregating more classes into a single class that has one instance for each class taking part in the aggregation. For example such a construct permits us to partition the class of integers between 0 and 100 in three intervals, $(0, \ldots, 10)$, $(11, \ldots, 50)$ and $(51, \ldots, 100)$, thus forming a new class with three instances, one for each interval.

A more sophisticated construct allows for the classification of instances of a class according to certain properties. For example we can aggregate people according to their sex and intervals of age.

Besides such constructs $\mathcal{SDM}$ provides one with the ability to refer to properties which are computed by means of suitable operators, such as COUNT, AVR, FREQ, etc. For example it allows for adding the property "average age" to the instances of the class formed by aggregating people according to their sex and intervals of age.

The rest of the paper is organized as follows. In Section 2 the basic conceptual model is presented and its main characteristics including the notion of polymorphic object are introduced. In Section 3 the constructs that denote statistical aggregations are added to the basic data model thus obtaining $\mathcal{SDM}$, and the power that the ability of repeatedly aggregating aggregates gives to $\mathcal{SDM}$, is illustrated. Finally, in Section 4, some concluding remarks end the paper.

2 The basic data model

In this section we formally define the object oriented data model $\mathcal{BDM}$ by specifying its syntax and its semantics.

2.1 Syntax

A $\mathcal{BDM}$ *schema* is a collection of class and view definitions over an alphabet $\mathcal{B}$, where $\mathcal{B}$ is partitioned into a set $\mathcal{C}$ of *class* symbols, a set $\mathcal{A}$ of *attribute* symbols (used in record structures), and a set $\mathcal{U}$ of *role* symbols (denoting binary relations over classes). We assume that $\mathcal{C}$ contains the special elements `Any` and `Empty`[1]. In the following C, A, and U range over elements of $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{U}$ respectively.

In defining classes and views we make use of complex links which are constructed starting from attributes and roles. An *atomic link*, for which we use the symbol l, is either an attribute, a role, or the special symbol `member` (used in the context of set structures). A *complex link* L is obtained from basic links according to:

$$L ::= b \mid L_1 \cup L_2 \mid L_1 \circ L_2 \mid L^* \mid L^- \mid \underline{\text{identity}}(C).$$

The construct $L_1 \circ L_2$ denotes the concatenation of link L_1 with link L_2, L^* denotes the concatenation of link L with itself an arbitrary finite number of times, L^- denotes the inverse of L (L taken in the reverse direction), and finally $\underline{\text{identity}}(C)$ denotes the identity role projected on C and is used to verify the occurrence of an instance of class C along a link.

Usually, in object oriented models every class has an associated type which specifies the structure of the value associated to an instance of the class. In $\mathcal{BDM}$, objects are not required to have a single structure. Instead, we allow for polymorphic objects, which can be viewed as having different structures corresponding to the different roles they can play in the reality modeled. Therefore we admit a rich set of expressions for defining structural properties. A *structure expression*, denoted with the symbol T, is constructed as follows:

$$\begin{aligned} T \quad ::= \quad & C \mid \neg T \mid T_1 \wedge T_2 \mid T_1 \vee T_2 \mid (o_1, \ldots, o_n) \mid \\ & [A_1 \colon T_1, \ldots, A_n \colon T_n] \mid \{T\}. \end{aligned}$$

The structure $(o_1, \ldots, o_n)$ represents the class whose instances are exactly $o_1, \ldots, o_n$. The structure $[A_1 \colon T_1, \ldots, A_n \colon T_n]$ represents all tuples which have at least $A_1, \ldots, A_n$ components having the structures $T_1, \ldots, T_n$, respectively. The structure $\{T\}$ represents sets of elements having structure T. Additionally, by means of $\wedge$, $\vee$, and $\neg$, we are allowed not only to include intersection and union in structure expressions (as in [1]), but also to refer to all objects that do not have a certain structure. Note that often object oriented models make, either explicitly or implicitly, the assumption that every object belongs to exactly one "most specific class". Under this assumption, intersection can be eliminated from the schema definition since if an object is an instance of two classes, the schema also contains a class that specializes both and of which the object is an instance [1]. In contrast, in $\mathcal{BDM}$ we do not want to enforce the "most specific class assumption", consistently with most knowledge representation formalisms [4] and semantic data models [13]. Such an assumption would go against the spirit of our notion of polymorphism, which allows an object to simultaneously have more than one structure (and thus to belong to different unrelated classes).

[1] We may also assume that $\mathcal{C}$ contains some additional symbols such as `Integer`, `String`, etc., that are interpreted as usual, with the constraint that no definition of such symbols appears in the schema.

Class and view definitions are built out of structure expressions by asserting constraints on links between the instances of the defining class and the instances of the other classes. A class definition expresses necessary conditions for an object to be an instance of the defined class, whereas a view definition characterizes exactly (by means necessary and sufficient conditions) the objects belonging to the defined view. Our concept of view bears similarities to the concept of *query class* of [18].

Class and *view definitions* have the following forms (C is the name of the class or of the view to be defined):

class C
  structure declaration
  link declarations
endclass

view C
  structure declaration
  link declarations
endview

Let us explain the different parts of a class (view) definition.

- A *structure declaration* has the following form:

  is a kind of T.

  It can be regarded as both a type declaration in the usual sense, and an extended ISA declaration introducing (possibly multiple) inheritance.

- *link declarations* are distinguished as follows:
  - *Universal* and *existential link declarations*, whose form respectively is:

    all L in T
    exists L in T.

    An universal declaration states that each object reached through link L from an instance of C has structure T, while an existential one states that for each instance of C there is at least one object of structure T reachable through link L. Note that link declarations represent a generalization of existence and typing declarations for attributes (and roles).
  - *Cardinality declarations* whose form is:

    exists (u, v) b in T
    exists (u, v) b^- in T

    where u is a nonnegative integer and v is a nonnegative integer or the special value ∞. A declaration of this kind specifies that for each instance of C there are at least u and at most v different objects of structure T reachable through an given atomic link l (l^-). Existence and functional dependencies can be seen as special cases of this type of constraint.
  - *Key declarations* whose form is:

    key $L_1, \ldots, L_m$.

    A declaration of this kind states that each object o in C is linked to at least one other object through each link that appears in the declaration, and moreover the objects reached through these links uniquely determine o, in the sense that C contains no other object o' which is linked, by means of $L_1, \ldots, L_m$, to exactly the same objects as o (for all links in the declaration).

In both class and view definitions link declarations are optional.

2.2 Semantics

The formal semantics of a $\mathcal{BDM}$ schema is based on the notion of *interpretation* $\mathcal{I} = (\mathcal{O}^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\mathcal{O}^{\mathcal{I}}$ is a nonempty set that constitutes the *universe* of the interpretation and $\cdot^{\mathcal{I}}$ is the *interpretation function* over such a universe. Note that an interpretation corresponds to the usual notion of database state. Traditional object oriented models distinguish between objects (characterized through their object identifier) and values associated to objects. The structure of an object is specified through its value which can be either a tuple, a set or an atomic value. Since an object has a unique value it is forced to have a unique structure. Instead, in $\mathcal{BDM}$ we have chosen not to distinguish between objects and values, and one is permitted to assign different structures to an element of the universe of interpretation. Indeed, we regard $\mathcal{O}^{\mathcal{I}}$ as a set of *polymorphic objects*, that is objects possibly having simultaneously more than one structure, i.e.:

1. The structure of *individual*: an object can always be considered as having this structure, and this allows it to be referenced by other objects of the domain.

2. The structure of *tuple*: an object o having this structure can be considered as a property aggregation, which is formally defined as a partial function from $\mathcal{A}$ to $\mathcal{O}^{\mathcal{I}}$ with the proviso that o is uniquely determined by the set of attributes on which it is defined and by their values. Below the term tuple is used to denote an element of $\mathcal{O}^{\mathcal{I}}$ that has the structure of tuple, and we write $[A_1 \colon o_1, \ldots, A_n \colon o_n]$ to denote any tuple t such that, for each $i \in \{1, \ldots, n\}$, $t(A_i)$ is defined and equal to o_i (which is called the A_i-component of t). Note that the tuple t may have other components as well, besides the A_i-components.

3. The structure of *set*: an object o having this structure can be considered as an instance aggregation, which is formally defined as a finite collection of objects in $\mathcal{O}^{\mathcal{I}}$, with the following provisos: (1) the view of o as a set is unique, in the sense that there is at most one finite collection of objects of which o can be considered an aggregation,

and (2) no other object o' represents the aggregation of the same collection. Below the term set is used to denote an element of $\mathcal{O}^{\mathcal{I}}$ that has the structure of set, and we write $\{\!|o_1, \ldots, o_n|\!\}$ to denote the collection whose members are exactly $o_1, \ldots, o_n$.

The interpretation function $\cdot^{\mathcal{I}}$ assigns an *extension* to classes, structure expressions, and links, as follows:

- It assigns to `member` a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that for each $\{\!|\ldots, o, \ldots|\!\} \in \mathcal{O}^{\mathcal{I}}$, we have that $(\{\!|\ldots, o, \ldots|\!\}, o) \in \texttt{member}^{\mathcal{I}}$.
- It assigns to every role U a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$.
- It assigns to every attribute A a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that, for each tuple $[\ldots, A\colon o, \ldots] \in \mathcal{O}^{\mathcal{I}}$, we have that $([\ldots, A\colon o, \ldots], o) \in A^{\mathcal{I}}$, and there is no $o' \in \mathcal{O}^{\mathcal{I}}$ different from o such that $([\ldots, A\colon o, \ldots], o') \in A^{\mathcal{I}}$. Note that this implies that every attribute in a tuple is functional for the tuple.
- It assigns to every complex link a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that the following conditions are satisfied (in the semantics, "$\circ$" stands for concatenation of binary relations, and "$*$" for their reflexive transitive closure):

$$
\begin{array}{l}
(L_1 \cup L_2)^{\mathcal{I}} = L_1^{\mathcal{I}} \cup L_2^{\mathcal{I}} \\
(L_1 \circ L_2)^{\mathcal{I}} = L_1^{\mathcal{I}} \circ L_2^{\mathcal{I}} \\
(L^*)^{\mathcal{I}} = (L^{\mathcal{I}})^* \\
(L^-)^{\mathcal{I}} = \{(o, o') \mid (o', o) \in L^{\mathcal{I}}\} \\
(\underline{\text{identity}}(C))^{\mathcal{I}} = \{(o, o) \in \mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}} \mid o \in C^{\mathcal{I}}\}.
\end{array}
$$

- It assigns to every class and to every structure expression a subset of $\mathcal{O}^{\mathcal{I}}$ such that the following conditions are satisfied[2]:

$$
\begin{array}{l}
\texttt{Any}^{\mathcal{I}} = \mathcal{O}^{\mathcal{I}} \\
\texttt{Empty}^{\mathcal{I}} = \emptyset \\
C^{\mathcal{I}} \subseteq \mathcal{O}^{\mathcal{I}} \\
(\neg T)^{\mathcal{I}} = \mathcal{O}^{\mathcal{I}} \setminus T^{\mathcal{I}} \\
(T_1 \wedge T_2)^{\mathcal{I}} = T_1^{\mathcal{I}} \cap T_2^{\mathcal{I}} \\
(T_1 \vee T_2)^{\mathcal{I}} = T_1^{\mathcal{I}} \cup T_2^{\mathcal{I}} \\
(o_1, \ldots, o_n)^{\mathcal{I}} = \{o_1, \ldots, o_n\} \subseteq \mathcal{O}^{\mathcal{I}} \\
[A_1\colon T_1, \ldots, A_n\colon T_n]^{\mathcal{I}} = \{[A_1\colon o_1, \ldots, A_n\colon o_n] \in \mathcal{O}^{\mathcal{I}} \\
\qquad\qquad\qquad\qquad \mid o_1 \in T_1^{\mathcal{I}}, \ldots, o_n \in T_n^{\mathcal{I}}\} \\
\{T\}^{\mathcal{I}} = \{\{\!|o_1, \ldots, o_n|\!\} \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_n \in T^{\mathcal{I}}\}.
\end{array}
$$

The elements of $C^{\mathcal{I}}$ are called *instances* of C.

In order to characterize which interpretations are legal according to a specified schema, we now define what it means that, given an interpretation $\mathcal{I}$, an object $o \in \mathcal{O}^{\mathcal{I}}$ *satisfies* a declaration which is part of a class or view definition:

- o satisfies "$\underline{\text{is a kind of}}\ T$", if $o \in T^{\mathcal{I}}$;
- o satisfies "$\underline{\text{all}}\ L\ \underline{\text{in}}\ T$", if for all $o' \in \mathcal{O}^{\mathcal{I}}$, $(o, o') \in L^{\mathcal{I}}$ implies $o' \in T^{\mathcal{I}}$;
- o satisfies "$\underline{\text{exists}}\ L\ \underline{\text{in}}\ T$", if there is $o' \in \mathcal{O}^{\mathcal{I}}$ such that $(o, o') \in L^{\mathcal{I}}$ and $o' \in T^{\mathcal{I}}$;
- o satisfies "$\underline{\text{exists}}\ (u, v)\ b\ \underline{\text{in}}\ T$", if there are at least u and at most v objects $o' \in \mathcal{O}^{\mathcal{I}}$ such that $(o, o') \in b^{\mathcal{I}}$ and $o' \in T^{\mathcal{I}}$; a similar definition holds for a cardinality declaration involving b^-.

Moreover, a class C satisfies a key declaration "$\underline{\text{key}}\ L_1, \ldots, L_m$", if for every instance o of C in $\mathcal{I}$ there are objects $o_1, \ldots, o_m \in \mathcal{O}^{\mathcal{I}}$ such that $(o, o_i) \in L_i^{\mathcal{I}}$, for $i \in \{1, \ldots, m\}$, and there is no other object $o' \neq o$ in $C^{\mathcal{I}}$ for which these conditions hold.

An interpretation $\mathcal{I}$ *satisfies a class definition* δ for the class C, if every instance of C in $\mathcal{I}$ satisfies all declarations in δ, and if C satisfies all key declarations in δ. $\mathcal{I}$ *satisfies a view definition* δ for the view C, if the set of objects that satisfies all declarations in δ is exactly the set of instances of C. In other words, there are no other objects in $\mathcal{O}^{\mathcal{I}}$ besides those in $C^{\mathcal{I}}$ that satisfy all declarations in δ.

If $\mathcal{I}$ satisfies all class and view definitions in a schema $\mathcal{S}$ then $\mathcal{I}$ is called a *model* of $\mathcal{S}$. A schema is said to be *consistent* if it admits a model. A class (view) C is said to be *consistent in* $\mathcal{S}$, if there is a model $\mathcal{I}$ of $\mathcal{S}$ such that $C^{\mathcal{I}}$ is nonempty. The notion of consistency is then extended to structure expressions in a natural way.

2.3 Example of a $\mathcal{BDM}$ schema

To illustrate the main characteristic of the basic data model we present a simple $\mathcal{BDM}$ schema. The schema models a condominium (instance of the class `Condominium`) as a set of apartments and simultaneously as a record having two fields, one for its address and one for the an integer representing its budget. The address, `loc` being declared a key, univocally identifies a condominium. Each condominium is also required to have a single manager who manages it. Similarly we introduce the class `Manager` and `Address`. We also define a view `CondominiumManager` as the collection of those managers who manage a condominium.

```
class Condominium
    is a kind of
        {Apartment}∧
        [loc: Address, budget: Integer]
    key loc
    exists (1,1) manages⁻ in Manager
endclass

class Address
    is a kind of
        [city: String, street: String,
         num: Integer]
    key city, street, num
endclass
```

[2] Note the slight notational abuse in the fourth equation: to be more precise we should write $(o_1, \ldots, o_n)^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots, o_n^{\mathcal{I}}\} \subseteq \mathcal{O}^{\mathcal{I}}$ where $o_1^{\mathcal{I}}, \ldots, o_n^{\mathcal{I}}$ are distinguished objects in $\mathcal{O}^{\mathcal{I}}$.

view CondominiumManager
  is a kind of Manager
  exists manages in Condominium
endview

class Manager
  is a kind of
    [ssn: String, loc: Address]
  key ssn
  exists manages in Any
endclass

Observe that, in $\mathcal{BDM}$, objects can be seen as having different structures simultaneously. In the example, the structure of the class Condominium is specified through a conjunction of the set structure {Apartment} and the record structure [loc: Address, budget: Integer]. Therefore, the designer is anticipating that each instance of Condominium will be used both as a set (in this case the set of apartments forming the condominium) and as a record structure collecting the relevant attributes of the condominium (in this case where the condominium is located and its budget). Moreover, each instance of condominium can also be regarded as an individual object that can be referred to by other objects through roles (in this case manages).

3 Statistical constructs

Next we extend the basic data model $\mathcal{BDM}$ with suitable constructs for aggregating data for statistical means. We call the resulting data model $\mathcal{SDM}$.

A *statistical aggregate* (or simply aggregate) is a specification of how to classify the instances of a given class, which is called *target of the aggregate*, according to the values of certain properties. For example we may define a statistical aggregate by classifying the instances of the class people according to the value of sex and age (possibly partitioning ages in intervals). The characterizing feature of the data model $\mathcal{SDM}$ is that statistical aggregates are indeed classes, on which we can operate exactly as we do for simpler classes. Aside from special structures for representing statistical aggregates, $\mathcal{SDM}$ is equipped with a new kind of atomic links, called computed links, since they are computed by making use of some predefined operators.

Similarly to $\mathcal{BDM}$, an $\mathcal{SDM}$ schema is a collection of class and view definitions over the alphabet $\mathcal{B}$, where $\mathcal{B}$ is partitioned into a set of symbols of *class* $\mathcal{C}$, a set of symbols of *attribute* $\mathcal{A}$, a set of symbols of *role* $\mathcal{U}$, a set of symbols of computed link $\mathcal{F}$, and a set of symbols of predefined operators $\mathcal{OP}$, such as COUNT, AVR, FREQ, etc. The symbols C, A, U, F, OP, denote generic elements of $\mathcal{C}$, $\mathcal{A}$, $\mathcal{U}$, F, e $\mathcal{OP}$ respectively. An *atomic link* in $\mathcal{SDM}$, denoted by the symbol l, is either an attribute, a role, a computed link, or one of the following special symbols: member (as in $\mathcal{BDM}$), in (denoting member$^-$), and target (a special attribute used in the context of statistical aggregates).

Class definitions of $\mathcal{SDM}$ are analogous to those of $\mathcal{BDM}$; in contrast view definitions can have in alternative to the structure declaration a *statistical aggregate declaration*. Such declarations are of two kinds: *simple aggregate declarations* or *complex aggregate declarations*. We illustrate them below.

3.1 Simple aggregate declarations

A simple aggregate declaration has the following form:

$$\underline{\text{is an aggregate of}}\ T_1, \ldots, T_n$$

where $T_1, \ldots, T_n$ are structural expressions. The target class in this case is understood to be the union of $T_1, \ldots, T_n$. Intuitively, by means of a simple aggregate declaration as the one above we define a class C having n instances, each such an instance is a set made up by the whole collection of objects in T_i.

The formal semantics is as follows: given an interpretation $\mathcal{I}$, a class C satisfies a declaration "is an aggregate of $T_1, \ldots, T_n$" if

$$C^{\mathcal{I}} = \{\{\!|o \mid o \in T_1^{\mathcal{I}}|\!\}, \ldots, \{\!|o \mid o \in T_n^{\mathcal{I}}|\!\}\}.$$

A typical example of simple aggregate declaration is the following:

view AgeIntervals
  is an aggregate of
    $(0, \ldots, 10), (11, \ldots, 100), \texttt{Integer} \wedge \neg(0, \ldots, 100)$
endview

where the declaration is used for partitioning Integer in three intervals; indeed AgeIntervals has three instances each denoting a set of objects: the first denotes $\{\!|0 \ldots, 10|\!\}$, the second denotes $\{\!|11 \ldots, 100|\!\}$, and the third denotes $\{\!|101, 102, \ldots|\!\}$.

3.2 Complex aggregate declarations

A complex aggregate declaration has the following form:

$$\underline{\text{is an aggregate of}}\ C\ \underline{\text{by}}\ att_1 = L_1 : T_1, \ldots, att_n = L_n : T_n$$

where C is the target class, $att_i \in \mathcal{A}$, L_i are (possibly complex) links, and T_i are structural expressions.

Intuitively by means of a complex aggregation declaration as the one above we define a class C' that denotes a classification of the instances of the class C according to the values of $L_1, \ldots, L_n$.

The formal semantics is as follows. Given an interpretation $\mathcal{I}$, a class C' satisfies the declaration "is an aggregate of C by $att_1 = L_1 : T_1, \ldots, att_n = L_n : T_n$" if

$$C'^{\mathcal{I}} = \{[att_1 : o_1, \ldots, att_n : o_n, \texttt{target} : S] \mid o_1 \in T_1^{\mathcal{I}}, \ldots, o_n \in T_n^{\mathcal{I}}, S = \{\!|o \in C^{\mathcal{I}} \mid (o, o_1) \in L_1^{\mathcal{I}}, \ldots, (o, o_n) \in L_n^{\mathcal{I}}|\!\}\}$$

Note that $att_1, \ldots, att_n$ form a key for the tuples in C'. Observe also that if for some $o_1 \in T_1^{\mathcal{I}}, \ldots, o_n \in T_n^{\mathcal{I}}$, we have that $[att_1 : o_1, \ldots, att_n : o_n, \texttt{target} : S] \notin \mathcal{O}^{\mathcal{I}}$, then it is not possible to assign to C' an extension which satisfies the declaration.

A typical example of complex aggregate declaration is the following:

view Agg
  is an aggregate of Person by
    s = sex : Sex, a = age ∘ in : AgeIntervals
endview

where AgeIntervals is the statistical aggregate defined in the previous example. The complex aggregate Agg denotes the classification of the instances of Person by sex and age intervals. The number of instances of Agg is equal to the cardinality of the extension of Sex multiplied by the cardinality of the extension of AgeIntervals, that is $2 \times 3 = 6$. An example of instance of Agg is the following:

$$[\mathtt{s} = male, \mathtt{a} = (0, \ldots, 10), \mathtt{target} = \{\!|o_1, \ldots, o_n|\!\}]$$

where $o_1, \ldots, o_n$ are all males having an age ranging from 0 to 10 that are contained in the extension of the class Person.

3.3 Computed links

As mentioned, beside statistical aggregate declarations, we also introduce the possibility of declaring computed links, i.e. atomic links that are computed by making use of a set of predefined operators. Such operators, which typically are statistical operators, operate on whole classes, computing for example the number of the instances of a class (COUNT), The average of a certain value for a class (AVR), etc.

A *computed link declaration* has the following form:

$$\underline{\text{compute}}\ F\ \underline{\text{as}}\ OP(\mathcal{E}_1, \ldots, \mathcal{E}_n)$$

where $F \in \mathcal{F}$ is the computed link introduced by the declaration, $OP \in \mathcal{OP}$ is one of the predefined operators, and $\mathcal{E}_1, \ldots, \mathcal{E}_n$ are expressions denoting the classes that constitute the input of OP.

A computed link F represents a function having as domain the class in which its declaration appears, and as codomain the codomain of OP.

The domain and codomain of OP are predefined. For example COUNT$(\cdot)$ has as domain the set of the subset of Any and as codomain Integer, similarly AVR$(\cdot)$. Instead FREQ$(\cdot, \cdot)$ accept as input two subsets $\mathcal{E}_1, \mathcal{E}_2$ of Any, such that $\mathcal{E}_1 \subseteq \mathcal{E}_2$, and returns a rational number, the ratio between the cardinality of $\mathcal{E}_1$ and the cardinality of $\mathcal{E}_2$.

The expressions $\mathcal{E}_1, \ldots, \mathcal{E}_n$ in input to the predefined operators can be either structural expressions or special expressions having the following form:

$$@L : T$$

where L is a (possibly complex) link and T is a structural expression.

For notational convenience, given an interpretation $\mathcal{I}$, for each object $o \in \mathcal{O}^{\mathcal{I}}$, we denote by $\mathcal{E}_i^{\mathcal{I}}(o)$ the set singled out by $\mathcal{E}_i$, defining $T^{\mathcal{I}}(o) = T$.

The semantics of a computed link declaration is as follows. Given an interpretation $\mathcal{I}$, an object $o \in \mathcal{O}^{\mathcal{I}}$ satisfy the declaration "compute F as $OP(\mathcal{E}_1, \ldots, \mathcal{E}_n)$" if

$$(o, o') \in F^{\mathcal{I}} \text{ where } o' = OP^{\mathcal{I}}(\mathcal{E}_1^{\mathcal{I}}(o), \ldots, \mathcal{E}_n^{\mathcal{I}}(o)).$$

3.4 Example of a $\mathcal{SDM}$ schema

We now present an example of a schema in $\mathcal{SDM}$ that shows the power that comes from the ability of treating statistical aggregates as any other classes. The example stems from a real case considered by AIPA as part of a study on the state of the information systems used by Italian Public Administration.

Norms issued by the Parliament institute certain processes, each of which is carried out in a given Ministry. It is of great interest to know the distribution of the quantity of norms wrt to the quantity of ministries they affect. Indeed to simplify administrative procedures it is desirable that most ministries are affected only by a limited number of norms. This simple observation makes the importance of having the ability to determine the above distribution apparent.

The classes of elementary data of interest, in this example, are the following: Norms, Processes, Process-Norm pairs (denoting that a norm contributes to the institution of a process) and Ministries. Moreover we also require that each process determines a ministry in which it takes place. The corresponding $\mathcal{SDM}$ schema is the following:

class Norm
  is a kind of Any
endclass

class Ministry
  is a kind of Any
endclass

class Process
  is a kind of Any
  exists (1,1) m in Ministry
endclass

class PN
  is a kind of [p:Process,n:Norm]
  key p,n
endclass

Observe that the last definition denotes a binary relation in a formally correct way.

Data that populate the classes Norm, Ministry, Process e PN, can be obtained directly through a survey. Instead the data we are interested in – the distribution of norms wrt ministries – require a sophisticated use of the statistical aggregates and, in particular they require the aggregation of statistical aggregates repeatedly.

We get the data of interest in three step of aggregation.

- The first step is to aggregate the class PN by norms and ministries.

  view A1
    is an aggregate of PN by
      n′ = n : Norm, pm = p ∘ m : Ministry
  endview

- The second step is to aggregate the class A1 (which is a statistical aggregate) by norm, adding to each instance of the resulting class a computed link that denotes the number of ministries to which a norm applies. Observe this number is the same as the cardinality of the subset of the target of the aggregate singled out by a given norm.

  view A2
    is an aggregate of A1 by
      $n'' = n'$: Norm
    compute num_ministry as
      COUNT(@target ∘ member : Any)
  endview

- The third and last step is to aggregate the class A2 (which is again a statistical aggregate) by number of ministries, adding a computed link that denotes the number of norms that affect a given number of ministries. Observe this number is the same as the cardinality of the subset of the target of the aggregate singled out by the number of ministries.

  view A3
    is an aggregate of A2 by
      num_ministry$'$ = num_ministry : Integer
    compute num_norm as
      COUNT(@target ∘ member : Any)
  endview

The class A3 obtained in this way contains exactly the summary data desired: the distribution of the quantity of norms wrt the quantity of ministries they affect.

4 Discussion and conclusion

$\mathcal{SDM}$ is a powerful formalism for modeling both statistical data and elementary data. One of the main features of $\mathcal{SDM}$ (inherited from $\mathcal{BDM}$) is the adoption of the notion of polymorphic object, which allows for considering objects having a complex structure as individual objects, thus exploiting their complex structure only when such structure is actually relevant. This characteristic allows for treating statistical aggregates in the same way as elementary data are treated, and hence permit us to form statistical aggregates of statistical aggregates.

We would like to conclude the paper by briefly mentioning some methodology issues. Often statistical aggregates are formed by partitioning a target class in equivalence classes induced by certain properties of the instances of the target class. The formalism introduced in this paper does not impose such a restriction, in the sense that its semantics remains perfectly coherent even if we do not enforce the above restriction. In particular, the properties of a given class used to aggregate its instances do not need to cover the whole class, i.e. it is admissible that some instances of the target class are not inserted in any of the instances of the aggregate class. Furthermore, such properties do not need to single out disjoint subsets of the target class, i.e. it is admissible that a given instance of the target class is inserted in more then one instance of the aggregate class. $\mathcal{SDM}$ leaves to the system designer the choice of which restrictions (if any) to enforce in forming statistical aggregates.

Acknowledgments

The authors would like to express their gratitude to Maurizio Lenzerini for valuable suggestions and comments on the work presented here.

References

[1] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 159–173, 1989.

[2] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach*. Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.

[3] C. Batini and G. Di Battista. Design of statistical databases: a methodology for the conceptual step. *Information Systems*, 13(4):407–422, 1988.

[4] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, 1992.

[5] A. W. Bragg. Data manipulation languages for statistical databases – the Statistical Analysis System (SAS). In *Proceedings of the 1st LBL Workshop on Statistical Data Bases Management*, 1981.

[6] D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning. In *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases (DOOD-95)*, LNCS 1013, pages 229–246, Springer Verlag, 1995.

[7] T. Catarci, G. D'Angiolini, and M. Lenzerini. Concept description language for statistical data modeling. In *Proceedings of Very Large Data Bases*, 1990.

[8] T. Catarci, G. D'Angiolini, and M. Lenzerini. Concept language for statistical data modeling. *Data and Knowledge Engineering*, 1995. To appear.

[9] P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[10] G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 205–212, 1994.

[11] G. De Giacomo and M. Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 801–807, 1995.

[12] S. P. Ghosh. Statistical relational tables for statistical database management. *IEEE Transaction on Software Engineering*, SE-12(12):1106–1116, 1986.

[13] R. B. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[14] G. Ozsoyoglu and Z. M. Ozsoyoglu. Statistical database query languages. *IEEE Transaction on Software Engineering*, SE-11(10):1071–1081, 1985.

[15] G. Ozsoyoglu, Z. M. Ozsoyoglu, and V. Matos. Extending relational calculus with set-valued attribute and aggregate functions. *Transactions on Database Systems*, 12(4):566–592, 1987.

[16] G. Ozsoyoglu and T.-A. Su. Rounding and inference control in conceptual models for statistical databases. In *Proceedings of the IEEE Security Symposium*, 1985.

[17] A. Shoshani and H. K. T. Wong. Statistical and scientific databases issues. *IEEE Transactions on Software Engineering*, SE-11(10), 1985.

[18] M. Staudt, M. Nissen, and M. Jeusfeld. Query by class, rule and concept. *Journal of Applied Intelligence*, 4(2):133–157, 1994.

[19] S. Y. W. Su. Semantic data model for statistical databases. In *Proceedings of the IEEE CS International Conference on Data Engineering*, 1984.