The P-type Model : from Databases to Knowledge Bases

Ana Simonet and **Michel Simonet**
Laboratoire TIMC-IMAG,
38706 La Tronche cedex - FRANCE
e-mail : (Ana,Michel).Simonet@imag.fr

1 The P-type Model

The p-type data model was conceived in the early eighties as an answer to database needs [12]. It was expressed within the Algebraic Data Types (ADT) paradigm [7] [8] and its main concern was the sharing of objects by several kinds of users seeing them through one or several views. A p-type is organized in a hierarchy of classes, where classes model database views. An object belongs to one and only one p-type, and to several views. Multiple specialisation is not necessary to express that an object belongs to several subclasses (views of a p-type). It is used only to specify a subset of the views intersection.

To specify a p-type one first gives its minimal view then its other views by simple or multiple strict specialisation, adding attributes and/or assertions. The root of the hierarchy of views is called the minimal view in that all the objects of the p-type must satisfy its properties. The ADT of a p-type is derived from its views declaration. This type contains all the attributes and methods which appear in the views of the p-type, including the minimal view. An object belongs to a view iff it satisfies its assertions. Objects which are instances of a p-type may belong to several views, among which only the minimal view is mandatory.

A p-type is defined as an algebraic data type $< S, F, E >$ where S is a set of sorts $\{s_1, ..., s_n\}$, the carrier of the type, F a set of functions $s_i \times s_j \times ... \times s_k \to sl$ and E a set of equations [12]. One sort, T, called the set of interest of the type, is central, in that the aim of the type definition is to establish the elements of the type and define their behaviour. In general, the type is given the name of its set of interest : T. Among all possible functions, we call attributes those of the form $T \to s, s \in S$. Other functions are called methods.

The algebraic type of the p-type is derived from the views declarations (including the minimal view). The type *PERSON* contains all the attributes and methods which appear in its views. The domain of an attribute in type *PERSON* is the union of its domains in the views where it is declared.

Let t_{min} : $< S, Fmin, Emin >$, t_1 : $< S, F_1, E_1 >$, t_2 : $< S, F_2, E_2 >$, ... be the views of a p-type T. T is defined as $< S, F, E >$ where S is the support set of T, $F = \bigcup_i F_i$ and $E = E_{min}$.

As a simple example, consider a p-type PERSON whose minimal view has the attributes Name, Age, and Sex, and its different views are ADULT : PERSON (Age$\geq$ 18), SENIOR : PERSON (Age>65), and STUDENT, characterized by specific attributes. In the graph presented in figure 1, a student aged between 18 and 65 belongs to the views PERSON (the minimal view is mandatory), STUDENT, and ADULT, provided it satisfies the properties of these views.

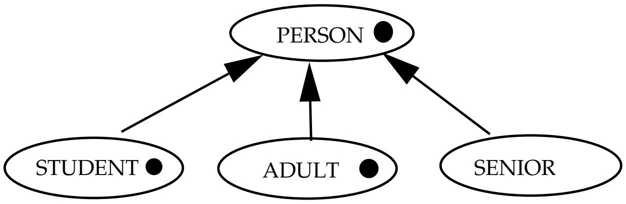

Figure 1: Graph of p-type *PERSON*

The set of interest (domain) of the minimal view person is identical to that of the p-type *PERSON*. The domain of another view is a subset of the domain of the view it specializes, or of the intersection of the domains of the views it specializes in case of multiple specialization.

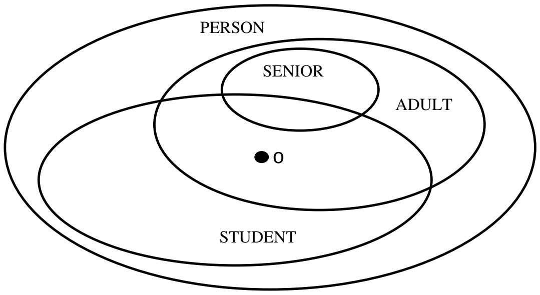

Figure 2: Inclusion set of p-type *PERSON*

In the general case, any view may be a strict specialisation of one or more views, and have its own attributes and/or assertions. Assertions are Horn clauses with literals of the form $\forall x$ *Attribute(x) in Domain*, called Domain predicate. An example of such an assertion is *Age(x)>18 → MilitaryService(x) in {done, deferred, exempt}*. Assertions reduced to a single Domain predicate, such as *Age(x) in [18, 65]* may stand for an attribute domain definition in a view.

Unlike most OODBMS such as the O2 proposal [9], attributes whose values are calculated by a method (or a procedure) are true attributes, and therefore are not themselves considered as methods. Any attribute may be stored or not, and may be calculated or not. A calc-stored attribute is calculated from the values of other attributes (e.g., Age from BirthDate and CurrentDate) and automatically updated whenever necessary.

2 An Example

A base schema is made up of several p-type definitions. In general, these p-types are not independent. In OSIRIS the interrelationships between different p-types of a schema are expressed by attribute definitions and by Inter-Object Dependencies (IODs).

We present the main features of the p-type description language and of the Inter-Object Dependencies through a very simple OSIRIS example. The universe modelled is that of persons and vehicles. Persons may be and/or students, teachers, trainee-teachers, professors, sportsmen. They are also either adults or minors according to their age. A given person is a model of the minimal view and may belong to none, any or several other views. The view TRAINEE, which inherits STUDENT and TEACHER, is not necessary to express that a person can be a student and a teacher at the same time. It has been created to designate a subset of their intersection, characterized by some more assertions, which restrict its domain.

```
class PERSON – Minimal view of p-type PERSON
attr
  Name : P_NAME; – P_NAME is declared elsewhere
   Children : setof PERSON;
   Sex : CHAR;
   Age : INT;
   MilitaryService : STRING;
   IncomeTax : REAL calc; – procedural attachment
   CarsOwned : setof CAR;
                – CAR is a view of a p-type VEHICLE
key  Name – External key
methods – other functions specification
assertions
– Domain Assertions
    Sex in { “f”, “m” };
    0 ≤Age ≤ 120
    MilitaryService in
       { “yes”, “no”, “deferred”, “exempt” };
– Inter-Attribute Dependencies
    Age < 18 ⇒ MilitaryService = “no”;
    Age ≥  18 ⇒ MilitaryService in
       { “yes”, “deferred”, “exempt” };
    Sex = ‘f’ ⇒ MilitaryService = “no”;
end;
```

The minimal view automatically contains a private attribute OID : t_{oid}.

```
view STUDENT : PERSON . . .
view TEACHER : PERSON . . .
view PROFESSOR : TEACHER . . .

view TRAINEE : STUDENT , TEACHER
        – specializes STUDENT and TEACHER
assertions
  Status = “trainee”;
  Studies = “graduate”;
  Diplomas contain  “degree”;
end;
⋮
view ADULT : PERSON
assertions
  Age ≥ 18;
end;
view SENIOR : ADULT
assertions
  Age > 65;
end;

  implementation PERSON

    – stored attributes
    – body of methods

end;
```

The attributes of the type *PERSON* are those of the minimal view, PERSON, plus those defined in other views : Studies, Year, Status, Diplomas. Within a given view, the user may only access the attributes inherited from its super-views and the attributes proper to the view, if any.

Objects which are instances of the p-type *PERSON* may satisfy one or several views, among which only the minimal view is mandatory.

A part of the description of the p-type *VEHICLE* migh be :

```
class VEHICLE
attr
  Type : STRING;
  Year : DATE;
. . .
assertions
  Type in { “car”, “truck”, “bus”, “tractor”};
end;

view CAR : VEHICLE
attr
 Owner : PERSON;
. . .
assertions
  Type = “car”;
end;
```

Within the scope of the definition of p-type *PERSON* and view CAR of p-type *VEHICLE*, the interrelationships between cars and persons are expressed through the attributes CarOwner and Owner of the p-types *PERSON* and *VEHICLE* respectively. To express that these two attributes are reciprocal, one writes an Inter Object Dependency :

```
PERSON.CarsOwned reverse  CAR.Owner
```

CarsOwned in p-type *PERSON* being declared as the reverse function of Owner in p-type *VEHICLE*, the OSIRIS system ensures integrity maintenance. In particular, every car whose owner is a person X must belong to the set of cars of X. For example, suppressing a car Y with owner X implies that Y no longer belongs to the set of cars owned by X. Similarly, adding a car Y with Owner X would trigger the checking that Y belongs to the set of cars owned by X, and adding it if necessary. Thus referential integrity is checked and automatically maintained. This deductive aspect (deducing a new CarsOwned value from the insertion of a new car) is also present in Inter Attribute Dependencies (e.g. value "no" for MilitaryService can be deduced from an Age less than 18).

When modelling the universe of persons, i.e., characterizing its subclasses, the modeller has to make choices. For example, the SENIOR view can be defined as an ADULT whose Age is > 65, or as a PERSON with the same constraint on the age. Both views would be considered equivalent by the Osiris system. However, different consequences might result from either choice. If the view ADULT is modified, e.g., enriched with some new property, the view SENIOR will inherit this property only if it has been explicitly defined from the view ADULT or any subview of it.

3 The Classification Space

The key to implementation is definition of the partitioning of the object space based on the Domain Predicates of the p-type. Each Domain Predicate defines a partitioning of the attribute it covers. The product of partition of an attribute by all the predicates of the p-type [13] [14], determines a partitioning of the domain of that attribute into Stable-Sub-Domains (SSD). An instance whose attribute values change within the same SSD satisfies the same Domain Predicates, hence the same assertions. This is the stability property on which the whole system relies.

In the example given above, the partitioning of the attribute domains is :

Domain (Age) = $d_{11} \bigcup d_{12} \bigcup d_{13}$
Domain (MilitaryService) = $d_{21} \bigcup d_{22}$
Domain (Sex) = $d_{31} \bigcup d_{32}$

where
d_{11} = [0, 18[, d_{12} = [18, 65], d_{13} =]65, 120]
d_{21} = {"no"},
d_{22} = {"yes", "deferred", "exempt"}
d_{31} = {"m" }, d_{32} = {"f"}

By definition, each subdomain d_{ij} has the following property : when the value of attribute $Attr_i$ changes within the subdomain d_{ij}, all domain predicates maintain their truth value and consequently the assertions do likewise. Divisions d_{ij} are therefore stability zones for the assertions, hence their name : Stable Subdomains (SSDs). Domain Predicates are transformed into elementary predicates of the form $Attr_i \in d_{ij}$, where the d_{ij} are the SSDs of $Attr_i$. Introducing a new assertion with predicate Age > 40 would cause the splitting of d_{12} into [18, 40] and]40, 65], and the corresponding internal rewriting of the concerned assertions.

The partitioning of each attribute domain is extended to the object space. This partitioning, whose elements are named Eq-classes, is called the Classification Space. It is the quotient space of the object space with respect to the equivalence relation *'satisfy the same subset of Domain Predicates'*. The classification space of a p-type is the cartesian product of the sets of Stable Subdomains of its classifying attributes. Elements of the classification space are called Eq-classes. For a p-type with n classifying attributes [1], Eq-classes are n-tuples $(d_{1i}, d_{2j}, \ldots d_{nl})$.

The classification space can be illustrated in a 3-dimensional space by the figure shown figure 3, obtained by considering only attributes Age, MilitaryService and Sex, and the domain constraints above, leading to a partitioning into 3x2x2 = 12 Eq-classes.

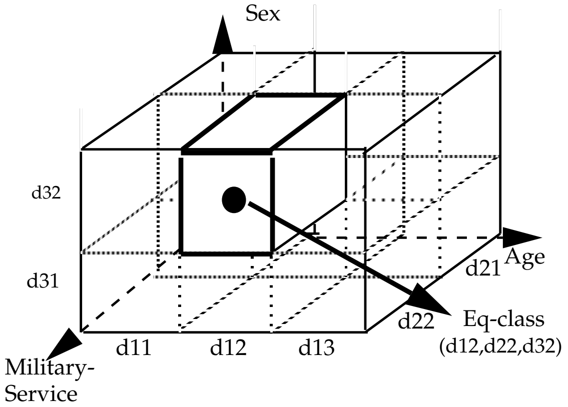

Figure 3: Space partitioning

It is possible to determine at compile time the set of views to which each Eq-class belongs. Classifying an instance, i.e., determining the set of views to which it belongs, is no longer performed by following the hierarchy of views. Classification is performed by a boolean (propositional) solver, based on the structure of the Classification Space, in time linear to the number of views and of SSD, and polynomial to the number of attributes and p-type assertions [2]. When the object is not completely known, its known attribute values determine several Eq-classes instead of a single one. These Eq-classes determine which views are valid, invalid, or potential, i.e., views whose validity still depends on missing attribute values.

Eq-classes are never represented explicitly in their totality. They up hold and direct the compilation process, and at execution time, they index the actual objects of the database. Ther number of actual Eq-classes is therefore limited by the objects which are really entered in the base [14].

Primary indexing through Eq-classes also enables semantic query optimization. The query (PERSON | Age < 30) would automatically select individuals from Eq-classes having SSD d_{11} as a component, and reject those corresponding to d_{13}. Only the elements of those indexed by d_{12} (18≤Age≤65) have to be checked for the condition Age < 30.

---

[1] Classifying attributes are attributes whose domain is partitioned in at least two SSDs.

4 Databases vs Knowledge Bases

Besides security and the ability to efficiently manage large quantities of data, concurrency and data sharing are important features of databases. In typical database applications, an object is assigned one class and the database has to deal with further evolution of its attribute values. In a knowledge base, objects are often not completely known, and object evolution mainly consists in the determination of unknown attribute values, but rarely in value changes. The objective is to obtain the most refined information about the object, including its valid and potential classes, deduced attribute values or value ranges, and explanations about all inferred information.

Another important feature of p-types is that an object can belong to several views and change views (not its p-type) in its lifetime, whereas is OODBMS instanciation is made in one class and is final, as in programming languages. Belonging to a view is a property which is defined as satisfying the assertions of the view, i.e., both its proper and inherited assertions. A mandatory assignation to a view, as is usually the case in a database situation (create p as V) will cause the assertions of V to be verified, considering them as integrity constraints. The way view determination has been designed of p-types [14] consists in determining all the valid views of a given object, extending this determination to that of those possible when the object is not completely known. This process is the very process of instance classification in knowledge bases.

Dealing with incomplete information is an important aspect of knowledge bases. In Osiris, it may happen that incomplete information leads to an absolutely certain conclusion, without having to make hypotheses about unknown values of attributes. In some way, all possible hypotheses have been “compiled” through the Eq-classes. When probabilistic information is available about the distribution of the values of the attributes in its SSDs, the classifying process is able to evaluate the probability assigned to each view when some attributes have unknown values [2]. Classifying a completely known instance is then a particular case of the probabilistic classification : the SSD of a known attribute value has a probability value 1 and the others 0. As a result of classification, views known to be certain have a probability 1, and those impossible a probability 0. When the actual probability of SSDs is not known, assigning to them an arbitrary probability value (e.g., equi-probability), will lead to 0, 1, and non-zero-one values, still characterizing impossible, certain, and potential views. However, in this case, the probability value is not significant and only indicates that the view is potential (i.e., neither certain nor impossible).

The consistency of the base is ensured by the integrity constraints expressed by the assertions. Integrity constraints verification is a by-product of the classification process. In effect, classifying an object in a given view means that the object is a valid interpretation of its assertions. When the user assigns an object to a given view, which is the usual situation in databases, checking the integrity constraints of that view is performed by checking that this view belongs to the objects views.

Other consistency aspects may be considered in a knowledge base context : class validity and assertion contradiction. We also define Domain-inconsistency which is weaker than logical inconsistency and indicates a probable distortion between several assertions (possibly written by several users).

Within a p-type a view may be defined with assertions which make it inconsistent, i.e. no object instance of the p-type can be a model of its assertions (inherited and proper assertions). This is detected by an empty set of valid Eq-classes for the view.

Assertions can be checked for logical inconsistency, which is possible in spite of their first order general form, because the static process enables their transformation into an equivalent set of propositional formulas. Assertions :

a1: Age$< 18 \Rightarrow$ MilitaryService = “no”
a2: Age $\geq 18 \Rightarrow$ MilitaryService $\in$
{“yes”, “deferred”, “exempt”}
a3: Sex = “f” $\Rightarrow$ MilitaryService = “no”

may be transformed into a propositional system where attributes are implicitly universally quantified, and where p_{ij} is the proposition expressing that attribute $Attr_i$ is in SSD d_{ij} :

a1': $p_{11} \Rightarrow p_{21}$
a2': $p_{12} \bigvee p_{13} \Rightarrow p_{22}$
a3': $p_{32} \Rightarrow p_{21}$

along with propositions of the form

$$p_{ij} \Rightarrow \text{not } p_{ik} \text{ for all k} \neq \text{j}$$

expressing the mutual exclusion of stable subdomains for the same attribute :

$$(\forall i) d_{ij} \bigcap d_{ik} = \emptyset \text{ for all } k \neq j.$$

Domain inconsistency is weaker than logical inconsistency. An assertion is said to be domain-inconsistent when its antecedent is always invalidated by other assertions of the type. In the context of the above example, the assertion 'Sex = “f” and $Age > 30 \Rightarrow$ some conclusion' is always valid, whatever its conclusion, because its antecedent is always false, being contradictory to assertions a1-a3, which impose that there cannot be any female aged over 18 in the base [2]. Assertion a2 should have been written : $Age \geq 18$ and $Sex = `m' \Rightarrow MilitaryService \in \{“yes”, “deferred”, “exempt”\}$. One can assume that such Domain-inconsistent assertions are not written deliberately and their detection is essential to the designer. Once they have been detected, it is up to the user to decide whether to maintain them or not. Domain-inconsistencies may be intended by the programmer; they may be harmless, but they may have unwanted hidden consequences, hence the interest of their detection.

P-types were designed in a database perspective and the Osiris implementation fulfills the usual database requirements. Persistency, transactions, concurrency, etc., are provided through the use of a set of persistent C++ classes (called the Osiris

---

[2] This is due to assertion a2: $Age \geq 18 \Rightarrow MilitaryService \in \{“yes”, “deferred”, “exempt”\}$

kernel) which will be implemented in two ways : by an object manager [1] and a relational database [10][11]. The relational version of the kernel will implement data sharing [3] and a nested transaction mechanism similar to that described in [4]. The main objective for a relational implementation was to inherit the qualities of the second generation relational DBMS. Among these, efficient storage of large data volumes, concurrency control, and confidentiality management.

5 Conclusion

To conclude, we add that the p-type data model resembles more nearly Terminological Logics which can classify an instance into several concepts, than the data model of most OODBMS in which an instance must be created in exactly one class and cannot change its class in its lifetime [4]. Work remains to extend OSIRIS to view subsumption, which may be expressed as the inclusion of sets of Eq-classes in the Classification Space. The complexity of view subsumption with respect to the class of assertions taken into account, i.e., Horn clauses with Domain Predicates as literals, is still to be evaluated.

Although no commercial OODBMS has until now incorporated a view mechanism, the idea that views need to be included is becoming widely accepted. In 1992, E. Bertino acknowledged that "several questions about a suitable view model for OODBMS still need to be addressed in current research" [3].

Views are a primary concept in p-types, and are not superimposed to a given object model. A p-type is a semantic unit for the grouping of subclasses, namely views. A real world entity is instanciated in one and only one p-type, and may belong to several views : those of which it satisfies the properties. Grouping subclasses as views of a p-type is the corollary of considering the unity of the object, which is indeed the basis of object modelling. A person is unique, whether considered as a student, a sportsman, an adult, etc. In P/FDM, a prolog-based implementation of a functional data model, a given object may also be instanciated in several subclasses, with the same OID [6]. P. Gray remarks that this approach is equivalent to views, which we acknowledge.

We would also like to mention Date's opinion that "the process of inserting a row can be regarded as a process of inserting that row into the database (rather than into some specific table)"[5]. In an object-oriented perspective, this argues well for automatic classification of objects in views.

References

[1] E. Abecassis, *YOODA user's guide*, APIC systeme Arcueil, France, 1994.

[2] C. G. Bassolet, *Reseaux de Neurones de Classement dans le modele des p-types*, Rapport de Recherche IMAG, Grenoble, to appear.

[3] E. Bertino, M. Negri, G. Pelagatti, L. Sbattella, Object-Oriented Query Languages : The Notion and the Issues, IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No 3, June 1992.

[4] R. G. G. Cattell, *The Object Database Standard : ODMG-93*, Morgann Kaufmann Publishers, 1994.

[5] C. J. Date and David McGoveran, *A New Database Design Principle*, In Database Programming and Design, July 1994.

[6] P. Gray, G. Kemp, *Object-Oriented Systems and Data Independence*, OOIS'94, London, Dec. 1994.

[7] J. Guttag, J. Horning, *The Algebraic Specification of Abstract Data Types*, Acta Informatica, 1978.

[8] B. Liskov, B. Zilles, *Programming with Abstract Data Types*, Proc. of a Symp. on Very High Level Language, Sigplan Notices 9, 4, April 74.

[9] O. Deux et al., The O2 System, CACM, October 1991, Vol 34, No. 10, pp 34-48.

[10] M. Quast,*Osiris et le modele relationnel*, Memoire d'ingenieur CNAM, to appear, TIMC-IMAG, 1995.

[11] M. Quast, A. Simonet, M. Simonet *A Relational implementation of a View-based Object System*, OOIS'95, Dublin, Dec. 1995 (accepted).

[12] A. Sales-Simonet, *Types abstraits et bases de donnees : formalisation du concept de partage et analyse statique de contraintes d'integrite*, These de Docteur-Ingenieur, USMG, Grenoble, 1984.

[13] A. Simonet, M. Simonet, *Objects with Views and Constraints : from Databases to Knowledgebases*, OOIS'94, London, Springer Verlag, 1994.

[14] A. Simonet, M. Simonet, *Osiris : an OO system unifying databases and knowledge bases*, KBKS'95 (Building and Sharing of Very Large-Scale Knowledge Bases), p217-227, IOS Press, 1995.

---

[3] The P of p-types stand for the french *partage*, which means *shared*. P-types were conceived to be shared by several users, while groups of users might have their own views of the set of objects represented by the p-type