Static Classification Schemes for an Object System

A.-M. Simonet, M. Simonet, C.-G. Bassolet, X. Delannoy, R. Hamadi

TIMC - IMAG - CNRS
Faculté de Médecine de Grenoble
38706 La Tronche Cedex - France
({Ana,Michel}.Simonet, Cyr-Gabin.Bassolet, Xavier.Delannoy, Riad.Hamadi)@imag.fr
Tel: +33 (0)4 76 63 71 54 Fax: +33 (0)4 76 51 86 67

Abstract

Instance (or object) classification in a knowledge base management system is the very same problem as view determination in an object DBMS, where views are subsets of classes intensionally defined by constraints. Maintaining the relationship between an object and its current views when the object evolves is a difficult and costly problem. We propose a solution for instance classification based on a partitioning of the object space obtained through a static analysis of the properties defining the views. Four kinds of properties are considered: domain constraints, inter-attribute dependencies, dynamic (past temporal) constraints, and linear constraints.

Introduction

Instance classification (or recognition) in a knowledge base management system consists in determining the current classes of an object -an instance- according to its attribute values and the properties characterizing the class. As the term "class" is polysemic, we shall consider classes as disjoint families of objects, and views as subsets of a class, characterized by constraints on the attributes of this class. Therefore instance classification consists in determining the current views of an object. We make a clear distinction between views and classes because in the standard object model, coming from the programming field, an object belongs to a unique class and never changes its class during its lifetime, whereas it can belong to several views according to its current attribute values. In a Description Logic perspective, views are *defined concepts* whereas classes correspond to *primitive concepts*. An object is assigned to a class but its views are automatically determined by the system, which is the process of instance classification. Note that instance classification is distinct from view (or concept) classification which consists in ordering views according to the subsumption relationship [2].

An object is classified when created, and re-classified after each update. The question of object updating is not usually addressed by KBMS, where the purpose is to determine the current subclasses of an object at a given time (e.g., a fault, a disease). However, the need to deal with large quantities of objects in knowledge bases leads to considering the problem of instance classification (or view determination) for changing objects [9].

Classification is a costly operation, all the more it has to be performed after each update. Our approach aims to minimize the re-classification cost by maximizing the static work done on the constraints which define the views. Although it can be applied to other systems (e.g., Description Logics, object-preserving OODBMS), we present it in the context of our system, Osiris, which answers both database requirements (large quantities of changing objects shared by several categories of users) and knowledge base requirements (deduction, classification).

Data Model

In Osiris, a class is defined by the set of its constituent views. The views are organized in a hierarchical manner. A root view, also called the minimal view, contains the necessary conditions for an object to belong to the class. Other views are defined by the views they specialize and their own properties (attribute and/or constraints).

The extension of a class is that of its minimal view. Assigning an object to a class implies verifying it satisfies the properties of the minimal view. The properties (own and inherited) of other views are necessary and sufficient conditions, thus making possible the automatic determination of the current view(s) of an object of the class, which is instance classification. The minimal view of a class is similar to a primitive concept and the other views similar to defined concepts in Description Logics. The properties of the minimal view of a class are the integrity constraints of the class. The properties of the other views guide the classification process.

In our approach, both the checking of integrity constraints and instance classification are part of the classification process, which is optimized due to a static analysis of the constraints which leads to a partitioning of the object space. Four kinds of constraints are considered:

(D) Domain constraints, which are Domain Predicates (DP), i.e., of the form *Attribute* $\in$ *Domain*. Domain constraints are used to restrict the definition domain

of an attribute and are the basic constituents of database constraints and of rule-based expert systems.

(I) Inter-Attribute Dependencies (IAD), which are Horn clauses whose literals are DPs. Examples of IADs are (i1) and (i2) below.
Domain constraints are a particular case of IAD.

(H) Historic constraints (e.g., h1), which are past temporal logic (propositional) formulas, constructed from state formulas to which past temporal operators and connectives are applied. Two past temporal operators are considered : the universal operator $\Box$ (« always in the past ») and the existential operator $\Diamond$ (« once in the past »).
Historic constraints are a superset of Inter-Attribute dependencies.

(L) Linear constraints on continuous domains (e.g., l1).
We present below the treatment of binary constraints. N-ary constraints (N>2) are also considered by first translating them into binary ones.

Here are examples of view definitions in the class PERSON.

Example

```
class PERSON                                  -- Minimal view
        ssNo: INT;
  (d1): age: INT in ]0, 140];                 -- Domain constraint
  (d2): sex: CHAR in {'m','f'};
  (d3): militaryService: STRING in {"no", "ongoing",
        "done", "deferred", "exempt"};
  (d4): income: REAL ≥ someConstant;
  (d5): nbChildren: INT ≥ 0;
  (i1): age<18 ⇒ militaryService = "no";
              -- Inter-Attribute Dependency
  (i2): age≥18 and sex="m"⇒ militaryService in
        {"ongoing", "done", "deferred", "exempt"};
end PERSON;

view ADULT: PERSON    age ≥ 18;  end ADULT;

view SENIOR: PERSON   age ≥ 65;  end SENIOR;

view HIGHLIFE: PERSON
  (h1)  □ ( age>30 ⇒ ◊ (salary > 200 ))
          ∧ ◊ age>30;          -- Historic constraint
end HIGHLIFE;

view LOWINCOME: PERSON
  (l1)  income - nbChildren * someFactor < 50;
              -- Linear constraint
end LOWINCOME;
```

To belong to the class PERSON, an object must satisfy the constraints of the minimal view, i.e., constraints (d1-d5) and (i1-i2). According to the current value of its attributes, it will be classified in other views, possibly none. We now present the building and use of the object space partitioning.

Classification Space

Instance classification in Osiris is performed on the basis of a partitioning of the object space, called the Classification Space, which is obtained by a static analysis of the Domain Predicates occurring in the constraints of type (D) and (I) in the views of a p-type.

Each DP on an attribute defines a partitioning of this attribute into two blocks: the extension of the DP and its complement to the definition domain of the attribute. The product of all the partitions defined by the DPs on an attribute is also a partition. We call its elements Stable SubDomains, in short SSD, because when an attribute changes its value while remaining in the same SSD, no DP changes its truth value. Some SSDs of the example given above are:

age: d_{11} = [0, 18[, d_{12} = [18, 30], d_{13} =]30, 65[,
d_{14} = [65,140]

sex: d_{21}={'m'}, d_{22}={'f'}

militaryService: d_{31}={"no"},
d_{32}={"ongoing", "done", "deferred", "exempt"}

Eq-classes of a class are then defined as the quotient space of the object space relative to an equivalence relationship called DP-equivalence. Two instances o1 and o2 are DP-equivalent iff $\forall\ p \in \Delta$, p(o1)=p(o2), where Δ is the set of DPs occurring in the views of the class.

Given an Eq-class, it is possible to determine its views at compile time, i.e., the views its objects satisfy and those they do not satisfy (when considering only domain-based constraints, i.e., assertions of type (D) and (I)). To do so, one has only to check the assertions of the view for an object of the Eq-class, because all the objects have the same behaviour relative to DPs, hence to assertions, and consequently to views.

The cartesian product of the SSDs of the attributes of a p-type is called its **classification space**. It is constituted by a set of eq-classes. As the number of eq-classes of a p-type is exponential to the number of attributes, they are never represented in their totality. Only the eq-classes which contain at least an object of the database are effectively created and used to index the objects [10]. However, the classification space underlies the classification process and much of the work done by the Osiris system. The classification Space of the p-type PERSON is shown Fig. 1.

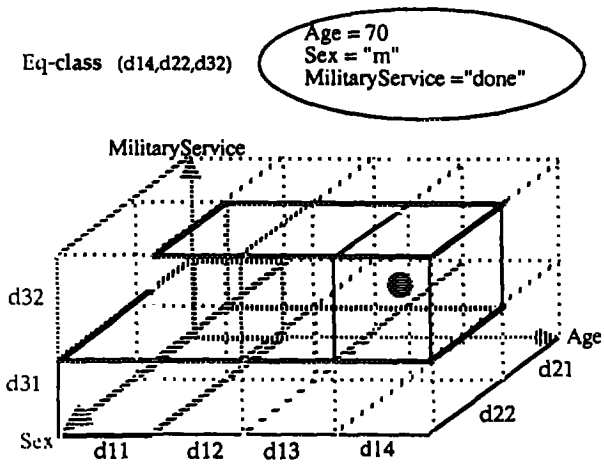

Fig. 1: Eq-classes of the class PERSON

Domain-based Constraints

We first consider instance classification in views defined by domain-based constraints, i.e., domain constraints (D) and inter-attribute dependencies (I).

Instance classification is performed by an automata with a connectionist architecture where all the nodes and arcs are determined statically (there is no learning). SSDs of the classifying attributes constitute its input layer, and views constitute the output layer. We present below the principle automata (Fig. 2) although it is not used in practice because of its size due to the explicit representation of the eq-classes. However, it is simpler to understand than the actual network, where the eq-class layer is replaced by two intermediate layers representing the logical connectives of the constraints.

For a completely known object, only one cell of the eq-class layer will be active, that representing its eq-class, and corresponding to its attribute values. Active cells have an output value 0, others 1.

The output function of the eq-class layer is the product of its input values. So, the active eq-class cell is that whose input is 1×1×...×1. The other cells have at least one input value which is 0. There are as many cells as the Cartesian product of the SSDs. All the connections are not represented on the diagram of Fig. 2.

The output function of the views layer is the sum of the input values of a view cell, thus expressing that a view is valid if and only if the object being classified belongs to one of the eq-class which validates that view. So, all the views which are connected to the valid eq-class are themselves valid and have an output value 1; the other views are invalid and have an output value 0.

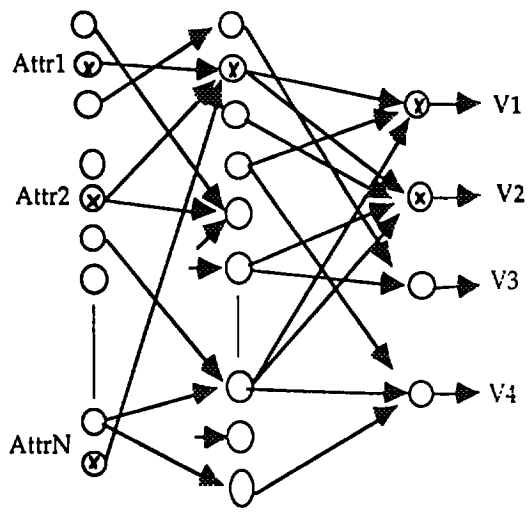

Fig. 2 : Principle Classification Network

In the case of object updating, when the modified attribute does not change its SSD, it is not necessary to perform further classification because of the stability property of SSDs: the object continues to satisfy exactly the same views.Therefore, following most updates, instance classification will be reduced to checking that the attribute remains in the same SSD.

The input to this automata is 0 or 1 for each SSD and the output is a value in [0,1] for each view. 0 and 1 represent valid and invalid views respectively. Given incompletely known objects, intermediate values represent views which are "possible", i.e., neither valid nor invalid. The size of the automata is polynomial to the number of classifying attributes.

Historic Constraints

The historic constraints we consider are expressed in Past Propositional Temporal Logic (PPTL). For a more comprehensive presentation of historic constraints see, e.g. [3] [4] [5]. Informally speaking, PPTL is propositional logic extended with two temporal operators: $\Box$ interpreted as « always in the past » and $\Diamond$ interpreted as « once in the past ». The universal nature of $\Box$ and the existential nature of $\Diamond$ make these two operators dual by the relation $\Box A \equiv \neg \Diamond \neg A$ (or $\Diamond A \equiv \neg \Box \neg A$). Therefore, the PPTL syntax only requires one of these two operators. It is common practice however to consider them both.

For example, to define in Osiris a view MONSTER to express that it is an animal whose weight has never come under 400kg one it attained the size of 20m, one would write the assertion:

$\Box (\Diamond \text{size}>20 \Rightarrow \text{weight}>400) \wedge \Diamond (\text{size}>20)$

PPTL – Consider a set Vp={p,q,...} of propositional symbols, the set $\{ \wedge, \vee, \Rightarrow, \Leftrightarrow \}$ of binary operator symbols, the set $\{ \neg, (), \Diamond, \Box \}$ of unary operator symbols and the two boolean **true** and **false**. Formulas of PPTL on Vp are defined as follows :

(*i*) **true** and **false** are formulas ;
(*ii*) any element of Vp is a formula ;
(*iii*) if F and G are formulas, then so are $\neg F$, $F \wedge G$, $F \vee G$, $F \Rightarrow G$, $F \Leftrightarrow G$, $\Box F$, $\Diamond F$, (F);
(*iv*) nothing else is a formula.

Temporal logic formulas are classically interpreted on models made of a 3-uple M=<W,R,m> (Kripke frames). In this 3-uple, W is a set of dates, R is a binary relation on W named relation of temporal precedence and $m: W \rightarrow P(Vp)$ is a function from W to the powerset of Vp. This function associates to each date the set of propositional symbols which evaluate to **true** at this date.

A Model for Evaluation – Given a historic assertion H with n elementary propositions p_i, $Vp=\{p_1, \ldots, p_n\}$. As each p_i can be evaluated to **true** or **false**, we can consider the set H_Σ of truth-vectors which is the cartesian product

$$\prod_{i=1}^{n} (\textbf{true},\textbf{false}) \text{ or more simply } \prod_{i=1}^{n} (+,-).$$

Each element of H_Σ is called a *truth-vector*. For example, if H is made of two elementary propositions p_1 et p_2, $H_\Sigma = \{ (+,+), (+,-), (-,+), (-,-) \}$.

Any state of an object O is associated with a unique truth-vector of H_Σ. For example, if a state makes p_1 true but not p_2, then this state is associated with the truth-vector (+,–). Therefore one can canonically associate to S(O) a unique sequence of truth-vectors noted $S_\Sigma(O)$. Now, the truth value of H in the current state only depends on the truth value of its elementary propositions on S(O) and therefore only depends on $S_\Sigma(O)$ which constitutes the model of interpretation of H. Taking the notations of the last §, one has :

W: the set of states of object O ;

R: the order relation on W which describes the sequence S(O) ;

m: the function which associates to each state of the object O the corresponding truth-vector.

Evaluation – A naïve evaluation of historic assertions relies on the examination of the past states of objects. The method we propose does not require accessing the past states of an object (*history-less* method), and its cost is therefore low and constant. It consists in associating a finite state automaton to each historic assertion. Each object has a pointer on this automaton which indicates its current state. When the object is updated, this pointer is moved accordingly. If the current state corresponds to a final node of the automaton, the historic assertion is verified, otherwise it is not.

Let H be a historic assertion. L(H) is the language made of the sequences of H_Σ for which H evaluates to **true**. For any historic assertion H, one proves (see [5], chapter 10) that L(H) is regular and given by the following compositional formulas :

$(i)\ R(p_i) = [\pm]^* . [+_i]$ $\quad (ii)\ R(\neg F) = [\pm]^+ - R(F)$

$(iii)\ R(F \vee G) = R(F) \cup R(G)$ $\quad (iv)\ R(\Diamond F) = R(F) . [\pm]^+$

As a consequence, for any historic assertion H there exists a regular expression R(H) which describes the language L(H). For example, for the historic assertion

$\Box(\Diamond p_1 \vee p_1)$, one gets $[\pm]^+([\pm]^+([\pm]^* . [+_1] . [\pm]^+) \cup ([\pm]^* . [+_1]))) . [\pm]^+)$

According to the definition of L(H), H evaluates to **true** for an object O iff $S_\Sigma(O) \in L(H)$. According to the preceding theorem, it is equivalent to check that $S_\Sigma(O) \in R(H)$. Evaluating a historic assertion is therefore equivalent to checking that a word belongs to the language described by a regular expression. It is quite usual to use finite state automata to perform this task.

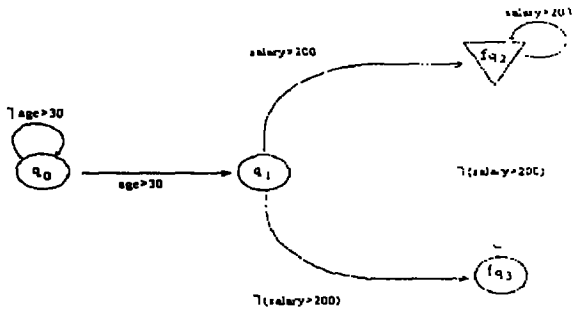

Fig. 3: Automaton for historic constraint (h1)

Indeed, it is a well known result that for any regular expression there is an automaton which recognises the same language. Since a non-deterministic automaton can always be translated to a deterministic one, we only consider here deterministic automata. The automaton for R(h1) shown Fig. 3 was generated with the Rank Xerox finite state compiler (http://www.rxrc.xerox.com/research/mltt/fst/fsinput.html). This automaton is made of three nodes. One of them, fq_2, is a final node.

Finally, the proposed method of evaluation for historic assertions has two parts :

i *building of the finite state automata* (one per historic assertion). This is performed by first translating the historic assertions to regular expressions and next to automata. A pointer to each of these automata is then added to each object. The pointers initially point to the initial node of the corresponding automata.

ii *monitoring*. After each update on an object, its pointers are moved according to the truth-vector associated with the new state of the object (since automata are deterministic only one new state is possible). If the new node reached is a final node, then it means that the corresponding historic assertion evaluates to true.

Classification – In view definition, historic assertions act as boolean attributes, therefore extending the classifying space by one dimension. For example, adding the view MANAGER to the P-type EMPLOYEE would result in adding an axis for the historic assertion (h1) in the Classification Space of Fig. 1. This axis would only be divided into two parts : **true** and **false**. This easy embedding of historic assertions in the Classification Space makes possible to consider views defined by static and historic assertions at no additional cost.

Linear Binary Constraints

As the treatment of domain-based constraints is highly optimized, classification w.r.t. more complex constraints is performed only on views already recognized as satisfying the domain-based constraints. In the following we consider linear binary constraints but the approach can be extended to more general constraints.

The classification space can further be used for the determination of views defined by linear constraints. Given a view with domain-based constraints and linear constraints (L), the partitioning of the object space by the constraints of type (D) and (I) defines eq-classes where the objects have the same comportment towards these constraints. Each eq-class may also be qualified w.r.t. a linear constraint Li(X,Y) as valid (v), invalid (i) or to be checked (?), as illustrated by Fig. 4. Valid (resp. invalid) eq-classes are entirely on the valid (resp. invalid) side of the constraint, wheras those to be checked intersect the constraint.

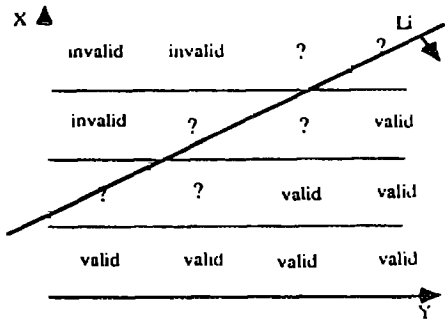

Fig. 4 Linear constraint and eq-classes

When an object belongs to a (v) or (i) eq-class, its validity remains unchanged as long as the object remains in the same eq-class. Therefore, when an object is updated, if it does not change its eq-class, it is not necessary to classify it again. Only when its eq-class is (?) the constraint has to be checked. The set of (?) eq-classes of a constraint (L) is called its triggering set.

One straightforward method to check linear constraints consists in extending the classification automata with a "constraint" layer containing one cell for each linear constraints in the p-type. The input of the cell for a binary constraint L(X,Y) is made of its triggering set of eq-classes and of the values of the two variables X and Y. Its output is 1 if the constraint is satisfied or not triggered, 0 otherwise. In the following, the constraints of a view include both own and inherited constraints.

In the classification network, each output cell corresponding to a view with linear constraints is replaced by an intermediate cell V_D, connected to the eq-classes of the view, which reflects the satisfaction of domain-based constraints. The link between V_D and a constituent eq-class is suppressed if the eq-class is invalid for at least one of the linear constraints of the view. The output cell for V is connected to V_D and to each cell corresponding to a constraint of the view. The transition function of the output cell is the product of its input values, i.e., its output is 1, meaning the view is satisfied, iff the domain based constraints are satisfied (output of V_D is 1) and all the linear constraints are satisfied.

Through this method, a constraint is evaluated only when the object to classify belongs to an eq-class of its triggering set. Moreover, when the object is modified while remaining in the same eq-class, it needs not be reclassified if this eq-class does not belong to any linear constraint triggering set.

Another approach has been explored in [7]. It consists in representing the eq-classes and the constraints by graph micro-structures, following Jegou's approach [8]. The subdomains are used instead of single variable values to build the micro-structures. The principle has been presented in [6].

Conclusion

In this paper we have presented a method for instance classification based on a static analysis of the properties defining the classes. In the case of domain-based constraints this analysis leads to a partitioning of the object space into a so-called classification space which guides the classification process, and also the representation of persistent objects [10]. The principle network which has been presented is not used in the Osiris system, but is replaced by a network based on the assertions, whose size is polynomial to the number of attributes [1]. The extension to this network for the linear constraints is the same as that presented above.

The approach using the micro-structure is a promising one which has still to be evaluated and compared with the one based on the classification network.

References

1- Bassolet, C.-G.; Simonet, A.; and Simonet, M. 1996. *Probabilistic Classification in Osiris, a View-based OO DBMS and KBMS*, In : DEXA'96, 7th International DEXA Workshop on Database and Expert Systems Applications, Zurich, pp 62-68.

2- Borgida, A,. and Brachman, R. 1993. *Loading Data into Description Reasoners*, in Proc. of the ACM SIGMOD International Conference on Management of Data, pages 217-226, Washington, DC, June 1993.

3 Chomicki, J. 1995. *Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding*, ACM Trans. on Database Systems, Vol.20, No.2.

4- Delannoy, X.; Simonet A.; and Simonet, M. 1996. *Database Views with Dynamic Assertions*, Third Int. Conference on Object-Oriented Information Systems (OOIS), Springer Verlag Ed., London, December 1996. *http://curie.imag.fr/~delannoy/perso/publications.html*

5- Delannoy, X. 1997.*Contributions to the study (i) of the tension between of consistency and confidentiality and (ii) to the classification of objets according to their history, in databases* , Doctoral Thesis, University of Grenoble, France.

6- Hamadi, R.; Simonet A.; and Simonet, M. 1997. *Domain Decomposition Methods to solve CSPs*, FLAIRS 97.

7- Hamadi, R. 1997.*Méthodes de Décomposition de Domaines pour la Résolution des CSPs*, Doctoral thesis, University of Grenoble, France.

8- Jegou, Ph. 1993. *Decomposition of Domains based on the Micro-Structure of Finite Constraint Satisfaction Problems*. In: Procs of the AAAI Conf., Washigton DC, pp 731-736.

9- Simonet, A. and Simonet, M. 1995. *OSIRIS : an Object-Oriented System unifying Databases and Knowledge Bases*, KBKS 95 : Toward very Large Knowledge Bases, IOS Press, pp 216-227.

10- Simonet, A. and Simonet, M. 1994.*Objects with Views and Constraints : from Databases to Knowledge Bases*, Int. Conf. on Object Oriented Information Systems (OOIS'94), London.