# Interactive Class Classification Using Types

Cécile Capponi[1]

[1] INRIA Rhône-Alpes, LIFIA-IMAG,
46 av. Félix Viallet, F-38031 Grenoble Cedex 1, France
tel: (33) 76 57 47 99
e-mail: Cecile.Capponi@imag.fr

**Summary:** *An object-based knowledge representation (OBKR) model organizes knowledge base into hierarchies of classes, each class being described by a set of slot specifications that the instances of the class must satisfy. Each hierarchy is given by the* specialization relation. *Class classification is a mechanism which computes the position of a class into the hierarchy, once this class has been described or modified. Class classification has to be seen as a knowledge base construction tool, while instance classification is an inference mechanism. Class classification has to be an interactive process and its results have to be correct according to the specialization relation definition. Thus, the structure of the knowledge base is correct and can support inference mechanisms. However, specialization semantics is not always clear and must be formalized. This is done by introducing types in an OBKR model, at the level of slots. At this point, a new structure is considered, namely the slot type graph, which allows the decomposition of processes working on class structures along the slot specifications of classes; such a decomposition is quite natural and speeds up all of these processes.*
**Key words:** *Knowledge representation, class, slot, specialization, subtyping relation, normalization.*

# 1 Introduction

This work stands into the object-based knowledge representation (OBKR) system area, with a class/instance approach. These systems distinguish *generic* knowledge, represented by classes of instances organized into hierarchies ordered by the specialization relation, and *specific* knowledge (instances). Class classification is a tool used to implement *structural* acquisition, i.e. acquisition of generic knowledge, which is an important step of the incremental knowledge base construction.

Due to the great size of a knowledge base, the addition of a new class is difficult: to give the class description and to find its correct position are complicated because the user does not know exactly the whole base entities, and might thus introduce some inconsistencies. It is proper to have an automation of the structural acquisition process, at least a partial one: it will be used both to make the description of the new class easier and to find its correct position in accordance with the specialization relation.

In this paper, the term *classification* is not used to design a hierarchical structure; in OBKR area, *classification* designs the *mechanisms* used to compute the relevant position of an object into hierarchies according to a particular relation. Thus, generic knowledge classification has to be distinguished from specific knowledge classification, unlike classification performed by terminological languages [BMPS+91] which joins classes (concepts) and instances (individuals) for the same algorithm based on the subsumption relation [Woo91].

Assuming that the categorization is a process which constructs a class definition from a set of instances, it is not related to class classification process which is applied to a given class description in order to find its position into a hierarchy.

# 2 Object-based knowledge representation models

OBKR and manipulation systems model the world in terms of object classes which are organized into hierarchies, ordered by the specialization relation. The term *classification* sometimes designs these hierarchies, while OBKR systems extend the word *classification* to all the mechanisms which work essentially on the hierarchies. More exactly, the action *to classify* becomes a powerfull process to organize the structure of a knowledge base, i.e. to establish the hierarchies.

## 2.1 Description and organization of the entities

The important entities of OBKR are the notions of classes, instances, slots and specialization. This section shows the relations there exist between these entities.

### 2.1.1 Class, instance and specialization

A class describes the common structure of a set of instances. This structure, called the class description, is made up of a record of slot definitions. A slot is defined by its name, its domain of possible values, and possibly by procedural attachments and default values. A slot value must belong to the domain of this slot. A class instance is defined by its name and a record of values corresponding to a subset of the slots defined in the class description. The extension of a class is its set of instances.

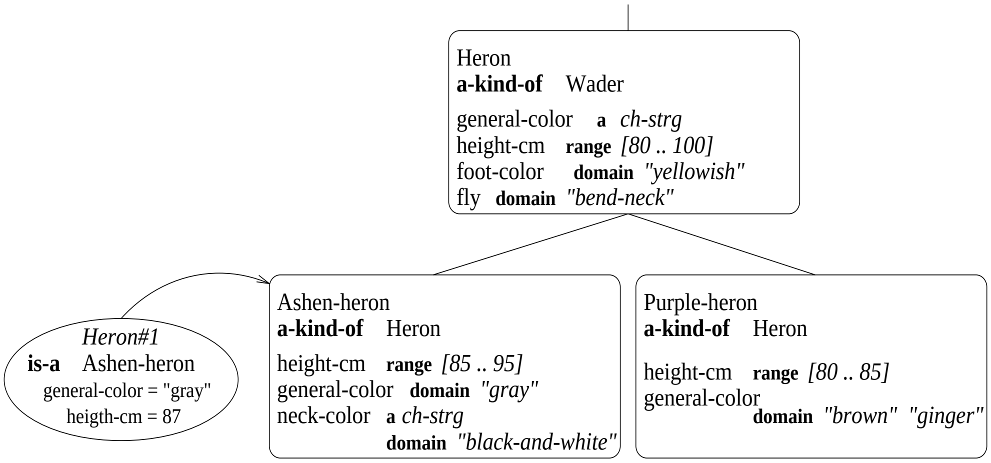


Figure 1: Classes, instance and specialization. *Heron#1* is an instance of the class *Ashen-heron* (**is-a** link). The class description expresses common properties of instances by the way of slots. An instance of this class gives value to slots, according to the specification. The **a-kind-of** link represents the specialization relation: the class *Ashen-heron* is a sub-class of *Heron*, i.e. the former is more specific than the later; thus *Heron#1* is also an instance of *Heron*.

Classes are organized according to the specialization relation. A class is more specific than its super-classes in the sense that it contains more informations, and/or more accurate ones, on the instances. Due to the link between classes and instances, namely instantiation, let us give the definition by extension of the specialization relation: *If class* C *is a specialization of class* C' *then every instance of* C *is an instance of* C'.

The *specialization relation* leads to the construction of the specialization graph, which gives the representation of the specialization relation and of its transitivity property.

### 2.1.2 Slot and class specialization

A class description defines a slot by associating to its name a complete or partial specification of its domain. The complete class description is built up by the inheritance mechanism; this mechanism considers that a class inherits slot definitions from its super-classes and it completes these definitions by the local ones. A class description may define new slots or/and give more specific (constrained) domains to the existing slots inherited from super-classes.
These conditions on slot definitions between a class and its super-classes gives the definition by intension of the specialization relation: *Class* C *can be a specialization of class* C' *if, for each slot* s *of* C', C *contains also* s *and the domain of* s *in* C *is included into the domain of* s *in* C'.

The specialization relation has to be semantically well defined to allow its checking. However, such a clear semantics does not exist in object-oriented knowledge representation. A formalization is thus needed, and should be at the level of slot definitions since they allow the checking of the intension of specialization relation.

## 2.2 Classifications

A classification may design a structure which organizes objects such that common properties of these objects are explicitly highlighted. However in OBKR area, the classification designs a powerfull reasoning *mechanism*: confronted to a new situation, the human being does classify it in a known class of situations. Expert systems follow this same reasoning principle: they classify a new problem in a virtual class of known problems to get some guides for finding a general solution which they refine using also a classification process.
Thus, classification *mechanisms* are used to elaborate the classification *structure*, that is to say to construct and organize the knowledge base entities. A this point, we distinguish the classification structure from two kinds of classification mechanisms: individual classification (as an inference mechanism) and class classification (as a construction tool).

### 2.2.1 Instance classification

The *instance classification mechanism* leads to classify a new instance in the class structure. Its outcome may be the degree of matching between the instance and each class or the determination of the class as being certain, possible or impossible [MRU90]. A class is *certain* if the instance satisfies all its constraints, it is *impossible* if there is a contradiction between an instance slot value and the class constraints, and it is *possible* if the instance is not in contradiction with the class but one cannot ensure its belonging to the class [Rec89].
Thus, the attachment of an instance to a class is not definitive and can change dynamically, even if the user does not know the right new class for the instance. Instance classification leads to "specialize" an instance which has been previously attached to a class anywhere in the hierarchy. This specialization allows to infer new slot values for that particular instance which are statically available from the new class.
Instance classification is an inference mechanism since it allows the deduction of information on individuals. In that sense, the instance classification is closer to the individual classification developped by the description logics community [BS85]. However, individual classification is generally achieved by computing the generic concept corresponding to the individual and then by finding the most specific subsumers of this concept.

### 2.2.2 Class classification and categorization

Due to the evolution of the world to be represented and the partial perception we have of it, a knowledge base is continuously modified; thus, having tools for the verification and the maintenance of links between knowledge base entities is very important. Indeed, these links (e.g. the specialization link between classes) hold up the consistency and the correctness of a knowledge base, and are challenged whenever a modification occurs on a structural entity. A management performed by the user could be very dangerous, even impossible, whenever the size of the knowledge base reaches a certain threshold, like for instance KB into the domain of molecular biology whose size raises more than 4000 classes [Per91].
Class classification allows to assist the user in adding a new class by giving him facilities to describe the structure of its class and by performing the computation of this new class position into the hierarchy: as the user gives the slot specifications, the system computes possible positions for the class, relatively to the described slots. Once the definition of the new class is complete, the system provides the final position, which is correct according to the specialization relation (definition 2).
The class classification is different from the categorization process since the static definition of the class is known. The task of class classification is not the construction of the specification of a class from a set of potential instances, but it is the finding of the right position of the known class. Instances are ignored by the class classification process which performs on the knowledge base structure. Once the class classification is achieved and validated by the user, the system activates the instance classification process on instances of super-classes of the new one.

# 3 Class classification: a correct and interactive approach

As it has already been said in section 2.2.2, class classification is a process which computes the position of a class $C$ in a hierarchy of existing classes, in accordance with the specialization relation. Such a computation must be performed over the definition by intension of the specialization. Indeed, that definition ensures the structural coherence of the hierarchy, allowing correct algorithms of instance manipulations. The definition by intension of the specialization shows that any mechanism performed on class descriptions in accordance with the specialization relation can be projected on the declarative definitions of the class slots.

## 3.1 The class classification algorithm

The class $C$ classification is implemented by decomposing it along $C$'s slots, i.e. by introducing slot types to represent structural definitions. The intersection of the relative positions of $C$ according to each of its slots gives the correct position.

### 3.1.1 Type graph of a slot

The type of a slot $s$ in a class $C$ is obtained first by reprocessing facet values which specify the domain of $s$, then by normalizing this domain [Cap93]; the normalization process is essential to allow further comparisons between types: indeed, there are several possibilities to express a value domain using facets of the representation language. For example, the value domain expressed by `nb-legs a integer range [0 8] except 1 3 5 7` is equivalent to the value domain expressed by `nb-legs a`

`integer domain 0 2 4 6 8`; normalization allows to compute a normal expression for the domain and then to compare two of these expressions we call *types*, without reducing the expressive power of the language. $T_C(s)$ is the type of slot $s$ in class $C$. By extension, we consider $T_C(s)$ the type of a class $C$ as the record of the types of the class slots, as in [AC91]. Several classes can give different definitions for a same slot name.
Then, for each slot $s$ the *type graph* of $s$ can be constructed. Graph nodes represent the different types of $s$ to which lists of classes giving the corresponding type for $s$ are associated. A link between two nodes represents the *subtyping relation* (noted $\leq_\tau$), the computation of which is assumed to be tractable. The sub-typing relation allows to check the specialization relation, due to the following condition: if $C \leq_{sp} C'$ then $T_C(s) \leq_\tau T_{C'}(s)$. The subtyping relation, both in the cases of classes and slots, has to be equivalent to the domain inclusion relation.

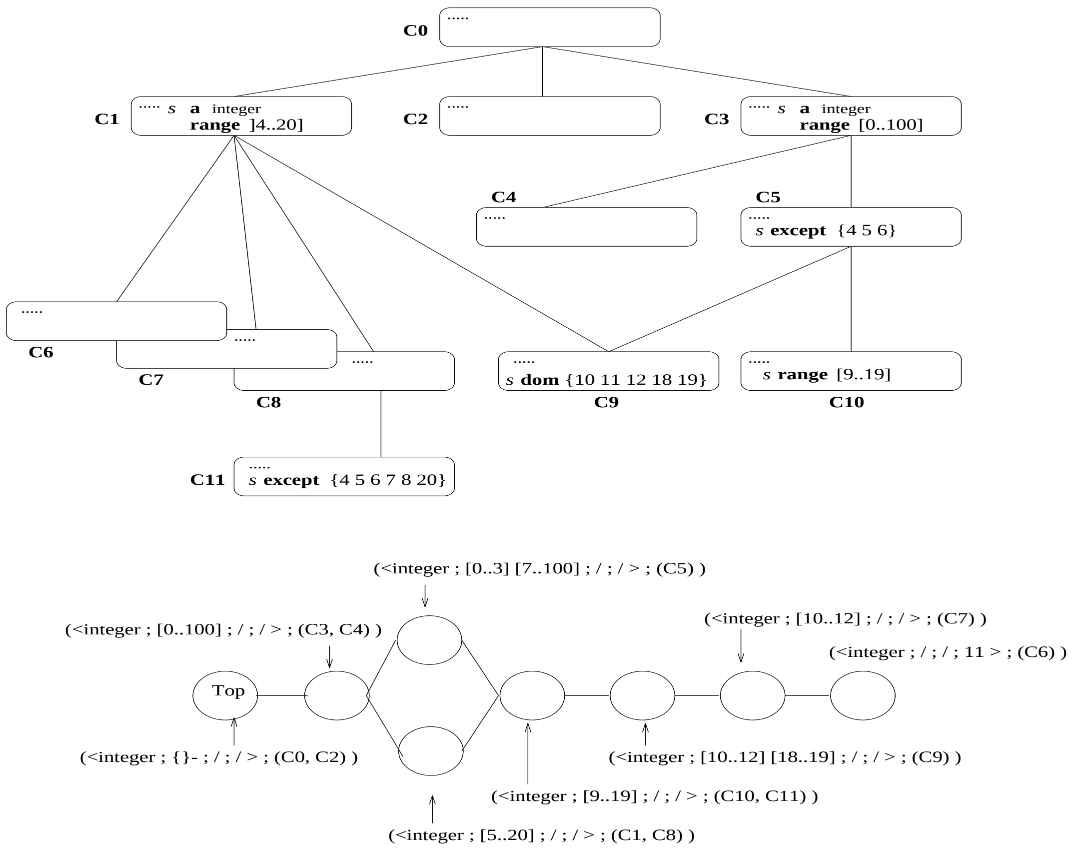


Figure 2: The specialization graph (for readability, only specifications of the slot $s$ are shown) and the type graph of $s$. The type graph shows all the subtyping links existing for $s$. Thus, the type graph allows the storage of structural information about the specialization relation, relatively to $s$.

### 3.1.2 Position of a class according to one slot

Each node of the type graph $GT_s$ of slot $s$ contains the list of the classes which give that type for $s$. Thus, provided with the type $t_0$ of $s$ into the class $C$ to be classified, $t$ is inserted into the type graph of $s$ and the position of $C$ in the specialization hierarchy relatively to $s$ is obtained by the following expressions:

$$sup(s,C) = \bigcup_{\substack{(t,l)\in GT_s \\ t \geq_\tau t_0}} l \qquad \text{and} \qquad inf(s,C) = \bigcup_{\substack{(t,l)\in GT_s \\ t \leq_\tau t_0}} l$$

Thus, the type graph $GT_s$ provides both the correct super-classes and sub-classes of the new class $C$ containing $s$, only with regard to $s$.

### 3.1.3 Computation of the final position of a given class definition

The definition of a class depends on several slots. Thus the position of a class in the specialization graph is computed by intersecting the available relative positions for each slot. However a sub-class $S_1$ of a class $S_2$ has to satisfy the complete inheritance property:
$C_1 \leq_{sp} C_2 \Rightarrow [\forall s \in A(C_2), s \in A(C_1)]$, where $A(C)$ is the set of the slot names existing in the class $C$. Thus, possible super-classes of a class $C$ and its possible sub-classes are obtained from the following expressions:

$$sup(C) = (\bigcap_{s \in A(C)} sup(s, C)) \quad \bigcap \quad \{C'/A(C') \subseteq A(C)\}$$

$$inf(C) = (\bigcap_{s \in A(C)} inf(s, C)) \quad \bigcap \quad \{C'/A(C) \subseteq A(C')\}$$

The last result provides the possible position of the class schema $C$. It may contain several solutions of super-classes. In the case of simple inheritance, the user has to choose one super-class; such a choice is also required in the case of sub-class conflicts (if $C_1$ is a possible sub-class of $C$ and $C_2$ is the current super-class of $C_1$, $C_2 \notin sup(C)$, which one of $C_2$ and $C$ should be the new sub-class of $C_1$ ?).
However, all the provided super-classes are acceptable they can thus be viewed as related solutions. In the case of multiple inheritance, the optimal position can be easily obtained without the user intervention by minimizing the solution according to the specialization transitivity property.

## 3.2 Example

The example on figure 3 is presented in order to show the interest of the slot type graphs in performing the class classification. Indeed, once the type graphs are available, the computation of the new class possible position leads to slot type insertions and list intersections.
The methods suggested in this paper were developed over the Shirka [Rec89] knowledge representation model and tested on knowledge bases containing more than 700 classes. These tests showed the real efficiency of the creation and insertion process both at descriptional and consistency checking level. Now, a type system is implemented over the multi-point of views model Tropes [GGS93]. This system is the main support of all structural modification management tools.

## 3.3 Some advantages of the type graph structure

To define class slots in an OBKR model, it is quite natural to use a formalism over types, since slot domains represent sets. Moreover, such a formalism provides a high abstraction degree, relevant to implement validation processes, since such processes work on the knowledge base structure [CC93]. Besides, such a study has already been recently done in the domain of terminological language, as a way to specify description logics [Bor92].
Slot type graph, first introduced to manage the class classification, provides another view of the specialization graph, introducing a new interactive power useful whenever

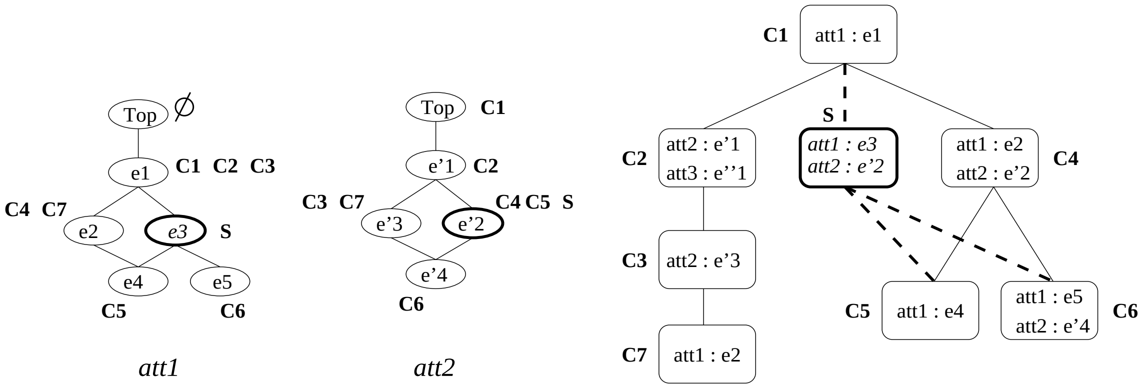


Figure 3: The new class $S$ has to be classified. It contains two slots, $att_1$ and $att_2$. $S$ gives to $att_1$ the specification whose type is $e_3$; this type is a new one and thus is inserted in the type graph of $att_1$, and the information about relative position of $S$ according to $att_1$ is retrieved: $S$ can be a sub-class of $C_1$, $C_2$ and $C_3$ and a super-class of $C_5$ and $C_6$. The same process is performed with the slot $att_2$ for which the user gives an existing type. Due to the non-strictness of the sub-typing relation, the position of $S$ according to $att_2$ is $C_1$, $C_2$, $C_4$ and $C_5$ as possible super-classes and $C_6$, $C_4$ and $C_5$ as sub-classes. The intersection of these two solutions gives the partial position, reduced by complete inheritance and transitivity, to obtain the final possible position ($C_2$ cannot be a super-class due to the complete inheritance property).

a user tries to perform modifications on the knowledge base structure [Fin86]. Indeed, seeing the class graph from the viewpoint of only one slot at a time allows to highlight the relationships between classes which do not appear within the specialization graph since they do not exist relatively to other slots. On one hand, a type graph is a support for describing type specifications of the new class; it helps the recognition of existing specifications of a given slot name, independently from other slots. On the other hand, the type graph encloses the complete sound information about the corresponding slot, since its construction is closely related to the subtyping relation, formalization of the specialization. Thus, it is obvious to prove the soundness of the classification algorithm according to the specialization semantics, and by extension to prove other validation management processes.
Even if it can be parallelized, a slot type graph computation is time-consuming. However, the best way is to construct and memorize all the slot type graphs as long as the base grows up using class classification: class classification then becomes faster and type structures are always available to manage the correction of base structure modifications.

# 4 Conclusion

Performing class classification by the way of slot type graphs offers a lot of advantages. It is achieved by decomposing the classification problem into $n$ successive sub-problems ($n$ is the number of slots of the new class); $n$ successive slot classifications are performed, then the intersection of computed solutions is considered. Such a decomposition allows slot definition and corresponding class classification tasks to be simultaneous; this is important in order to understand result explanations and error messages, since they are provided in the definition context.
The class classification process is a special case of knowledge base modification. In fact, each structural base modification checking could be projected over slots; subtyping relation is a natural projection of the structural constraints of the specialization relation, so a natural formalization of it. Thus subtyping relation allows the checking

of the base structural consistency.
Beyond the specialization relation formalization, types could allow the formalization of many other object relations, such as composition or instantiation, since the formal definitions of all inference and deduction mechanisms which can be applied to representation model structures.
In conclusion, types seem to be well suited to perform a partial formalization of object-oriented model in the domain of knowledge representation, allowing the performing of many processes working on slot domain specifications, as for example class classification or instance classification. In that sense, J. Euzenat works on the formalization of the generalization of classification and categorization processes [Euz93].

# References


[AC91] R.M. Amadio and L. Cardelli. Subtyping recursive types. In *18th POPL*, Orlando (FL, US), 1991.

[BMPS+91] R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L.A. Resnick, and A. Borgida. *Living with CLASSIC, When and how to use a KL-ONE-like language*, chapter 14, pages 401–456. Principles of semantic networks. Morgan Kaufmann, 1991.

[Bor92] A. Borgida. From type systems to knowledge representation: natural semantic specifications for description logics. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):93–126, April 1992.

[BS85] R.J. Brachman and J.G. Schmolze. An overview of the kl-one knowledge representation language. *Cognitive Science*, (9):171–216, 1985.

[Cap93] C. Capponi. Classification des classes par les types. In EC2, editor, *Représentations Par Objets*, pages 215–224, La Grande Motte, France, June 1993.

[CC93] C. Capponi and M. Chaillot. Construction incrémentale d'une base de classes correcte du point de vue des types. In *journée Validation, JAVA*, Saint-Raphaël, France, March 1993.

[Euz93] J. Euzenat. Définition abstraite de la classification et son application aux taxonomies d'objets. In *Représentations Par Objets, Journée Classification*, pages 235–246, La Grande Motte, France, 1993.

[Fin86] T.W. Finin. Interactive classification: a technique for acquiring and maintaining knowledge bases. In *IEEE*, volume 74, October 1986.

[GGS93] J. Gensel, P. Girard, and O. Schmeltzer. Integrating constraints, composite objects and types in the knowledge representation system. In *5th IEEE International Conference on Tools with Artificial Intelligence*, Cambridge (MA, US), November 1993. to appear.

[MRU90] O. Marino, F. Rechenmann, and P. Uvietta. Multiple perspectives and classification mechanism in object-oriented representation. In Luigia Carlucci Aiello, editor, *9th European Conference on Artificial Intelligence*, pages 425–430, Stockolm, Sweden, August 1990.

[Per91] G. Perrière. Coligène, traitement de l'information des séquences biologiques. *Cahiers IMABIO*, 2:61–66, 1991.

[Rec89] F. Rechenmann. Shirka: un modèle de connaissance centré-objets. rapport interne, laboratoire ARTEMIS/IMAG, Grenoble, France, 1989.

[Woo91] W.A. Woods. *Understanding subsumption and taxonomy: a framework for progress*, chapter 1, pages 45–94. Principles of semantic networks. Morgan Kaufmann, 1991.