Brief overview of T-TREE: the TROPES Taxonomy building Tool

Jérôme Euzenat

INRIA Rhône-Alpes
IMAG-LIFIA, 46, avenue Félix Viallet, 38031 Grenoble Cedex 1 (France)
Jerome.Euzenat@imag.fr

ABSTRACT: *TROPES is an object-based knowledge representation system. It allows the representation of multiple taxonomies over the same set of objects through viewpoints and provides tools for classification (identification) of objects and categorisation (classification) of classes from their descriptions. T-TREE is an extension of TROPES for the construction of taxonomies from objects. Data analysis algorithms consider TROPES objects for producing TROPES taxonomies. Thus, data analysis is integrated into the knowledge representation system. Moreover, the original bridge notion permits the comparison and connection of adjacent taxonomies.*

KEYS: *Automated techniques to assist in creating classification scheme — Knowledge representation schemes — Classification algorithms — Software for management of classification schemes — Comparison and compatibility between classification schemes.*

TROPES is an object-based knowledge representation system. It is thus a general-purpose knowledge repository whose basic entities are objects (i.e. entities structured into fields and belonging to a class). Such a system is provided for integration into a wider application system which uses it for containing both the objects (data) and the inference mechanisms tied to the object semantics. The application system consists in a set of programs which access the knowledge base for consulting and modifying the objects that which, in turn, can trigger some of the inference methods (knowledge).

TROPES heavily relies on the classification mechanism (i.e. the mechanism which recognises the class to which an object can belong). The classes are hierarchically organised into taxonomies and, as knowledge about an object is acquired, it can be further classified under more specific classes. Classification is very important since the more specific the class, the more adapted to the object is the knowledge attached to that class.

The TROPES Taxonomy building Tool (T-TREE for short) is an extension of the TROPES object-based knowledge representation system enabling the integration of data analysis and automatic classification methods for the construction of object taxonomies. As a matter of fact, an important part of data analysis research is devoted to automatic classification.

Here, we show how this integration can be achieved and what are the benefits which can be expected from the integration. In particular, it is shown how T-TREE takes advantage of exclusive features of TROPES such as viewpoints and bridges. Further on, the integration can be seen as an important outlet for automatic classification algorithms in directly fetching the data and storing the result of classification in the place where it is used: the knowledge base.

The first section presents the TROPES system as representative of available object repositories but also as proposing some special features. Section 2 is devoted to the classification and categorisation mechanisms in TROPES. It emphasises the importance of the taxonomies in the TROPES system. Section 3 presents the principles of T-TREE: how the TROPES notions can be accounted for by data analysis algorithms, how these algorithms can produce TROPES expressions and how the control of the algorithms can be passed on to the user. Section 4 describes the use of a special feature of TROPES (bridges) for correcting and comparing the results of the data analysis phase.

in: Philip Smith, Clare Beghtol, Raya Fidel, Barbara Kwasnik (eds.), Proc. 4th ASIS workshop on classification research workshop, Columbus (OH US), pp69-87, 1993

1. TROPES

TROPES is an object-based representation system which favours classification. It is here presented through its basic notions. As TROPES is still under development, references to previous experiences [4, 8, 15] are made on the basis of its ancestor SHIRKA.

1.1. Objects and concepts

TROPES [10] is an object-based knowledge representation model (see [11] for a comprehensive introduction and review of such systems). This means that individuals are represented as objects.

The objects are partitioned into *concepts* (an object is an instance of one and only one concept). As an example, the `employee` concept concerns all the individual employees. Ontological prerogatives are attached to the concept. They are those which warrant the integrity of an object (i.e. that it cannot be modified in a way which would lead it to cease being an instance of the concept). Thus the concept defines the structure and the identity of its instances.

The structure of an object is uniquely determined by a set of *fields* and their basic domains independently of the classes to which the object can be attached. The fields relevant to objects are described in the concept. The instances of the `employee` concept have a `name`, a `salary` and a `publications` field. The basic domain of a field is either a primitive type (real, integer, string, boolean), a concept or a type constructed from these ones (and only them) with the help of set and list constructors.

Moreover, the field values are part of the objects and do not depend upon the classes to which they are attached. Thus, inference methods which represent implicit knowledge (valid deduction or value computation which must return the true value of a field) are attached to the concept itself.

The concept prerogatives also include the management of object identity. In TROPES, objects are identified with the help of a key: a list of field values which uniquely corresponds to one object. The usual consequences with keys are related to integrity: an object without a key cannot exist and there cannot exist two objects with the same key (thus accidental co-reference is avoided). Concepts also have in charge the management of existential dependencies from objects to components they cannot exist without (for example, it is not possible to change the hiring-date of some employee).

These prerogatives play an important role at instantiation when the object is created and registered. During its lifetime, an individual, created under some concept, is ensured that none of the above postulates is violated.

1.2. Viewpoints

Objects can be seen under several *viewpoints*. Some `employee` instances can be though of as `employees` from the `Accounting` viewpoint, as `researchers` from the `Functional` viewpoint, as `customers` from the `Restaurant`... The viewpoints allow to restrict the view on instances (as noted in [14], there are 970 fields (slots in CYC) in the person concept (class), thus users are often discouraged in their search for the adequate one) and organise the concept into a particular taxonomy. A viewpoint determines:

- The set of fields which are relevant under the viewpoint (the `publications` are not relevant from the `Restaurant` viewpoint as well as the `diet` under the `Functional` one, and thus the corresponding fields are hidden under the respective viewpoints).
- A hierarchy of classes under which the instances of the concept can be classified. Classes introduce constraints on the field values of their members. They are related through the specialisation relation and determine progressive subsets of the set of instances of the concept. This progression is parallel to the strengthening of the constraints added to sub-classes. Under the `Functional` viewpoint, `employees` are divided into `engineers`, `researchers`, `clerks`, and other classes; `researchers` can

be divided into `directors`, `assistants` and `PhD students`. An object is attached to (to be opposed to "is member of") only one more specific sub-class under a viewpoint, but is member of all the classes of which this class is a specialisation. Each viewpoint offers to the user a new taxonomy under which the classification operation depends on different criteria. They allow to focus on particular aspects of objects without being disturbed by others.

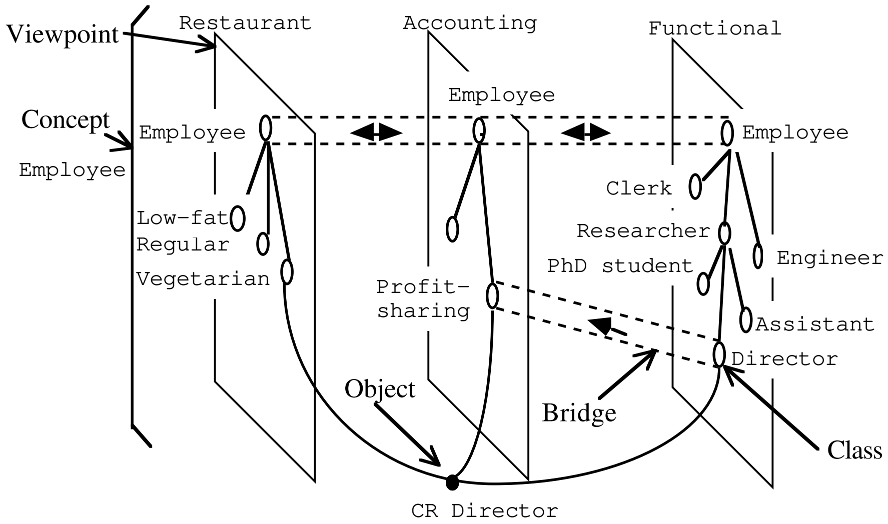

Figure 1: The `employee` concept is visible under the `Restaurant`, `Accounting` and `Functional` viewpoints. Each of them determines a hierarchy of classes whose root class is named `employee`. For example, under the `Functional` viewpoint there is a decomposition of the set of employees following their functions. The `CR Director` object is attached to the `vegetarian` class under the `Restaurant` viewpoint, to the `profit-sharing` holder under the `Accounting` viewpoint and to the `director` class under the `Functional` viewpoint.

The advantages of multiple taxonomies are manifold:

- Under a particular viewpoint, an object is attached to only one class (this was motivated by the argument that an object belonging to two unrelated classes is so under different viewpoints [10]). This organisation is very useful since, from a particular viewpoint, one can classify an object with regard only to the relevant fields. As a consequence, the set of classes to be considered is small, while considering the whole lattice of possible structures (e.g. `Vegetarian-Profit-sharing-Clerk`) would confuse the user.
- It allows to access taxon of a particular taxonomy from a specialised access device such as dichotomous diagnostic key [13] or decision trees. In [8], two decision methods have been used in connection to zoological classification. It is also suggested that the viewpoints can be specific geographic areas which drastically restrict the set of available species and allow to build faster and simpler restriction key systems.
- It is sometime useful to use different related taxonomies. This is the case in genetics when considering functional, chemical and evolutionary viewpoints on genes: there is a taxonomy for each viewpoint. This also allows the representation of alternative taxonomies in the same system. The need for representing "diverse overlapping classifications" has already be accounted for in [1].
- It enables the building of a new taxonomy without discarding the current one(s) (when testing a new hypothesis about the knowledge manipulated by the taxonomist). For instance, from an already existing knowledge base containing an important set of objects, it is possible to run clustering algorithms in order to build another taxonomy on a new viewpoint and to compare it with other taxonomies.

1.3.Classes and taxonomy

A *class* determines the relevant fields of its members and defines constraints that objects must satisfy in order to belong to the class. It is a projection of the structure of the concept retaining only relevant fields and a restriction of the possible values of these fields. This is achieved with the help of:

- Primitive domain restriction provided by domain enumeration, exclusion or bounding (`diet` has a string value enumerated by `"regular"`, `"vegetarian"` and `"low-fat"`; the `salary` of a `clerk` can be between `£750` and `1400`).
- Attachment restrictions for concepts: the field values must not only be instances of a particular concept but can be constrained to belong to particular classes of that concept (the value of the field `project-director` of a `researcher` must be member of the `director` class).
- Constraints on field values. These constraints are membership constraints or constraints between fields (the `share` of an `employee` cannot exceed `20%` of (her)his `basic salary`).
- Cardinality restriction on sets (resp. lists) by bounding the cardinality. It is noteworthy that constructed field values are true sets (resp. lists) and not multiple values. It means that the value of a field is given by the corresponding set (resp. list) and not by a subset (resp. sub-list) of it which could be completed later.

The interpretation of the specialisation relation is twofold: first, the members of a class are members of its super-classes (extension inclusion property); second, the constraints defined in a class apply to all the members of that class (and thus to the objects attached to all of the less general classes). In the above example, it means that:

- all `PhD students` are `researchers`, and
- all `PhD students`, as `researchers`, are subject to the confidentiality clause.

But this does not imply the converse assertions. As a matter of fact, the interpretation of classes is descriptive, i.e. the satisfaction of the constraints associated to a class expresses a necessary but not sufficient condition for being a member of that class. The sufficient condition is the voluntary act of attachment produced by the user (or an application program).

As opposed to instantiation, objects can be attached to a class and can be detached from it at anytime. Classes also allow the expression of hypothetical knowledge in terms of default values or default inference methods. The hypothetical knowledge is relevant here since default values, for example, are given with regard to the specificity of the object class. These values are not to be seen as the true value of the field but as some hypothesis which can be made under a particular viewpoint. As a matter of fact, default values returned from different viewpoints do not have to be identical. For instance, it could be the case that research `directors` are usually fat while `vegetarian` are usually thin: what about a `vegetarian director`?

1.4.Bridges

As it has been said above, the classification of objects is considered as contingent. Thus, there can be several different classifications of the instances of the same concept: in a particular firm, the individuals are not considered the same way by the project management staff, the account office and the restaurant. Therefore the same individual is seen differently under several viewpoint and each viewpoint can have its own classification of the same set of individuals. However, viewpoints are not totally disconnected. *Bridges* are knowledge components which link a set of classes under pairwise disjoint viewpoints (called source) to a class in yet another viewpoint (called destination). A bridge means that an object which is member of each of the source classes is also member of the destination class. This allows, for instance, to infer that if an object is attached to the `director` class under the `Functional` viewpoint, it is member of the `profit-sharing` holders under the `Accounting` viewpoint (see figure 1). Bridges can be though of as either integrity constraints or inference mechanisms.

2. CLASSIFICATION AND CATEGORISATION

The aim of TROPES is not to be a taxonomic database system [1, 13] but an object representation system which provides tools for taxonomy management. However, TROPES has emphasised classification for long now [8, 9, 10] so that its classification mechanisms are elaborate. It provides classification and categorisation. The semantics of these operations are given in [6]. Here is an informal presentation of TROPES classification abilities.

2.1. Classification

The system is able, given a particular object *i* with several field values, to find *Cl(i)*, the set of classes (specialisation of its initial class) under which it can be attached (this is called "classification" in knowledge representation and "identification" in data analysis).

This is determined by an adequacy test which checks whether the field values of the object are included in the domain associated to the fields in the class and if the object as a whole satisfies the constraints associated to the class. The classification operation is based on a type interpretation of classes and algorithms which transform the classification process in a type inference process have been designed recently for TROPES [2]. However, some important particularities of the classification in object-based knowledge representation system must be noted:

- *The value of a field can be another object*. Since the field type can be a set of classes, this object must in turn be classified in order to determine if it can belong to the specified classes [9].
- *The value of a field can be unknown*. In such a case, it is possible that the algorithm cannot decide if the object satisfies the constraints of the class. Then the algorithm divides the set of classes into three: those which are known to be possible classes for the object (possible), those which are known not to be possible classes for the object (impossible) and the others (undetermined).

A correct classification algorithm could test any of the classes and check if the object satisfies the class description. However, the implemented algorithm usually takes advantage of the taxonomic structure and the respect of extension inclusion.

A classification algorithm checks, for a particular class the super-class of which is a possible or undetermined class for the object, if the type constraints are satisfied by the object. If some field values are missing (and need to be checked) the system can ask them to the user. For classic domains, this test is carried out the usual way (type checking). For an object value, if the object must belong to specified classes, the algorithm triggers a new classification process on that value in order to check if this is possible. If every constraint is satisfied for every field, then the class is declared a possible class for the object; if any of the constraints is not satisfied, then the class is declared impossible; if the super-class is undetermined or some fields still do not have a value (and the others satisfy the constraints) then the class is undetermined. Then, it can proceed on sub-classes if the class is possible or undetermined. The classification process also takes into account the bridges which allow to infer the membership to a class from the membership to other classes in other viewpoints.

For instance, if a company hires a new employee, the corresponding object is created in the information system. The system will find the object class under the `Functional` viewpoint with the help of its `function` but also its `diploma` and other constraints on research `directors`. Since the project of the director (another object) is classified as a `computer science project`, the employee is classified under those which must have a login account on scientific computers. Once the object is attached to the research director class, it can be inferred, with the help of the bridge, that the individual is a `profit sharing` holder (without knowing anything from the `Accounting` viewpoint). Since there is no value about the kind of `diet`, the classification is carried on under the `Restaurant` viewpoint by asking the value.

It must be noted that the simplification introduced by viewpoints allows to restrict the taxonomies under which an object is classified. Moreover, even if the classification takes into account all the viewpoints, the user can distinguish the viewpoints under which (s)he is able to answer queries from those under which (s)he is not (for example, if the new employee is not at hand, the user can tell that the questions about the `Restaurant` viewpoint cannot be answered).

2.2. Categorisation

Categorisation corresponds to what is called classification in data analysis or machine learning. However, it is still a bit different since automatic classification builds the class and its description from the set of objects attached to that class and categorisation starts from a class description and tries to find the place of the class in the taxonomy. This is related to incremental conceptual clustering [7].

The categorisation in TROPES consists in finding the set of more general and the set of more specific classes than the class to insert in the taxonomy. The only constraint that a taxonomy must always satisfy is the extension inclusion property. As a consequence, the class can be a sub-class of one (or more) of its more general classes and a super-class of some (or all) of the more specific classes of its super-class(es) which are also in its more specific classes. The number of classes to which the newly inserted class is connected depends on the characteristics of the representation system. In TROPES, it is always a sub-class of only one class.

Thus, the algorithm starts with the class description and a starting class (usually the top of the taxonomy). It goes down the taxonomy and compares the classes. Again, the potential extension of the classes are abstracted through types and sub-typing is regarded as the test that the extension of a class is included in the extension of another (and thus that the former is more specific than the latter). The search is proceeded until the considered class is no more general than the class to insert. When this has been done for all the branches of the tree, the last encountered classes in each branches are the most specific more general classes than the class to be inserted. From here the search can continue until the encountered classes are more specific than the class to insert. Once these two sets of classes have been found, the new class can be inserted in the taxonomy by making it a specialisation of its more general classes and making some of its more specific classes a specialisation of it.

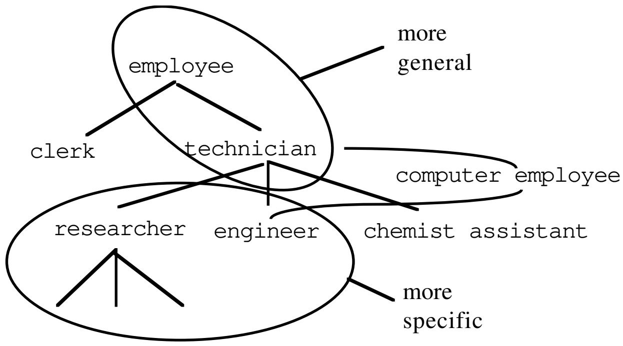

Figure 2: The categorisation aims at introducing, in the already existing taxonomy, the `computer employee` class characterised as a class of employees working with computers whose work is technical. It first tests if this class can be a sub-class of the `employee` class, that which is trivially true. It then tests for `clerk` class but this is not possible since the nature of the work is not technical: `clerk` is thus eliminated. It then tests for `technician` which is selected as a possible super-class. The algorithm tries to continue downward but must eliminate `researcher`, `engineer` and `chemist assistant` since they have fields (`publications` for instance) that the `computer employee` class does not have. The most specific more general classes is thus reduced to `technician`. The algorithm then tries to find possible sub-classes by going downward from that class. `Researcher` and `engineer` are selected this time since they share all the fields in `computer employee` and only add new fields and constraints to the existing fields.

All these algorithms are skeleton algorithms since they can be different depending on whether the graph search is breadth-first, depth-first or adaptive. They can also optimise the number of questions asked to the user.

Thus TROPES makes an extensive use of the classification mechanisms both for the building of the knowledge base and for its use in identifying the objects. However, there is nothing available for classifying from the data. This lack is filled by T-TREE.

3. DATA ANALYSIS-BASED CLASSIFICATION

TROPES has been designed for hand made taxonomies. It also provides help in order to insert hand-crafted classes into an existing taxonomy. However, when a lot of objects have been collected for a particular concept, dedicated algorithms can be used to build new taxonomies out of these individuals. This is the aim of T-TREE.

Our purpose here is not the presentation of new specialised hierarchical classification algorithms but rather the description of how such algorithms can be integrated into an object-based representation system (note that from that point, the examples concern a real estate knowledge base). First it is shown how some object-based concepts can be easily translated into data analysis concepts and how some others cannot be simply modelled. Then, it is shown how the results of data analysis algorithms are taken into account by TROPES. Last, the possible interactions between the user and data analysis methods are presented.

3.1. TROPES notions as data analysis ones

Clustering algorithms usually work on a set Ω of individuals affected of weight $w_1,\ldots w_m$ (whose sum is equal to 1), with characteristics (or variables) $v_1,\ldots v_n$ which are functions from Ω to structured sets of values $O_1,\ldots O_n$ (see [5] for example). In TROPES, Ω corresponds to the set of instances of a concept and the variables are the fields to which corresponds a domain. Since objects with the same characteristics are grouped into one individual, its weight corresponds to the importance of the set of objects it represents. As it can be found in data analysis, the domain of a field can be qualitative or quantitative, structured or not by a partial order.

```
Flat
  fields =
    district
    area-1
    monthly-rent
    annual-rent
    return-rate
  constraints =
    annual-rent = monthly-rent * 12
    return-rate = (annual-rent / price) * 100
    area-1 = determine-inhabitation-area( district )
```

```
flat#4
  district = "Meylan"
  monthly-rent = 3700
  price = 80
  nb-rooms = 3
  surface = 74
```

```
flat#4 \ Income
  area-1 = 3
  return-rate = 55500
  annual-rent = 44400
```

```
flat#4 \ Comfort
  floor = 6
  price = 80
  area-2 = 1
```

Figure 3: Part of the `Flat` concept with the `flat#4` instance as it is provided by the user and as it is seen from viewpoints `Income` and `Comfort`. The constraints allow to express relations between field values: the first one concerns a change of unit; the second one the elaboration of a new field, expressing the ratio `annual-rent` on acquisition `price` of a flat; the last one attributes an `area-1` number to a flat regarding to the `district` it is built in. It must be noted that the `Income` viewpoint only displays data which are not provided by the user but computed by the system.

It is sometime useful to change the kind of a variable in order to use particular methods or to simplify the results. This can be achieved in TROPES by creating a new field and defining an inference method allowing to compute its value with regard to the values of other fields. This is the case when one wants to obtain intervals from real values for instance or to use the `return-rate` (of an investment) variable instead of the two variables `price` and `annual-rent`.

The unit in which characteristic values are expressed is of great significance. It is thus useful to change variable units (rather than changing distance definitions). For the sake of experiments, the values of variables are divided by their standard deviation thus being independent from units.

Automatic classification algorithms produce clusters which are sets of objects. They correspond to the classes of a particular viewpoint in TROPES and the set inclusion relation between clusters produced by a hierarchical clustering algorithm corresponds to the specialisation relation. Thus, TROPES is able to describe a problem for clustering algorithms and to store the result of such an algorithm. If already implemented programs make use of particular formats such as matrices, section 3.3 shows how the conversion can be worked out. However, TROPES has limited capabilities for expressing global constraints (i.e. constraints taking into account a set of objects). More exactly, such constraints cannot concern a set of objects defined intentionally (if the set of object evolves, everything must be computed from scratch). This is important in a number of data analysis operations:

- the definition of a class (with regard to data analysis algorithm output) can express both metric and topologic conditions for class membership, but these conditions cannot refer to the set of members of a class or the classes themselves. For instance, the definition of a class can be that of a precise n-dimensional sphere (for n fields) but not that it must be a sphere of limited diameter (that which depends on all the members of the class).
- the definition of intervals which are balanced (grouping the same number of objects) or centred (grouping a set of homogeneous objects).
- the weighting of characteristic values with regard to their variance is not possible through inference methods since it requires the computation of variance over the complete set of objects. It is here carried out during the translation step.

This is related to the problem of interpreting the results and methods of data analysis algorithms and its capability to produce conceptual descriptions. The result cannot be translated as such into a knowledge representation formalism. Thus TROPES only plays the role of a repository of the class hierarchy and the associated members.

3.2. Integrating algorithms

In order to build a new taxonomy, the user creates a new viewpoint. (S)he then chooses the fields relevant under that viewpoint and a method for constructing the taxonomy. Here, several methodological problems arise which are related to choice and guidance of the user (choice of a method, choice of a metrics, choice of the other parameters of the method…). These problems are out of the scope here since they are independent from data analysis; §3.3 addresses them.

Two methods have been implemented so far: the divisive k-means method and the agglomerative method of closest neighbours [5]. In both methods, the Minkowski metrics is used and reduced to Euclidean distance function of the standard deviation of each dimension.

Figure 4 shows the result on a sample real estate base with two viewpoints modelling the buyer-for-rental and the buyer-for-inhabiting behaviour. The following table gives the object field values. Under the first double bar are the first viewpoint fields (`floor`, `price`, `area-1` — is it close to work and facilities —, `surface` and number of rooms) and under the second double bar, those of the second viewpoint (`price`, `return-rate`, and another partition into area considering if the inhabitants are usually wealthy…). Note that the area values are ordered numbers.

field\flat	1	2	3	4	5	6
floor	3	3	0	6	0	3
price	31	20	35	80	75	100
area-1	3	2	2	3	1	1
surface	35	20	60	74	80	90
nb-rooms	1	1	2	3	3	4
area-2	1	3	2	1	2	2
return-rate	0,8	0,9	0,9	0,5	0,5	0,4

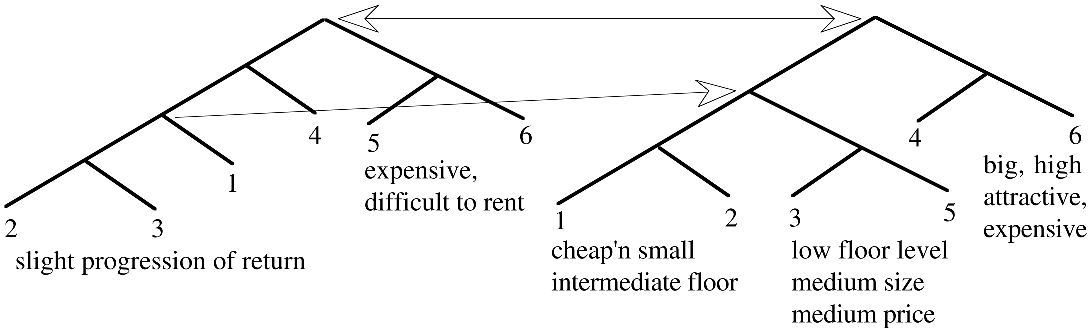

Figure 4 (The lines represent specialisation; the arrows represent bridges which can be inferred, see §4): Both hierarchies have been computed with the same (closest neighbours) algorithm. The results are quite good for the second viewpoint in which three classes are obtained (the interpretation is given under the classes). The first viewpoint provides two relevant classes one of which is quite wide and would require more tuning of the algorithm (for example by giving less importance to the area and using it only for distinguishing two close individuals).

Once supplied with the relevant parameters, the method is able to build the new taxonomy. The resulting taxonomy is made of a set of new classes (with dummy names) plus the set of objects attached to their most specific class (usually a singleton class). This taxonomy can be used in many ways:

- As usual output of data analysis methods, it can be seen as an abstract representation of the set of objects.
- As a usual knowledge representation taxonomy, it can be seen as a set of abstract classes ordered by progressive specialisation.
- But, it can be seen as only one step of the more elaborate process for achieving a more precise classification by:
    - — Examining the relations of the resulting classes with other classes on other taxonomies (this is detailed in §4).
    - — Revealing the lack of particular individuals in the panel for achieving a meaningful classification.
    - — Revealing the inadequacy of the parameters provided to the method.

3.3. Integrating external libraries and controlling algorithms

So far, it has just been said that data analysis algorithms are able to produce new taxonomies when run adequately thanks to the correspondence given at §3.1. One important point in using data analysis algorithms is the use of numerous parameters (e.g. distance definition, number of classes per level, initial classes…). When the knowledge base is used by a taxonomist, it is useful for (her)him to have at hand the choice of the method to use for classifying. Moreover, the taxonomist would certainly appreciate having some help for controlling the available methods and the opportunity to interrupt the system and lead (her)himself the classification process.

For instance, our implementation of the *k*-means method allows the user to provide the initial centre of gravity of the classes, the number of classes and the distance. Moreover, the *k*-means method is a partition method and not a hierarchical clustering method. Thus the user is allowed to change the parameters of the methods at each call (i.e. at each class which require more partitioning) such that the number of class sub-divisions is not always the same.

SCARP [4, 15] is a problem solving environment developed in our research team and which must be ported on top of TROPES. It has already been used for integrating a set of data analysis algorithms in the domain of simple linear ordination (principal component analysis, factorial correspondence analysis and multiple correspondence analysis). The idea beside SCARP is the representation of the tasks to be carried out as objects and the interpretation of these particular objects through a specific process (called engine). Each task is the composition of more elementary tasks or the representation of an implemented algorithm. There is a set of generic tasks such as sequence, iteration, alternative, parallel… which are executed by the sequential, iterative, alternative or parallel execution of their sub-tasks. This stops at the leaves of the task decomposition tree which are implemented by programmed methods. A problem to solve is supplied to SCARP as a high-level task. The engine dynamically monitors the task decomposition, classification and execution until the problem is solved. SCARP is co-operative in that it is able to carry out the task on its own but can be interrupted and directed by the user.

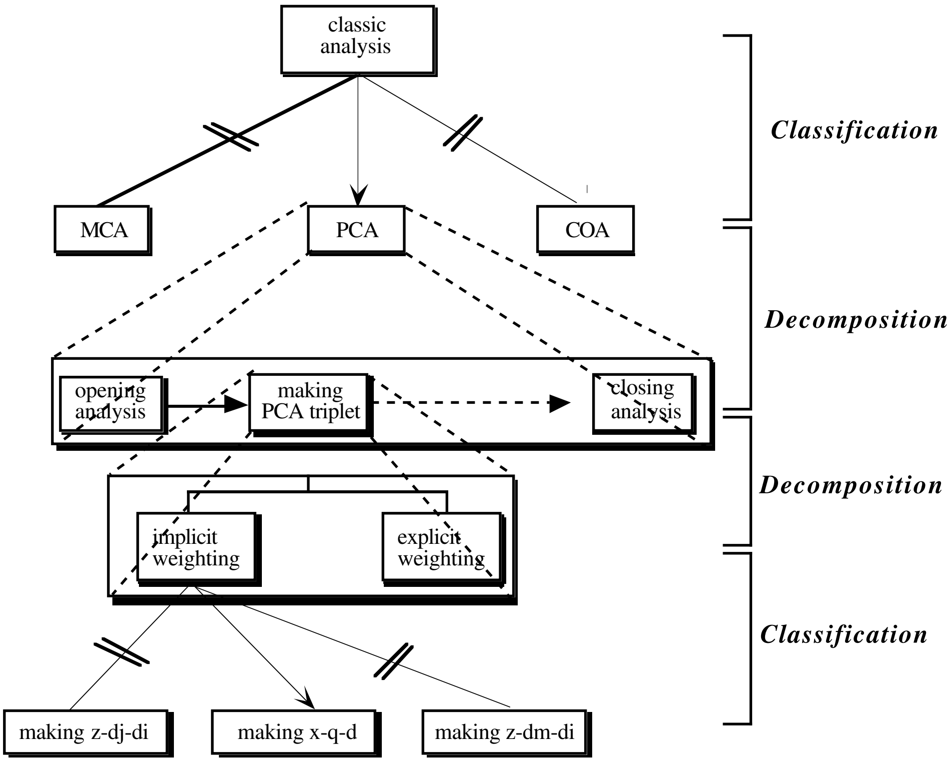

Figure 5 (Reprinted from [4] with permission; a crossed line represents a taxonomic link which has been eliminated by classification, dashed lines represent task decomposition and arrows represent the classification of a task into a more specific task): A *classic analysis* is first specialised according to the problem context (especially the characteristics of the matrix to analyse) into a *principal component analysis*. This task is then decomposed into its sub-tasks. After the execution of *opening analysis* the next sub-task, *making PCA triplet*, has to be executed. It has no sub-class and is therefore directly decomposed into its sub-tasks; in fact, the solving strategy indicates a choice between *implicit* and *explicit weighting*. *Implicit weighting* is chosen and in turn classified.

SCARP deals with several problems arising in T-TREE:

First, it allows external library integration. As a matter of fact, one of the advantages of describing the tasks into sub-tasks is the ability to add pre- and post-treatments to the task itself. Indeed, the standard format for data analysis algorithms is not the set of objects but the matrix. A formatting task can produce the matrix required by the chosen method.

Second, SCARP provides the help which the user needs for choosing (for example) among the available methods and to build (her)himself a resolution tree (a step-by-step mode is available). Moreover, SCARP is designed as a co-operative problem solving environment. It allows the user to stop the current resolution, change parameters or backtrack to a previous choice before starting the resolution again. For instance, it allows to implement the *k*-means method as a task and the classification algorithm as another task using the *k*-means method through iteration. Thus the configuration of the *k*-means method is done before each call to the task.

Third, the pre-tasks of SCARP allow to ask the user for the configuration of the method to be used or to acknowledge the choice made by a specialised task. For instance, the choice for a metric should be taken into account by a specialised task.

Up to now, the algorithms are run by hand with the user supplying the parameters directly to the algorithm. But, in the future, each method must be supplied with its own configuration interface. Also, only the classification mechanisms have been considered. Of course, any data analysis method such as correspondence analysis or principal component analysis can be integrated as well. Such methods do not produce any new classification but they can output data in the form of matrix file or graphic display. However, the output can be the filling of new fields corresponding to the dimensions of the result. These fields can be the input of an automatic clustering method in order to integrate the result in a taxonomy, achieving automatically what is usually left to the user's eyes.

4. BRIDGE INFERENCE

So far, the emphasis has been put on the building of a new taxonomy under a new viewpoint. However, one of the strength of TROPES is the ability to consider objects under several viewpoints. The comparison between viewpoints can be really useful for a taxonomist. Not only, T-TREE allows to build and use the taxonomies but it also allows to compare different taxonomies. An algorithm has been developed which is able to infer bridges. The bridge inference algorithm, given a set of source viewpoints and a destination viewpoint (built by T-TREE or by any other mean), returns all the bridges (in a minimal fashion) which are satisfied by the available data. That is the set of bridges for which the objects in every source class are indeed in the destination class. Establishing bridges helps the taxonomist in:

- considering if bridges are legitimate and so retaining them,
- checking if some particular kind of data, which invalidates bridges, is missing, or
- demonstrating the differences between methods (or their sensitivity to particular data).

The algorithm compares the extension (set of members) of the presumed destination to the intersection of these of the presumed source classes. If there is no inclusion of the latter in the former, the algorithm is re-iterated on all the sets of source classes which contain at least one class which is a sub-class of the tested source classes. If the intersection of the extension of the presumed source classes is included in that of the presumed destination class, then a bridge can be established from the latter (and also from any set of sub-classes of the source classes) to the former (and also any super-class of the destination class). But other bridges can exists on the sub-classes of the destination. The algorithm is thus re-iterated on them. It stops when the bridge is trivial, i.e. when the source is empty. The algorithm is as follows:

```
find-bridges-under(classes,class) =
if ∩classes=∅ then exit; /* avoid trivial bridges */
if ∩classes⊆class
then if ∃c∈subclasses(class); ∩classes⊆c
     then /* the bridge is not minimal */
          find-bridges-under(classes,c)
     else add-bridge(classes,class);
          /* the bridge is also true for all sub-classes of classes;
             and all super-classes of class;
             look for more specific bridges (∩classes⊄c) */
          for every c∈subclasses(class) do find-bridges-under(classes,c)
else /* search for bridges from more specific set of classes */
     for every c∈classes
     do   for every c'∈subclasses(c)
          do find-bridges-under(classes\{c}∪{c'},class)
```

The algorithm is extension-correct (only valid bridges are inferred), extension-complete (all valid bridges are inferred) and extension-minimal (only more general bridges are inferred). The proof is carried out in the classification scheme framework [6] and the "extension-" prefix just tells that what is considered is only the extension of the classes (the algorithm tests set inclusion on classes). Thus the result is not considered as valid in the logical sense of the term (two classes can share the same set of members, but are not necessarily equivalent). For instance, is that a coincidence that all directors have formerly been at the same university? Maybe, maybe not. Hence the user has to decide the validation of inferred bridges. This has to be contrasted with a stronger kind of bridge inference based on the structural constraints on classes.

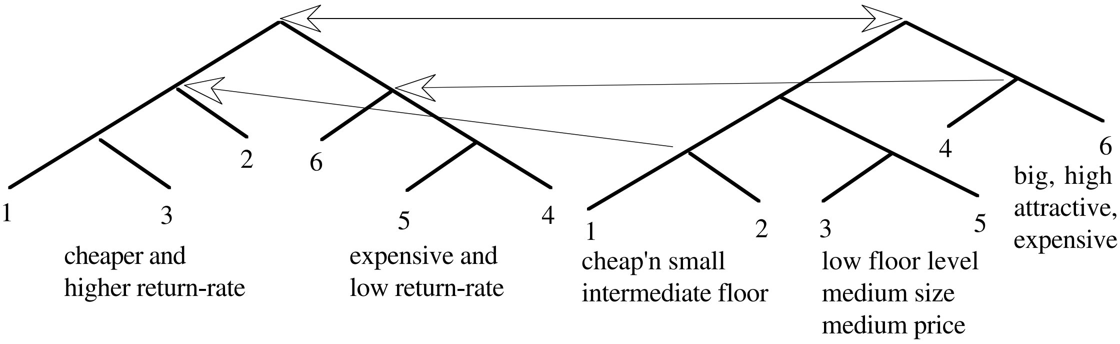

Figure 6: The example of figure 4 does not introduce relevant bridges. As suggested there, the classification is constructed here with the help of `price` and `return-rate` only: it is a more natural representation of the buyer-for-rental view of the examples. Bridges can be though of as rules and the inferred bridges here can be interpreted as: big, attractive and expensive flats have a poor return-rate; small and cheap flats have a high return-rate (another bridge which is inferred the other way tells that flats with the highest return-rate are either small or medium sized). All these rules are quite correct. However, this is not ensured by the algorithm.

The bridge inference method is designed for a large number of objects and a relatively small number of classes. It is exponential in the number of classes (the exponent being the number of viewpoints) and linear in the number of objects. The worst case complexity is $O(n_C * n_{CV}^{n_V} * n_I)$ in which:

n_I is the maximum number of objects in a class;
n_C is the number of subclasses of the initial class;
n_V is the number of viewpoints, and
n_{CV} is the maximum number of classes per viewpoint.

Bridge inference is nothing else than the search for correlation between two sets of variables. This correlation is particular from a data analysis point of view since it does not need to be valid on the whole set of individuals (the algorithm looks for subsets under which the correlation is valid) and it is based on strict set equality (not similarity). However, even if the bridge inference algorithm has been described with set inclusion, it can be helped by other measurements which will narrow or broaden the search. A solution for search narrowing consists in adding a threshold on the bridge cardinal (the number of instances concerned by the bridge) in order to obtain more likely bridges. A solution, in order to broaden the search, consists in adding a threshold on the cardinal of the difference (some elements can miss the bridge without invalidating it). More sophisticated tools can take the typicality of the excluded members as a mean to decide if the bridge can be retained: such a result can help tuning the data analysis method which has been used for classifying. More generally, the inclusion and emptiness tests can be replaced out by tests based on the similarity of two sets of objects (as it is usual in data analysis). In fact, many parameters can be taken into account when inferring bridges; for that purpose, the algorithm is function of the meaning of the operators $\subseteq$, $\cap$ and $=\emptyset$.

5. CONCLUSION

Taxonomies are produced in a systematic way by data analysis algorithms from a set of formatted data. T-TREE aims at reducing the distance between data analysis algorithms and the taxonomic repository. It is an extension of the TROPES knowledge representation system which allows to use data analysis classification algorithms whose input is a set of objects of the representation and output is a new taxonomy into the representation. The following advantages are brought by T-TREE:

- Numeric and symbolic properties are merged into the same base and manipulated the same way;
- Objects and classes are ready to be used like in any knowledge-based system: T-TREE enables in the same framework to build, compare and use taxonomies;
- The result of the classification process can be compared under several viewpoints. These viewpoints can differ in the method used for establishing their taxonomies, the parameters of the methods or the fields of individuals which are selected in order to build the taxonomy.
- The integration of tasks into object-based knowledge models should allow to easily integrate important data analysis libraries and combine them.

At that point of development, we are looking for data analysis libraries and significant workbench in order to evaluate the advantages of T-TREE.

An important problem is due to the difficulty to interpret the clusters provided by data analysis. there is no systematic way to provide a class description from the cluster: this has to be carried out by hand. Anyway, this remains a cornerstone problem to build symbolic description out of numeric characterisation of classes. Using machine learning conceptual clustering algorithms [12] for this task could provide the answer.

Further work remains to be carried out on T-TREE:
- adding graphic user interface (with the building of TROPES interface);
- implementing the methods as tasks (after the port of SCARP);
- integrating more important data analysis or conceptual clustering algorithms. A relevant result would be the interfacing with a complete classification library (e.g. SICLA [3]);
- testing various distances taking into account non valued characteristics and studying their relationships with incompleteness handling;
- testing methods for automatically building consensus trees from several taxonomies can also be applied as a way of evaluating the classification results.

6.REFERENCES

[1] James Beach, Sakti Pramanik, John Beaman, **Hierarchic taxonomic databases**, in Renaud Fortuner (ed.), Advanced computer methods for systematic biology, The Johns Hopkins university press, Baltimore (ML US), pp241-256, 1993

[2] Cécile Capponi, **Interactive class classification using types**, Proc. 4th conference of the international federation of classification societies, Paris (FR), 1993 to appear

[3] Gilles Celeux, Edwin Diday, Gérard Govaert, Yves Lechevallier, Henri Ralambondrainy, **Classification automatique de données**, Dunod, Paris (FR), 1989

[4] François Chevenet, François Jean-Marie, Jutta Willamowski, **A development shell for cooperative problem solving environments (extended abstract)**, in Elias Houstis, John Rice, Robert Vichnevetsky (eds.), Proc. 3rd international conference on expert systems for numerical computing, Technical report CSD-TR-93-028, Purdue university, West Lafayette (IN US), pp1-7, 1993 complete version to appear in *Mathematics and computers in simulation*

[5] William Day, Herbert Edelsbrunner, **Efficient algorithms for agglomerative hierarchical clustering methods**, *Journal of classification* 1(1):7-24, 1984

[6] Jérôme Euzenat, **Une définition abstraite de la classification et son application aux taxonomies d'objets**, Proc. 2nd journées représentations par objets (RPO), La Grande Motte (FR), pp235-246, 1993

[7] Douglas Fisher, **Knowledge acquisition via incremental clustering**, *Machine learning* 2:139-172, 1987

[8] Nicole Gautier, Alain Pavé, François Rechenmann, **Object-centered representation and fish identification in Antartica**, in Renaud Fortuner (ed.), Advanced computer methods for systematic biology, The Johns Hopkins university press, Baltimore (ML US), pp181-195, 1993

[9] Olga Mariño, **Classification d'objets composites dans un système de représentation de connaissances multi-points de vue**, Proc. 8th RFIA, Villeurbanne (FR), pp233-242, 1991

[10] Olga Mariño, François Rechenmann, Patrice Uvietta, **Multiple perspectives and classification mechanism in object-oriented representation**, Proc. 9th ECAI, Stockholm (SE), pp425-430, 1990

[11] Gérald Masini, Amedeo Napoli, Dominique Colnet, Daniel Léonard, Karl Tombre, Les langages à objets, InterÉditions, Paris (FR), (tr. eng. **Object-oriented languages**, Academic press, London (GB), 1991) 1989

[12] Ryszard Michalski, Robert Stepp, **Learning from observation: conceptual clustering**, in Ryszard Michalski, Jaime Carbonell, Tom Mitchell (eds.), Machine learning: an artificial intelligence approach, Tioga publishing company, Palo Alto (CA US), pp331-363, 1983

[13] Richard Pankhurst, **Practical taxonomic computing**, Cambridge university press, Cambridge (GB), 1991

[14] Larry Stephens, **The classification of semantic relations based on primitive properties**, in Suzanne Humphrey, Barbara Kwasnik (eds.), Advance in classification research (proc. of the 1st ASIS SIG/CR classification research workshop, Toronto (CA), 1990), Learned information, Medford (NJ US), pp159-168, 1990

[15] Jutta Willamowski, François Chevenet, **Modelling methodological knowledge in data analysis**, Proc. 13th international conference on artificial intelligence, expert systems and natural language, Avignon (FR), pp211-220, 1993