Building classes in object-based languages by automatic clustering

Petko Valtchev

INRIA Rhône-Alpes
655 av. de l'Europe, 38330 Montbonnot Saint-Martin, France

Abstract. The paper deals with clustering of objects described both by properties and relations. Relational attributes may make object descriptions recursively depend on themselves so that attribute values cannot be compared before objects themselves are. An approach to clustering is presented whose core element is an object dissimilarity measure. All sorts of object attributes are compared in a uniform manner with possible exploration of the existing taxonomic knowledge. Dissimilarity values for mutually dependent object couples are computed as solutions of a system of linear equations. An example of building classes on objects with self-references demonstrates the advantages of the suggested approach.

1 Introduction

Object-based systems provide a variety of tools for building software models of real-world domains : classes and inheritance, object composition, abstract data types, etc. As a result, the underlying data model admits highly structured descriptions of complex real-world entities. With the number of object applications constantly growing the need of analysis tools for object datasets becomes critical [7]. Although the importance of the topic has been recognized [4, 1], it has been rarely addressed in the literature : a few studies concerning object knowledge representation (KR) systems [3, 9, 1], or object-oriented (OO) databases [7] have been reported in the past years.

Our own concern is the design of automatic class building tools for objects with complex relational structure. A compact class can be built on-top of an object cluster discovered by an automatic clustering procedure [5]. The main difficulty that clustering application faces is the definition of a consistent comparison for all kinds of object attributes. Relations connect objects into larger structures, object *networks*, and may lead to (indirect) self-references in object descriptions. Self-references make it impossible to always compare object attributes before comparing objects. Consequently, most of the existing approaches to clustering [6, 10] cannot apply on self-referencing descriptions.

We suggest an approach towards automatic class building whose core element is an object dissimilarity measure. The measure evaluates object attributes in a uniform way. Its values on mutually dependent object couples are computed

as solutions of a system of linear equations. Thus, even object sets with self-references may be processed by a clustering tool. The discovered clusters are turned to object classes and then provided with a intentional description.

The paper starts by a short presentation of an objects model together with a discussion of aspects which make objects similar (Section 2). A definition of a suitable dissimilarity measure is given in Section 3. The computation of the measure on mutually dependent object couples is presented. Section 4 shows an example of an application of the measure. Clustering and class characterization are discussed in Section 5.

2 Object formalism

Object languages organize knowledge about a domain around two kinds of entities, classes and objects. Domain individuals are represented as objects, whereas groups of individuals, or categories, give rise to classes. Thus, each class is associated to a set of member objects, its *instances*. Both classes and objects are described in terms of attributes which capture particular aspects of the underlying individuals. A class description is a summary of instance descriptions.

In the following, only the descriptive aspects of object languages will be considered. The object model of TROPES, a system developed in our team [8] will be used to illustrate the object specific structure. However, the results presented later in the paper hold for a much larger set of object models.

2.1 Objects, concepts and classes

A *structured object* in our model is a list of attribute values which are themselves objects or simple values. For instance, an object representing a flat within a real-estate knowledge base, may have fields for flat's rent, standing, owner, etc. Fig. 1 shows example of an object, `flat#5`, with three attributes, `rooms`, `owner` and `rent`.

Three kinds of attributes will be distinguished: *properties*, *components* and *links*. Properties model features of the individual being modeled, for example the rent of a flat, whereas components and links model relations to other individuals. Composition, or `part-of`, relation between individuals is expressed through component attributes. Such an attribute relates a composite object to one or a collection of component objects. Composition is distinguished due to its particular nature : it is a transitive and non-circular relation. On the above example, `owner` is a link and `rooms` is a component. Apart for nature, attributes have a `type` which delimits their possible values. Object-valued attributes are typed by object concepts whereas simple values are members of data types, further called *abstract data types* (ADT). Finally, attributes may have a single value or a collection of values in which case they are called *multi-valued* attributes. Collections are built on a basic type by means of a constructor, `list` or `set`. For instance, the values of `rooms` attribute in `flat#5` is a set of three instances of the `room` concept.

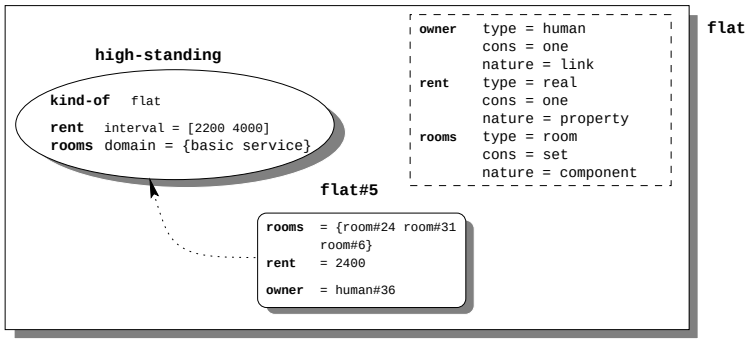

Fig. 1. Example of a concept, `flat`, a class, `high-standing` and an instance, `flat#5`. All three entities provide lists of attributes : concept attributes specify type, nature and constructor, which are necessary in interpreting values in instance attributes; class attributes provide value restrictions on instance attributes; instance attributes contain values.

Objects of a KB are divided into disjoint families, called *concepts*, e.g. `flat`, `human`, `room`. An object is thus an instance of a unique concept, for example, `flat#5` of the `flat` concept drawn as a rectangle. Concepts define the structure (set of object attributes with their types) and the identity of their instances. This may be seen on upper right part of Fig. 1 : `flat` defines the type, the nature and the constructor of the three attributes of the example. Concepts are comparable to big classes in a traditional OO application, or to tables in relational databases. A *class* defines a subset of concept instances called class *members*. Classes provide value restrictions on attribute values of member objects, that is sets of admissible values among all the values in the attribute basic type given by the concept. Restrictions on property attributes correspond to sub-types of the ADT, whereas restrictions on relations are classes of the underlying concept. For example, the class `high-standing` on Fig. 1 restricts the `rent` attribute to values in the interval `[2200,4000]` and the rooms attribute to sets composed of members of the classes `basic` and `service`. Classes of a concept are organized into a hierarchical structure called *taxonomy*.

An object KB is made up of a set concepts. For each concept, a set of instances are given which are organized into one or more class taxonomies. Objects and classes of different concepts are connected by means of relational attributes.

2.2 Summary on objects as data model

An object is a member of the entire set of instances defined by its concept. A class defines a sub-set of instances. In quite the same way, simple values are members of the domain of their ADT whereas types define subsets of ADT values. Unlike objects, simple values have neither identity, nor attributes.

Objects can be seen as points in a multi-dimensional space determined by the object's concept where each dimension corresponds to an attribute. A class describes a region of that space and each member object lays exclusively within the region. Furthermore, dimensions corresponding to relational attributes are spaces themselves. Thus, a relation establishes a dependency between its source concept and its target concept : instances of the former are described by means of instances of the latter. An overview of all inter-concept dependencies is provided by a graphical structure, henceforth called *conceptual scheme* of the KB, composed by concepts as vertices and relational attributes as (labeled) edges. For example, the (partial) conceptual scheme of the real estate KB given on the left of Fig. 2 is made up of three concepts, `human`, `flat` and `room`, and three attributes: two links composing a circuit, `owner` and `house`, and a multi-valued component relation, `rooms`.

The scheme summarizes the relations that may exist between instances of different concepts in the KB. In fact, each object is embedded into a similar relational structure where each concept is replaced by one or a collection of its instances. The structure, which we call the *network* associated to an object o, includes all objects which are related to o by a chain of attributes. In a network, an edge corresponding to a multi-valued attribute may link a source object to a set of target objects. For example, on the left of Fig. 2, the network of the object `flat#5` is drawn. Within the network, an edge `rooms` connects `flat#5` to the three objects representing flat's rooms. Observe the similar topologies of both structures on Fig. 2. [1]

Finally, the network of o extended with classes of all its objects contains the entire amount of information about o. It is thus the maximal part of the KB to be explored for proximity computation.

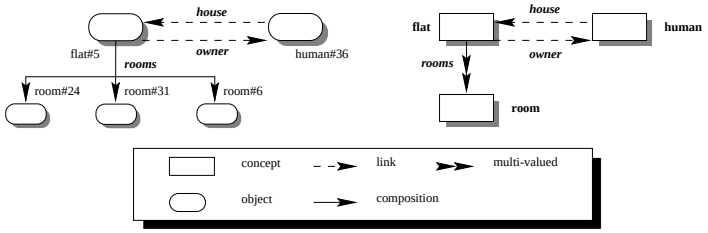

Fig. 2. Example of a KB conceptual scheme, on the right, and an object network that reflects the scheme structure, on the left. The network is obtained from the scheme by replacing a concept by an instance or a collection of instances.

[1] Actually, there is a label-preserving morphism from each instance network to the respective part of the conceptual scheme.

2.3 Analysis issues

We are concerned with class design from a set of unclassified objects within an object KB. The problem to solve is an instance of the conceptual clustering problem [10] and therefore may be divided into two sub-tasks: constitution of member sets for each class and class characterization [6]. A conventional conceptual clustering algorithm would typically carry out both sub-tasks simultaneously using class descriptions to build member sets. With an object dataset where link attributes relate objects of the same concept this approach may fail. Let's consider the example of the `spouse` attribute which relates objects representing a married couple. Within a class of `human`, the `spouse` attribute refers to a possibly different class of the same concept. The attribute induces two-object circuits on instances of `human` and therefore may lead to circular references between two classes in the taxonomy. When classes are to be built in an automatic way, it is impossible to evaluate two classes, say c and c', referring to each other since the evaluation of each class would require, via the `spouse` attribute, the evaluation of the other one.

We suggest an approach to automatic class building which deals with both tasks separately. First, member sets are built by an automatic clustering procedure based on an object proximity measure. The measure implements principles suggested in [3] to compare object descriptions with self-references. Once all class member sets are available these are turned into undescribed classes. The characterization step is then straightforward since class attributes in particular those establishing circuits, may refer to any of the existing classes.

As our aim is to find compact classes, we require the discovered clusters to be homogeneous on all object attributes, inclusive relational ones. Thus, an object proximity measure is to be designed which combines in a consistent way the differences on both properties and relations. Keeping in mind the analogy between objects and simple values we can formulate the following general comparison principle. Given two points in a space, we shall measure their mutual difference with respect to the relative size of the smallest region that covers both of them. The smallest region is a type in case of primitive values and a class in case of objects. It is unique for an ADT, but it may be interpreted in two different ways for concepts. In a first interpretation, it corresponds to an existing class, the most specific common class of both objects. Thus, the corresponding region is not necessary the smallest possible one, but rather the smallest admissible by the taxonomy of the concept. The second interpretation is straightforward : the region is the effective smallest one, and is thus independent form the existing taxonomy.

In the next section we describe a dissimilarity function based on the above principle which deals successfully with circularity in object descriptions.

3 Dissimilarity of objects

A proximity measure, here of *dissimilarity* kind [12], is usually defined over a set Ω of individuals ω described by n attributes $\{a_i\}_{i=1}^{n}$. Attribute-level functions

δ_i compute elementary differences on each attribute a_i, whereas global function $d = Aggr(\delta_i)$ combines those differences into a single value. In case of object KB, each concept C_l is assigned a separate object set Ω_l and hence a specific function $d^o_{C_l}$.

3.1 Object-level function

Let C be a concept, e.g. `flat`, with n attributes $\{a_1, .., a_n\}$ (we assume that each object has all attribute values). In our model, the *object dissimilarity function* $d^o_C : C^2 \rightarrow \mathbb{R}$ is a normalized linear combination of the attribute dissimilarities $\delta^f_i : type(a_i)^2 \rightarrow \mathbb{R}$. For a couple of objects o and o' in C, dissimilarity is computed as :

$$d^o_C(o, o') = \sum_{i=1}^{n} \lambda_i * \delta^f_i(o.a_i, o'.a_i) \ . \tag{1}$$

where λ_i is the weight of attribute a_i ($\sum_{i=1}^{n} \lambda_i = 1$). It is noteworthy that all real-valued functions are normalized.

With respect to what has been said in the previous section, the value of $d^o_C(o, o')$ may be interpreted in the following way. Suppose each δ^f_i evaluates the size of the smallest region that contains the values $o.a_i$ and $o'.a_i$ within the corresponding dimension. Then, $d^o_C(o, o')$ computes the weighted average of all such sizes. This is an estimation of the size of the smallest region covering o and o' within the space of C. The form of d^o has been chosen to enable the computation in case of circularity. In the following, possible definitions for δ^f_i are presented which are consistent with the general principle of the previous section.

3.2 Attribute-level functions

Following the attribute type, δ^f_i is substituted in Formula 1 either by a property dissimilarity, $\overline{\delta}^{pr}_T$ or by a relational dissimilarity $\overline{\delta}^r_{C'}$.

The dissimilarity of simple values depends on the ratio between the size of their most-specific common type and the size of the whole ADT value set. Formally, let T be an ADT of domain D. We shall denote by $range(t)$ the size of the sub-set of D described by the type t. This value may be a set cardinal, when T is a nominal or partially ordered type (structured as in [10]), or an interval length T is a totally ordered type. For a couple of values in D, v and v', let $v \vee v'$ denote their most specific common type within T. Then :

$$\overline{\delta}^{pr}_T(v, v') = \frac{range(v \vee v') - 0,5 * (range(v) + range(v'))}{range(D)} \ . \tag{2}$$

The subtraction of the average of value ranges in the above formula allows $\overline{\delta}^{pr}_T$ to remain consistent with conventional functions on standard types. For example, with an integer type $T =$ [0 10] the value of $\overline{\delta}^{pr}_T(2, 4) = 0,2$.

In case of object-valued attribute, the relative size of the effective smallest region is estimated by applying a function of d^o kind. Doing that, the dissimilarity computation goes a step further in the object network structure, from

objects to attribute values. In case of a strongly connected component[2] of the network the computation comes back to the initial couple of objects, o and o'. For example, comparing two flats requires the comparison of their owners which in turn requires the comparison of flats. In other words, the self-references in object structure lead to recursive dependence between dissimilarity values for object couples. Section 3.3 presents a possible way to deal with recursion.

The class dissimilarity δ^{cl} compares objects as members of a class and no more as attribute lists. More precisely, δ^{cl} estimates the relative size of the most specific common class of two objects. Of course, the function can only be used on an attribute a if a class taxonomy is available on $C' = type(a)$. Moreover, the computed values only make sense if the taxonomy structure reflects the similarities between instances of C'.

Formally, let C' be a concept, the type of an object-valued attribute a and let $root(C')$ be the root class of C'. Let also o'_1, o'_2 be a couple of instances of C' and let $c = o'_1 \vee o'_2$ be the most specific common class of o'_1, o'_2. Class dissimilarity is then the ratio of the number of objects in the class and the total number of concept instances. Thus, the more specific the class, the less dissimilar the objects.

$$\overline{\delta}^{cl}_{C'}(o'_1, o'_2) = \frac{\|(members(c))\| - 1}{\|members(root(C'))\|} \ . \tag{3}$$

where $members()$ returns the set of member objects of a class. Here one stands for the average of $members()$ on both objects.

The above function allows the existing taxonomic knowledge in the KB to be used for clustering, that is for building of new taxonomies.

In case of multi-valued attributes, both relational and of property nature, a specific comparison strategy must be applied since collections may have variable length. The resulting collection dissimilarity relies on a pair-wise matching of collection members prior to the computation. Space limitations do not allow the point to be extended here, but an interested reader will find a description of a multi-valued dissimilarity in [11]. All functions defined in that paper are consistent with the above dissimilarity model in that they are normalized and represent valid dissimilarity indices.

3.3 Dealing with circularity

Circularity arises when strongly connected components occur in object networks. A simple example of such component is the two-way dependency between `flat` and `human` concepts established by the `owner` and `house` attributes. If the dissimilarity of a couple of humans who own their residences is to be computed, this depends, via the `house` attribute on the dissimilarity of the respective flats. The flat dissimilarity depends, in turn, on the dissimilarity of the initial objects via `owner`. Both values depend recursively on themselves. A possible way to deal with such a deadlock is to compute the values as solutions of a system of linear equations (see [2]).

[2] not to mix with component attributes

A single system is composed for each strongly connected component that occurs in *both* networks. In the system, the variables x_i correspond to pairs of objects which may be reached from the initial pair o, o' by the same sequences of relational links (no composition). For each couple, let's say there are m couples, an equation is obtained from Formula 1.

$$x_i = b_i + \sum_{i=j, i \neq j}^{m} c_{i,j} * x_j \ . \tag{4}$$

Here, b_i is the *local part* of the dissimilarity d^o_C, that is the sum of all dissimilarities on properties and components plus those links which does not appear in the strongly connected component (so there are no variables representing them in the system). The remaining dissimilarities are on link attributes which take part in the circular dependence.

The coefficients $c_{i,j}$ in each equation are computed as follows. If the objects of a couple corresponding to x_j are the respective values of an attribute a in the objects of the x_i couple, then $c_{i,j}$ is the weight of the attribute a. Otherwise, $c_{i,j}$ is 0.

The obtained system is quadratic (m variables and m equations).

$$\mathbf{C} * \mathbf{X} = \mathbf{B} \ . \tag{5}$$

The matrix C is *diagonal dominant* so the system has a unique solution. It can be computed in a direct way or by an iterative method.

The measures d^o_C, one per concept C, obtained by such a computation are valid dissimilarity indexes, since positive, symmetric and minimal. Moreover, it can be proved that if all δ^f_i functions are metrics, then d^o_C are metrics too.

4 An example of dissimilarity computation

In the following, we shall exemplify the way d^o is computed and used to cluster objects. Due to space limitations, we only consider a sample dataset made exclusively of `human` instances with three attributes: `age`, `salary` and `spouse` (see Table 1). `spouse` attribute is of a link nature and its type is the `human` concept itself. It thus establishes tiny cycles of two instances of the `human` concept. The `salary` attribute indicates the month's income of a person in thousands of euros, it is of float type and ranges in `[1.8, 3.3]` whereas the `age` attribute is of integer type and ranges in `[20, 35]`.

Let's now see how the value of d^o is computed for a couple of objects of the set, say `o1` and `o4`. First, we fix the attribute weights at `0.4` for `spouse`, and `0.3` for both `salary` and `age`. According to Formula 1 the dissimilarity for `o1` and `o4` becomes:

$$d^o_{human}(o1,\ o4) = 0.3\ \overline{\delta}^f(26,\ 30) + 0.3\ \overline{\delta}^f(3,\ 2.7) + 0.4\ \overline{\delta}^f(o2,\ o3) \ . \tag{6}$$

Table 1. A sample dataset of `human` instances

attribute	o1	o2	o3	o4	o5	o6
age	26	23	27	30	32	35
salary	3	2.3	2.2	2.7	2.9	3.2
spouse	o2	o1	o4	o3	o6	o5

The first two differences are computed by a property function of $\overline{\delta}^f$ taking into account the respective domain ranges. The total of both computations amounts to `0,18`. As no taxonomy is provided, $\overline{\delta}^f$ on `spouse` is replaced by d^o_{human} :

$$d^o_{human}(o2,\ o3) = 0.3\ \overline{\delta}^f(23,\ 27) + 0.3\ \overline{\delta}^f(2.3,\ 2.2) + 0.4\ \overline{\delta}^f(o1,\ o4)\ . \qquad (7)$$

Here we come to the mutual dependency of $d^o_{human}(o2,\ o3)$ and $d^o_{human}(o1,\ o4)$ due to the `spouse` circuit. Substituting a variable for each of them, lets say x_1 and x_2, the following linear equation system is obtained :

$$x_1 \ = \ 0.18 \ + \ 0.4x_2 \qquad (8)$$

$$x_2 \ = \ 0.14 \ + \ 0.4x_1 \qquad (9)$$

Table 2. Dissimilarity values for $\underline{d}^o_{human}$ and d^o_{human}

	o1	o2	o3	o4	o5	o6
o1	0	0.38	0.32	0.3	0.33	0.51
o2	0.38	0	0.23	0.48	0.65	0.9
o3	0.36	0.26	0	0.31	0.48	0.73
o4	0.28	0.43	0.31	0	0.17	0.42
o5	0.5	0.61	0.46	0.33	0	0.25
o6	0.56	0.74	0.57	0.44	0.25	0

The solutions for x_1 and x_2 are `0.28` and `0.26` respectively. The values of d^o_{human} for the whole dataset are given in Table 2. The table should be read as follows : whereas the entries below the main diagonal represent the results of the above d^o_{human} function, another function, lets call it $\underline{d}^o_{human}$, is given in the upper part of the matrix. $\underline{d}^o_{human}$ is computed only on `age` and `salary` attributes taken with equal weights. We put it here to exemplify the specific features of the relational measure in case of circularity.

A detailed examination of respective values for d^o_{human} and $\underline{d}^o_{human}$, leads to the following observation. Given a couple of objects o' and o'' which represent a couple of spouses, lets consider an arbitrary third object o and its dissimilarity with each of the initial objects. Suppose also $\underline{d}^o_{human}(o', o)$ is less than $\underline{d}^o_{human}(o'', o)$. Then, the following inequalities hold :

$$\underline{d}^o_{human}(o', o) \ < \ d^o_{human}(o', o) \ < \ \underline{d}^o_{human}(o'', o) \ . \tag{10}$$

The same inequalities hold for $d^o_{human}(o'', o)$. Besides, the values of both measures for the initial couple remain the same:

$$d^o_{human}(o', o'') = \underline{d}^o_{human}(o', o'') \ . \tag{11}$$

The underlying phenomenon is a kind of attraction between the objects of a couple, here o' and o''. Actually, the mutual influence of dissimilarity values for both objects to a third one tends to minimize their difference. This means that, within the space induced by the relational measure, both objects will lie somehow "nearer" than in the space induced by the simple measure, even if the absolute values of both functions remain equal.

Globally, the attraction results in a more compact dissimilarity matrices in the sense that the total variance of the values tends to decrease with respect to a non-relational measure with the same relative weights for property attributes. Of

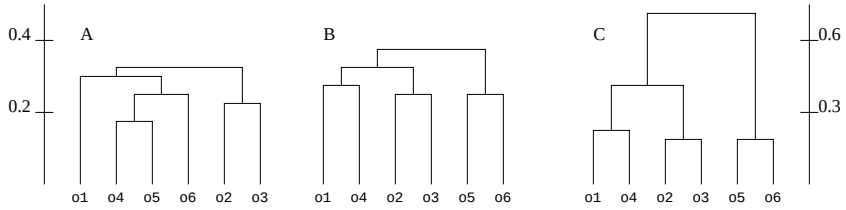

Fig. 3. Clustering results for the dataset: A. single linkage clustering with $\underline{d}^o_{human}$ as input; B. single-linkage clustering with d^o_{human} as input; C. complete-linkage clustering with d^o_{human} as input. The scale for first two dendrograms is given on the left and the scale of the last one on the right of the figure.

course, the above attraction phenomenon has been detected in a very particular situation where couples of objects of the same concept are strongly connected. However, a similar tendency can be observed in case of strongly connected components of greater size and of heterogeneous composition.

5 Clustering

The matrix obtained by the computation of d^o is used as an input for a hierarchical clustering algorithm which detects homogeneous object groups. An example

of clustering results may be seen on Fig. 3 where input data has been taken from Table 2.

Thus, Fig. 3.A shows the result of a single linkage clustering on $\underline{d}^o_{human}$ values, whereas the dendrograms on Fig. 3.B and Fig. 3.C are obtained with d^o_{human} by a single-linkage and complete linkage algorithms respectively.

A first remark concerns the attraction between related objects which goes as far as to change the dissimilarity-induced order between object couples. This could be observed on the matrix, but the dendrogram shows it even better. In fact, whereas `o4` forms a compact class with `o5` in the first case, on the second dendrogram it is combined with `o1`. This shift is undoubtfully due to the influence of the `o4`'s spouse, `o3`, which is nearer to the `o1`'s spouse then to `o5`'s. Both `o1` and `o4` are attracted to form a class with their spouses which may be seen on both dendrograms B and C.

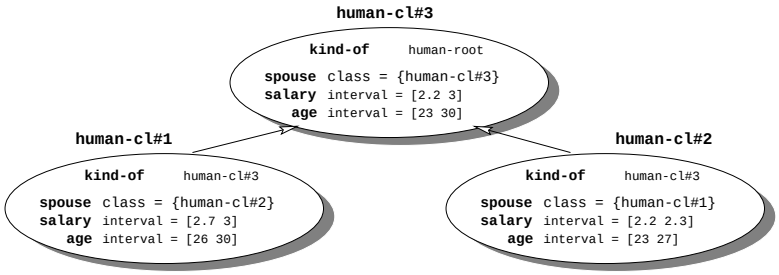

Fig. 4. Part of the class hierarchy obtained after characterization. of classes found by the hierarchical clustering.

As far as class inference is concerned, the clusterings with d^o_{human} suggest the existence of four classes below the root class. Before presenting them for user's validation, a characterization of each class in terms of attribute restrictions has to be provided. The description shows the limits of the region in the concept space represented by the class and thus helps in the interpretation. Fig. 4 shows the hierarchy made of three classes, corresponding to the object clusters {`o1,o4`}, {`o2,o3`} and {`o1,o4,o2,o3`}. Observe the cross-reference between classes `human-cl#1` and `human-cl#2` established via the `spouse` attribute.

6 Conclusion

An approach towards the automatic class design in object languages has been presented in the paper. Classes are built in two steps: first, member sets of classes are discovered by a proximity-based clustering procedure, then, each class is provided a characterization in terms of attributes.

The dissimilarity measure used for clustering compares objects with respect both to their properties and their relations. In case of self-references in object

descriptions, the values of the measure are computed as solutions of a system of linear equations. In addition, the measure allows available taxonomic knowledge to be explored for object comparison, but does not require taxonomies to exist. In sum, the measure is complete with respect to object descriptions and therefore allows the detection of clusters which are homogeneous on all object attributes.

References

1. H.W. Beck, T. Anwar, and S.B. Navathe. A conceptual clustering algorithm for database schema design. *IEEE Transactions on Knowledge and Data Engineering*, pages 396–411, 1994.
2. G. Bisson. Conceptual clustering in a first order logic representation. In *Proceedings of the 10th European Conference on Artificial Intelligence, Vienna, Austria*, pages 458–462, 1992.
3. G. Bisson. Why and how to define a similarity measure for object-based representation systems. In N.J.I. Mars, editor, *Towards Very Large Knowledge Bases*, pages 236–246, Amsterdam, 1995. IOS Press.
4. G. Booch. *Object-oriented analysis and design with applications.* Benjamin-Cummings, 1994.
5. J. Euzenat. Brief overview of t-tree: the tropes taxonomy building tool. In *Proceedings of the 4th ASIS SIG/CR classification research workshop, Columbus (OH US)*, pages 69–87, 1993.
6. D.H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.
7. J. Han, S. Nishio, H. Kawano, and W. Wang. Generalization-based data mining in object-oriented databases using an object-cube model. *IEEE Transactions on Knowlwdge and Data Engineering*, 25(1):55–97, 1998.
8. INRIA Rhône-Alpes, Grenoble (FR). *Tropes 1.0 reference manual*, 1995.
9. J.-U. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(2):193–217, 1994.
10. R. Michalski and R. Stepp. *Machine learning: an Artificial Intelligence approach*, volume I, chapter Learning from observation: conceptual clustering, pages 331–363. Tioga publishing company, Palo Alto (CA US), 1983.
11. P. Valtchev and J. Euzenat. Dissimilarity measure for collections of objects and values. In P. Coen X. Liu and M. Berthold, editors, *Proceedings of the 2nd Symposium on Intelligent Data Analysis.*, volume 1280 of *Lecture Notes in Computer Science*, pages 259–272, 1997.
12. B. van Cutsem. *Classification and dissimilarity analysis*, volume 93 of *Lecture notes in statistics.* Springer Verlag, New York, 1994.