OBJECT ORIENTED APPROACH IN CLUSTER ANALYSIS

František HUŇKA
Department of Computer Science, University of Ostrava, Dvořákova 7, 701 03 Ostrava 1,
Czech Republic, tel.: +420 596 160 221, E-mail: frantisek.hunka@osu.cz

SUMMARY

This paper deals with object oriented modelling in cluster analysis based on the Mjolner BETA System and the BETA object oriented language. The model is designed in layers, which enables more flexibility and clear understanding. Object oriented modelling enables one to create a model that is more natural, and it is easy to extend or change its functionality. These features predetermine such a model for making experiments. Cluster analysis has many methods and techniques for helping to solve classification problems. Object oriented models enables one to exploit them in a more complex way and thus receive interesting results. We use this model on medical data for making a diagnosis. With the benefit of the Mjolner BETA System we used rich means for composition, virtual mechanism and persistence. Persistence allows one to split the whole application into rather simple parts and to store the results for the next usage.

Keywords: *object oriented modelling, numerical clustering, BETA*

1. INTRODUCTION

One of the benefits of object-oriented programming is modelling. Object oriented languages provide excellent facilities for modelling phenomena and concepts from the real world. Using classes, subclasses and virtuals it is possible to describe classification structures and composition structures of phenomena and concepts. Classification and composition are fundamental means for most people to organize knowledge about a given application domain.

Cluster analysis is an exploratory method for helping to solve classification problems. Its use is appropriate when little or nothing is known about the category structure of a body of data. The objective of cluster analysis is to sort a sample of cases under consideration into groups such that the degree of association is higher between members of the same group and lower between members of the different groups. Cluster analysis is a modern statistical method of partitioning an observed sample into disjoint or overlapping homogeneous classes and provides an operational classification [1]. This classification may help to:

- assist identification (e.g. diagnosing);
- formulate hypotheses concerning the origin of population;
- predict the future behaviour of population type etc.

Using the object-oriented perspective we design a more complex and versatile tool suitable for sophisticated utilization. The system we chose for realization provides rich facilities for conceptual modeling. We used our model for making a diagnosis.

2. THE BETA PROGRAMMING LANGUAGE

The Beta programming language was designed as a successor to Simula67. Like Simula, Beta provides built-in support for co-routines and block structure. However, Beta improves and generalizes Simula in a number of ways. While procedures and classes can still be thought of as separate entities, they can both be specialized through inheritance and declared to be virtual. In fact, Beta removes any syntactic distinction between these two concepts and treats them as one. More general model and program structuring mechanism is called pattern. In Beta patterns are therefore employed to describe all of the classical programming concepts, such as procedures, functions, classes and processes. Beta also extends Simula by allowing singular entities, which can function as one-of-a-kind objects [9].

3. BASIC STRUCTURE OF THE MODEL

In order to make the model more flexible and clear we design a layered model. Each layer provides a specific function. Classes creating the object oriented model can be divided into three layers. The classes dealing with measured data of the clusters create the bottom most layer. In this layer there is a declared *generic-list* class, which is exploited as a super class for many other classes. Further specialized classes of *generic-list* class help to work with a row of measured values that belong to each cluster and help to organize this data into a list. This layer provides for principal component analysis and is exploited by its neighboring layer [4].

The classes declared as the basic classes necessary for clustering itself create the next layer. The main classes are classes describing the cluster and the list of clusters. The nature of clustering is an association between each pair of clusters. This association is mostly expressed by a real value. For describing this association we introduce an association class called *proximity*. Objects of this class are stored in a *proximity-list* class.

In the most top layer there are declared classes realizing different methods of clustering. We use a virtual mechanism for describing these methods [7].

3.1 Basic classes

Basic classes provide a necessary service for the next layers of the model. The main class is an abstract *generic-list* class that provides data structure and methods for the list of elements with a direct access. This list is created using repetition which is BETA construct similar to an array in other languages. Repetition has built in methods, which enable one to declare the dimension of the actual repetition at run time and extend current repetitions. Elements of the generic list are further specialized in subclasses using BETA's virtual mechanism [9].

Subclasses of the *generic-list* class declare the structure and operations carried out on the input data for clustering. For each cluster there is a row of real attributes with identification. The number of attributes depends on the application. These data are stored in *CVector* class. *CMatrix* class, which is mainly exploited in principal component analysis, is composed of *CVector* objects. At the beginning of clustering proximity values between each pair of clusters have to be calculated. There are many different algorithms to solve this problem. The other subclasses of *CVector* class are declared to solve the problem using a virtual procedure mechanism. Structure of the main classes of this layer is shown in the following listing of the reduced BETA code.

```
CGenericList: (* GenericList and some of its subclasses *)
(#   content:< Object; (* virtual class pattern *)
     Table: [0] ^Content; (* repetition *)
     Top,incr: @integer;
     increment: (#  do incr->Table.extend #); .. #);

CVector: CGenericList (* Declaration of vector *)
  (# ident: @integer;
     content::< realObject (* virtual class pattern further binding *) ;
     dissim:< (# .. #); .. #); (* virtual procedure *)

CSquareVector: CVector (* square Euclid distance for dissimilarrity *)
  (# dissim::< (# .. #) #); (* virtual procedure further binding *)

CWardVector: CSquareVector (* Ward coefficient dissimilarrity *)
          (# dissim::<  (#  ..  #) #);
CEklVector: CVector (* Euclid distance dissimilarity *)
  (# dissim::<  (# ..  #) #);
CMatrix: CGenericList (* Class matrix and its methods *)
  (# Content::< CVector (* virtual class pattern further binding *) ;
     noObj: (* number of clusters *) (#  exit N #);
     noAtr: (* number of attributes *) (#  exit M #);
     noObjNext: (* number of further added clusters *)
     (#  exit NX #); ..
  #);
```

3.2 Basic classes for clustering

This layer describes all general classes that are necessary for hierarchical clustering itself. At first it is the *cluster* class describing clusters themselves and the *list of clusters* class, which is a container of clusters. The *cluster* class contains two main attributes. A reference to a vector of measured values and a list of "merged" clusters. During the hierarchical clustering process clusters are grouped together. So the list of "merged" clusters is used for this purpose. This list allows for keeping a history of clustering.

The nature of the hierarchical clustering is an association (called proximity) between each pair of clusters. This association is expressed by a real value, which is changed during the clustering process.

The traditional hierarchical clustering approach works only with triangular matrix of associations and omits clusters completely. Clusters are used only at the beginning of the clustering process.

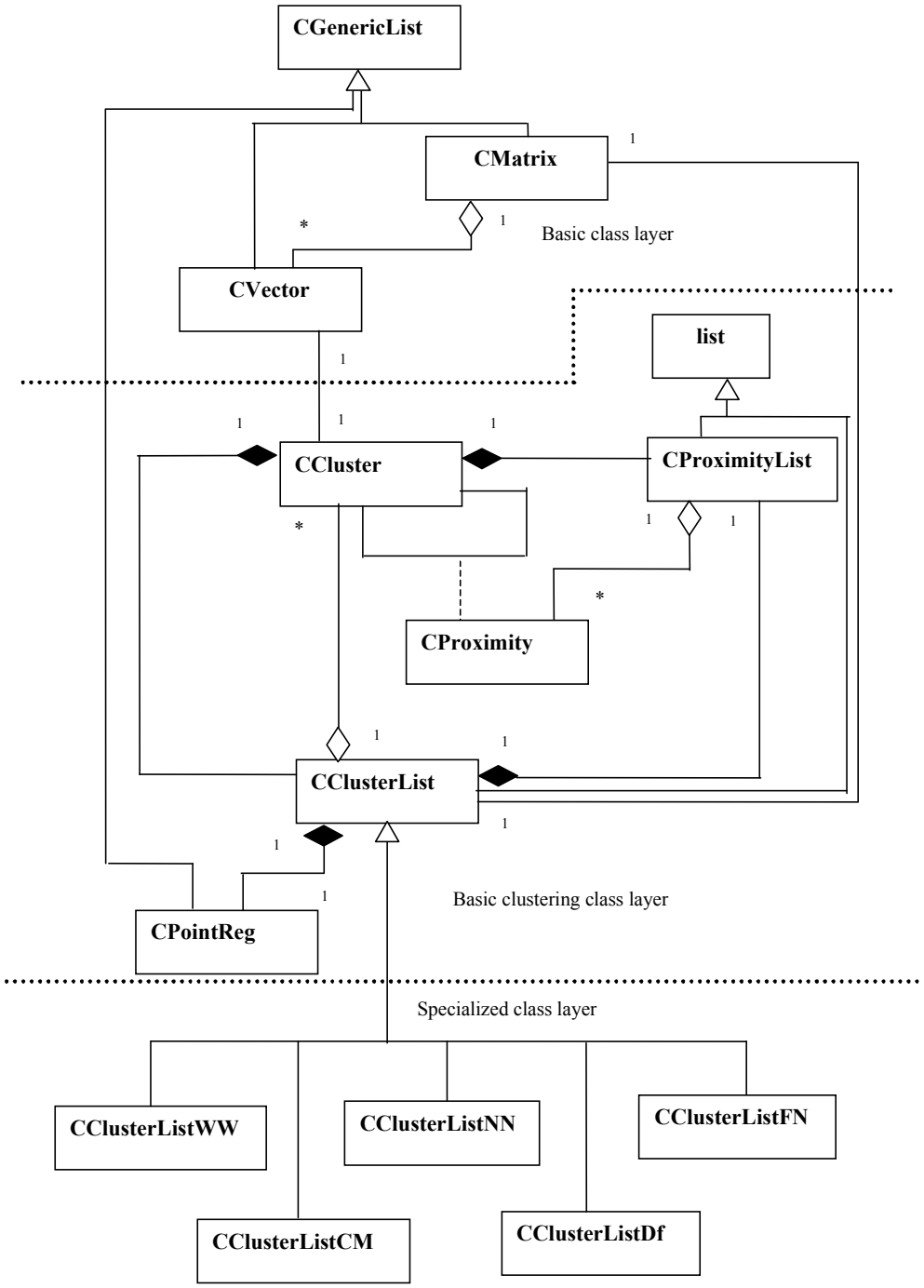
To cope with the association problem between each pair of clusters we introduce an association class between each pair of them.

This class holds association values between clusters and references to them. This approach allows one to compute association between a pair of clusters by different methods.

These associations are arranged in a list of proximities. In addition, each cluster has the list of its own proximities. (Actually it has only a list of references). This is done for making computation faster. The class diagram of all the layers is shown in the picture 1.

3.3 Specialized classes

Classes declared in this layer realize individual hierarchical clustering methods. In general we distinguish two types of hierarchical classification methods: traditional and definit methods. The traditional methods work only with a pair of clusters with minimum proximity value in each step of



Picture 1. Class diagram – hierarchical clustering

clustering. Contrary to the traditional methods definit methods work with a set of pairs of clusters, fulfilling a given interval in each step of clustering.

Both these main types have a lot of individual methods. For each declared method a special subclass of the *cluster list* class is created. Inside these classes virtual procedures are exploited. The common part of these methods is declared in the *cluster list* class and further binding is declared in a given subclass. In addition, there are two forms of algorithms for each individual hierarchical clustering method, recursive form and explicit form. Consequently each subclass contains at least virtual methods for explicit and recursive forms of computing the given proximity value.

4. PERSISTENCE

Beta is a persistent programming language and fulfils basic requirements for persistence:

- persistence is type orthogonal;
- persistence applies equality to all kinds of objects;
- object access is location independent.

A persistent object is an object that is saved on secondary storage during a program execution and thus survives the program execution in which it was created. A persistent object may be read by another program execution. By default, when an object is made persistent, all objects that can be reached through references are made persistent as well. The set of objects that can be reached from that object in this way is called the transitive closure of the object. Persistent objects are saved in a persistent store, which is a collection of persistent objects. An object may be pointed out to become a persistent root.

In the current BETA version it is possible to use one persistent store for one application [6].

Our intention of using persistence was motivated by two reasons. The first was the size of the data we wanted to process and the second was splitting the whole application into smaller parts (i.e. to separate graphical outputs from the computational parts). In terms of the first reason of using persistence, our model contains several lists (some of them "merged") and during the hierarchical clustering in some lists the elements are deleted while in the others lists they are added. During the experiments mainly the operation deletion proved to be "time consuming". Perhaps the way we used persistence is not quite the traditional way. On the other hand, in terms of splitting an application into smaller parts it was without any problems.

5. RESULTS

The designed model was tested on medical data, which are aimed at judging the risk of cardiovascular diseases. 16 attributes were measured for each tested person. In the first step we created 5 final clusters out of 500 single element clusters (tested persons). Each of these 5 final clusters was represented by the supposed risk of cardiovascular diseases (low, medium, high etc.) and in this way these clusters represent hypothetical classification structure. Then we calculated proximity values between new single element clusters and the classification structure. The results obtained in this way may be used for making a diagnosis and are presented in the table 2. The first column represents a new tested cluster. In the second column there are identifications of the clusters belonging to the hypothetical classification structure. The next three columns represent results of three different clustering computations [5].

new cluster identification	classification structure cluster-identification	partitional proximity	partitional nearest neighbour	hierarchical proximity
892	1	3.761314	3.064040	14.031
892	4	3.520515	2.289107	8.508
892	5	**2.427633**	**1.165088**	**2.361**
892	6	3.557192	1.792499	10.119
892	66	5.432594	3.393863	24.514
894	1	**2.283625**	**0.878928**	**5.172**
894	4	4.651128	3.614552	19.069
894	5	4.515607	2.425641	15.661
894	6	6.033694	4.029887	31.558
894	66	7.743661	4.803328	50.934

Table 2. Clustering results for making a diagnosis

6. CONCLUSION

Simplification of tasks is a principal way in which we are trying to cope with the complexity of models and programs. It is based on abstraction, i.e. the removal or aggregation of details, components and/or relationships. To decrease mental effort, we can divide a system into substructures, which can be then treated as primitives at this level and defined as composites at the level below.

Modularisation and hierarchies play an important role in this process, leading to the notion of the layered design as a basis for a powerful metaphor concerning organizing such layers of knowledge through locality of description [7]. Encapsulation ensures that objects can be insulated from a surrounding context by allowing access to internal representations only through well-defined interfaces; the so-called message protocols.

Finally, inheritance of structure and behaviour along with class/subclass links make it easy to extend systems by adding new classes (new layers). In this way special purpose libraries can be built, whose components are defined, tested and debugged incrementally and interactively.

The cluster analysis has many different methods and approaches for creating classification structure of the unknown data. It is done by the fact that

different data require different methods. Our experience convinced us that object-orientation is an appropriate style for modelling this problem.

Object oriented modelling enables one to create a more complex model of cluster analysis that brings new possibilities of exploiting cluster analysis than the traditional approach. It covers parallel calculations of unimeasure dissimilarity, interactive hierarchical clustering and combination of hierarchical and partitional clustering. Interactive hierarchical clustering enables one to add new clusters to a classification structure of clusters and continue with clustering without the necessity to start from the very beginning.

The nature of combination of the hierarchical and partitional clustering lies in the fact that classification structure produced by the hierarchical clustering is transformed into partitional clustering and is exploited for making diagnoses between a new single element cluster and the classification structure.

REFERENCES

[1] Backer, E.: Computer-Assisted Reasoning In Cluster Analysis, Prentice Hall, 1995

[2] Everitt, B.S., Landau S., Leese M.: Cluster Analysis, Arnold, 2001

[3] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996

[4] Huňka, F.: Aspects of Object-Oriented Modeling, CE&I '99 Conference proceedings, The Technical University Košice, Slovak Republic, pp.91-96, 1999

[5] Huňka, F.: Exploiting Clustering for Making a Diagnosis. MENDEL 2001. 8th International Conference on Soft Computing. Brno 2002. pp. 244-248. ISBN: 80-214-2135-5

[6] Knudsen, J. L.: Object-oriented Environment, Prentice Hall 1994

[7] Kreutzer, W., Osterbye, K.: BetaSIM A framework for discrete event modelling and simulation. Simulation Practice and Theory, 6 (1998), pp. 573-599

[8] Lukasová, A., Šarmanová, J.: Cluster Analysis Methods, SNTL Praha 1985

[9] Madsen, O. L., Moller-Pedersen, B., Nyggard, K.: Object-Oriented Programming in the BETA programming Language. Adison Wesley, 1993

BIOGRAPHY

František Huňka was born on 27.2.1953. In 1977 he graduated (MSc.) at the department of Cybernetics, Faculty of Mechanical and Electrical Engineering University of Transport and Communication in Žilina. He defended his PhD. in 1994 at the Faculty of Management University of Žilina. Title of his thesis was Distributed Systems Control Modelling. He is employed at the department of Computer Science University of Ostrava. He focuses his research in the field of object oriented technologies and simulation.