VARIOUS KINDS OF INHERITANCE

S. MOTOGNA* and V. PREJMEREAN*

Received: November 18, 1992
AMS Classification:68065, 68099

REZUMAT. Diverse tipuri de moştenire. Lucrarea de faţă se vrea un mic studiu comparativ, în funcţie de avantajele şi dezavantajele lor, a diverselor tipuri de moştenire implementate în limbajele orientate obiect. Studiul s-a făcut în principal pe limbajele Smalltalk, Beta şi CLOS deoarece exemplifică cel mai bine moştenirea simplă, multiplă şi în CLOS introducerea claselor abstracte numite mixin.

1. Introduction : A variety of inheritance mechanisms have been developed for object-oriented programming languages. These mechanisms range from Smalltalk single inheritance to the complex and powerful multiple inheritance combination of CLOS.

These languages have similar object models and also share the view that inheritance is an incremental modification mechanism, but they differ widely in the kind of incremental changes supported.

Inheritance is a hierarchical incremental modification mechanism that transform a parent P with a modifier M into a result R : $R = P + M$ as in figure 1.

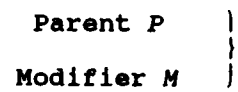


fig. 1

The parent P, modifier M and result R have a set with finite number of attributes :

* "Babeş-Bolyai" University, Department of Computer Science, 3400 Cluj-Napoca, Romania

$P = (P_1, P_2, \ldots, P_p)$

$M = (M_1, M_2, \ldots, M_m)$

$R = (R_1, R_2, \ldots, R_r)$

When the attributes of the modifier M differ from those of the parent P then the result R has $p+m$ attributes, contains the union of the P and M attributes. For the overlapping attributes, the modifier attributes redefine the parent attributes, in the same way as the identifiers declared in an inner encapsulated module redefine those declared outside the module.

The inheritance in Smalltalk, Beta, CLOS are representative of three kinds of inheritance. The inheritance mechanisms seem to be very different but they have a common structure. This mechanism combines two sets of attributes P and M such that duplicate attribute definitions are given a value from one set.

2. Single Inheritance : If we consider single inheritance, each class has at most one superclass, the determination of the class procedure list is trivial : we only need to traverse a linear path to the most general superclass of its inheritance hierarchy.

Inheritance in Smalltalk is a mechanism for incremental derivation of classes, it is a single inheritance and was adopted from Simula.

In Beta the inheritance is single, too and is designed to provide security from substituting of a method by a completely different method. Inheritance is supported by prefixing of definitions.

These two mechanisms are the same, only the direction of modification is different. In Smalltalk the new attributes are favored and may replace the inherited ones, in Beta the original attributes are favored.

3. Multiple Inheritance: The real world has in many situations to deal with multiple inheritance. The natural inheritance comes from two parents more than from one.

Let's consider a class T which inherits from superclasses T_1, T_2, ..., T_n :

class T inherits (T_1, T_2, ..., T_n) in T - body.

Some multiple inheritance systems, as CLOS extend the inheritance hierarchies, i.e. by ordering T_1, ..., T_n in a linear order from left to right.

The problem of the multiple inheritance is at the moment of invoking a method. If a method is defined in more than one superclass which of them should be invoked? There must be no conflict between characteristics inherited from independent classes (even if these characteristics have the same name).

There are some algorithms for linearization the hierarchy, for reaching a method but each of them has some disadvantages.

There are two kinds of strategies to solve the conflict problem in multiple inheritance : linear strategy and graph-oriented strategy.

The principle of linear strategy is that the inheritance graph should be transformed to a linear structure, without duplicates and treat the resulting graph as a single inheritance

one. This goal is realised ordering the superclasses list of a class using depth-first search or breadth-first search.

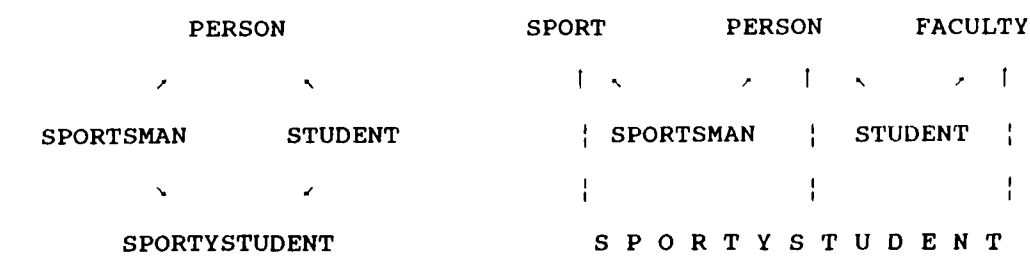
The graph-oriented strategy works directly on the inheritance graph without modifying it, allowing to access each inherited characteristic. When a conflict occurs the superclass from which we want to inherit must be specified . We mention that this class is not necessary the one which defines the characteristic. A technic which is used in extended Smalltalk is the selector composition. A composed selector is a selector preceded by the class name.

Using linearization, a CLOS multiple inheritance hierarchy could be reduced to a collection of inheritance chains, each of which can be interpreted using single inheritance.

4. Mixins : A mixin is an abstract subclass that may be used to specialize the behavior of a variety of parent classes. In contrast to classes, mixins are no objects which can create own instances. In mixins we may define new methods that perform some actions and then call the coresponding parent methods, but these methods are only textually. When a class is defined, the methods of its mixins will be defined for that class.

Stroustrup[6] had the following argument for using multiple inheritance : "it might be useful to have class *B* inheriting from two classes A_1 and A_2 ...". This is not possible using mixins but if we use factorization we obtain a solution for this.

VARIOUS KINDS OF INHERITANCE



using multiple inheritance using mixins and factorization

fig. 2

CLOS supports mixins and it seems to be not so difficult to extend Beta and Smalltalk to support mixins and generalized inheritance.

R E F E R E N C E S

1. Wegner, P. and Zdonik, S. B., *Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like,* in ECOOP'88 Proceedings,pp 55-77.
2. Ducournau, R. and Habib, M., *On Some Algorithms for Multiple Inheritance in Object Oriented Programming,* in ECOOP'87, pp 243-252
3. Bracha, G. and Cook, W., *Mixin-based Programming,* in ECOOP/OOPSLA'90 Proceedings, pp 303-311.
4. Bretthauer, H., Christaller, T. and Kopp, J., *Multiple vs. Single Inheritance in Object-oriented Programming Languages.* What Do We really Want, 1989.
5. Wegner, P., *Concepts and Paradigms of Object Oriented Programming* - Expansion of Oct. 4 OOPSLA'89 Keynote Talk, OOPS Messenger, vol 1, no 1, pp 7-87, Aug 1990.
6. Stroustrup,B., *The C++ Programming Language.* Addison Welsey, 1986.