Bul.St.Univ.Baia Mare
Seria B,Matematică-Informatică,Vol.VII,Nr.1-2,133-137

OBJECTS TYPES IN OBJECT-ORIENTED PROGRAMMING

by
Dan CHIOREAN

Object oriented programming,introduced in the 1960's by Simula has become an important programming framework for large software systems. Among the attractions of object programming languages is the potential for software reuse. Programming environments such as the Smalltalk-80 environment provide programming tools and examples in a set of pre-defined system classes,similar to a software library. Programming in such a system consists of directly using existing classes by creating derived classes by subclassing existing ones.

Subclassing,or inheritance,allows the programmer to define a new class of objects which inherit the procedures and state descriptions of an existing class. The programmer may then modify and extend the behavior of the base class,and add new state information,if desired. Because state information is static,subclassing state is straightforward:simply append new state information. However,behaviour is dynamic; the problem of subclassing behavior is a bit more complicated, since it may not be appropriate to simply concatenate the behaviors of the superclass and the subclass and execute them sequentially. This issue is referred to as the method combination problem.

An Overview of Object Oriented Programming

The main difference between object oriented programming and traditional programming is that object oriented programs are composed of objects,packages of data and related procedures. Typically,a programmer using an object oriented language thinks primarily of the data structures,followed by the procedures that act upon those structures. Traditional programs are composed of a collection of procedures which are independent of the data and related only by conventions; programmers using traditional languages often consider first the procedures,and then the data structures.

Typically procedures act only on certain types of data;for example, the procedure DRAWSTRING(str) expects a String as its argument and DRAWRECTANGLE(rect) expects data of type Rectangle. Although compilers can check to see if arguments are correctly typed,it is still up to the programmer to determine which procedure is the correct one to use in each case. By closely relating data and the procedures that act

upon the data,object oriented languages alleviate this problem. For example,in an object oriented system,drawing a string could be done by executing Draw(str,location) and drawing a rectangle by executing Draw(rect,location);finding the correct procedure to execute is handled by the support system of the language and is based on procedure's actual parameters. This seems like a minor advantage, but having language support for choosing the appropriate procedure contributes strongly to the usability of the language and allows the programmer to focus on the semantics of the expression,rather than the appropriate syntax.

Object oriented programming has been used traditionally in symulation systems,due to the easy of describing simulation entities and their relationship as objects. More recently,user interface systems,including window managers,have been successfully designed using object oriented languages and programming styles.

Concerning the object oriented languages,their classification can be made according to the possibility in which they implement the following seven concepts:

-- object
-- class
-- inheritence
-- data-abstraction
-- strongly typed
-- concurrency
-- persistence

There is a potential language class,correspondind to each of the 128 existing subsets and which results from the mixage of the concepts presented above,some of these classes having a particular interest.

The aim of this paper is to establish the place which object oriented languages ocupie in this class and to present some of their main problems.

Terminology

Object oriented programming is infamous for introducing new terminology and there are many different terms that are used for similar concepts in different languages. Smalltalk-80 is one of the earliest and most successful object oriented systems; for consistency throughout this paper the Smalltalk-80 terms,usage and syntax will be used. However,the syntax of program examples will diverge slightly from true Smalltalk syntax for readability.

An object is the fundamental data structure in object oriented

languages.

Object,method,message

In object oriented languages,an object is the fundamental data structure. In paper "Smalltalk-80" , the object is defined as "a component of Smalltalk-80 system,represented by a storage location and an operations set". In a homogeneous system as Smalltalk or Oaklisp,everything in the system is an object. However,most object oriented systems are not homogeneous. They are implementations of object oriented programming added to existing languages;in these systems programming can be a mix of object oriented,procedural and functional styles(see Turbo Pascal 5.5 and 6.0. , Turbo C++ , etc).

In [3] , the object is defined as "a set of operations and a local shared state that remembers the effect of operations". Different language subcultures have fundamentally different notions of what an object is . So, there are the following types:

1)Functional objects arise in functional object oriented languages such as OBJ2,and in logical ones,such as Vulcan. Their interfaces are object-like,but the objects lack an identity that persists between operations. An operation that transforms the state results in the creation of a new object with the given interface and a new state.

2)Passive or "server" objects become active when a synchroneous message or a remote procedure call invokes their operations. These are traditional objects of Smalltalk and C++ and are the default when I use the term Object whithout qualification. This lets me include the packages of Ada,the modules of Modula-2 and the objects of Simula and CLU among the entities I call objects.

3)Active or "autonomous" objects can execute autonomously even when other objects do not invoke them. They include monitors,strongly coupled concurrent objects and distributed concurrent objects.

4)Slot-based objects are defined in terms of their instance variables,or slots,as in Flavors and CLOS(Common Lisp Object System). You can dynamically add methods for slot-based objects with DEFMSETHOD operations,which specify the class,or flavor,to which you want to add them.

The physical structure of an object is defined by its instance variables. These are the slots in the object which may be filled with values or references to other objects. Object behavior is defined through methods , procedures which are run in response to a message sent by another object. Methods may directly access the instance variables of the receiver , the object to which the message was sent. The receiver is the entity which,along with the message

name, determines the correct method to run. We may say, then, that the "method" describes the realisation mode of an operation that an object knows to do and the "message" is a request addressed to an object to do one of its operation.

Messages may have arguments which are passed along to the method. In the previous example, executing Draw(str, location) would consist of sending the message Draw to the object str, the receiver with the argument location.

In most cases only the receiver and the message are used to determine which method to run. Some systems, such as CommonLoops, may use the classes of all the arguments to the message to find the correct method.

The syntax of message sending varies from language to language. Sending the message fill with arguments color and mode to the object myRectangle is done in several different languages as follows:

```
myRectangle fill:color transfer:mode        Smalltalk
(fill myRectangle color mode)               Oaklisp, CommonLoops
(= myRectangle:fill color mode)             CommonObjects
```

Note that "Sending the message fill to the object myRectangle" means "run myRectangle 's method named fill on itself". This terminology is unfortunate in that it implies objects may concurrently execute methods along with other objects, which is quite uncommon in object oriented languages.

The names of messages sent to objects, called message selectors, are the object's interface to the outside world. The important aspect of a message selector is that it is only a name for the desired action, describing what the programmer wants to happen, not how it should happen. Dan Ingalls, one of the creators of Smalltalk, states that "The message sending metapher provides modularity by decoupling the intent of a message embodied in its name from the method used by the recipient to carry out the intent".

The Smalltalk syntax is quite different than the different Lisp dialects; the receiver is first, followed by the message and arguments in infix form. The full message selector in the Smalltalk example is fill:transfer:; the arguments appear after keywords that terminate with colons.

The set of messages that can be handled by a particular object is defined by the object's class and is called a Message protocol .

Objects can be modeled by automata whose state represents the object's state and whose input symbols represent operations with their arguments. Figure 1 illustrates an object with operations f_1,

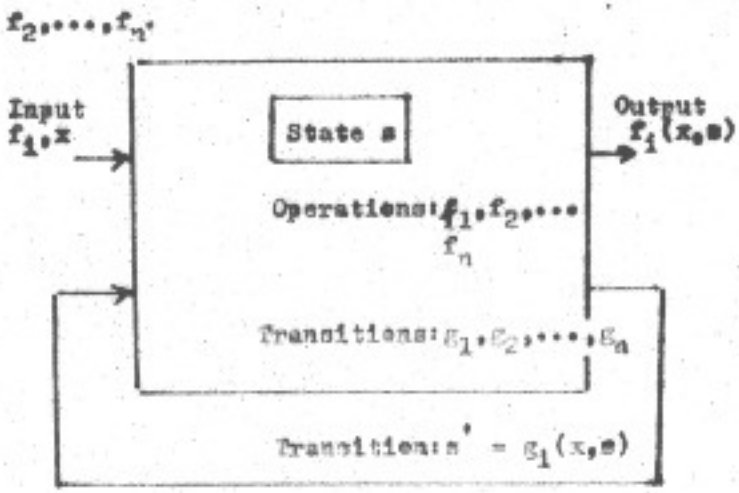

Figure 1. An object with operations $f_1,f_2,\ldots,f_n$. Thus operations symbols f_i are associated with symbols on the input tape, internal actions are associated with the state transition function $g_i(x,s)$ and outputs are associated with operations $f_i(x,s)$

An operation f_i with argument x in state s results in output $f_i(x,s)$ and state transition $s' = g_i(x,s)$. Thus operation symbols f_i are associated with symbols on the input tape, internal actions are associated with the state-transition function $g_i(x,s)$ and outputs are associated with operations $f_i(x,s)$.

REFERENCES

1 Horn, B.L., An Introduction to Object Oriented Programming, Inheritance and Method Combination, Carnegie Mellon Univ., Computer Science Dept., Jan. 1988

2 Thomas, D., What's in an Object, Byte, March 1989, pp. 231-240

3 Wegner, P., Learning the Language, Byte, March 1989, pp. 245-253

UNIVERSITY OF CLUJ-NAPOCA
3400 Cluj-Napoca ROMANIA