4A-4 On the Principles behind Object-oriented Facilities of Common Lisp for Alien Languages

Masayuki Ida, Satoshi Uchida
Aoyama Gakuin University

This paper describes the following two points:
(1) The essences of CommonLOOPS as an object-oriented language.
(2) How to export the essences to other alien languages.

1. Introduction

This paper describes the principles and key designs of CommonLOOPS-in-alien languages (CL-in-Alien), for languages such as assembly language or C. CommonLOOPS[Bob85] is an object oriented facility proposal for Common Lisp [Ste84], and is discussed as a candidate for the standard facility at IJCAI'85 Common Lisp object oriented subcommittee where the first author attended.

Though CommonLOOPS is mostly based on LOOPS, the notations of the syntax are different from LOOPS[Bob83]: the syntax of Common LOOPS is quite simpler than that of LOOPS, nevertheless the functionality supported by CommonLOOPS is more flexible and stronger than that of LOOPS. Some aspects of CommonLOOPS are influenced by Flavors[Wei83]. The specification of CommonLOOPS is provided by the two stages: kernel and extension.

The first author has continuous interest to Common Lisp as the translator of the book[ida85a], as the chair of the Jeida committee [Jei85], and as a researcher [Ida85b] [ida85c]. He traced the discussions of Common Lisp object oriented subcommittee since October 1984. The second author has an academic interest in data abstraction in assembly language environment [Uch85a] [Uch85b] and in reusable environment [Uch85c]. This paper is the first step of them to find simple and common mechanisms for object oriented facility.

In chapter 2, the brief survey of the design goals of CommonLOOPS, the fundamental reasoning of the object oriented facilities in CL-in-Alien, are described. In chapter 3, the most principal mechanisms of CL-in-Alien are described. In chapter 4, the algorithmic essences of CL-in-Alien comparing CommonLOOPS are described. In chapter 5, as a summary, the future plans are presented.

2. The brief survey of the principles of CommonLOOPS and the fundamental reasoning of CL-in-Alien

2.1 the basics

We define object oriented programming as a programming style (methodology). It leads the conception that the problem is how to supply rich set of primitives. On the other hands, many programming languages became to have three style: 1) structured programming, 2) modular programming, 3) data abstraction. By adding object oriented facility to above three methodologies, the language will have another kind of richness. They are supported by:
1) class definition, method definition
2) method specialization
3) message sending feature
4) multiple inheritance
5) method combination
6) annoted value (active value)

2.2 Goals of CommonLOOPS and CL-in-Alien

i) CommonLOOPS has the following major goals[Bob85].
1. Compatibility
2. Small kernel
3. Powerful base
4. Universal kernel
5. Common Lisp style
6. Efficiency
7. Extensibility

ii) CL-in-Alien has the following goals and meanings:
1. Compatibility: the mechanisms are compatible with Common Lisp based CommonLOOPS. This gives us the same training base, and the same terminology, and the same OS primitives.
2. Small kernel: Simple and small enough to attach object oriented facility to many application written in other languages.
3. Powerful base: There are two ways for powerful base. One is an approach like exploratory programming [She83], in which all the support tools should be included in a running environment, and environments should be dynamically changeable. The other is in static environment and more tend to efficiency. In the alien environments, some may be in rich environment and some may be in a slim environment. Then the goal should not restrict to supply of rich environments.
4. Universal kernel: In the CommonLOOPS document, Universality means that Flavors, LOOPS, and Smalltalk can be implemented upon the same kernel. In our context, universality means the same kernel for various languages.

5. Common Lisp style: This goal cannot be achieved as for syntactic notations.
6. Efficiency: For the application which needs high performance, the system should supply optimized codes without overhead. But for the applications which needs guards and run time checking, the system should supply rich environments.
7. Exensibility: We need extensibility with the same context as CommmonLoops.

3. the Principal mechanisms of CL-in-Alien

3.1 Dynamic vs static

i) Dynamic environment

When an object orient facilities is incorporated into an alien language, the important factor is whether the language has dynamic facilities: dynamic memory allocation, dynamic function call changing, etc. Since object oriented programming style is mainly evolved in the Lisp environment which has more dynamic environment among various programming languages, object oriented programming style has a lot of dynamic facilities such as instance creation, addition or modification of method in running time, and method combination.

Sorry to say, most languages except Lisp does not have a mechanism which realizes a dynamic environment. So a run-time manager module has to be introduced. The basic facilities of the run-time manager module is as follows.

(a) a run-time memory management:
mainly for creation or deletion of an instance
(b) a run-time function invocation mechanism:
mainly for method specialization, method combination

The other features of object oriented programming style such as classes, methods, message sending may be realized by using the built-in mechanism of conventional languages.

ii) Static environment

Object Oriented Programming system is desirable to be established in a conversational style in a dynamic environment. But from the view point of a compactness of a processor and run-time efficiency, it is a desirable case that the system is still in a static environment. To do this, the following steps are introduced.

(a) All the instances of classes are created in compiling time.
(b) A variable should be strongly typed. And the type of a variable is never changed.
(c) Method search is all solved in compiling or linking stage.
(d) A database for classes and methods is introduced so that an optimizing compiler and linker can utilize at any time.

In this way, a compact and efficient code can be generated.

3.2 Principal mechanisms for method search

The principal mechanisms of CL-in-Alien should be compatible with those of CommonLOOPS. In CommonLOOPS, message sending has the same syntax as function call. As for multiple inheritance, CommonLOOPS has both LOOPS style and Flavors style. Method combination of Flavors style is introduced in the extension part.

Key mechanisms of CL-in-Alien are the algorithms for method search. They may be in compile phase or execution phase as requests. There are two assumptive data structures:

1) A method name has a method-precedence-list, a pointer-list or a table whose entries are pointers for the method's definition (or body). See Fig.1. Upon the defining method, the definition will be added to the proper position in the precedence-list for the method name. See Fig.2. The proper position means the position which reflects the correct precedence for method search.

2) there should be a database which has informations concerning the positional relations and types of slots(instances) of each classes. See Fig.3.

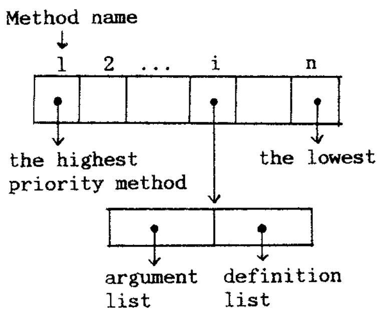

Fig.1 Method-precedence-list

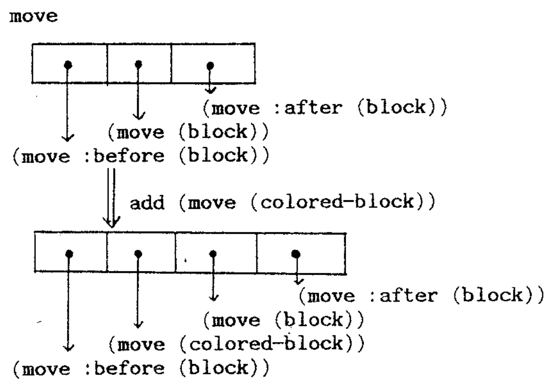

Fig.2 Adding a method to a method-precedence-list

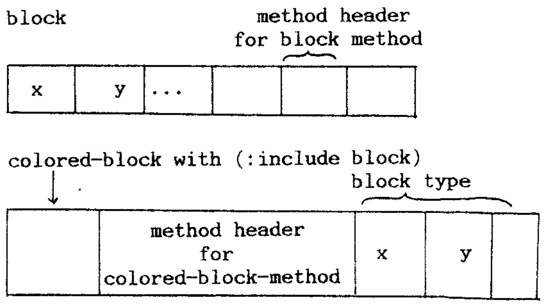
block
method header for block method
x
y
...
colored-block with (:include block)
block type
method header for colored-block-method
x
y

Fig.3 classes and slots

With these assumptive data structures, following steps take place on method search;
a) Get the types of the arguments (or the specific value assumption to the arguments). Then find the first method in the precedence-list.
b) If inherited or combined, there is a possibility to execute further methods which mate the situations, after investigation into the precedence-list.
c) If Flavors like :before or :after is specified, it needs primary method. But, this type of method combination is easily possible with the proper order of the method definition list. Flavors type combination needs to check the completeness upon finishing the definition input, or to dynamic check upon execution.
d) If there is a run-super specification, the next method in the precedence-list is sequentially executed.

The basic mechanisms are integrated and lie in the run-super type execution rule.

4. Key elements of CL-in-Alien and CommonLOOPS

CommonLOOPS has following functionalities.
(1) Class Definition
(2) Method Definition
(3) Method Specilization
(4) Message Sending
(5) Multiple inheritance
(6) Method Combination
(7) Annoted value
In the following sections, we describe the above 7 features respectively.

4.1 Class definition

i) CommonLOOPS spec

Classes are defined by defstruct. As the defstruct-options,
:class class [LOOPS style class],
:class flavor [Flavors style class],
:include (superclass-name-list) [for multiple inheritance]
are introduced.

The form of slot-option is same as COMMON Lisp. As for option in slot-description, :allocation is introduced. The :allocation option can have values class (for class variable), instance (for instance variable), dynamic (for allocated as an instance variable in running time), or none (for not inherited from its superclass).

ii) CL-in-Alien features

The mechanism for classes may be realized by using the built-in facilities of conventional languages. As CommonLOOPS does, a facility for struct or a record can realize classes by using the extension of them. In the language which has neither a struct of a record, a facility which creates an offset structure can be used for the purpose. Take an assembly language for example, EQU pseudo operation is adoptable to describe classes. See Fig.4.

```
(defstruct moving-object
           ((x-velocity float)
            (y-velocity float)
            (mass float))
           )
```
(a) A Class Definition in CommonLOOPS

```
; Example description of classes in an assembly
; language stored in a file named "moving-object"
  moving-object-size equ 12
  x-velocity         equ 0
  y-velocity         equ 4
  mass               equ 8
```
(b) A Class Definition using EQU pseudo op.

Fig.4 Class Definition in CL-in-Assembler

4.2 Method Definition

i) CommonLOOPS spec

Method is defined as a Lisp function using "defmethod". If you want to define as Flavors style method combination, :after or :before is specified as an additional option.

ii) CL-in-Alien features

The mechanism for methods can be realized by using the function definitions. In the simplest case, as an actual name of a method, the name consisted of "class-name.method-name" is used if there is no method specialization and method combination. The name may be accepted by a conventional linker.

4.3 Method Specialization

i) CommonLOOPS spec

One of the most particular features of CommonLOOPS is that the type of argument can be specialized. In other words, same method name on the same class may be defined if the argument types are different. The most "specific" method is executed if no method combination is specified.

ii) CL-in-Alien features

In most of the conventional languages, method specialization had better solved in compiling time or linking time. This is possible if the type of a variable is not changed in a running time. The basic mechanism to search the most "specific" method in compilation is implemented by using a database which treats method names and method argument type as described in 3.2.

4.4 Message sending

i) CommonLOOPS spec

In CommonLOOPS, the notation of message sending is just like as a function call in Lisp. However, when the options for method combination such as :after or run-super is specified, the other method will be executed.

ii) CL-in-Alien features

The message sending itself is realized by a simple function call with argument check. But in the case when the method search is required by the specification of method specialization and method combination, the dynamic mechanism for controlling the method invocation is needed as described in 3.2.

4.5 Multiple inheritance

i) CommonLOOPS spec

In CommonLOOPS, multiple inheritance is defined by :include option in defstruct. The searching algorithm is same as that of LOOPS.

ii) CL-in-Alien features

In most of the conventional languages, an extension of include statement or copy statement are used for a realization of multiple inheritance. The basic mechanism to implement multiple inheritance is already showed in 3.2.

4.6 Method combination

i) CommonLOOPS spec

The basic mechanism for method combination in CommonLOOPS is run-super. run-super is same as send-super in LOOPS. And in the extension part, method combination of Flavors style such as :after or :before is also discussed.

ii) CL-in-Alien features

The mechanism for a pointer to a function (in an assembly language, C, FORTRAN, Pascal, etc) can be used to realize method combination in an executing time using the algorithm described in 3.2. In the case, some run-time manager for method invocation will be required.

4.7 Annoted value

i) CommonLOOPS spec

An annoted value is same as an active value in LOOPS.

ii) CL-in-Alien features

Creation of slot-access function is used for the implementation of an annoted value. For an annoted value, get-function or put-function is prepared. The most simplest steps in slot-access function are load(store)-value instruction and return instruction. If a procedure for get(put) is coded, it is concatenated to the above two instructions. When the value is accessed, the required function will be invoked and the slot access will occur.

5 Final remarks

This paper showed the way to port the principles and abstract common mechanism of CommonLOOPS to alien languages. We are going to clarify the ambiguity of the CommonLOOPS specification.

After that, our next steps are

(a) Implementation of CommonLOOPS on APCL (Aoyama Personal Common Lisp).

(b) Incorporation of object oriented programming style to an alien language, especially an assembly language.

References

[Bob83] Bobrow.D.G.,Stefik.M.J.: The Loops Manual, Xerox, 1983.

[Bob85] -,et.al.: CommonLOOPS, Xerox, 1985.

[Ida85a] Ida,M.:Japanese version of [Ste84], Kyoritu Pub. Corp. 1985.

[Ida85b] -: Comments on Common Lisp, bit, 1985 (in Japanese)

[Ida85c] -: A Proposal of a subset of Common Lisp WGSYM IPSJ, Nov. 1985, (to be appeared)

[Jei85] The Report of the Jeida Common Lisp Committee 1985, (to be appeared)

[She83] Sheil,B.: Power Tools for Programmers,Datamation, February, pp.131-144,1983.

[Ste84] Steele,G,L.: Common Lisp: the language, Digital Press. 1984.

[Uch85a] Uchida,S.,Mano,K.: A Structured Programming and Modular Programming Facility for the CP/M-86 Assembly Language (in Japanese),WGMC IPSJ, No.36-1,1985.

[Uch85b] -,-: A Data Abstraction Facility for the CP/M-86 Assembly Language (in Japanese),WGMC IPSJ,No.36-2,1985.

[Uch85c] -,-: A reusable environment for an Assembly Language of Micro Computer Based on Data Abstraction, (in Japanese), The Summer Symposium of IPSJ, July 1985.

[Wei83] Weinreb.D.Moon D. Lisp Machine Manual, Symbolics Inc.1983.