The Common Lisp Object System: An Overview

by

Linda G. DeMichiel and Richard P. Gabriel
Lucid, Inc.
Menlo Park, California

1. Abstract

The Common Lisp Object System is an object-oriented system that is based on the concepts of generic functions, multiple inheritance, and method combination. All objects in the Object System are instances of classes that form an extension to the Common Lisp type system. The Common Lisp Object System is based on a meta-object protocol that renders it possible to alter the fundamental structure of the Object System itself. The Common Lisp Object System has been proposed as a standard for ANSI Common Lisp and has been tentatively endorsed by X3J13.

2. History of the Common Lisp Object System

The Common Lisp Object System is an object-oriented programming paradigm designed for Common Lisp. The lack of a standardized object-oriented extension for Common Lisp has long been regarded as a shortcoming by the Common Lisp community. Two separate and independent groups began work on an object-oriented extension to Common Lisp several years ago. One group is Symbolics, Inc. with New Flavors, and the other is Xerox PARC with CommonLoops. During the summer of 1986, these two groups met to explore combining their designs for submission to X3J13, a technical working group charged with producing an ANSI standard for Common Lisp.

At the time of the exploratory meetings between Symbolics and Xerox, the authors of this paper became involved in the technical design work. The major participants in this effort were David Moon and Sonya Keene from Symbolics, Daniel Bobrow and Gregor Kiczales from Xerox, and Richard Gabriel and Linda DeMichiel from Lucid.

By March 1987 this three-way collaborative effort had produced a strong draft of a specification for the bulk of the Object System. X3J13 has voted an endorsement of that specification draft, stating that it would almost certainly be adopted as part of the standard and encouraging implementors to proceed with trial implementations. This paper is a report on the specification that was presented to X3J13.

© Springer-Verlag Berlin Heidelberg 1987
3. The Common Lisp Object System View of Object-Oriented Programming

Several aspects of the Object System stand out upon inspection: a) it is a layered system designed for flexibility; b) it is based on the concept of generic functions rather than on message-passing; c) it is a multiple inheritance system; d) it provides a powerful method combination facility; e) the primary entities of the system are all first-class objects.

3.1 The Layered Approach

One of the design goals of the Object System is to provide a set of layers that separate different programming language concerns from one another.

The first level of the Object System provides a programmatic interface to object-oriented programming. This level is designed to meet the needs of most serious users and to provide a syntax that is crisp and understandable. The second level provides a functional interface into the heart of the Object System. This level is intended for the programmer who is writing very complex software or a programming environment. The first level is written in terms of this second level. The third level provides the tools for the programmer who is writing his own object-oriented language. It allows access to the primitive objects and operators of the Object System. It is this level on which the implementation of the Object System itself is based.

The layered design of the Object System is founded on the meta-object protocol, a protocol that is used to define the characteristics of an object-oriented system. By using the meta-object protocol, other functional or programmatic interfaces to the Object System, as well as other object systems, can be written.

3.2 The Generic Function Approach

The Common Lisp Object System is based on generic functions rather than on message-passing. This choice is made for two reasons: 1) there are some problems with message-passing in operations of more than one argument; 2) the concept of generic functions is a generalization of the concept of ordinary Lisp functions.

A key concept in object-oriented systems is that given an operation and a tuple of objects on which to apply the operation, the code that is most appropriate to perform the operation is selected based on the classes of the objects.

In most message-passing systems, operations are essentially properties of classes, and this selection is made by packaging a message that specifies the operation and the objects to which it applies and sending that message to a suitable object. That object then takes responsibility for selecting the appropriate piece of code. These pieces of code are called methods.

With unary operations, the choice of a suitable object is clear. With multiary operations, however, message-passing systems run into problems. There are three general approaches to the problem of selecting a suitable object to which to send a message: currying, delegation, and distribution of methods.