

Four Metaphors for Programming

Daniel G. Bobrow
Mark J. Stefik

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304
USA

You may find
words here.
This package is
document answer
Rule

Daniel
Here's the current
version of the
part of the
metaphors

Abstract. Programming metaphors provide us with ways of thinking about programming, and with ways for organizing programs. The most familiar metaphor in computer programming languages is the *procedure-oriented* metaphor. In this metaphor, programs are composed of procedures that call each other, and operate on data organized in data structures. Alternatively, in the *object-oriented* metaphor programs are composed of objects that have aspects of both procedures and data. Objects communicate with each other by sending messages. This approach makes it convenient to define program interfaces in terms of message protocols. The *data-oriented* metaphor is dual to this approach. In the object-oriented metaphor objects send each other messages, causing data to change by side effect. In the data-oriented metaphor an object may change its data, causing a message to be sent by side-effect. The data oriented approach makes it convenient to create programs that monitor the workings of other programs, and which "wake up" when certain data are changed. Lastly, in the *rule-oriented* metaphor programs are composed of rule-sets that can be presented with data. Within a rule-set the invocation of rules is guided largely by patterns in the data. In the typical case, rules correspond to nearly-independent patterns in the data. The rule-oriented approach is convenient for describing flexible responses to a wide range of events.

Our experience suggests that programs are easier to describe in a language when there is an available programming metaphor that matches the structure of the problem. In the paper we seek to identify the sources of power and the breadth of application of the different metaphors. Examples are drawn from our experience in designing and using LOOPS, a programming system that adds data oriented and object oriented programming to Interlisp. We also survey some variations that have been used in various programming languages.

1. Introduction

The thesis of this paper is that the different metaphors for programming fit different kinds of applications, and that a programming system that provides a variety of metaphors allows users to pick ones that best fit their applications. Metaphors provide us with ways of thinking about programming, and with ways of organizing programs. AI Perlis once remarked that three things define a programming language: data, operations, and control [Perlis69]. This paper is about metaphors for programming computers oriented around procedures, objects, data, and rules. Each of these metaphors emphasizes an approach to organizing the data, operations, and control of programs, and each provides a different way to divide up a large program.

Core to read it with...
in light of the...
on in the Rule paper

Procedure-oriented Metaphor. The procedure oriented metaphor is the dominant metaphor provided in most programming languages today. Two kinds of entities are distinguished: procedures and data. Procedures are active and data are passive. Programs are composed out of procedures that act on data structures. The procedure-oriented metaphor encourages a style of programming in which a programmer visualizes the tasks to be done, and divides the tasks into smaller and smaller steps corresponding to individual procedures and eventually to individual instructions.

Object-Oriented Metaphor. In contrast with the procedure-oriented metaphor, programs are not primarily partitioned into procedures and separate data. Rather, a program is organized around entities called objects that have aspects of both procedures and data. Objects have local procedures (methods) and local data (variables). All of the action in these languages comes from sending messages between objects. Objects provide local interpretation of the message form. The object-oriented metaphor encourages a style in which programs are divided into classes of objects that contain certain kinds of information and which have specified behaviors that are invoked by certain kinds of messages.

The object-oriented approach is well suited to applications where the description of distinct kinds of entities is simplified by the use of uniform protocols. For example in a graphics application, windows, lines and composite structures could be represented as objects that respond to a uniform set of messages (i.e., *Display*, *Move*, and *Erase*). A program to manipulate these different kinds of objects can work with each of them uniformly, using the protocol. An important feature of these languages is an inheritance network, which makes it convenient to define objects which are *almost like* other objects. Objects are arranged in the network so that specialized classes of objects inherit much of their structure and protocols from superclasses. This works together with the use of uniform protocols because specialized classes of objects usually inherit the protocols of their super classes.

This metaphor was pioneered by Smalltalk ([Goldberg82], [Goldberg81], [Ingalls78]), and has its roots in SIMULA and in the concept of data abstraction. Smalltalk is a system based completely on the object-oriented paradigm. In addition, packages have been added to various systems to facilitate programming in that style. For example, the Flavors package [Cannon82] supports this style of programming in the MIT Lisp Machine environment [Weinreb81].

Data-Oriented Metaphor. In both of the previous metaphors, the invocation of procedures (either by direct procedure call or by message sending) is convenient for creating a description of a single process. Data oriented programming is appropriate for interfacing between nearly independent processes. In the data-oriented metaphor, action is potentially triggered when data are accessed.

A good example of this is the construction of a viewer for an independent traffic simulation process. The viewer provides a visual display of the changing traffic simulation process without affecting the code for the simulation. This independence means that the code for the two processes can be written and understood separately. It also means that the interactions between them can often be controlled by changing the attachments to particular data, but without changing the code.

Variations of the data-oriented programming metaphor have appeared in several frame

languages (e.g., KRL ([Bobrow77a], [Bobrow77b]). The idea also draws on the notion of *demons* as in Planner [Sussman70].

Rule-Oriented Metaphor. Emphasis on control. Each rule-set characterizes decision-making as triggered by a set of data. Emphasis is on the choice. Programs are composed hierarchically out of rule-sets, which may invoke each other.

Characterize as situation-action. Give example of simulation.

Site Post, Hayes-Roth, Davis and King, Newell.

Organization of this Paper. This paper is intended as a tour of the three metaphors for programming in terms of programming concepts, language features, and example applications. Our vehicle for the tour is a programming system, call LOOPS, that we have developed and started to use during the past two years. Using LOOPS, all three metaphors are available within the Interlisp environment [Teitelman78]. The term LOOPS is an acronym for Lisp Object and Data Oriented Programming System. We will use the LOOPS language to illustrate language features that seem essential for supporting the object and data-oriented metaphors, and LOOPS syntax for the examples of applications. Along the way we will compare LOOPS features to those found in closely-related languages.

Section 2 begins the discussion of object-oriented programming by introducing the concepts of classes and instances. Section 3 introduces the notion of an inheritance network, and section 4 discusses issues for using an inheritance network that allows multiple super classes. Section 5 discusses *knowledge bases* which provide a virtual memory for objects, and *environments* which provide important support for system design using objects. Section 6 illustrates the notion of active values, the basic language feature that supports the data-oriented metaphor in LOOPS. Sections 7 and 8 introduce some useful programming concepts for these systems. Section 7 considers the notion of *perspectives* and contrasts it with the use of multiple superclasses. Section 8 considers the creation of embedded *description languages* and illustrates the idea with two examples of composite objects. Section 9 illustrates an important programming concept, called *perspectives*, which is closely related to the notion of multiple super classes.

2. Structure of Classes and Instances

Classes. A class is a description of one or more similar objects. An instance is an object described by a particular class. Every object within LOOPS is an *instance* of exactly one *class*. Classes whose instances are themselves classes are called *metaclasses*. Most classes are instances of a metaclass called *Class*.

Variables. LOOPS supports two kinds of variables - class variables and instance variables. Class variables are used to contain information that is shared by all instances of the class. A class variable is typically used for information about a class taken as a whole. Instance variables contain the information specific to an instance. Both kinds of variables have names and values. A class describes the structure of its instances by specifying the names and default values of instance variables, and other properties. LOOPS also allows "variable length" classes, which have some