Integrating a Network-Structured Database Into an Object-Oriented Programming Language

Ira Goldstein

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

Smalltalk is an object-oriented language (Ingalls78, KayGoldberg77, Hewitt73). PIE is a subsystem that extends Smalltalk's descriptive power by supporting the creation, storage, retrieval and manipulation of network structures (GoldsteinBobrow80a,b,c; BobrowGoldstein80). These networks have been employed to represent software, documentation, electronic mail, calendars, people, addresses, bibliographic references and other items that together comprise the personal information space of a user of an office information system. By employing a common network representation, PIE supports an integrated environment for software development and office-related tasks. PIE has been developed collaboratively with Dan Bobrow, and is presently being used on an experimental basis by a small community at Xerox PARC.

Smalltalk represents entities in the external world as **objects**. An **object** has a state—i.e. an assignment of values to a set of state variables—and a **class**. The class of an object defines the behavior of the object in terms of a set of methods. Thus the class is a generic description of a collection of objects, while the objects associated with a class provide a particular description of the state of individual instances.

Smalltalk classes implement the notion of data abstraction. For example, consider the Smalltalk class Set which defines the generic behavior of sets. Objects that are instances of this class represent particular sets. A user interacts with one of these instances by sending it messages. These messages invoke methods defined in the class that manipulate the state of the instance, and then return some value. At no time need the user know the internal Smalltalk representation for a set. All that a user must know is the set of messages understood by class Set. In this fashion, the user is freed of the idiosyncracies of the chosen representation and is confronted only by the semantics of the data type as defined by its behavior.

The network database defined in PIE ameliorates various deficiencies in Smalltalk's representation mechanisms. One such deficiency is Smalltalk's limited ability to describe the internal structure of a class. The names of state variables can be specified, but no information regarding their expected type, default values or dependencies can be assigned.

PIE is predicated on the observation that Smalltalk objects can be viewed as nodes in a network with the field variables linking one object to another. From this point of view, Smalltalk objects are analogous to the *units* of a representation language (BobrowWinograd77). This suggested that we add the descriptive machinery developed in such languages to improve Smalltalk's capability to describe its own objects. PIE is the Smalltalk implementation of this machinery. Using it, a programmer can add description to a class regarding default values for variable assignments, expected types for these variables, constraints among the variables, demon procedures triggered by manipulating the variables, and commentary regarding their purpose. This machinery is internal to the class. Consequently, a client that employs an instance of a class—for example, a set—need still only know that object's message set. Data abstraction from an external viewpoint is maintained.

By improving Smalltalk's description of itself, a variety of benefits were obtained. (1) Classes were in a more informed position to check variable assignments for their reasonableness; (2) Consistency among internal variables was more easily maintained by explicit constraints than by the implicit behavior of methods; (3) Modifications by other programmers were facilitated by the increase in self-description.

Since the description is at the generic level, it has a low overhead for a Smalltalk system in terms of space. It is created for classes, not for each instance. It can be costly in cycles, if the interpreter is required to check these descriptions whenever a procedure owned by a class is executed. To accomodate this, we have various runtime modes that execute procedures with and without checking the relevant descriptions.

PIE also overcomes other deficiencies in Smalltalk's representational power. These deficiencies arise from the sacrifice of flexibility for efficiency in the present implementation of Smalltalk. A more powerful machine has recently become available that has allowed us to alter this tradeoff and expend space and time for more descriptive power. Consequently we have explored the following generalizations to Smalltalk.

* Smalltalk allows a class to inherit behavior from only one superclass. PIE supports multiple inheritance.

* Smalltalk requires that the class of an object be assigned when the object is created. PIE allows one or more classes to be assigned to an object after its creation.

* Smalltalk requires that the number of instance variables be fixed in the class. PIE allows instances of a class to have arbitrary instance variables.

Here is an example of the use of these generalizations. Initially a programmer might create a Smalltalk object as an instance of a dictionary, i.e. a list of attributes with their associated values. Subsequently he decides that this record is to be assigned a particular semantics—for example, that it represents a description of an employee. In PIE, this semantic model can be imposed by adding a second class assignment to the object so that it is treated as both a dictionary and an employee description. If, at still a later time, a new attribute is added to the semantics of employee records, then this attribute can be added to the record as well.

PIE has been employed to build a representation environment for office-related information objects. The generic descriptions for these objects built from PIE-extended Smalltalk classes constitutes a semantic data model. The data records are the instances of these classes. The size of this database is presently limited by Smalltalk's 16 bit address space. A new implementation of Smalltalk is underway that extends the virtual memory to 31 bits, thus allowing PIE networks to grow to a more significant size.

Generic descriptions of Smalltalk classes and procedures have also been written in PIE. As a result, it is possible to conduct Smalltalk programming projects by creating and manipulating node descriptions. A user can create a node that describes a procedure or a class, then compile this description into executable Smalltalk code. A corollary of this capability is that the development of PIE can be conducted within the system.

The descriptive advantages of PIE raise new technical problems that we are currently exploring.

* How are the ambiguities raised by multiple inheritance handled?

* How can we move gracefully between highly efficient but description-insensitive execution and less efficient, description-sensitive computation?

* How can we design efficient search procedures when while still preserving the privacy of the internal structure of objects and allowing individual classes to define idiosyncratic matching programs?

Bibliography

Bobrow, D.G. and Goldstein, I.P. "Representing Design Alternatives", *Proceedings of the AISB Conference*, Amsterdam, 1980.

Bobrow, D.G. and Winograd, T. "An overview of KRL, a knowledge representation language", *Cognitive Science* **1**, 1 1977.

Goldberg, A. and Robson, D. "A Metaphor for User Inteferace Design", *Proceedings of the 13th Hawaii International Conference on System Science*, Jan. 1979, pp. 148-157.

Goldstein, I.P. and Bobrow, D.G., "Extending Object Oriented Programming in Smalltalk", *Proceedings of the Lisp Conference.* Stanford University, 1980a.

Goldstein, I.P. and Bobrow, D.G., "A Layered Approach to Software Design", *Xerox PARC CSL-5-80,* 1980b.

Goldstein, I.P. and Bobrow, D.G., "Descriptions for a Programming Environment", *Proceedings of the First Annual Conference of the American Association for Artificial Intelligence, August,* 1980c.

Hewitt, C., Bishop, P., and Steiger, R., "A Universal Modular ACTOR formalism for artificial intelligence", *Proceedings of the Third International Joint Conference on Artificial Intelligence,* 1973, pp. 235-245.

Ingalls, Daniel H., "The Smalltalk-76 Programming System: Design and Implementation," *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages,* Tucson, Arizona, January 1978, pp. 9-16.

Kay, A. and Goldberg, A. "Personal Dynamic Media" *IEEE Computer,* March, 1977.