Editor: Bertrand Meyer, Interactive Software Engineering, 270 Storke Rd., Ste. 7, Goleta, CA 93117; voice (805) 685-1006; fax (805) 685-6869; ot-column@eiffel.com

Reality: A cousin twice removed

Bertrand Meyer
Interactive Software Engineering

One of the most common misconceptions about the object style of software construction is that it lets developers model the real world. This idea has even found its way into the Object Management Group's definition of *object*: "An object models a real world entity."

This assertion is too general because it ignores the many classes that represent purely internal objects, such as lists, buffers, arrays, semaphores, and binary trees, which usually account for the majority of object classes in an actual object-oriented system.

Many people who would not endorse the OMG's definition still like to think of *some* classes as representing objects from the real world. Indeed, a commonly advertised benefit of the object-oriented approach is its ability to yield software systems based in part on modules (classes) that embody concepts closely related to the outside reality. An elevator control system has *CABIN* and *BUTTON* classes; an accounting system has *EMPLOYEE* and *PAYCHECK* classes.

This immediate connection to objects in everyday life is part of the attraction of object technology, but it should not confuse us: We developers are not talking directly about the real world. (A cynic might take this to be good news: Wasn't getting away from the real world one of the reasons for becoming a programmer in the first place?)

In this column, part of which is extracted from my upcoming book (*Object-Oriented Software Construction*, Prentice Hall, 1996), I explore some of the many reasons developers should be careful whenever they are tempted to say that their objects come from the real world.

The eye of the beholder

Reality is in the eye of the beholder. Consider a program that performs mathematical computations—proving the four-color conjecture in graph theory, integrating some differential equations, or solving geometrical problems in a four-dimensional Riemann surface. Do we want to quarrel with our mathematician friends over which artifact is more real, a piece of software or a complete subspace with negative curvature?

The notion of the real world collapses anyway in the not infrequent case of software that solves software problems—reflexive applications, as they are sometimes called. For example, the "real" objects that a C++ compiler written in Eiffel processes are C++ programs. Why should we consider these programs to be more real than the compiler itself? This would be absurd. The same observation applies to any software that primarily manipulates objects that only exist in a computer: operating systems, interpreters, editors, CASE tools, even document-processing systems (which essentially manipulates digital objects; the printed text being only a particular form of output).

OBJECTS AROUND THE WORLD

Several important object technology conferences are coming up:

- ECOOP (European Conference on Object-Oriented Programming), July 8-12, Johann Kepler University, Linz, Austria; voice +43 732-2468-230; ecoop96@ifs.uni-linz.ac; http://www.ifs.uni-linz.ac.at/ecoop96/home.html.
- TOOLS USA (Technology of Object-Oriented Languages and Systems), July 29-Aug 2, Santa Barbara, California; voice (805) 685-1006; tools@tools.com; http://www.tools.com.
- Object Expo, August 5-9, New York; voice (212) 242-7578; info@sigs.com; http://www.sigs.com.
- Object World West, August 18-22, San Jose, California; voice (508) 820-4300; kwatson@ow.com.
- OOPSLA (Object-Oriented Programming, Systems, Languages and Applications), October 6-10, San Jose, California; voice (503) 578-4640; oopsla@acm.org; http://www.acm.org/sigplan/oopsla/oopsla96/oopsla96.html.

Computers in reality

While it may have been legitimate in the early days of computers to think of software as being superimposed on an existing, independent reality, today software is more and more a part of that reality. This is especially evident in the MIS domain. It may have been true 40 years ago that software simply automated existing procedures, but today many existing procedures already involve software. To describe the operations of a modern bank is to describe mechanisms that have software as a fundamental component. The same is true in most other domains. For example, many of the activities of physicists and other natural scientists rely on software, not as an auxiliary tool but as a fundamental part of the operational process.

It is tempting here to think of modern physics and its recognition that the world and the study of the world are not independent. Quantum mechanics recognizes that observation is part of the experiment; unobtrusive though the observer may try to be, the measurement always affects the measure, however minutely. In software construction, the epistemological significance of the corresponding properties is perhaps less portentous, but the practical consequences are just as important: We can seldom separate the software from the external reality. Consider the notion of virtual reality and its implication that what software produces is no less real than the outside world. In all such cases, the software is not disjoint from the reality; it is as if we had a feedback loop in which operating the software injects some new and important inputs into the model.

Methods on the Web

In the last issue, we published some Web addresses of general object-oriented pages. This month we focus on OO methods. These Web sites contain official information about some of the best-known methods for OO analysis and design.

- Unified Modeling Language (Booch-Jacobson-Rumbaugh): http://www.rational.com/ot/ot.html (at Rational Software Corporation)
- Business Object Notation (BON): http://www.eiffel.com/products/bon.html (at ISE)
- Coad method: http://www.oi.com (at Object International)
- MOSES method: http://linus.socs.uts.edu.au/~cotar/moses.html (at the University of Technology, Sydney)
- OSA (Object-Oriented Systems Analysis): http://osm7.cs.byu.edu/OSA.html (at Brigham-Young University)
- Shlaer-Mellor method: http://www.projtech.com/smmethod/smmethod.html (at Project Technology Inc.)

Objects and concepts

Even classes that do represent external concepts ("real world" concepts, if you insist) do not represent objects in the physical sense. As Kim Waldén and Jean-Marc Nerson put it in *Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems* (Prentice Hall, 1995):

> A class representing a car is no more tangible than one that models the job satisfaction of employees. What counts is how important the concepts are to the enterprise, and what you can do with them.

Keep this in mind as you look for external classes. They can be quite abstract. A class *SENIORITY_RULE* for a parliamentary voting system and a class *MARKET_TENDENCY* for a trading system may be just as real as *SENATOR* and *STOCK_EXCHANGE*.

A model of a model

Yet another—and perhaps the most fundamental—reason to avoid talking about the real world when dealing with software is that software is at best only a model of a model of some part of some reality. A patient-monitoring system is not a model of a hospital; it is the implementation of someone's view of how certain aspects of hospital management should be handled. In other words, it is a model of a model of a subset of the hospital's reality. An astronomy program is not a model of the universe; it is a software model of someone's model of some properties of some part of the universe. A financial information system is not a model of the stock exchange; it is a software transposition of a model devised by a certain company to describe those aspects of the stock exchange that it finds relevant.

The central OO notion of *abstract data types* helps explain why we are dealing with the real world. As expressed by the ADT theory, the first step to object orientation is to discard reality in favor of something less grandiose but more palatable: A set of abstractions characterized by externally applicable operations (features). Never is there any pretense that these operations are the only ones possible; we choose the ones that serve our purposes of the moment and reject the others. To model is to discard.

The reality that a software model addresses is, at best, a cousin twice removed.

Abstraction

Reasoning too much in terms of the real world can actually be detrimental to software quality. What matters is not how closely we model today's reality but how extensible and reusable our software is, so it can be adapted to a new or changed reality. Reusability and extensibility are object technology's central goals, and we should never lose sight of them. Adhering to a posited "real world" will not always help us achieve them.

Like the people in Plato's cave, we know reality only through our perception of it. When we base software on reality, what counts is the quality of our perception. Software will model only part of the reality. (Otherwise, we would not need software at all. We could let reality stand for itself, like the Academicians of Laputa in *Gulliver's Travels* who decided that "since words are only names for things, it would be more convenient for all men to carry about them such things as were necessary to express the particular business they are to discourse on.")

Modeling the real world too closely is dangerous if we choose aspects that are too low-level, circumstantial, or very subject to change. One example that grows more ominous every day is the subject of a flood of conferences, the so-called "millennium problem." How can we upgrade the myriad date-sensitive programs written by developers who did not consider dates beyond the twentieth century? Thus we now have conferences, conference series, even an entire industry devoted to *a single decimal digit*. But the people who wrote the two-digit-date programs were faithfully modeling the real world! In 1965, the real world of a 55-year-old programmer involved only two-digit dates.

Object technology is not about modeling the real world. Object technology is about producing quality software, and the way to obtain this is to devise the right abstractions, whether or not they model what someone sees as the reality.