Object Identity

A Position Paper for I-WOOOS '93

Andrew P. Black
Cambridge Research Laboratory
Digital Equipment Corporation

This position paper discusses the rôle of Object Identity in object-oriented systems. A distinction is drawn between object identity and object identifier; the former is an intrinsic part of an object-oriented system, while the latter is not. An equality test on object identifiers breaches encapsulation; such a test should therefore only be enabled at the specific request of the implementor of an abstraction.

1 Object identity is fundamental

I believe that object identity is fundamental to object-oriented systems: the existence of some concept of object identity is one of the attributes that makes a system object-oriented.

Consider the semantic equations that give meaning to the constructs of an object-oriented language. Clearly, the meaning of **self** depends on the identity of the containing object. The meaning of any identifier i also depends on the identity of the containing object. Thus, the semantic function that defines the meaning of an expression like i must take amongst its parameters some indication of the identity of the current object.

An application built on an object-oriented operating system may have its own notion of identity that differs from that provided by the underlying system. For example, clients of a replicated directory service may wish to regard all the replicas of a particular directory as identical. Because these replicas are (by design) distinct system-level objects, they have different identities as far as the underlying system is concerned. Indeed, understanding that they are distinct objects is vital to the correct implementation of the replication protocol. The fact that different notions of identity exist at different levels of abstraction is not an argument against the centrality of the concept of identity to object-orientation.

2 "Identity" should be distinguished from "Identifier"

Just because identity is essential, it does not follow that it must be possible to reify the identity of an object into a form that can be manipulated in the language itself. For example, it is possible to give a perfectly satisfactory semantics for an object-oriented language by representing each object as a separate state function: rather than having a single global state from which the right part must be extracted by indexing with some sort of object identifier, each object can instead be represented as a separate state. These states, being functions from locations to values, might not themselves be expressible in the language. Even if they are expressible, they cannot be compared for equality.

3 Should Objects have Identifiers?

The question that we should ask, then, is not whether the concept of object identity should be manifest to programmers, but in what ways should programmers be allowed to manipulate object identity. Given a reference to an object, what can be done with it, beyond invoking the object to perform one of the operations in its protocol? At one extreme is the answer "nothing at all"; the other extreme is to give object references a full set of operations, like those available on integers.

The minimal set of operations that enable a programmer to implement more complex operations seems to be

$$\begin{aligned} \mathit{hash} &: \text{object} \to \text{integer} \\ \mathit{equal} &: \text{object} \times \text{object} \to \text{Boolean} \end{aligned}$$

In theory, *hash* is superfluous: it is possible to implement *hash* using *equal* and exhaustive search. Although the inefficiency that this would introduce makes *hash* desirable in practice, for the purpose of theoretical analysis we can confine discussion to the

availability of an equality operation on object references.

4 The Case against Object Reference Equality

Providing an equality operation on object references breaches encapsulation. Consider a *file* object with an operation *openForReading* that returns a byte stream. Although the file and stream objects are distinct in the abstract, they may well be implemented by (two different interfaces of) the same concrete object. But perhaps they are implemented by distinct concrete objects, or perhaps this choice has been made differently by different implementors. Whatever choice is made, it should be the private concern of the implementor. The client of the file and the stream should not be able to ask if they are in fact the same object.

If unrestricted equality on object references is permitted, implementation secrets such as this cannot be safeguarded. In contrast, if object reference equality is not provided as a primitive, the implementor of a particular object is free to export various equality operations. Should unique identifiers be required, the implementor can even provide an *oid* operation, by the simple expedient of storing a sufficiently large unique integer in one of the fields of the object.

5 The Case for Object Reference Equality

Caching is a tremendously useful technique for improving the performance of distributed systems. Consider a system that lets a user browse through catalogues held on a large collection of remote computers. Potentially interesting catalogue cards can be tagged and set aside for further examination.

Getting the data associated with a particular catalogue card is an expensive remote operation. But the second time that the user selects a given card, the data can be displayed immediately if it has been cached on the local computer. Moreover, a cross-reference card that refers to the same object should be able to take advantage of the cached data too. The simplest way of organizing the cache is to key it on object identity: this requires the ability to test object references for equality (and, in practice, to hash them).

"Simulating" object identifiers by storing large integers as one of the fields of the data objects is not a practical alternative. Since accessing such an "oid field" would require a remote operation, and access to the cache is keyed by oid, searching the cache would take just as long as obtaining the remote data. If the performance penalty involved here were minor, we might dismiss this argument as seeking to sacrifice conceptual clarity on the alter of efficiency. But the cost of a remote operation may be a thousand times greater than the cost of a local operation. Without an efficient way to maintain a cache, many distributed applications become impossible.

6 A Compromise?

One possible compromise is to allow the implementor to decide whether to provide an oid operation, and thus whether or not to expose the identity of the underlying object, but to provide system and language support to make this operation particularly efficient. In particular, going from an object to an oid should not require a (possibly remote) invocation on the object; the oid is actually extracted from the object reference itself. However, the implementor of an object would retain the discretion not to provide an oid operation at all, if that were inappropriate. So it might be possible for a client to discover that two directory entries actually referred to the same file, but not that two streams were actually reading the same file, even though all four references might actually name the same implementation-level object.

Even with this compromise, it should be realised that requesting system support for oids is likely to impose a cost. For example, the implementation might not normally use uids in its object references, so requesting system support for oids might increase the size and complexity of the object reference, even for clients that do not use oids.

7 Summary

Object identity, as an abstract concept, is essential to object-oriented systems. The reification of object identity as some form of object identifier is sometimes very convenient, but never essential.

Object-oriented operating systems should consider the costs and benefits of providing object identifiers and comparison operations thereon. There are costs as well as benefits, and the costs may impinge on all clients, even on those that do not use object identifiers. One of the costs is the breaching of object encapsulation; for this reason, the designers of abstractions must be able to inhibit the use of the identity operation on the references to their objects.