Why Object Oriented Operating Systems are Boring

Graham Hamilton, Yousef A. Khalidi and Michael N. Nelson

Sun Microsystems Laboratories Inc.
2550 Garcia Avenue 29-112, Mountain View, CA 94043

1 Introduction.

"Object oriented" has become a very popular buzz phrase. We continually read about "object oriented window systems", "object oriented toolkits" and even "object oriented operating systems".

There are two things that are being confused. One is merely building systems using object oriented techniques. The other is building interesting systems, which may happen to use an object metaphor as a construction device.

It is clear that you can design and implement an operating system kernel as a collection of Eiffel/C++/Smalltalk classes. You can even build a distributed system using object oriented RPC. This is neither surprising nor interesting. There is some evidence that object oriented languages (or at least the object metaphor) are useful for building large programs. This is a fruitful source of employment for language designers, software engineering pundits, etc. Operating systems are large programs, so they should probably benefit from this technology. This is useful, but it has no more to do with operating systems research than writing an operating system in say Cedar or CLU or Modula-2. It is merely enabling technology rather than OS technology itself.

If the best thing one can say about an operating system is that it is "object oriented", then it is a boring operating system.

2 So what are the real goals?

So what constitutes an interesting operating system in the early 90's?

The main challenges facing OS designers in the 1990s are breaking up the monolithic single-machine operating system into a collection of distributed services. This includes such topics as:

2.1 Micro-kernels

These are depressingly fashionable. However they are a symptom that people are at least thinking about splitting up the OS, although it is all too easy to build a conventional monolithic system on top of a micro-kernel. (VMS had a well engineered "micro kernel" lurking under those enormous system services.)

2.2 Location transparency

We should be able to access a service in a uniform way regardless of whether it is in our address space, another address space in the same machine or on another machine entirely. This implies that service semantics are based on network semantics, with the local case being a lucky optimization.

2.3 Distributed security

It is simply not acceptable to ignore security. A single machine system that is a collection of cooperating services should be just as secure as a conventional monolithic OS. It must be possible to build multi-machine systems that are similarly secure.

2.4 Systems services are applications too

It should be possible for users to buy and install services that are in every way as much part of the system as the "standard" services. Shrink-wrap filesystems should be as easy as shrink-wrap spreadsheets. This implies both well designed interfaces between system
services and real support for installing new services dynamically.

2.5 Relish mistrust

In a distributed system there is no reason to trust another service simply because someone happens to be using it as their "vm" or their "filesystem". Learn to love it! Users should be able to sit down and write their own system services and start using them, without in any way compromising the security of the overall system.

2.6 Truly open systems

It is not enough to be single language or single architecture or single mind-set. We must build systems where services written in different languages with different philosophies can still play together. Just as the VAX and Fortran may seem like quaint remnants today, so may C++ and UDP/IP in ten more years. We have to be able to build systems that can cope with great diversity and with unexpected evolution.

3 Is the object metaphor useful?

Despite its unfortunate buzzing, the object metaphor is still useful in OS research. Making the interface King, separating implementations from interfaces, permitting multiple implementations to co-exist, these are all parts of the object metaphor which we can exploit in building interesting systems.

But the object metaphor is a means, not an end! Let us banish the discussions of what inheritance means or what are the rules for type equivalence to language or software engineering conferences where they belong. Let us instead talk about the kinds of systems we are building on top of this technology, rather than the technology itself. An object-oriented distributed filesystem must be interesting as a filesystem rather than as an example of the use of "object oriented technology". Being "object oriented" does not by itself solve any of the hard problems we are facing.