Summary of

A Note on Distributed Computing

The paper is mainly about the view that the objects that interact across address spaces, either a single address space, separate address spaces on the same machine, or separate address spaces on different machines, need to be handled differently from each other and that distributed object-oriented systems which are based on a model that ignores such differences is meant to fail.

The authors starts by talking about basic terminology about local computing and distributed computing, and then about the potential vulnerabilities or difficulties in a distributed system that is based on a vision of unified objects where there is no distinction between local and remote objects from the programmer's point of view. One such scenario described, whether a given object invocation is local or remote is a function of the implementation of the objects being used, and could possible change form one method invocation to another on any given object. Meaning that the vision of the system would be "objects all the way down"; that is, the system will distinguish the calls to a local object and an object "lives" on some other machine.

Afterwards the paper present various real world problems and the most critical features of distributed programming that need to be addressed and to prove the previous principle wrong. The paper claims that the hard problems in distributed computing are not the problems of how to get things on and off the wire but those that deals with latency, memory access, partial failure, concurrency and the lack of a central resource manager.

The paper says that the way an object deals with latency, memory access, partial failure, and concurrency control is an aspect of the implementation of that object, and describes this as part of the "quality of service". E.g. if reliability is important for the system, the programmer should choose to use reliable implementations for the underlying interfaces making up the system. Robustness, reliability and performance properties needs to be expressed at the interface level. Otherwise, a sloppy interface design might put an upper bound on these properties. NFS is mentioned as an example that wasn't very scalable, because of the limitations on the robustness and performance on large networks. However, the paper says that NFS was still one of the most successful distributed applications when the use was limited to a small number of geographically co-located and centrally administered machines.

There are fundamental irreconcilable differences between local and remote objects, and programmers need to accept this and take this into consideration when implementing the different system components, rather than trying to merge local and remote objects.

The paper also talks about a middle ground where there are objects that are in different address space, but on the same machine. The programming model for these "local-remote" objects can be made more similar to the programming model for local objects than distributed

objects. The objects will then be managed by a single resource manager even though they are in different address spaces, and because of this partial failure and indeterminacy can be avoided.