A Note on Distributed Computing

The main idea of the paper is to put forth the view that the objects that interact across address spaces need to be dealt differently from objects that interact within a single address space and that distributed object-oriented systems which are based on a model that ignores such differences is destined to fail.

The authors initially talk about the potential vulnerabilities in a distributed system that is based on a vision of unified objects where there is no difference between local and remote objects from the programmer's (client side) perspective. This unified view may have arisen due to the widespread misconceptions and negligence of the criticalities involved in distributing programming. In such scenarios, the assumption is that, whether a given object invocation is local or remote is a function of the implementation of the objects being used, and could possibly change from one method invocation to another on any given object. In other words, the vision of the system would be "objects all the way down"; that is, the system takes care of differentiating the calls to a local object and an object residing on some other machine and the programmers would remain oblivious of all that is happening behind the scenes.

The authors then present various real world scenarios and the most critical and inherent features of distributed programming to prove the above assumptions wrong. The authors assert that the hard problems in distributed computing are not the problems of how to get things on and off the wire but those that deal with latency, memory access, concurrency, partial failure and the lack of a central resource manager. Considering all these intricacies involved in distributed systems, the authors talk about the potential implications of creating a unified model. The first path is to treat all objects as if they were local and design all interfaces as if the objects calling them, and being called by them, were local. The other path is to design all interfaces as if they were remote. The former would not take care of the critical factors in a distributed system as discussed above while the latter model would involve unnecessary error checks and a lot of computational overhead involved in distributed systems even for the local object model.

One could also take the position that the way an object deals with latency, memory access, partial failure, and concurrency control is really an aspect of the implementation of that object which the authors call "The Myth of Quality of Service". The authors talk about the implementation of NFS, Sun's distributed computing file system to falsify such assumptions. They prove that the reliability of NFS cannot be changed without a change to the interface to reflect the distributed nature of the application.

Thus, the authors cite that the better approach is to accept that there are irreconcilable differences between local and distributed computing, and they urge all the players to be conscious of those differences at all stages of the design and implementation of distributed applications.

The authors conclude by talking about another class of objects - those that assume a middle-ground; those that belong to different address spaces but are residing on a single hardware. The main reason for treating this class of objects separately is that the entire system (e.g. compiler for an interface definition language) can be significantly optimized which could turn out to be an efficient and cost-efficient solution to the organizations following such a model.