Summary of “A note on Distributed Computing”

By Torgeir Lien and Oeyvind Rein

The authors start by criticizing the view that distributed and local objects can be thought of as the same thing. Because of the way distributed objects are handled and the way that they can fail they should not be modeled like local objects. They fear that overlooking this important detail in further work with distributed systems will make systems unreliable and useless for industry in the future.

A goal for many programmers seem to be to create a system where usage of remote objects should be entirely transparent such that code can be written in a very general fashion. That is they want to abstract away handling of distributed objects and let the underlying systems handle it. Writing such object oriented code would make it easy or trivial to move a project from a local setting to a distributed setting since objects are simply objects no matter where they are actually located and computed. As long as the interfaces that glues the different systems together are well made this should work just nice. The authors then says that there are fundamental faults in this way of thinking.

Further the authors note that the obsession with making distributed programming easier and more equal to local computing seems to resurface about every ten years when people discover that there are not that many distributed programs out there. The problem with distributed computing however is not that the communication protocols are too difficult to use but rather that the complexity of combining different architectures and models is enormous. In order to make distributed computing easier and more used by everyone is not that try and make it look like local programming but rather create good frameworks to help programmers deal with the complexity of creating a fail-safe distributed system.

The biggest difference between local and remote objects is not the delay although it may matter a lot for system performance. A big problem is that memory access across different platforms are handled differently and because of that pointers are difficult or impossible to implement transparently. Languages and handlers can be constructed to deal with this but then the programmer will have to learn a new way of programming since some utility is removed. The other way is the explain to the programmer that there are pitfalls when using address references across distributed systems but this breaks the unification between local and remote objects.

Another big problem in distributed computing is partial failure which just doesn't happen in local computing. There is no way to distinguish between a broken internet connection or a processor failure in one of the nodes. If one were to handle distributed partial failures as if they were local failures then any failure in any of the nodes would be catastrophic and this is obviously a huge drawback for the unified system. The other way is to handle all objects and calls as if they were remote but this brings a lot of overhead and a new way to program entirely which defeats the purpose of the unified model.

An example of QoS in software used to build distributed systems is given and then shot down by the argument that the systems will never be robust and fast enough to handle all the necessary tasks when communicating between remote nodes. NFS and it's pros and cons are then discussed. The NFS protocol is not very robust and uses hard-mounting which may cause entire networks of clients to hang in order to avoid corrupting data. NFS may work well for rather small networks in a close geographic area as long as there is a system admin to trace and fix errors and hang-ups when they occur. Still it is mentioned that NFS was probably the most successful distributed application at the time.

Since there are fundamental differences between local and remote objects, care must be taken to treat them as such. Ignoring the problems with distributed objects is unwise and engineers should be taught

the difference so that they can know how to handle the two types of objects correctly. A unified system of programming would be nice but may prove to be very difficult or impossible to implement robustly.

Systems to improve compatibility between local and distributed objects exists and are useful but there are limitations to this approach and it is not the same thing as unifying local and distributed objects.