Roman Kazarin
Michael Nekrasov
CS290B – Peter Cappello

A Note on Distribute Computing Summary

The paper "A Note on Distribute Computing" by Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall argues that the unified view of objects in a distributed system is incorrect, that the interaction of distributed vs. non-distributed objects is different, and that models that ignore this difference are set to fail. The paper calls this unified view of objects "the vision" – to the programmer there is no difference between objects that exists in the same address space and objects that are on remote computers. The vision follows from the main goal of RPC systems, which tries to make the invocation of remote methods look just like calling local methods. Everything in this system is "objects all the way down" where an object is defined by some interface and the object's implementation is hidden from other objects.

The vision dictates a distributed application to be pieced together in three stages. First, an application is written without paying attention to whether objects are local or remote, with the main goal being to set up the objects' interfaces. Second is object location "concretization". Interfaces are exported to their appropriate machines and communication between them is set up as well. Finally, the system is tested: machines are brought down and networks are partitioned. The system is then updated with means to respond to such failures. This system is advantageous in terms of maintenance (upgrades would be at the level of individual objects). The authors conclude their description of the main vision with three principles: there is a single OOD for an application (which doesn't depend on whether it is distributed or non-distributed), failure and performance issues should only be addressed after the initial design, and the interface of an object does not depend on the context in which the object is used. However, all three principles are actually false, and the rest of paper is spent explaining why that is.

The main issues with distributed computing revolve around partial failure and lack of a central resource whose role is to manage the system. From this follows the main differences between local and distributed computing: latency, memory access, partial failure, and concurrency. When comparing local and remote object invocation latency, the difference is between four and five orders of magnitude. This difference cannot be ignored: an application must be fine tuned with latency in mind either by relying on the speed of hardware or by first observing communication between objects and then bringing objects that are in constant contact to the same machine. The problem with memory access is that pointers in local address space are not accessible by a remote address space. One solution is to provide distributed shared memory. Another solution is forcing the developer to use "objects all the way down" so that only references to objects are passed around; if this is the solution used, the model must also enforce that the program cannot use pointers. The problem with partial failure in a distributed system is that if a component fails, there is no way to pinpoint what exactly went wrong (an exception or network failure). Since it is not possible to know the state of the system after a crash, the interfaces must be designed to be able to react to such failures, like reconstructing an appropriate state. The two solutions presented are either to treat all objects as local objects and ignore such failures, or treat objects as remote objects which would not only make the system more complex but also go against the main point of the vision. Similar issues arise with respect to concurrency: in a multi-threaded distributed system, there is no central system that aids the communication between objects therefore synchronization and failure recovery is difficult.

The authors state that reliability and robustness are components of the interface level and do not depend completely on the implementation of the interfaces. Some issues cannot be fixed by adding a "more robust implementation" but can be addressed by having a better interface.

It is in our best interest to acknowledge that local and distributed computing is fundamentally different and that these differences must be addressed during the implementation of a distributed system. In the end, it will be up to the programmer to be familiar with the best distributed system implementation models, such as knowing the difference between sending messages to local and remote objects. The paper looks at this separation positively in that it is good that the programmer be reminded which objects, local or remote, he/she is dealing with, and that this separation will be beneficial to the overall design of the system. The paper concludes by introducing a new class of objects: “local-remote”. These objects live on the same machine but in different address spaces. Access to these objects still has greater latency than accessing local objects, but because “local-remote” objects are managed by the same machine, partial failure and indeterminacy can be avoided.