HW 1, Summary of “A Note on Distributed Computing”[1]

Greg Parsons, Ethan Preble
CS290B Java-Centric Distributed Computing, UCSB, Spring 2015

In this paper, the authors argue against what must have been a prevailing pattern of thought at the time, that object-oriented design could encompass objects not just on a local machine, but those distributed across potentially many networks and machines. This erroneous “unified” paradigm, they state, results in very serious issues being ignored, namely: latency, differing memory access models, partial failure, and concurrency.

There are three general modes of computing discussed: local, distributed, and a “middle ground.” Local object-oriented programming involves invoking objects on the same machine and in the same memory space. Distributed implies the opposite. Objects can be physically located on remote machines across networks. A space in the middle is briefly also discussed as an option where objects are located on the same machine but not in the same memory space.

In this emerging unified model, the general development steps are to (1) design and implement without concern for location of objects; (2) analyze object interactions then tune performance; and (3) then test against real-world challenges and failure modes. The authors argue that the principles underlying this process are incorrect. These are: that there needs to be only one object type (both local and distributed), and that performance issues do not need to be considered in initial design.

The authors consistently argue for it to be clear to the programmer whether they are working with local or distributed objects. This is evident in their analysis of the differences between the two with regard to latency, memory access, partial failure and concurrency. Latency is obviously higher for remote objects where network links have to be transited, contrasted with only processing time for local objects. The unified model which hopes for the difference to go away with hardware improvements also has to contend with memory access. The authors argue that if you’re going to make it truly transparent to the programmer, then for example, pointers should be disallowed altogether. Thus, whereas the goal of unified objects is to abstract away remote-ness for programmer ease, actually doing so fully makes it more difficult as traditional programming methods have to change. The authors find the last two problems irreconcilable. Problems of partial failure and concurrency make it “clear that such a unification is [not] conceptually possible.”

The paper exemplifies these problems in several hypothetical examples, then go into detail with a concrete one in NFS. The thesis of the paper is that local and distributed objects should be treated differently. These examples show some reasons why. The NFS interface, originally intended for local use only, when applied in a distributed fashion caused rare errors to manifest much more often, because new distributed failure modes had to be ignored. This is an example of the unreliability that results when the distributed model is made to fit to a local computing interface. The other option is to make local objects follow distributed models, requiring changes to programming languages and making local computing unnecessarily complex.

The solutions proposed by the paper are to generally allow programmers to be aware of local and remote objects, and the consequences of using each. By keeping object types separate, compilers can also optimize appropriate, as can companies when determining where to focus resources. Finally, they suggest that a middle ground where objects are located on the same machine, but have different memory spaces, could lead to advances in modularity.

[1] Waldo, Wyant, Wollrath, and Kendall. “A Note on Distributed Computing.” Sun Microsystems Laboratories, Inc, 1994.