Brain Storm for Processing Efficiency

by
Daniel D. Beatty

One possibility: Service Oriented Programming could be used as a means to optimize the process management scheme. What would be needed?

- A Service Library
- A comparable computational model to any other Algorithm on a Turing Machine.
- A comparison to procedural models:
  - Data
  - Method
  - Control
- Or Object Oriented Models
  - Object
  - Control

However, this does not answer the Application and Object Paradox.

- How do we manage applications for zero configuration and zero install?
- How do we manage remote resources?
- How do maximize compute power?

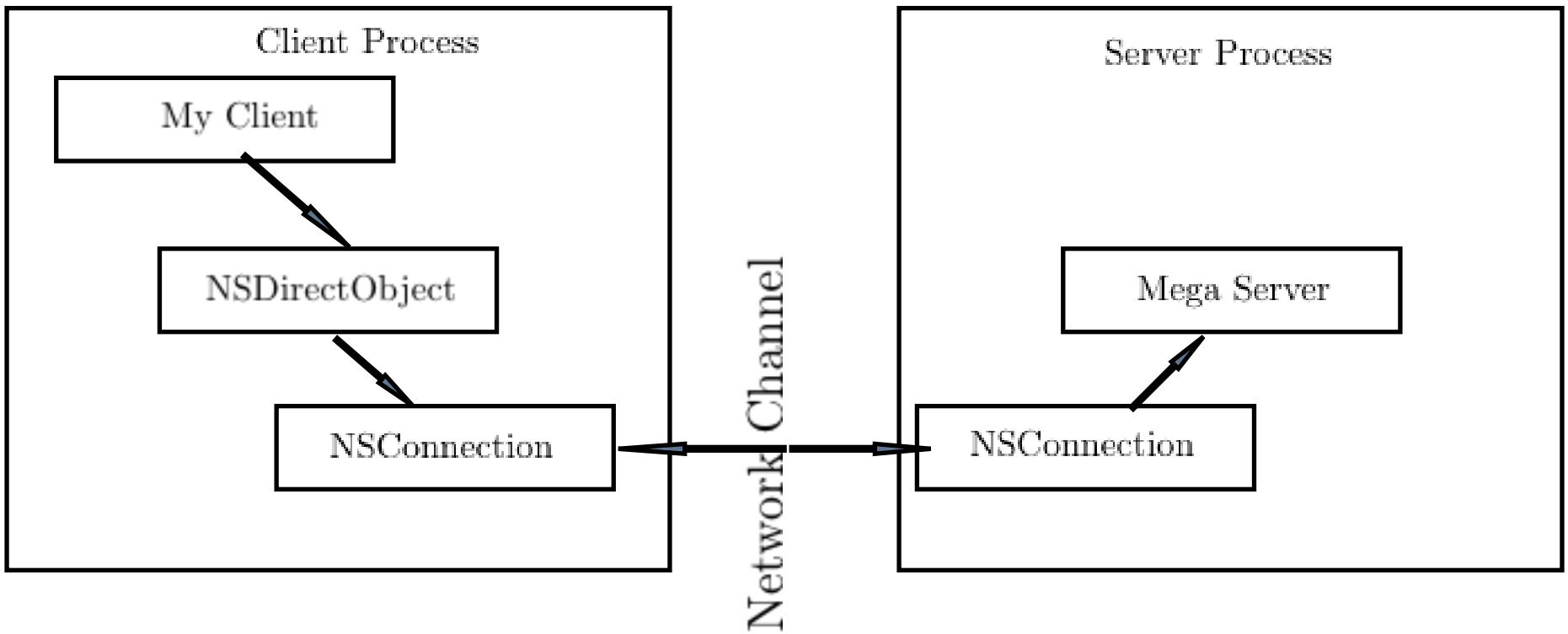
Part of the Application and Object Paradox is the nature of many batch submitted jobs such as numerical simulations. One example is in MPI programming. The basic idea is set an army of program drones that talk with each other to accomplish a task. However, the what is the application. In many respects, the MPI batch engine could be seen as the actual application since it sets the work in motion. Now consider the average application. Every bit of work is set in motion by the user operating on the application and the application calls any method necessary to fulfill the users wish.

If the desire is to maximize computational power, then a model needs to add a more techni-color spectrum to the idea of computational spectrum. Up until now, the idea computing hardware was either server or desktop. This black and white picture has been portrayed with commodity machine and super-computer and very little gray area in the middle. The consequences to maximized computation with data includes:

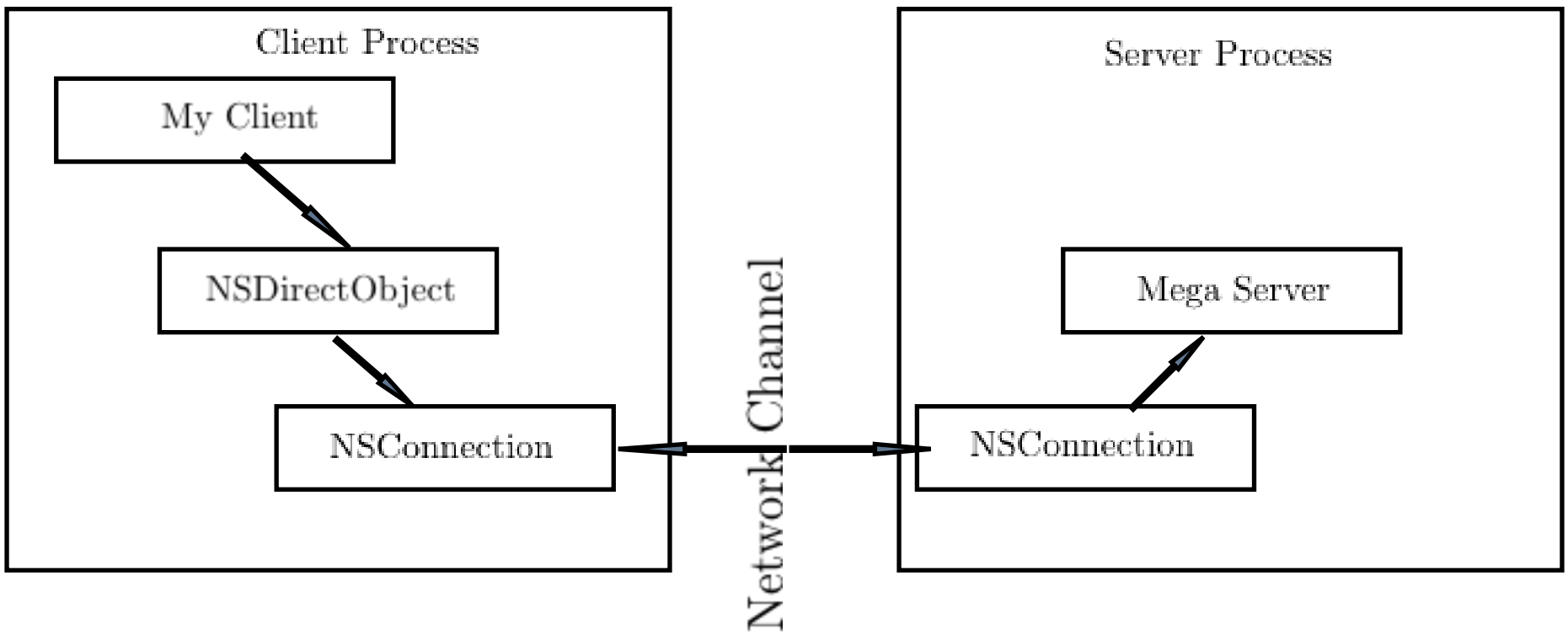
- Discovery Services
- Look Up services
- Secure access to the data
- Confidence that proper services are being called.
- Accessible common file system
- Platform issues:
  - Platform Dependent Code
  - Non-platform dependent code such as JVM, .net, and intepreted languages.

OSX is a product of NeXTStep with the Mach micro-kernel. As such it also has NSPorts. One feature that is also present is another service registration scheme called Rendezvous.

OSX uses NSPorts to provide distributed objects (DO(s)) and uses the run loop and or thread to achieve a non-blocking solution. Such DO are called via normal message passing routines associated with Objective C and NS Objects. This mechanism provides a sort of proxy for which there are two classes" NSDistributedObject and NSProxy.



An NSConnection object has two instances of NSPort: one receives data and the other sends data. An NSPort is a superclass to all other ports. NSMachPort uses Mach messaging and is typically used solely on the machine itself. NSSocketPorts use socket to go between machines.



There are addition identifier/ modifier types applied to distributed objects: functions, methods and members alike. These key words are as follows:

- oneway void (client does not wait for a response.
- in (A receiver is going to read the value but not change it.)
- out (A value is changed by the receiver by not read)
- inout (receiver is to both read and write the value).
- bycopy (argument is archived before sent and de-archived in the receiver's process space)
- byref (the argument is represented by proxy).

Each connection can have a delegate. Each time the connection spawns a new "child" connection, the "child" will have its delegate outlet set to point to its parent delegate. The connection monitor is a class for logging delegates and their connections.

Of course, there is nothing wrong with calling distributed tasks either. An example was provided by O'Reilly's articles and written by Drew McCormack May 11, 2004 [**?**]. This analysis examines the crucial parts.

Apply Filters is the method that calls Distributed Task. There are many nuggets of value in addition to the calls for:

- Add Sub Task with Identifier . This call includes
    1. The identifier
    2. Launch path
    3. Working Directory
    4. Output Directory
    5. Standard Input
    6. Standard Output
- Launch

The methods of how "Photo Industry" provides these values are somewhat interesting.

- The first section of Apply Filters acquires the time.
- Next initiates local instances of the file manager.
- The output directory is fed into Apply Filters and is not interesting.
- The temporary directory segment is interesting.
    1. It uses the processes own information (supplied by NS (OSX) which identifies the process in all of its details. The way this is used to access programs is with in the application itself.
    2. The temporary directory of functions which acquires the temporary directory as specified by the OS. (Any where NS applies).
- The next section claims to produce standard input for the filters which are actually programs and the parameters to those programs. The means for this is the typical array/ dictionary scheme of Objective-C.
- The next segment produces input and temporary directories for the input data (the photos). Features of these production(s) is the production of directories for the sub-tasks. Thus a scheme for dividing the work judiciously is being applied.

The question of the thread oriented submission becomes an issue.

Also, the feeding of data structures becomes an issue for the parent application:

- The manner the sub-tasks are divided up as a list of files (input). Items copied into these directories into these directories are the data (photos) and the programs to work on them.
- These structures include message forwarding which is the purpose of a NeXTStep delegate.
- "A delegate is an object directed to carry out an action by another object." page 456 [**?**]
- Once the sub-tasks and its data are determined, the sub job is copied out of the bundle (app), and the executable (script), then the sub-task queue is loaded.
- The rest of the methods are delegate methods.

5.3 Small Question

- Can xGrid be adapted for distributed process submission and management?
- Can xGrid provide means to identify the owner of such a job?
- How does xGrid handle processes and threads?
- Can xGrid advertise or show its process tables and subsequent to other Grid middle ware?
- Can xGrid advertise or show which processes can run and where?
- How are "safe nodes" handled?
- How are remote objects published in temporary fashions?
- How to distribute threads and parallel tasks?
- How to control jobs at a user level?
- How to deploy apps such that when a user starts them, and return to those application after leaving and coming back to it on a different terminal? Note the idea is that the application should still be working while in this nebulous space.

5.4 Big Questions

What contributions on this xGrid idea are worthy of a Ph.D. dissertation?

What synergy can be tapped to make a more complete product?

How can this product be made such that it is non-threatening by design to accomplish this computing maximizing in a non-threatening manner?