UNIVERSITY OF NEWCASTLE

Some Middleware Beginnings

Brian Randell

UNIVERSITY OF NEWCASTLE

The first known reference – NATO 1968

Alex d'Agapeyeff (CEO of Computer Analysts & Programmers, Ltd.)

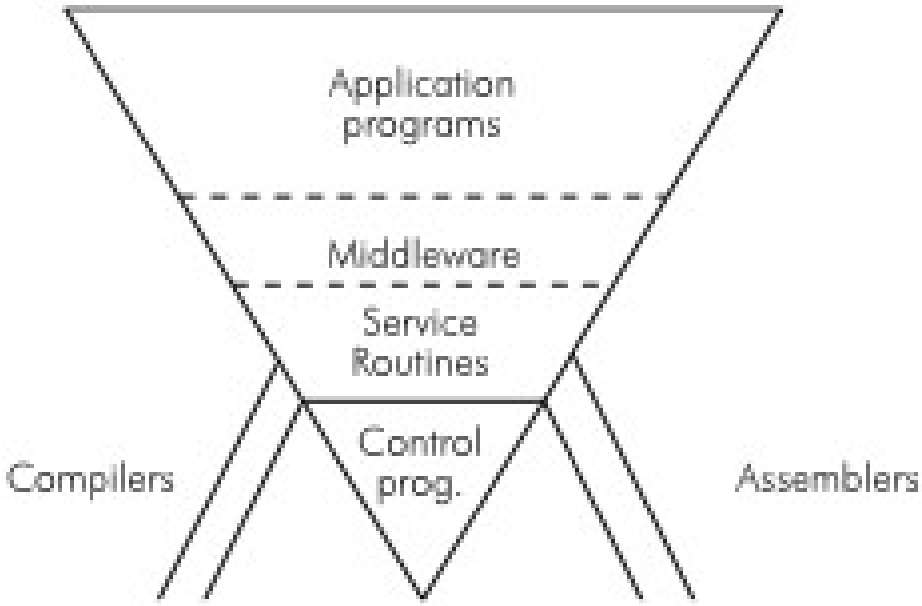


Figure 3. d'Agapeyeff's Inverted Pyramid

UNIVERSITY OF NEWCASTLE

d'Agapeyeff on Middleware

"This sensitivity of [applications to weaknesses in the] software can be understood if we liken it to what I will call the inverted pyramid. The buttresses are assemblers and compilers. They don't help to maintain the thing, but if they fail you have a skew. At the bottom are the control programs, then the various service routines. Further up we have what I called middleware. This is because no matter how good the manufacturer's software for items like file handling it is just not suitable; it's either inefficient or inappropriate. We usually have to rewrite the file handling processes, the initial message analysis and above all the real-time schedulers, because in this type of situation the application programs interact and the manufacturers' software tends to throw them off at the drop of a hat, which is somewhat embarrassing. On the top you have a whole chain of application programs."

UNIVERSITY OF NEWCASTLE

Remote Procedure Call

- Origins - Greg Nelson 1981 PhD Thesis at CMU
- Also in 1981, at Newcastle, Fabio Panzieri and Santosh Shrivastava designed a “Reliable Remote Procedure Call” protocol.
- From their Tech. Report:

  “We discuss various design issues involved, including the choice of a message passing system over which the remote call mechanism is to be constructed and the treatment of various abnormal situations such as lost messages and node crashes. We also investigate what the reliability requirements of the Remote Procedure Call mechanism should be with respect to both the application programs using it and the message passing system on which it itself is based.”

UNIVERSITY OF NEWCASTLE

Burying the Panzieri & Shrivastava RPC

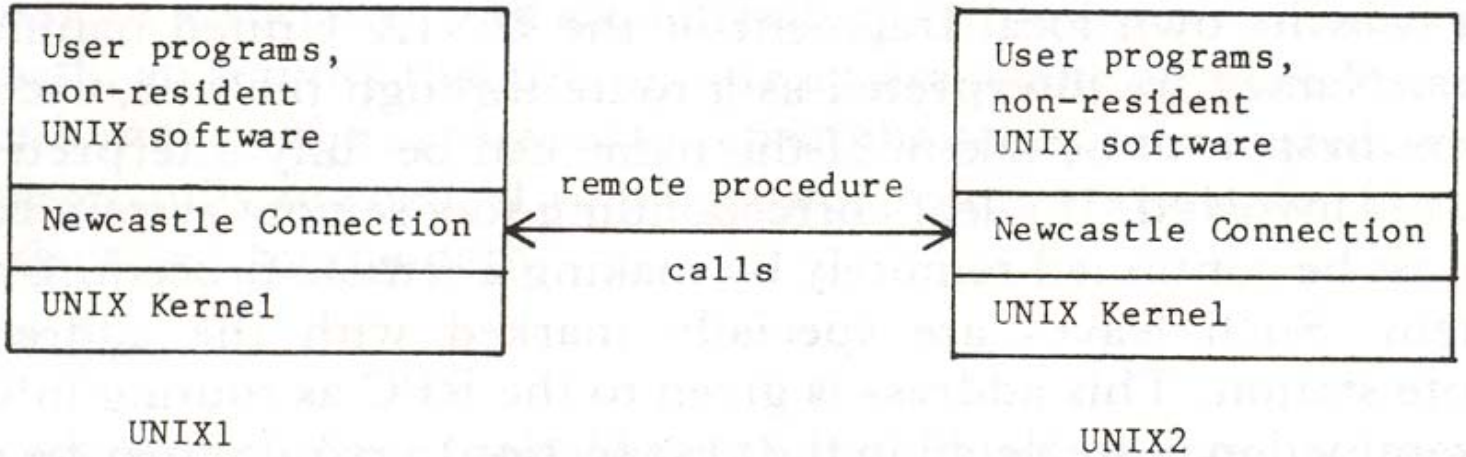
- They were enthusiastically starting to design a whole set of facilities for using and exploiting their RPC, but then:

> *"The crowd in the Daysh Building got to know of this software, and started experimenting with it. Soon the sixth floor of Daysh was buzzing with activity with group discussions going on in the corridors and offices, with Brian managing to be present simultaneously everywhere. David Brownbridge, Lindsay Marshall experimented with the crazy idea of gluing Unix systems together to provide a single Unix abstraction and John Rushby (now a senior scientist at SRI, but then an RA like me) used our RPC system to build a distributed secure system. Fabio and I felt slightly miffed, as these systems were far more interesting than our RPC!"*
>
> [Shrivastava - (Early) Distributed Computing Research at Newcastle]

UNIVERSITY OF NEWCASTLE

The Newcastle Connection

(A transparent middleware layer, inserted between the application programs and the UNIX operating system)

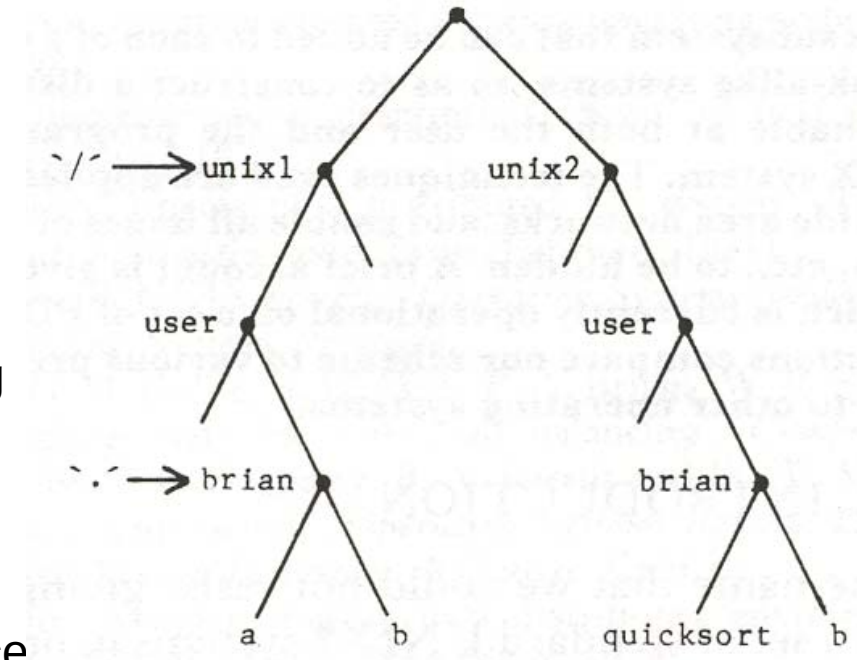


[D.R. Brownbridge, L.F. Marshall, B. Randell, The Newcastle Connection, or - UNIXes of the World Unite!, *Software Practice and Experience*, 12, 12 (1982), pp.1147-1162.]

UNIVERSITY OF NEWCASTLE

UNIX United

- The name for distributed systems built using the Newcastle Connection
- Individual UNIX systems looked like directories
- All issues concerning network protocols and inter-processor communications completely hidden
- All standard UNIX conventions, e.g. for protecting, naming and accessing files and devices, for inter-process communications, for input/output redirection, etc., apply to the distributed system as a whole
- No modification to any existing source code, of either the UNIX operating system, or any user programs



UNIVERSITY OF NEWCASTLE

Transparency & Recursion

- Transparency is great property for middleware:
  - The NC was just one of our middleware layers
  - Another, the hardware reliability layer, provided Triple Modular Redundancy (in just 600 lines of C)
  - Another provided load-balancing
  - And using “Trusted Network Interface Units” we built a Distributed Multi-Level Secure System.
- It aids composability - layers can be mixed and matched.
- It prompted a design principle:

  “*a distributed computing system should be functionally equivalent to the individual computing systems of which it is composed*”
- But this requires a suitably elegant OS (employing relative naming) as a starting point.
- After this work, in the mid-80s I left the middleware world to its own search for standards (and complexities).