Distributed Programming Abstractions

http://wiki.esi.ac.uk/Distributed_Programming_Abstractions

Shantenu Jha\textsuperscript{12}, Murray Cole\textsuperscript{3}, Daniel Katz\textsuperscript{1}, Manish Parashar\textsuperscript{4}, Omer Rana\textsuperscript{5} & Jon Weissman\textsuperscript{6}

\textsuperscript{1}Louisiana State University
\textsuperscript{2}e-Science Institute, Edinburgh
\textsuperscript{3}University of Edinburgh
\textsuperscript{4}Rutgers University
\textsuperscript{5}Cardiff University
\textsuperscript{6}University of Minnesota
Workshop Summary

- Separate the functional vs. non-functional
  - “Behavioural skeletons” (AC + AM)
  - Embarrassingly parallel apps (image processing, parameter sweep, number “crunching”)
- Data Mining-oriented programming (Domenico)
  - Data streaming – something that has emerged as being of interest recently
- Migratable objects provide an interesting abstractions – how could applications benefit from these
  - Calcium abstraction (1-1, 1-N,N-1, etc) + future objects
- Starting point → component model (then abstractions applied to components)
  - NxM components, master-server, data sharing, hierarchical distributed data, collective comms, “spatial-temporal” compositions
- Abstractions → Aspect of Transparency
  - Consensus abstractions (groups, virtual synchrony)
- Diversity of Abstractions
  - Basis for selection – which one is the most important
- Dynamic Groups – as abstractions for coordination in “joint action” – problems with optimization issues (GroupLog)
New Abstractions from workshop:

• Migratable objects + future objects ("wait by necessity" concept) – ProActive
• Adaptive components → Behavioural skeletons
• Group-based coordination abstractions
Some questions

• **How much to hide vs. expose to programmer**
  programmer only wants to see what's important to him, but how does middleware person know this, since it may vary from one programmer to another still a lot

• Abstractions can be at different levels:
  • Infrastructure, Middleware, Applications
  • Interactions between layers

• Abstractions are good for specific types of applications (“Abstraction Bloat”)
  • Identifying what “not” to include (e.g. BPEL is over-loaded with operators)
  • Can we identify abstractions that an application would need – application should be driver

• Relation of abstractions to execution language
  • What should these be
  • Should we also consider an “execution model”
Some questions ... 2

• What is a “useful” abstraction.
  • Performance penalties (QoS) issues with the use of abstraction
  • Useful vs. feasible (if it is not feasible then it is not useful – is this true)

• How does one choose the “right” abstraction in a particular context.
  • Abstractions should still be useful, even if they do not support performance improvements (as they may help better structure the application)

• Adapting abstraction to constraints imposed by an architecture
  • Relation of the two is important – especially with emerging architectures (such as multi-core and Clouds)
  • Architecture classes – rather than specific architectures (provide some mechanism to adapt the abstraction to specific architecture → transforming the abstractions)
  • “Styles of architectures” – (Field, Darlington et al. + @ Pisa)