Fine-Grained Mobility in the Emerald System

Eric Jul, Henry Levy, Norman Hutchinson, Andrew Black

SOSP, 1987 & TOCS, 1988

Short summary

- A solution in search of a problem!
- We can send objects anywhere, almost for free, but why would you want to?

Goals

- Experiment with mobility in distributed systems
  - Objects, not processes
  - Language not system support
- Single object model
- Excellent local performance
- Two Ph.D. theses!

Benefits

- Process migration
  - Load sharing
  - Communications performance
  - Availability
  - Reconfiguration
  - Specialized hardware
- + object migration
  - Data movement
  - Invocation performance
  - Garbage collection

Objects

- Name
- Representation
- Operations
- Process (optional)

Types

- Abstract (think Java interface)
- Concrete (think Java class)
- Conformity is proved by the system rather than declared by the programmer
- Objects are self-describing
  - don’t need classes

Mobility

- Locate
- Move
- Fix
- Unfix
- Refix
- Attach
- Call by move / visit

Gore

- Processes and mobility
  - Stack ripping
- Global, local and direct objects
- Object location – forwarding chains
- OIDs, pointers, templates, and swizzling
  - Registers

Performance

- Local invocation 19.4 µsec

  C: 13.4 µsec

  CE: 16.4 µsec
- Remote invocation 27.9 msec

  + 1 parameter 3.1 msec

  + 1 call by move/visit parameter ~5 msec
- Migration 12 msec

  + process: 28 msec

Performance (2007)

- Local invocation 0.67 µsec

  C: 0.02 µsec

- Remote invocation 0.37 msec

  + 1 parameter 0.01 msec

- Migration 0.37 msec

  + process: 0.11 msec

Discussion

Objects vs. classes

- What is the conceptual difference between an object-based language like Emerald and some other class based language (like Java)?

Yet another language?

- For what kind of applications would Emerald be better than any other programming language? Is it justified to come up with a new language every time a new system is designed?

Homogeneity

- Is it possible to scale the system to run on large heterogeneous networks?

How much transparency?

- Emerald makes it easy to ignore where your objects are, is this a good idea in a distributed system?
- How does location transparency affect failures?

Historical excuses?

- The authors state ‘historical reasons’ for using network communication routines that are slow. Is this a good reason? How often are things done ‘for historical reasons’?

Attachment

- How does moving objects handle the case where there are circular attachments? (e.g., X attached to Y, Y attached to Z, and Z attached to X, and the system decides to move one of them).

Compiler analyses

- The compiler chooses appropriate addressing mechanisms, storage strategies, and invocation protocols based on its knowledge of an object's use. Where does it get this knowledge?

Agents?

- Can you compare objects in the Emerald system with mobile agents?