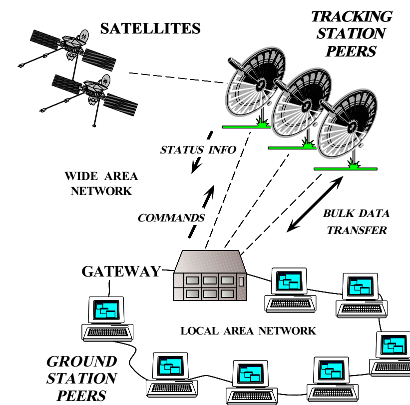
Mastering Software Complexity with Frameworks, Components, and Patterns

Douglas C. Schmidt

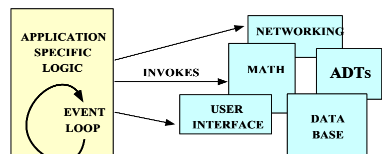
schmidt@cs.wustl.edu
Washington University, St. Louis
http://www.cs.wustl.edu/∼schmidt/

Motivation: the Distributed Software Crisis

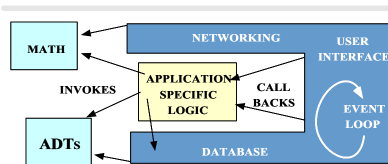


- **Symptoms**
  - Hardware gets smaller, faster, cheaper
  - Software gets larger, slower, more expensive
- **Culprits**
  - *Accidental* and *inherent* complexity
- **Solutions**
  - Frameworks, components, patterns, and architecture

Techniques for Improving Software Quality and Productivity



(A) CLASS LIBRARY ARCHITECTURE



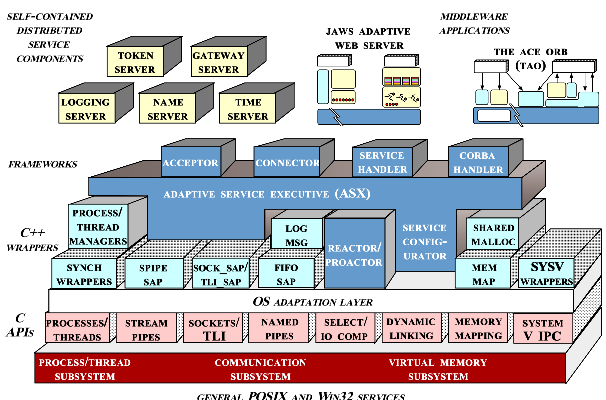
(B) APPLICATION FRAMEWORK ARCHITECTURE

- **Proven solutions**
  - *Components*
    - ∗ Self-contained, "pluggable" ADTs
  - *Frameworks*
    - ∗ Reusable, "semi-complete" applications
  - *Patterns*
    - ∗ Problem/solution pairs in a context
  - *Architecture*
    - ∗ Families of related patterns and components

Reality Sets In...

- **Components**
  - "Things that everyone wants to use, but very few are willing/able to build or can afford"
- **Patterns**
  - "An excuse to be vague"
- **Frameworks**
  - "Tangled webs of components that give up all pretense of modularity or separation of concerns"
- **Software Architecture**
  - "Those who can not develop become architects..."

The ADAPTIVE Communication Environment (ACE)



http://www.cs.wustl.edu/~schmidt/ACE.html

- **ACE Overview**
  - A concurrent OO networking framework
  - Available in C++ and Java
  - Ported to VxWorks, POSIX, and Win32
- **Related work**
  - x-Kernel
  - SysV STREAMS

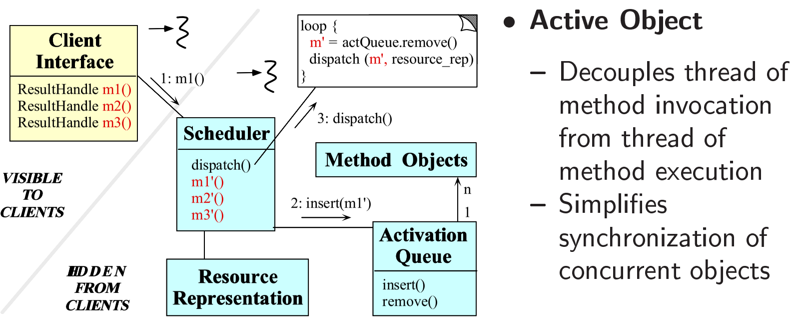
ACE Statistics

- ACE contain > 125,000 lines of C++
  - Over 10 person-years of effort
- Ported to UNIX, Win32, MVS, and embedded platforms
  - *e.g.*, VxWorks, Chorus, LynxOS, pSoS
- Large user community
  - www.cs.wustl.edu/~schmidt/ACE-users.html
- Currently used by dozens of companies
  - Bellcore, Boeing, Ericsson, Kodak, Lucent, Motorola, Nortel, SAIC, Siemens, StorTek, etc.
- Supported commercially
  - www.riverace.com

Patterns for Communication Middleware

- **Observation**
  - *Failures rarely result from unknown scientific principles, but from failing to apply proven engineering practices and patterns*
- **Benefits of Patterns**
  - Facilitate design reuse
  - Preserve crucial design information
  - Guide design choices

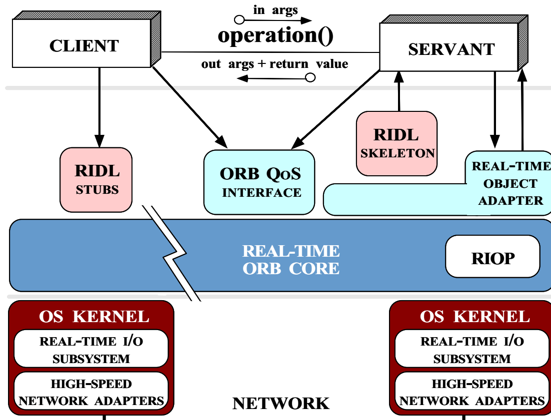
The Active Object Pattern



http://www.cs.wustl.edu/~schmidt/Active-Objects.ps.gz

- **Active Object**
  - Decouples thread of method invocation from thread of method execution
  - Simplifies synchronization of concurrent objects

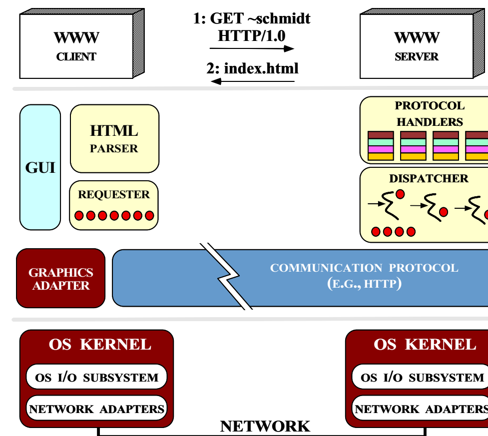
The ACE ORB (TAO)



http://www.cs.wustl.edu/~schmidt/TAO.html

- **TAO Overview**
  - A high-performance, real-time ORB
    - ∗ Networking and avionics focus
  - Leverages the ACE framework
    - ∗ Ported to VxWorks, POSIX, and Win32
- **Related work**
  - QuO at BBN
  - ARMADA at U. Mich.

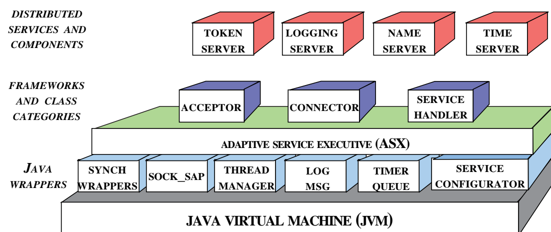
JAWS Adaptive Web Server



http://www.cs.wustl.edu/~jxh/research/

- **JAWS Overview**
  - A high-performance Web server
    - ∗ Flexible concurrency and event dispatching mechanisms
    - ∗ Full HTTP 1.0 and CGI support
  - Leverages the ACE framework
    - ∗ Ported to most OS platforms

Java ACE



http://www.cs.wustl.edu/~schmidt/JACE.html

http://www.cs.wustl.edu/~schmidt/C++2java.html

http://www.cs.wustl.edu/~pjain/MedJava.ps.gz

- **Java ACE Overview**
  - A version of ACE written in Java
    - ∗ Used for medical imaging prototype

Lessons Learned Building OO Communiation Frameworks

- Be patient
  - Good components, frameworks, and software architectures take time to develop
- Produce reusable components by generalizing from working applications
  - *i.e.*, don't build components in isolation
- Reuse-in-the-large works best when:
  - The marketplace is competitive
  - The domain is complex
  - Building middleware in-house costs too much
  - Corporate culture is supportive
- The best components come from solving real problems
  - Just like the best systems research...

The Good News

- Components are becoming mainstream
  - *e.g.*, GUIs and ADTs
- Less "Not Invented Here" syndrome
  - *e.g.*, due to increased complexity and competition
- Users are more sophisticated
  - *e.g.*, OOP/OOD, event loops, templates, applets
- More attention to performance
  - *e.g.*, STL Big-O notation
- Software architecture is gaining substance
  - *e.g.*, design patterns

The Bad News

- Lack of breadth
  - *e.g.*, focus is mostly on a few areas (GUIs)
- Lack of component integration
  - *e.g.*, incompatible event loops, name space pollution, non-robust tools
- Lack of education
  - *e.g.*, most universities don't teach software skills
- Lack of experience and training
  - *e.g.*, developers rarely apply component principles to their code
- Lack of standardized *semantics*
  - *e.g.*, design patterns

The Ugly News

- Lack of useful and truly open standards
  - *e.g.*, CORBA, ODP, ISO OSI, DCOM, TINA
  - Often leads to proprietary systems sold under guise of open systems
- Lack of adequate payoff
  - *i.e.*., cost of building components 'in-house' can be prohibitive
  - Leads to cancelled projects
- Lack of effective leadership and management
  - *e.g.*, organizations often focus on *Process* at expense of *Product*
  - Leads to the *Dilbert Principle*

Towards a Product-Oriented Process

- Develop complex systems incrementally
  - Rather than sequentially
- Emphasize qualitative reviews
  - *e.g.*, use systematic design/code inspections
- Emphasize reverse-engineering tools
  - *e.g.*, auto-generate documentation
- Invest in continuous education and training
  - Components and frameworks are only as good as the people who build and use them

Traits of Dysfunctional Software Organizations

- **Process Traits**
  - *Death through quality*
    - * "Process bureaucracy"
  - *Analysis paralysis*
    - * "Zero-lines of code seduction"
  - *Infrastructure churn*
    - * Programming to low-level APIs
- **Organizational Traits**
  - *Disrespect for quality developers*
    - * "Coders vs. developers"
  - *Top-heavy bureaucracy*
- **Sociological Traits**
  - *The "Not Invented Here" syndrome*
  - *Modern method madness*

Traits of Highly Successful Software Organizations

- Strong business and technical leadership
  - *e.g.*, understand the role of software technology
- Clear architectural vision
  - *e.g.*, know when to buy vs. build
- Commitment to/from skilled developers
  - *e.g.*, know how to motivate software developers
- Effective use of demos
  - *e.g.*, reduce risk and get user feedback

Concluding Remarks

- **Lessons Learned**
  - Not all problems require complex solutions
  - Beware simple(-minded) solutions to complex problems
  - Don't settle for proprietary *open systems*
  - Learn from past success
- **False Prophets**
  - *Languages*
  - *Methodologies*
  - *Process*
  - *Middleware*
- There is no substitute for *thinking*
  - Ultimately, *thoughtware* is our greatest resource