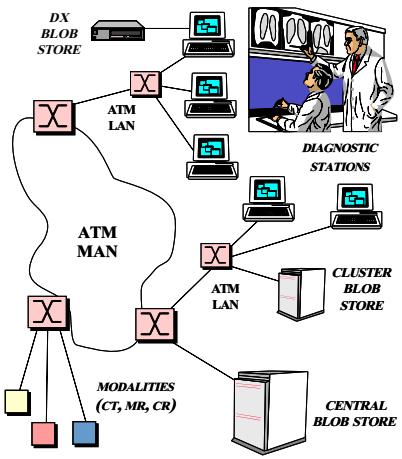
Motivation: the Distributed Software Crisis



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

Mastering Software Complexity with OO Frameworks, Components, and Design Patterns

Douglas C. Schmidt
schmidt@cs.wustl.edu

Washington University, St. Louis
http://www.cs.wustl.edu/~schmidt/TAO.html

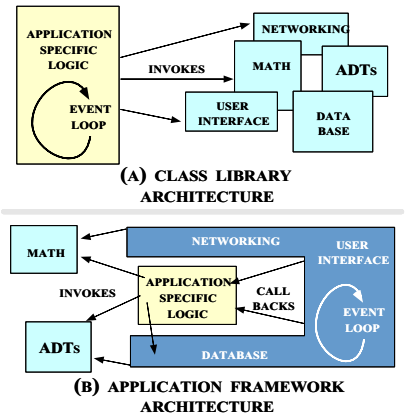
June 10, 1997

Sponsors include Siemens MED, Siemens ZT, Bellcore, McDonnell Douglas Aerospace, NSF, and US Sprint.

Why We Need Communication Middleware

- **System call-level programming is wrong abstraction for application developers**
  - *Too low-level* → error codes, endless reinvention
  - *Error-prone* → HANDLEs lack type-safety, thread cancellation woes
  - *Mechanisms do not scale* → Win32 TLS
  - *Steep learning curve* → Win32 Named Pipes
  - *Non-portable* → socket bugs
  - *Inefficient* → *i.e.*, tedious for humans
- **GUI frameworks are inadequate for communication software**
  - *Inefficient* → excessive use of virtual methods
  - *Lack of features* → minimal threading and synchronization mechanisms, no network services

Techniques for Attacking Complexity

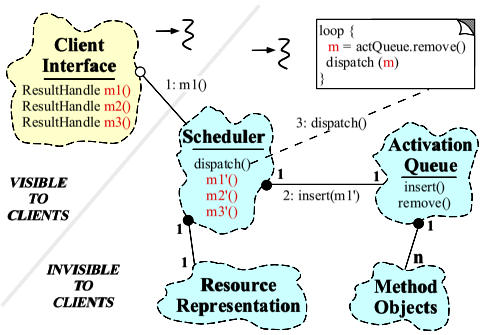


- **Proven solutions**
  - *Component*
    - Self-contained, "pluggable" ADTs
  - *Framework*
    - A reusable, "semi-complete" application
  - *Pattern*
    - Problem/solution pairs in a context

ACE Statistics

- Core ACE frameworks and components contain > 100,000 lines of C++
- > 10 person-years of effort
- Ported to UNIX, Win32, MVS, and embedded platforms (VxWorks)
- Large user community
  - http://www.cs.wustl.edu/∼schmidt/ACE-users.html
- Currently used by dozens of companies
  - Siemens, Motorola, Ericsson, Kodak, Bellcore, McDonnell Douglas, StorTek, etc.
- Supported commercially
  - http://www.riverace.com/

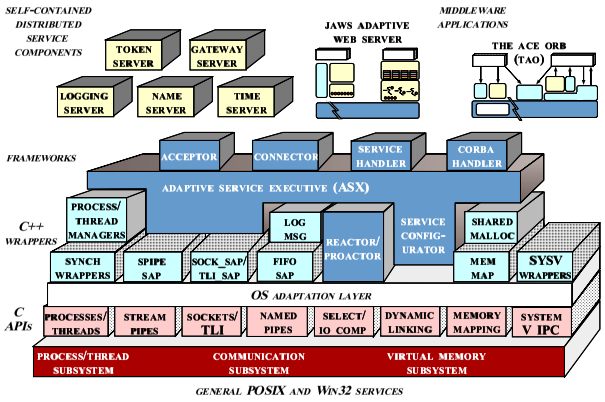
The Active Object Pattern



- **Active Object**
  - Decouples thread of method invocation from thread of method execution
  - Simplifies synchronization of concurrent objets
  - Widely used in CSA

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

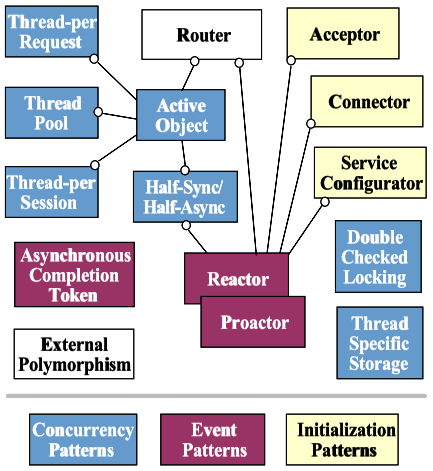
The ADAPTIVE Communication Environment (ACE)



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

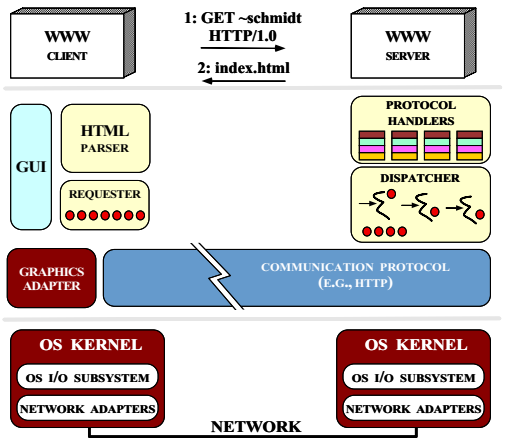
- **ACE Overview**
  - A concurrent OO networking framewor
  - Widely used in industry
  - Available in C++ an Java
  - Ported to VxWorks, POSIX, and Win32
- **Related work**
  - x-Kernel
  - System V STREAMS
  - Conduit+

Strategic Patterns for Communication Middleware



- **Benefits of Patterns**
  - Facilitate design reuse
  - Preserve crucial design information
  - Guide design choices
  - Document common traps and pitfalls
- /∼schmidt/patterns.html

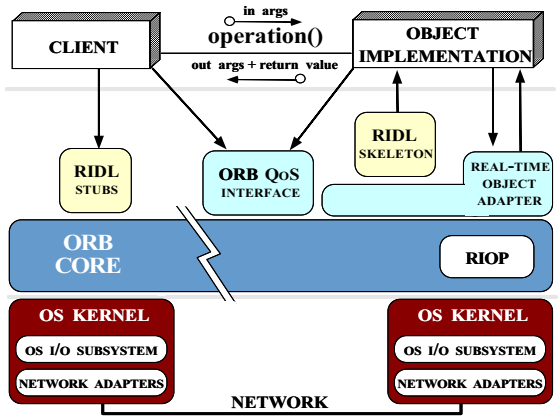
JAWS Adaptive Web Server



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

The ACE ORB (TAO)



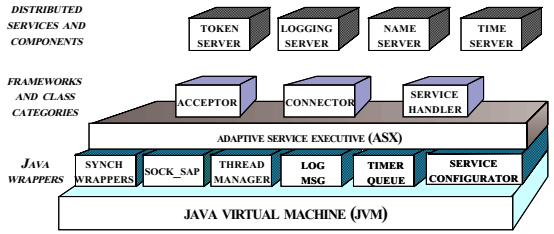
http://www.cs.wustl.edu/~schmidt/TAO.html

- **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.

Lessons Learned Building OO Frameworks

- Good components, frameworks, and software architectures take time to develop
- 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
- Produce reusable components by generalizing from working applications
  - *i.e.*, don't build components in isolation
- The best components (and systems research) come from solving real problems

Java ACE



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

Concluding Remarks

- Developers of distributed systems confront recurring challenges that are largely application-independent
  - *e.g.*, service initialization and distribution, error handling, flow control, event demultiplexing, concurrency control
- Successful developers resolve these challenges by applying appropriate *design patterns* to create communication *frameworks* and `components`
- Frameworks, components, and patterns are an effective way to achieve broad reuse of software in practice

Towards a Product-Oriented Process

- Develop complex systems iteratively rather than sequentially
- Focus on qualitative (rather than quantitative) reviews
  - *e.g.* use systematic design/code inspections
- Auto-generate documentation and use reverse-engineering tools more than forward-engineering tools
- Invest in continuous education and training
  - Components and frameworks are only as good as the people who build and use them