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

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
Strategic Patterns for Communication Middleware

Benefits of Patterns

ACE: Adaptive Communication Environment (ACE)

ACE Frameworks, Components, and Patterns

ACE Statistics
The ACE ORB (TAO)

- **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

Java ACE

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

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

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

http://www.cs.wustl.edu/~jxh/research/
Towards a Product-Oriented Process

Build and use them

- Components and frameworks are only as good as the people who use them.

Invest in continuous education and training.

Automated generation of documentation and reverse-engineering tools more than forward-engineering tools.

- Focus on qualitative (rather than quantitative) reviews.

Develop complex systems iteratively rather than sequentially.

Developers of distributed systems confront recurring challenges that are

Concluding Remarks

Successful developers resolve these challenges by applying appropriate design patterns to create communication frameworks and components.

- Service lifecycles, error handling, concurrency control, event demultiplexing, etc.

Frameworks, components, and patterns are an effective way to achieve broad reuse of software in products.

- Don’t assume that dependencies between components are a zero-cost artifact.

- Use reverse-engineering tools more than forward-engineering tools.

- Systematic design/code inspections.

- Use systematic design/code inspections, service lifecycles, and distributed, event-driven approaches to achieve.

- Successful developers resolve these challenges by applying appropriate design patterns to create communication frameworks and components.

- Service lifecycles, error handling, concurrency control, event demultiplexing, etc.

- Use reverse-engineering tools more than forward-engineering tools.

- Systematic design/code inspections.

- Use systematic design/code inspections, service lifecycles, and distributed, event-driven approaches to achieve.