Why Telecom Software Reuse Has Failed and How to Make It Work for You

Douglas C. Schmidt

Associate Professor &
Director of the Center for
Distributed Object computing

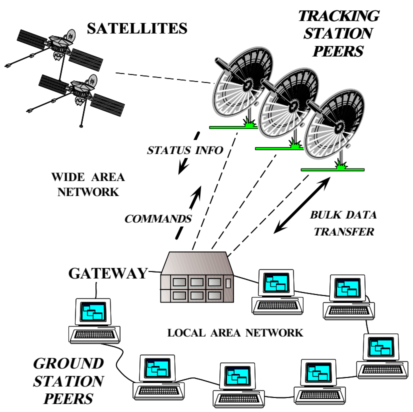
Computer Science Dept.
Washington University, St. Louis
www.cs.wustl.edu/∼schmidt/

Sponsors

NSF, DARPA, Bellcore/Telcordia, BBN, Boeing, CDI/GDIS, Hughes, Kodak, Lockheed, Lucent, Microsoft, Motorola, Nokia, Nortel, OTI, SAIC, Siemens SCR, Siemens MED, Siemens ZT, Sprint, USENIX

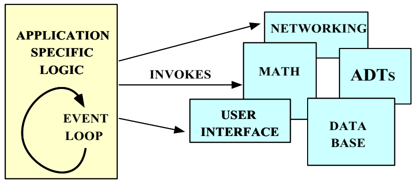
April 22^{nd}, 1999

Motivation: the Communication Software Crisis

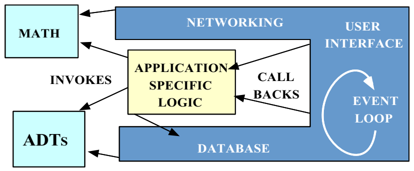


- **Symptoms**
  - ***Hardware*** gets smaller, faster, cheaper
  - ***Software*** gets larger, slower, more expensive
- **Culprits**
  - ***Inherent*** and ***accidental*** complexity
- **Solutions**
  - Frameworks, components, patterns, and architecture

Techniques for Improving Communication Software Quality, Reuse, and Productivity



(A) CLASS LIBRARY ARCHITECTURE



(B) FRAMEWORK ARCHITECTURE

Proven solutions →

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

When Reality Sets In...

Components

- "Artifacts everyone wants to use, but very few are willing/able to build or afford"

Frameworks

- "Tangled webs of components that give up all pretense of modularity or separation of concerns"

Patterns

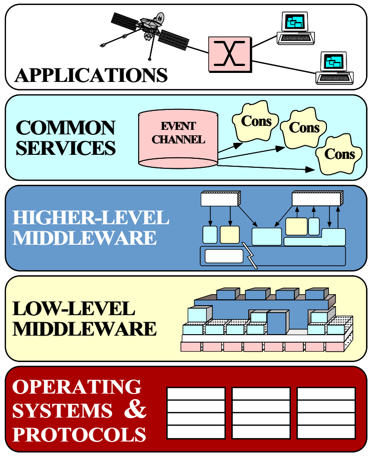
- "An excuse to be vague"

Architecture

- "Those who can no longer develop become architects... ;-)"

Bottom-line: systematic reuse is hard...

Why Systematic Software Reuse has (Largely) Failed

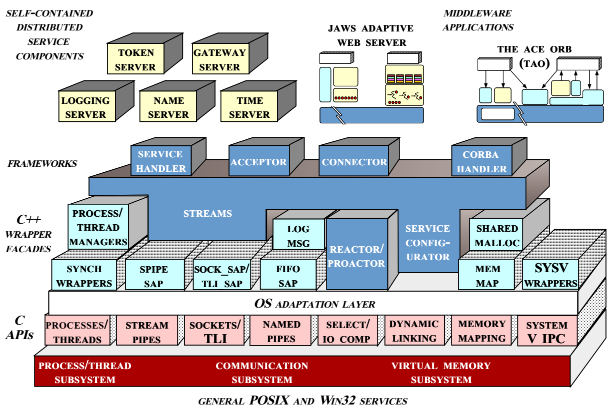


- **Improper software process**
  - Reuse techniques are often decoupled from reality...
  - Poor "expectation management"
- **Lack of organizational support**
  - *e.g.*, no economic incentives
- **Lack of technical expertise**
  - *e.g.*, limited knowledge of patterns and design principles

Why We Need Reusable Communication Middleware

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

The ADAPTIVE Communication Environment (ACE)



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

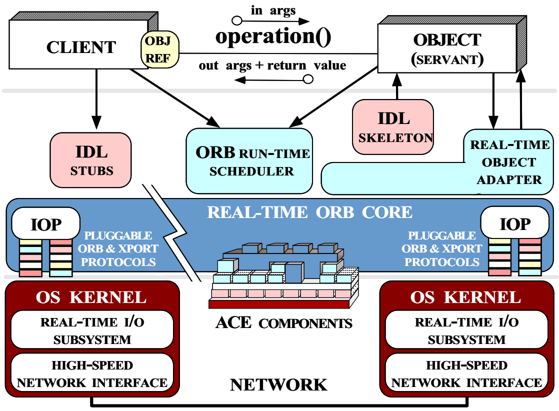
ACE Overview →

- A concurrent OO networking framework
- Available in C++ and Java
- Ported to POSIX, Win32, and RTOSs

Related work →

- x-Kernel
- SysV STREAMS

The ACE ORB (TAO)



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

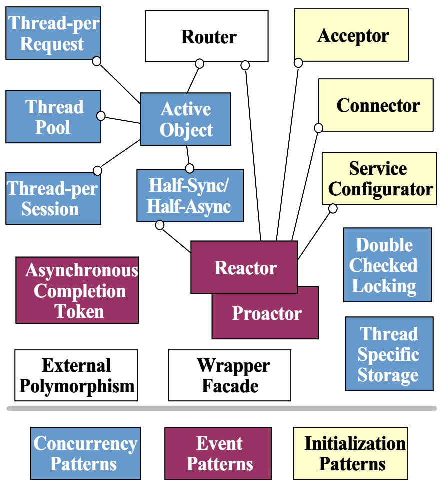
TAO Overview →

- An open-source, standards-based, real-time, high-performance CORBA ORB
- Runs on POSIX, Win32, & embedded RT platforms
  - *e.g.*, VxWorks, Chorus, LynxOS
- Leverages ACE

ACE and TAO Statistics

- Over 30 person-years of effort
  - ACE > 185,000 LOC
  - TAO > 100,000 LOC
  - TAO IDL compiler > 100,000 LOC
  - TAO CORBA Object Services > 150,000 LOC
- Ported to UNIX, Win32, MVS, and RTOS platforms
- Large user community
  - www.cs.wustl.edu/∼schmidt/ACE-users.html
- Currently used by dozens of companies
  - Bellcore, Boeing, Ericsson, Kodak, Lockheed, Lucent, Motorola, Nokia, Nortel, Raytheon, SAIC, Siemens, etc.
- Supported commercially
  - ACE → www.riverace.com
  - TAO → www.ociweb.com

Patterns for Communication Middleware



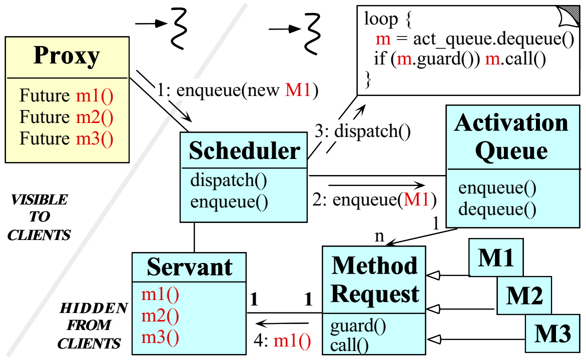
Observation →

- *Project 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



www.cs.wustl.edu/∼schmidt/patterns/Act-Obj.ps.gz

Active Object

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

How to Make Reuse Work for You

- **Be patient**
  - Good components, frameworks, and software architectures take time to develop
- **Reuse-in-the-large works best when:**
  1. The marketplace is competitive
  2. The domain is complex
  3. Skilled middleware developers
  4. Supportive corporate culture
  5. “Reuse magnets” exist
  6. Open source development models
- **The best components come from solving real problems**
  - Keep feedback loops tight to avoid “runaway” reuse efforts
- **Produce reusable components by generalizing from working applications**
  - *i.e.*, don’t build components in isolation

The Good News

- **Frameworks and components are becoming mainstream**
  - *e.g.*, GUIs and ADTs
- **Less "Not Invented Here" syndrome**
  - *e.g.*, due to increased complexity and competition
- **Developers are more sophisticated**
  - *e.g.*, OOP/OOD, event loops, templates, applets
- **More attention to performance**
  - *e.g.*, good ORBs are *very* efficient, predictable, & scalable
- **Software architecture is gaining substance**
  - *e.g.*, patterns and architectural styles

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, poor tools
- **Lack of education**
  - *e.g.*, most universities don't teach software skills
- **Lack of experience and training**
  - *e.g.*, developers rarely apply reuse principles/patterns to their code
- **Lack of standardized semantics & performance**
  - *e.g.*, design patterns & optimization principle patterns

The Ugly News

- **Lack of useful and truly open standards**
  - *e.g.*, ODP, ISO OSI, CORBA, DCOM, TINA, Java
  - 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 Software Process

- **Develop complex systems incrementally**
  - *i.e.*, not sequentially
- **Emphasize qualitative reviews**
  - *e.g.*, use systematic design/code inspections
- **Reward software development skills**
  - Both *generalization* and *customization* skills
- **Use 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*
  - *e.g.*, 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 leadership in business and technology**
  - *e.g.*, understand the role of software technology
  - Don't wait for "silver bullets"
- **Clear architectural vision**
  - *e.g.*, know when to buy vs. build
  - Avoid worship of specific tools and technologies
- **Effective use of prototypes and demos**
  - *e.g.*, reduce risk and get user feedback
- **Commitment to/from skilled developers**
  - *e.g.*, know how to motivate software developers and recognize the value of *thoughtware*

Concluding Remarks

Take-home Points

- Not all problems require complex solutions
- Beware simple(-minded) solutions to complex problems
- Don't settle for proprietary *open systems*
- Systematic reuse is achievable, though non-trivial

False Prophets

- *Languages*
- *Methodologies*
- *Process*
- *Middleware*
- *Organization-central solutions*
- *Technology-centric solutions*

There is **no** substitute for **thinking** and **hard work**!

Web URLs for Additional Information

- **These slides:** `www.cs.wustl.edu/~schmidt/keynote4.ps.gz`
- **More information on patterns:** `www.cs.wustl.edu/~schmidt/patterns.html`
- **More information on CORBA:** `www.cs.wustl.edu/~schmidt/corba.html` `www.omg.org`
- **More info on ACE:** `www.cs.wustl.edu/~schmidt/ACE.html` `comp.soft-sys.ace`
- **More info on TAO:** `www.cs.wustl.edu/~schmidt/TAO.html`