Experience Using Design Patterns to Evolve Communication Software Across Diverse Platforms

Douglas C. Schmidt

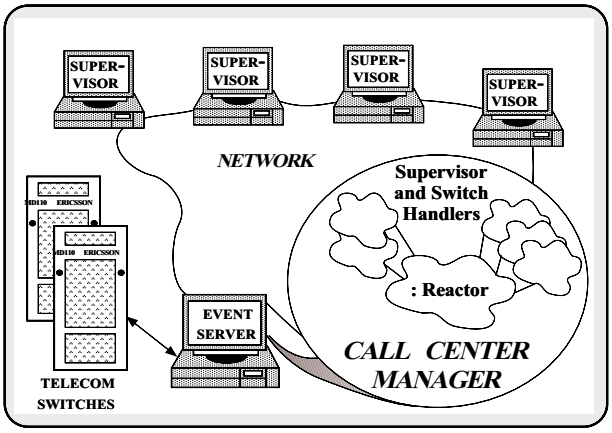
Washington University

schmidt@cs.wustl.edu

Introduction

- Developing portable, reuseable, and efficient communication software is hard
- OS platforms are often fundamentally incompatible
  - *e.g.*, different concurrency and I/O models
- Thus, it may be impractical to directly reuse:
  - *Algorithms*
  - *Detailed designs*
  - *Interfaces*
  - *Implementations*

Case Study



- OO framework for call center management developed for Ericsson

Problem: Cross-platform Reuse

- Original OO framework was developed for UNIX and later ported to Windows NT
- UNIX and Windows NT have fundamentally different I/O models
  - *i.e.*, synchronous vs. asynchronous
- Thus, direct reuse of original framework was infeasible...

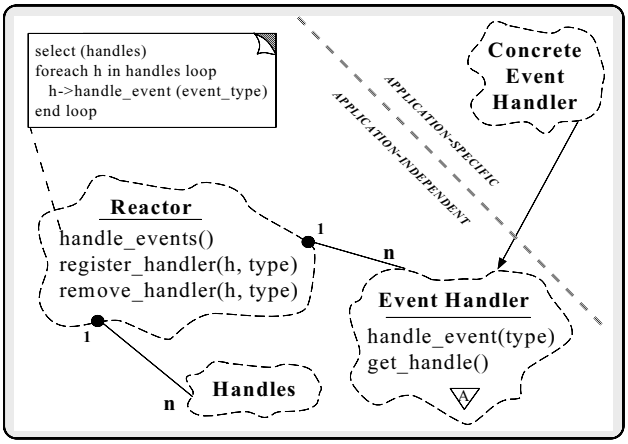
Solution: Reuse Design Patterns

- Design patterns support reuse of *software architecture*
- Patterns embody successful *solutions* to *problems* that arise when developing software in a particular *context*
  - They are particularly useful for articulating how and why to resolve *non-functional forces*
- Design patterns greatly reduced project risk at Ericsson by leveraging proven design expertise

Example Pattern: Reactor

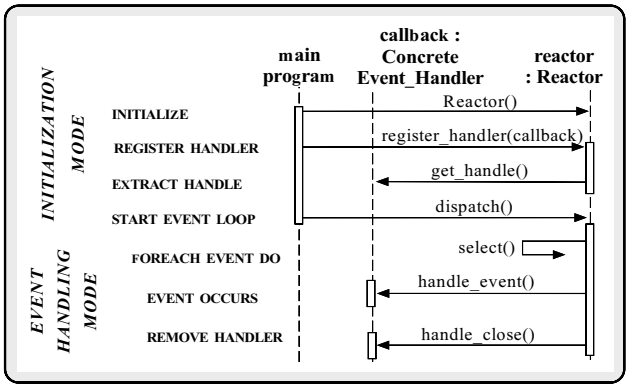
- *Intent*
  - Decouple event demultiplexing and event handler dispatching from the services performed in response to events
- This pattern solves a key problem for single-threaded communication software:
  - *How to efficiently demultiplex multiple types of events from multiple sources of events within a single thread of control*
- A pattern description captures the static and dynamic *structure* and *collaboration* among key *participants* in a micro-architecture

Structure of the Reactor Pattern



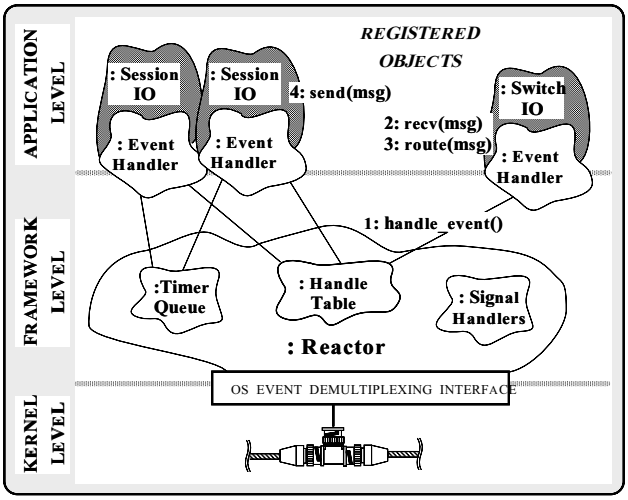
- Participants in the Reactor pattern

Collaborations in the Reactor Pattern



- Dynamic interaction among participants in the Reactor pattern

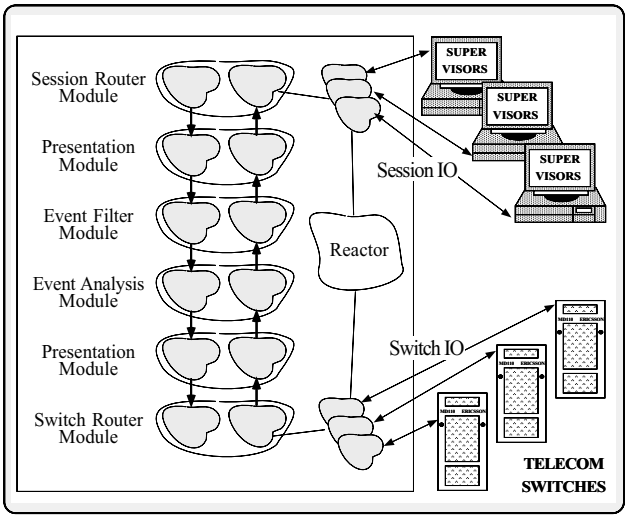
Using the Reactor Pattern for CCM



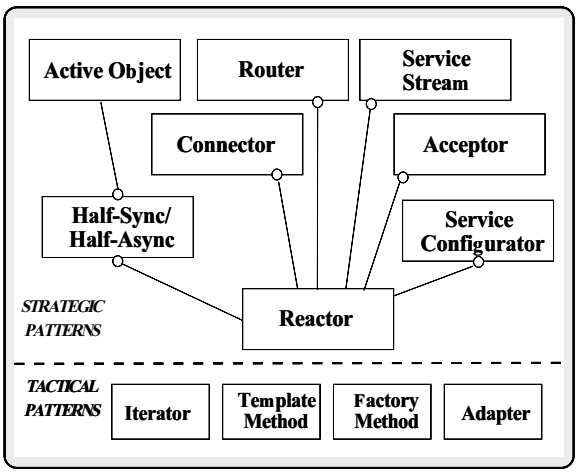
Implementing the Reactor on UNIX and Windows NT

- Major difference is UNIX *reactive* I/O vs. Windows NT *proactive* I/O
  - Reactive I/O is synchronous
  - Proactive I/O can be asynchronous
    - Requires additional interfaces to "arm" the I/O mechanism
- Other differences include
  - *Resource limitations*
    - *e.g.*, Windows NT limits number of HANDLEs per-thread
  - *Demultiplexing fairness*
    - *e.g.*, `WaitForMultipleEvents` always returns the lowest active HANDLE

CCM Software Architecture



Patterns used in CCM



- The CCM components are based upon a family of design pattern

Strategic and Tactical Patterns

- *Strategic* design patterns have an extensive impact on the software architecture
  - Typically oriented to solutions in a particular domain
  - *e.g.*, Reactor pattern in the domain of event-driven, connection-oriented communication software
- *Tactical* design patterns have a relatively localized impact on a software architecture
  - Typically domain-independent
  - *e.g.*, Wrapper, Adapter, Bridge, Factory Method, and Strategy
- It is important to understand both types of patterns

Summary of Case Study

- Real-world constraints of OS platforms can preclude direct reuse of communication software
  - *e.g.*, must often use non-portable features for performance
- Reuse of design patterns may be the only viable means to leverage previous development expertise
- Complex telecommunication software systems contain hundreds of reusable design patterns

Benefits of Design Patterns

- *Design patterns enable large-scale reuse of software architectures*
- *Patterns explicitly capture expert knowledge and design tradeoffs*
- *Patterns help improve developer communication*
- *Patterns help ease the transition to object-oriented technology*

Drawbacks to Design Patterns

- *Patterns do not lead to direct code reuse*
- *Patterns are deceptively simple*
- *Teams may suffer from pattern overload*
- *Patterns are validated by experience rather than by testing*
- *Integrating patterns into a software development process is a human-intensive activity*

Suggestions for Using Patterns Effectively

- *Do not recast everything as a pattern*
  - Instead, develop strategic domain patterns and reuse existing tactical patterns
- *Institutionalize rewards for developing patterns*
- *Directly involve pattern authors with application developers and domain experts*
- *Clearly document when patterns apply and do not apply*
- *Manage expectations carefully*

The Future of Patterns

- *Integration of patterns together with frameworks*
  - Achieve reuse of both design *and* code
- *Integration of patterns to form systems of patterns*
  - Focus more on strategic, rather than tactical, patterns
- *Integration with popular object-oriented methods and software process models*
  - More focus on patterns throughout the software lifecycle

Books and Magazines on Patterns

- *Books*
  - Gamma et al., "Design Patterns: Elements of Reusable Object-Oriented Software" Addison-Wesley, Reading, MA, 1994.
  - "Pattern Languages of Program Design," editors James O. Coplien and Douglas C. Schmidt, Addison-Wesley, Reading, MA, 1995
- *Special Issues in Journals*
  - "Theory and Practice of Object Systems" (guest editor: Stephen P. Berczuk)
  - "Communications of the ACM" (guest editors: Douglas C. Schmidt, Ralph Johnson, and Mohamed Fayad)
- *Magazines*
  - C++ Report and Journal of Object-Oriented Programming, columns by Coplien, Vlissides, and De Souza

Conferences and Workshops on Patterns

- Joint *Pattern Languages of Programs* Conferences
  - 3rd PLoP
    - ▷ September 4–6, 1996, Monticello, Illinois, USA
  - 1st EuroPLoP
    - ▷ July 10–14, 1996, Kloster Irsee, Germany
  - http://www.cs.wustl.edu/~schmidt/jointPLoP-96.html/
- USENIX COOTS, June 17–21, Toronto, Canada
  - http://www.cs.wustl.edu/~schmidt/COOTS-96.html
- IEEE GLOBECOM '96, November 18–22, London, England
  - http://crg.eee.kcl.ac.uk/comchap/gc96cfp.html

Obtaining ACE

- The ADAPTIVE Communication Environment (ACE) is an OO toolkit that implements many communication software patterns
- All source code for ACE is freely available
  - Anonymously ftp to `ics.uci.edu` (128.195.1.1)
  - Transfer the files `gnu/C++_wrappers.tar.gz` and `gnu/ACE-documentation/*.gz`
- Mailing list
  - ace-users@ics.uci.edu
  - ace-users-request@ics.uci.edu
- WWW URL
  - http://www.cs.wustl.edu/~schmidt/