Koala

A case study in component-oriented, product line software architecture

"The Koala Component Model," van Ommering et al, IEEE Computer 2000

"Building Product Populations," van Ommering, ICSE 2002

Why read these papers?

- General issues: software reuse, software architectural design, components
- Particular approaches: architecture design language, automation of integration and parameterization
- A peek at the research side of software engineering

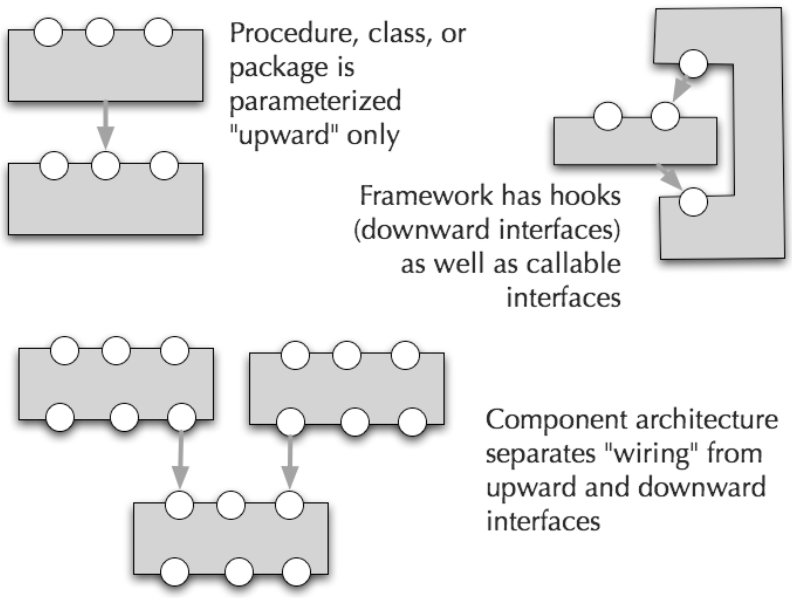
CE Problem & Opportunity

- Challenge:
  - Rapid development (market pressure)
  - Changing hardware environment
  - Cost constraint
- Opportunity:
  - Commonality across products
  - Reusable components
    - not quite the same thing: application domain vs. technical infrastructure

Exploiting Commonality

- General theme: Find what is the same, factor it out, reuse it
  - Includes design and methods as well as code (e.g., design patterns)
- Reusable pieces (code) require parameterization
  - A fixed part with variable parameters
  - Key question: What can we parameterize?

Objects, frameworks, and component architectures



Framework vs Component

- Framework: common structure, parameterize by “filling in”
  - Easy to fill in; hard to evolve structure
- Components: encapsulated, documented interfaces
  - “downward” as well as “upward” interfaces; note similarity to framework “hooks” or “slots”
  - Less given than frameworks, but easier to evolve

Provides & Requires

- Provides: upward interface
  - You’re used to this: class & method signatures, etc. ; extend naturally to larger constructs
- Requires: downward interface
  - Contrast to explicit naming in Java, C++, etc
  - You can simulate (clumsily) with interfaces and abstract classes

Binding Time

- In programming languages:
  - Run time (dynamic), compile time (static)
  - e.g., “static type” does not change during execution, “dynamic dispatch” is call through object reference
- In software engineering
  - Same concept, extend to various points in design
  - Same trade-offs: flexibility vs. performance, dependability

Binding in Koala

- Separate “wiring” from components
  - Explicit “downward” interfaces
  - Defer binding of calls
- Configuration-time binding
  - More flexible than component design time
    - as in conventional programming language constructs
  - Cheaper than run-time binding
    - as in COM, Corba, et al