OO Design: Component Architecture

7 Mar 2008
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

What are components?

- Pre-fabricated, reusable building blocks for software systems
- Allows for rapid development and consistent reuse
- They do not tie together libraries (chunks of code)
  - But do coordinate running code (dynamic cooperation of live objects with active state)
- Bigger than single objects, may be combinable
- Relate "peer-to-peer" rather than in hierarchy
- May operate across a suite of applications
  - To get consistent interfaces into those applications

Component concepts

- Applications use a palette of components
  - The programmer/user composes or “wires” them together to make a complete application
- Requires:
  - Mechanisms for defining new components
  - Standard ways to specify component interfaces
  - Each component has “hooks” (methods) by which other components interact with it
- Compare to hardware components:
  - Transistors, integrated chips, etc.

Components vs. code

- Hardware components are black boxes with spec sheets: wires connect them together
- Software components are represented as black boxes with interfaces: you write code to connect them up
- Software companies may sell components as binaries (black boxes) with API documentation
  - Need not sell the actual source code
  - e.g., NVIDIA binary graphics drivers for Linux
  - e.g., Scantron ClassClimate and myTWU portal

Vocabulary

- Components are assembled into containers
  - The finished assembly constitutes the application
- May also take document-centric view:
  - e.g., a container document may hold text, images, videos, buttons, etc.
  - Editing any item passes control to the appropriate component: text editor, image editor, etc.
  - The document *is* the application!
  - Peer-to-peer: no one component is “boss”

Component-based development (CDB)

- Delivering solutions by building or buying interoperable components
- Don't reinvent the wheel: write once, deploy many times at various levels (server, desktop, handheld, ...)
- Requires rigid adherence to software infrastructure:
  - Standard way for components to work together
- Fits naturally with distributed, multi-language, multi-platform heterogeneous environments:
  - Don't care what language it's written in
  - Don't care where it runs

Examples of component applications

- Tying together departments within a company (enterprise resource software):
  - Accounting, invoicing, human resources
- Leveraging rich, complex data types / data stores:
  - Data mining, pattern recognition, image analysis, genomics, StatsCan, ...
- Adding multimedia to a field salesperson's laptop/handheld
  - Interface to same back-end applications as at the head office

Layering

- Sometimes component architecture is deployed as "middleware":
  - A set of components that allow a variety of database stores or applications to be manipulated by a common interface
- Examples:
  - Application plug-in interface: Firefox, Photoshop
  - LAPACK/BLAS: standard library for linear algebra
  - ActiveX/COM: interoperation of MS applications
    - e.g., graphics, outlining, cut-and-paste, MS-SQL
  - JavaBeans: components for Java

Example: JavaBeans

- JavaBeans: component architecture for Java
  - JavaStudio: environment to build Beans
  - Components list which features (public methods, events, etc.) can be manipulated by the builder
  - Introspection: a “JavaBeans-enabled” builder tool examines Beans to see what features are exposed
  - What events can a Bean fire (send) or handle (receive)
  - Drag-and-drop application development
  - Persistence: Beans can save/restore state

Example: ODBC

- Open DataBase Connectivity
  - Standard API to many database systems: MS-SQL, Oracle, DB/2, mySQL, PostgreSQL, ...
  - Simplifies use of standard SQL commands
    - Can also access vendor-specific commands
  - Cross-platform, cross-language
    - Although Java also has its own: JDBC
  - Sybase ACA (Architecture for Competitive Advantage): similar, using Transact-SQL