Indefinitely Evolvable Architectures: Event-Based Systems

Ted Faison
Faison Computing Inc.
ted.faison@computer.org
EBSs have Superior Non-Functional Characteristics

- **Manageability** - Independent teams
- **Maintainability** – Independent changes
- **Deployability** – Independent updates
- **Testability** – Independent parts
- **Verifiability** – Independent implementations
- **Flexibility** – Independent designs
EBSs are cheaper and faster to build

• **EBSs are cheaper**
  
  Lots of small and independent parts are cheaper to build than fewer large and dependent parts

• **EBSs are easier to build**
  
  Development teams can work largely in parallel, due to the independence between parts. Final integration is much easier.

• **EBSs can evolve indefinitely**
  
  The parts are small and independent, so changes in the system requirements tend to have much smaller impacts on the individual parts. Changes often require only changing the system wiring and adding new parts.
Important EBS Definitions

• Event
   A detectable occurrence

• Notification
  Messages triggered by events

• The Observer design pattern
What is an EBS?

• It’s all about the system connectivity

*The constituent parts interact primarily or solely via notifications*

Part 1

Notifications

Part 2

Part 3

Ted Faison - GSAW 2005
What is an EBS?

• It’s all about coupling (and how to avoid it)
  – Static Coupling
    Occurs at compile-time
    Greatly affects development teams
  – Dynamic Coupling
    Occurs at run-time
    Has little affect on development teams
Improving the Observer pattern

• Self-subscribing Observers are coupled to Subjects

• Binders decouple the parts
Firing Events
(aka sending notifications)

• Sending Messages

• Using Procedure Calls
  – Typed calls
    *Introduce type coupling, which is static*
    Example: \texttt{myTypedReference.DoSomething()}
  – Untyped calls
    *Introduce signature coupling, which is dynamic*
    Example: \texttt{myMethodReference.Execute()}
Interaction dynamics: active and reactive patterns

- Active interactions
- Reactive interactions
Complexity Versus Size

- Heavily coupled systems: complexity grows exponentially with size
- Decoupled systems: complexity grows linearly with size
Case Study 1: A Distributed Workflow System

ASAP Cars - Order Processing System

Client

Send Request

Distributed Servers

Handle Order

Assemble Car

Generate Invoice

Invoice Received
System Coupling Diagram

ASAP Cars - Order Processing System

- Order Processor
- Vehicle Assembly
- Client
- Invoicing
- CommonTypes
Case Study 2: A System Browser

User Interface - File Browser
File Searcher

User Interface - File Searcher
User Interface Structure

![Diagram of User Interface Structure]

- **Panel Structure**:
  - **panelMain**
  - **panelToolBar**
  - **panelNavigator**
  - **panelContent**
  - **panelStatusBar**

- **System Browser**

Ted Faison - GSAW 2005
• There are no relationships between the main classes, meaning there is no static coupling between them

• Objects interact using event notifications
The Binder

- The Binder is coupled to all the classes in the system
Signal Wiring Diagrams

- Use Case: User selects a folder in the Folders navigator
The Wiring Diagram as a blueprint of connectivity

Ted Faison - GSAW 2005
Advantages of EBSs

• Most parts of a system are statically decoupled from the others

• Decoupled parts are easier to design, because they don’t call other parts

• Decoupled parts are easier to develop and maintain, because they can be tested in isolation from the rest of the system

• Decoupled systems are easier to extend and evolve, since the main parts are not aware of the others