Introduction to Software Engineering

10. Software Architecture
Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles:
  - Layered
  - Client-Server
  - Blackboard, Dataflow, ...
- Model-Driven Architecture
- UML diagrams for architectures
Sources

> *Objects, Components and Frameworks with UML*, D. D'Souza, A. Wills, Addison-Wesley, 1999
Roadmap

> **What is Software Architecture?**
> **Coupling and Cohesion**
> **Architectural styles:**
  — Layered
  — Client-Server
  — Blackboard, Dataflow, ...
> **Model-Driven Architecture**
> **UML diagrams for architectures**
What is Software Architecture?

A neat-looking drawing of some boxes, circles, and lines, laid out nicely in PowerPoint or Word, does not constitute an architecture.

— D’Souza & Wills
What is Software Architecture?

The architecture of a system consists of:

> the *structure(s) of its parts*
  — including design-time, test-time, and run-time hardware and software parts

> the *externally visible properties* of those parts
  — modules with interfaces, hardware units, objects

> the *relationships and constraints* between them

**in other words:**

The set of *design decisions* about any system (or subsystem) that keeps its implementors and maintainers from exercising “needless creativity”.
How Architecture Drives Implementation

> Use a 3-tier client-server architecture: all business logic must be in the middle tier, presentation and dialogue on the client, and data services on the server; that way you can scale the application server processing independently of persistent store.

> Use Corba for all distribution, using Corba event channels for notification and the Corba relationship service; do not use the Corba messaging service as it is not yet mature.
How Architecture Drives Implementation ...

> Use Collection Galore’s *collections* for representing any collections; by default use their List class, or document your reason otherwise.

> Use *Model-View-Controller* with an explicit *ApplicationModel* object to connect any UI to the business logic and objects.
A **sub-system** is a system in its own right whose operation is *independent* of the services provided by other sub-systems.

A **module** is a system component that *provides services* to other components but would not normally be considered as a separate system.

A **component** is an *independently deliverable unit* of software that encapsulates its design and implementation and offers interfaces to the out-side, by which it may be composed with other components to form a larger whole.
Roadmap

> What is Software Architecture?
> **Coupling and Cohesion**
> Architectural styles:
>   — Layered
>   — Client-Server
>   — Blackboard, Dataflow, ...
> **Model-Driven Architecture**
> **UML diagrams for architectures**
Cohesion

Cohesion is a measure of *how well the parts of a component “belong together”*.  

> Cohesion is **weak** if elements are bundled simply because they perform similar or related functions (e.g., `java.lang.Math`).

> Cohesion is **strong** if all parts are needed for the functioning of other parts (e.g. `java.lang.String`).

—Strong cohesion *promotes maintainability* and adaptability by *limiting the scope of changes* to small numbers of components.

There are many definitions and interpretations of cohesion.  

**Most attempts to formally define it are inadequate!**
Coupling is a measure of the *strength of the interconnections* between system components.

> Coupling is **tight** between components if they depend heavily on one another, (e.g., there is a lot of communication between them).

> Coupling is **loose** if there are few dependencies between components.

—Loose coupling *promotes maintainability* and adaptability since *changes in one component are less likely to affect others.*
Tight Coupling

Subsystem A

Shared data area

Subsystem B

Subsystem C

Subsystem D
Loose Coupling

Subsystem A

A’s data

Subsystem B

B’s data

Subsystem A

D’s data

Subsystem A

C’s data
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> **Architectural styles:**
  — Layered
  — Client-Server
  — Blackboard, Dataflow, ...
> Model-Driven Architecture
> UML diagrams for architectures
Architectural Parallels

> Architects are the *technical interface* between the customer and the contractor building the system

> A bad architectural design for a building *cannot be rescued by good construction* — the same is true for software

> There are *specialized types* of building and software architects

> There are *schools or styles* of building and software architecture
An architectural style defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style defines a vocabulary of components and connector types, and a set of constraints on how they can be combined.

— Shaw and Garlan
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> Architectural styles:
  — Layered
  — Client-Server
  — Blackboard, Dataflow, ...
> Model-Driven Architecture
> UML diagrams for architectures
Layered Architectures

A **layered architecture** organises a system into a set of layers each of which provide a set of services to the layer “above”.

> Normally layers are *constrained* so elements only see
  — other elements in the same layer, or
  — elements of the layer below

> **Callbacks** may be used to communicate to higher layers

> Supports the *incremental development* of sub-systems in different layers.
  — When a layer interface changes, *only the adjacent layer is affected*
OSI reference model

Application
Presentation
Session
Transport
Network
Data link
Physical

Communications medium

Application
Presentation
Session
Transport
Network
Data link
Physical
Roadmap

- What is Software Architecture?
- Coupling and Cohesion
- Architectural styles:
  - Layered
  - **Client-Server**
  - Blackboard, Dataflow, ...
- Model-Driven Architecture
- UML diagrams for architectures
A client-server architecture distributes application logic and services respectively to a number of client and server subsystems, each potentially running on a different machine and communicating through the network (e.g., by RPC).
Client-Server Architectures

**Advantages**

> *Distribution* of data is straightforward

> Makes *effective use of networked systems*. May require cheaper hardware

> Easy to *add new servers* or upgrade existing servers

**Disadvantages**

> *No shared data model* so sub-systems use different data organisation. Data interchange may be inefficient

> *Redundant management* in each server

> May require a *central registry* of names and services — it may be hard to find out what servers and services are available
Film and picture library

Client 1

Client 2

Client 3

Client 4

Wide area network

Video server

Music server

Photo server

Web server
Four-Tier Architectures
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> Architectural styles:
  — Layered
  — Client-Server
  — Blackboard, Dataflow, ...
> Model-Driven Architecture
> UML diagrams for architectures
A blackboard architecture distributes application logic to a number of independent sub-systems, but manages all data in a single, shared repository (or “blackboard”).
Advantages
> *Efficient way to share* large amounts of data
> Sub-systems need not be concerned with how data is produced, backed up etc.
> Sharing model is published as the *repository schema*

Disadvantages
> Sub-systems must agree on a repository data model
> *Data evolution* is difficult and expensive
> No scope for specific management policies
> Difficult to distribute efficiently
CASE toolset architecture

- Design editor
- Code generator
- Project repository
- Design translator
- Design analyzer
- Report generator
- Program editor
In an **event-driven architecture** components perform services in *reaction to external events* generated by other components.

> In **broadcast models** an event is broadcast to all sub-systems. Any sub-system which can handle the event may do so.

> In **interrupt-driven models** real-time interrupts are detected by an interrupt handler and passed to some other component for processing.
Broadcast model

> Effective in *integrating sub-systems* on different computers in a network

> Can be implemented using a *publisher-subscriber* pattern:
  — Sub-systems register an interest in specific events
  — When these occur, control is transferred to the subscribed sub-systems

> *Control policy is not embedded* in the event and message handler. Sub-systems decide on events of interest to them

> However, sub-systems don’t know if or when an event will be handled
Selective Broadcasting

Subsystem 1

Subsystem 2

Subsystem 3

Subsystem 4

Event and message handler
Dataflow Models

In a dataflow architecture each component performs functional transformations on its inputs to produce outputs.

> Highly effective for reducing latency in parallel or distributed systems
  — No call/reply overhead
  — But, fast processes must wait for slower ones

> Not really suitable for interactive systems
  — Dataflows should be free of cycles
## Pipes and Filters

<table>
<thead>
<tr>
<th>Domain</th>
<th>Data source</th>
<th>Filter</th>
<th>Data sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unix</td>
<td>tar cf - .</td>
<td>gzip -9</td>
<td>rsh picasso dd</td>
</tr>
<tr>
<td>CGI</td>
<td>HTML Form</td>
<td>CGI Script</td>
<td>generated HTML page</td>
</tr>
</tbody>
</table>
Invoice Processing System

- Invoices
  - Read issued invoices
  - Identify payments
  - Issue receipts
  - Receipts

- Payments
  - Find payments due
  - Issue payment reminder
  - Reminders
Compilers as Dataflow Architectures

- Lexical analysis
- Syntactic analysis
- Semantic analysis
- Code generation

Symbol table
Compilers as Blackboard Architectures

- Lexical analysis
- Syntactic analysis
- Semantic analysis

- Pretty printer
- Abstract syntax tree
- Grammar definition
- Symbol table
- Output definition
- Repository

- Editor
- Optimizer
- Code generation

© Ian Sommerville 2000
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> Architectural styles:
  — Layered
  — Client-Server
  — Blackboard, Dataflow, ...
> Model-Driven Architecture
> UML diagrams for architectures
The Vision of MDA

software developer
The Vision of MDA
The Vision of MDA

Platform Independent Model

C. Atkinson, U Mannheim
The Vision of MDA

software
developer

Platform
Independent
Model

automatic
translation

C. Atkinson, U Mannheim
MDA in a nutshell

- One unique Metametamodel (the MOF)
- An important library of compatible Metamodels, each defining a DSL
- Each of the models is defined in the language of its unique metamodel

J. Bézivin, ATLAS group, U Nantes
The OMG/MDA Stack

- **M₀**: "the real world"
- **M₁**: model
- **M₂**: metamodel
- **M₃**: meta-metamodel

The MOF

The UML metamodel ++

Some UML Models ++

Various usages of these models

Slide 42

Thursday, September 15, 11
Multi-target code generation

Write Once, Run Anywhere
Model Once, Generate Anywhere

PIM

CORBA
Java/EJB
C#/DotNet
Web/XML/SOAP

Platform-Independent Model

data grid computing
pervasive computing
cluster computing

SMIL/Flash

SVG, GML, Delphi, ASP, MySQL, PHP, etc.
Roadmap

> What is Software Architecture?
> Coupling and Cohesion
> Architectural styles:
  — Layered
  — Client-Server
  — Blackboard, Dataflow, ...
> Model-Driven Architecture
> **UML diagrams for architectures**
UML support: Package Diagram

Decompose system into **packages** (containing any other UML element, incl. packages)
UML support: Deployment Diagram

*Physical layout* of run-time components on hardware nodes.

*Figure 3-8. Deployment diagram (descriptor level)*

*Figure 3-9. Deployment diagram (instance level)*
What you should know!

> How does software architecture constrain a system?
> How does choosing an architecture simplify design?
> What are coupling and cohesion?
> What is an architectural style?
> Why shouldn’t elements in a software layer “see” the layer above?
> What kinds of applications are suited to event-driven architectures?
Can you answer the following questions?

> What is meant by a “fat client” or a “thin client” in a 4-tier architecture?

> What kind of architectural styles are supported by the Java AWT? by RMI?

> How do callbacks reduce coupling between software layers?

> How would you implement a dataflow architecture in Java?

> Is it easier to understand a dataflow architecture or an event-driven one?

> What are the coupling and cohesion characteristics of each architectural style?
Attribution-ShareAlike 3.0

You are free:
- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor.

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

http://creativecommons.org/licenses/by-sa/3.0/