SE310
Analysis and Design of Software Systems

Lecture 4, Part-2 – Architectural Design
OOA/OOD vs. SA/SD

Last Year’s Consensus - 50% OO Helps, 50% Undecided

Discussion of OOA & OOD CASE Tools - Last Year’s Comments

1. Aid for Smart and Efficient Designs, well suited for a larger project
2. Helps to Clarify and Universalize
3. Abstracts concepts – “what” a system is – clear specification and design models
4. Helps to organize
5. Aids SQA – allows you to visualize software, develop a structured plan, confirm logic prior to implementation
**OOA/OOD vs. SA/SD**

Negative Regarding OOA & OOD CASE Tools - Last Year’s Comments

1. Cumbersome at times – Modelio feels heavy weight for class assignments
2. Sometimes so abstract … making it also very difficult … hard to translate into an implementation
3. As a software engineer you should be able to fill in the blanks
4. Mainly useful in the modeling phase
Architectural Work

- Goals, Objectives, Concepts
- Requirements Analysis - Define, Review, Check Completeness, Consistency, Accuracy, etc.
- Block Diagrams
- Present Architecture for Review
- Test Driven Design - Early Prototyping and Testing

E.g. ICARUS Drone Net (System) - Drone Net Architecture
- Transformation - Machine Vision features
- Transactional - Image files and aerial object catalog (database)
- Event Driven - Embedded ADS-B, IMU, detection, coordination between nodes, calibration
- Interactive - Interface to ATC for notifications
Architecture and Design Patterns

Focus on What is Being Designed and Built

OO Has Goal of Design and Software Re-Use
- Encapsulation of Data and Operations
- Class Hierarchy and Object Instances
- Well Understand Use Cases
- Well Understand Interaction Between Objects

Study 4 Key System Types

1. Interactive – E.g. GUI, CLI
2. Event Driven – E.g. Anit-lock Breaking System Software
4. Transaction Oriented – E.g. DBMS, Client-Server
Four Common Types of Systems

(a) Interactive subsystem

(b) Event-driven subsystem

(c) Transformational subsystem

(d) Database subsystem
[Transactional - RDBMS Client, SQL Server]
Architectural Design

Transition from Requirements, Supports Requirements Analysis, Supports Design Activity

Focus on Domain Analysis, Use Cases and Higher Levels of System Rather than Details

UML
- Use Cases – Supported by Activity Diagrams
- Basic Class Hierarchy (Draft of Class Definitions with Encapsulation of Attributes + Methods, Class to Sub-class Inheritance and Polymorphism – Parametric and Late-Binding / Over-rides or refinement)
- Class Diagrams – Supported Interaction Diagrams

SA/SD – Dataflow, ER/EER Diagrams, High Level State Machines (Flowcharts are Typically Too Detailed)
Traditional SA/SD – Useful, But Not OO

- Data Flow Diagrams – Data [Messages] Between Processes and is Transformed
- Entity Relationship Diagrams – Lacks Operations, but Defines Entities [Objects] and Relationships
- State Machines [in Common, but Typically for Each Process in DFD]
- Flow-Charts – Detailed Procedural Design [Interaction, Logic]

Stores, Flows, Processes, External Entities

http://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model
http://en.wikipedia.org/wiki/Flowchart
Domain Models – Use Case Details

- UML is Universal Modeling Language [OMG, UML.org]
- Use to Support Requirements Analysis

Start Here! https://www.modelio.org/

USE Modelio 3.7 SD as your DESIGN TOOL

OMG UML 2.5 Standard

Structural Diagrams
- Start with Class Diagram and CRC
- Then Object Diagram
- Package and Deployment

Behavioral Diagrams
- Start with Use Case Diagram
- Interaction Sequence Diagram after Class and Object Done
- Add State Machine and Activity Diagrams for concurrency and statefulness

Helpful Validation and Verification Features for Design
- Integrated Models
- Checklists – Completeness
- CPP and Java Code Generation
Tool-Based Activities

- Bring Up Modelio and Start Entering ATM Design – Use Case and Class Diagram, Compare to UML Reference

- Look at More Reference Designs in UML - http://mercury.pr.erau.edu/~siewerts/se310/design/

- Consider How UML Models Support Analysis and Design of Software Systems
  - Limitations?
  - Concurrent Software, Hardware, and Test Process?
  - Walk-throughs and Inspections with Stakeholders
Why Do We Need Model?


No, you are all wrong. Elephant is like a fan.

Elephant is like a wall.

No, I think elephant is a cylinder.

No, elephant is like a rope.

We perceive the world differently due to differences in backgrounds and viewpoints. Modeling facilitate collective understand of the application.
## System Types and Architectural Styles - Updated

<table>
<thead>
<tr>
<th>Type of System</th>
<th>Architectural Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Interactive System</td>
<td>N-Tier</td>
</tr>
<tr>
<td>2) Event-Driven System</td>
<td>Event-Driven</td>
</tr>
<tr>
<td>3) Transformational System</td>
<td>Main Program and Subroutines</td>
</tr>
<tr>
<td></td>
<td>Persistence Framework</td>
</tr>
<tr>
<td>Object-Persistence Subsystem (CS317)</td>
<td></td>
</tr>
<tr>
<td>4) Client-server [Transactional]</td>
<td>Client-server (e.g. TCP session)</td>
</tr>
<tr>
<td>Distributed, decentralized</td>
<td>Peer-to-peer (e.g. UDP messaging)</td>
</tr>
<tr>
<td>Heuristic problem-solving</td>
<td>Blackboard</td>
</tr>
<tr>
<td>Distributed, asynch., key-value</td>
<td><strong>REST</strong>, micro-services, web services</td>
</tr>
</tbody>
</table>
Characteristics of Interactive Systems

- The interaction between system and actor consists of a relatively fixed sequence of actor requests and system responses.
- The system has to process and respond to each request.
- Often, the system interacts with only one actor during the process of a use case.
- The actor is often a human being although it can also be a device or another subsystem.
- The interaction begins and ends with the actor.
- The actor and the system exhibit a “client-server” relationship.
- System state reflects the progress of the business process represented by the use case.
N-Tier Architecture

Graphical User Interface Layer → Controller Layer → Business Objects Layer → Database Layer → Network Communication Layer

requests → requests → requests → requests → requests
Characteristics of Event-Driven Systems

• It receives events from, and controls external entities.
• It does not have a fixed sequence of incoming requests; requests arrive at the system randomly.
• It does not need to respond to every incoming event. Its response is state dependent—the same event may result in different responses depending on system state.
• It interacts with more than one external entity at the same time.
• External entities are often hardware devices or software components rather than human beings.
• Its state may not reflect the progress of a computation.
• It may need to meet timing constraints, temporal constraints, and timed temporal constraints.
Event-Driven Architecture

Component 1

Component 2

State Based Controller

Component 3

Component 4
Characteristics of Transformational Systems

• Transformational systems consist of a network of information-processing activities, transforming activity input to activity output.

• Activities may involve control flows that exhibit sequencing, conditional branching, parallel threads, synchronous and asynchronous behavior.

• During the transformation of the input into the output, there is little or no interaction between system and actor—it is a batch process.

• Transformational systems are usually stateless.

• Transformational systems may perform number crunching or computation intensive algorithms.

• The actors can be human beings, devices, or other systems.
Data Transformation - Storage-as-a-Service

**Examples-RAID-Unit-Test, stripetest.c**

```
[siewerts@localhost Examples-RAID-Unit-Test]$ ./stripetest Baby-Musk-Ox.ppm Baby-Musk-Ox.ppm.replicated
read full stripe
...
hit end of file
FINISHED

[siewerts@localhost Examples-RAID-Unit-Test]$
[siewerts@localhost Examples-RAID-Unit-Test]$
```

```
[siewerts@localhost Examples-RAID-Unit-Test]$ diff Baby-Musk-Ox.ppm Baby-Musk-Ox.ppm.replicated
```
Data Transformation – MPEG Streaming

VLC Server/Player – remote MPEG Transport Streaming

Encode elementary Video, Audio, PSI

Decode, Present, provide Guide

Big Buck Bunny
MPEG4 Stream File

Big Buck Bunny
Movie

VLC Stream Service

MPEG Program (MPTS)

Sockets API

UDP

IP

IEEE 802.3z - GigE

Cat-6 UTP

L7

L6

L5

L4

L3

L2

L1

VLC Player

MPEG Program (MPTS)

Sockets API

UDP

IP

802.11abgn - Wireless

Cat-6 UTP

IP Router/Switch

Internet

© Sam Siewert
Characteristics of Transaction Systems

- Numerous Asynchronous Requests for Information and Updates to Information (Bank balance and Transactions)

- Consistency and Determinism of Transaction is Key
  - E.g. DBMS 2-Phase Commit or Time-Stamp Ordering
  - ACID – Atomicity, Consistency, Isolation, Durability
  - Deposit, Check Processing, Balance Inquiry – Ok by Bank
  - Check Processing, NSF Fee, Deposit, Balance Inquiry – Ok by Bank
  - Lost Updates and Dirty Reads or Inconsistent Analysis Not Ok by Bank

- ATM Reference Design (Security Protocols)

- A More Specific Version of Client-Server with ACID Requirements

- Web Services (micro-services, stateless) – Simpler Client Server, Most Often Without Transaction Requirements (E.g. Web pages need refresh – inconsistent analysis not a concern (maybe it should be) – lost update, handled by transaction system + client server hybrid architecture)
Client-Server Architecture

<<client>>
Passenger Check-In Client 1

<<client>>
Passenger Check-In Client 2

<<client>>
Passenger Check-In Client 3

<<remote procedure call>>

<<remote procedure call>>

<<remote procedure call>>

<<server>>
Flight Information System

UML notation for a software component

UML stereotype for a user-defined modeling construct.
Characteristics of Object-Persistence Systems

- It provides object storage and retrieval capabilities to other subsystems.
- It hides the implementation from the rest of the system.
- It is responsible only for storing and retrieving objects, and does little or no business processing except performance considerations.
- It is capable of efficient storage, retrieval, and updating of a huge amount of structured and complex data.

Studied in CS317 as Alternative to RDBMS

- Integrates Behavior with Data Structural (Relational) Modeling
- Allows for OOA/OOD to Extend EER
- Integrates Query with Application Data Processing and Client Interfaces (Interactive, Transformational, Event Driven – Hybrid with Transaction)
Modelio UML Use Case

- For ATM Reference Design – [Here](#)
- Add Activity and State Diagrams for More Detail

© Sam Siewert 23
Interaction Diagram for Use Case Scenario – Supports Class Diagram
SafeHome Example (Pressman)

- UML Example Design Provided in Textbook
- Done with ArgoUML (Only use Argo to Review Pressman Design)
- [http://mercury.pr.erau.edu/~siewerts/se300/design/ArgoUML/](http://mercury.pr.erau.edu/~siewerts/se300/design/ArgoUML/)
MySQL Workbench Example

CS317 – We Used CASE for DBMS, Limited to EER

Forward and Reverse Engineering – SQL code generation and EERs from SQL Deployed Database

Database: siewertsDHV1

<table>
<thead>
<tr>
<th>Table</th>
<th>Engine</th>
<th>Collation</th>
<th>Data Length</th>
<th>Index Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advert</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>32,768</td>
</tr>
<tr>
<td>Branch</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>32,768</td>
</tr>
<tr>
<td>BusinessOwner</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>32,768</td>
</tr>
<tr>
<td>Lease</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>32,768</td>
</tr>
<tr>
<td>Manager</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>32,768</td>
</tr>
<tr>
<td>Newspaper</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>0</td>
</tr>
<tr>
<td>PrivateOwner</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>0</td>
</tr>
<tr>
<td>Property_For_Rent</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>65,536</td>
</tr>
<tr>
<td>Registration</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>49,152</td>
</tr>
<tr>
<td>Renter</td>
<td>InnoDB</td>
<td>latin1_swedish_ci</td>
<td>16,384</td>
<td>0</td>
</tr>
</tbody>
</table>

SQL Command Line – Interpreter, Coding

Browswer, SQL Command Builder

EER Design
Quiz #1 - This Week (Canvas)

- 10 to 12 Problems
  - True / False, 4 Points Each
  - Short Answer, 6 Points Each

- Take up to 3x on Canvas

- Average score is recorded

- Normally we have 2 quizzes before Exam #1, and 2 or 3 after