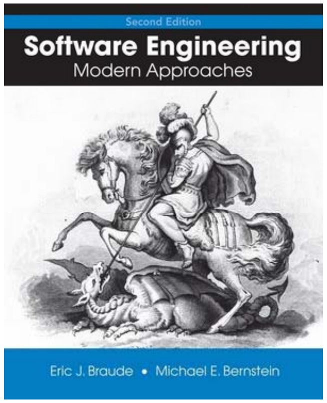
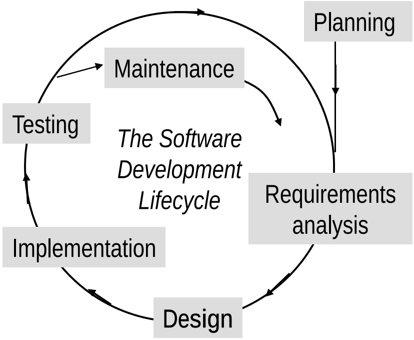
Software Engineering

Modern Approaches



Eric Braude and Michael Bernstein

Chapter 18: Software Architecture



Planning

Maintenance

Testing

The Software Development Lifecycle

Requirements analysis

Implementation

Design

Phase most relevant to this chapter is shown in bold

Learning Goals of This Chapter

- How do you classify "software architectures?"
- What are data flow architectures?
- What are three-tier architectures and their generalizations?
- What makes database-centric systems a separate type of architecture?
- What are service-oriented architectures?
- What are the IEEE standards for expressing designs?
- What do real-world architectures look like?

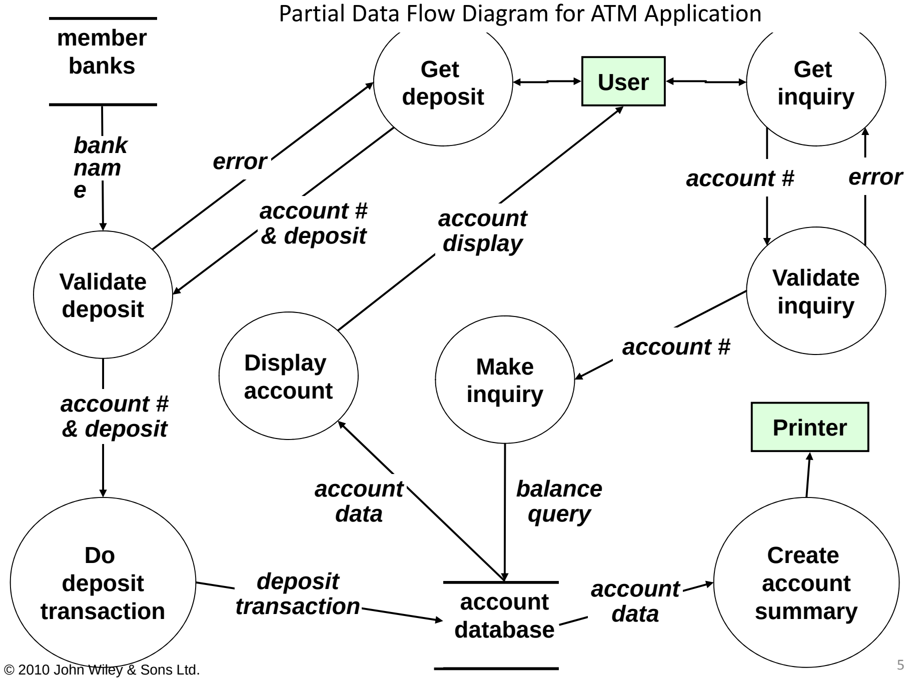
Categorization of Software Architectures

(Shaw & Garlan)

- Dataflow architectures
  - Pipes and Filters
  - Batch sequential
- Independent components
  - Client-server systems
  - Parallel communicating processes
  - Event systems
  - Service-oriented (added)

- Virtual machines
  - Interpreters
  - Rule-based systems
- Repository architectures
  - Databases
  - Hypertext systems
  - Blackboards
- Layered architectures

Partial Data Flow Diagram for ATM Application



member banks

bank nam e

error

account # & deposit

Get deposit

User

Get inquiry

account display

account #

error

Validate deposit

Validate inquiry

Display account

Make inquiry

account #

account # & deposit

Printer

account data

balance query

Do deposit transaction

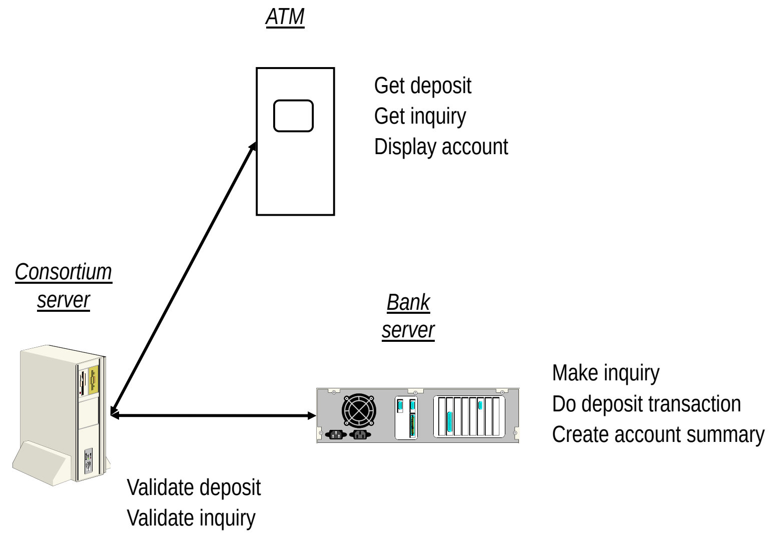
deposit transaction

account database

account data

Create account summary

Platforms in Data Flow Architectures: An Example



ATM

Get deposit
Get inquiry
Display account

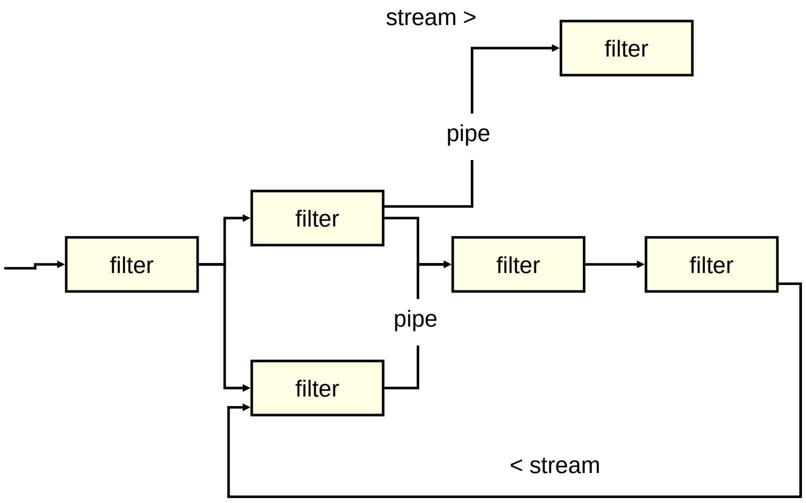
Consortium server

Bank server

Make inquiry
Do deposit transaction
Create account summary

Validate deposit
Validate inquiry

Pipe and Filter Architectures



stream >

filter

pipe

filter

filter

filter

filter

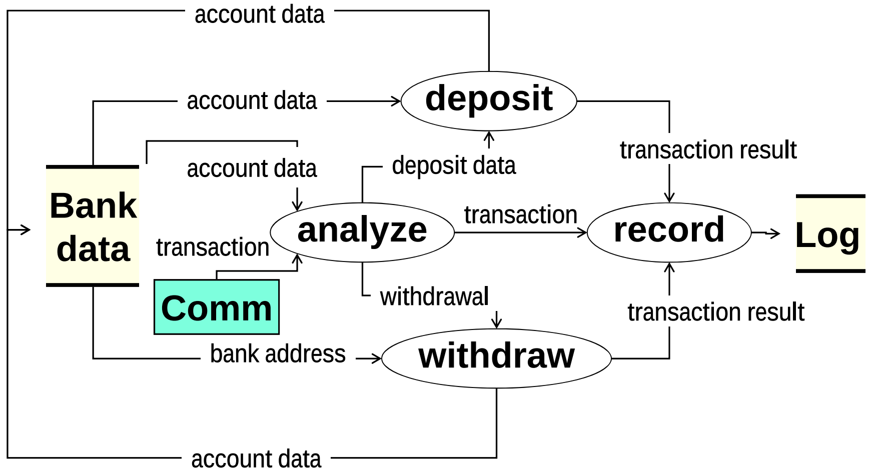
pipe

filter

< stream

Example of *Pipe & Filter Data Flow* Architecture

Requirement: **Maintain wired financial transactions.**



account data

account data

deposit

account data

deposit data

transaction result

Bank data

transaction

analyze

transaction

record

Log

Comm

withdrawal

transaction result

bank address

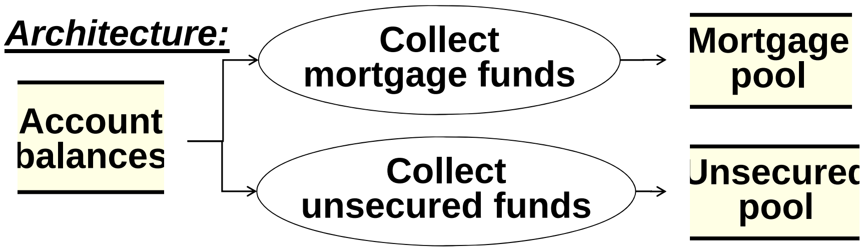
withdraw

account data

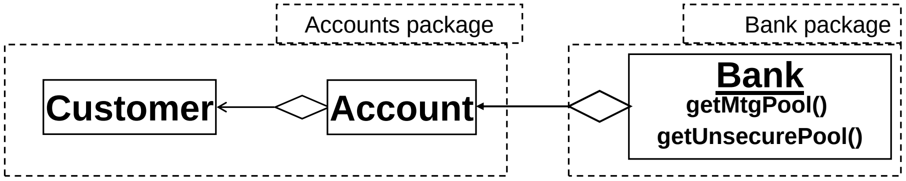
Example of *Batch Sequential Data Flow* Architecture

Requirement: **Manage bank funds available for mortgages & unsecured lending.**

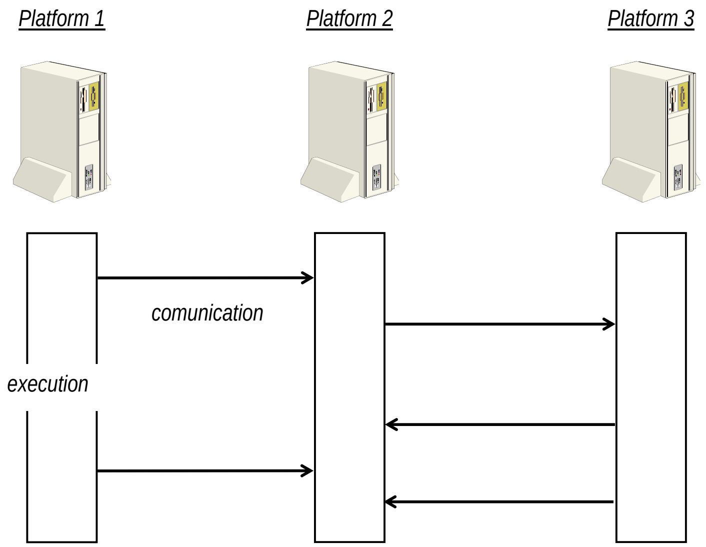
Architecture:



Class Model for *Batch Sequential Data Flow*



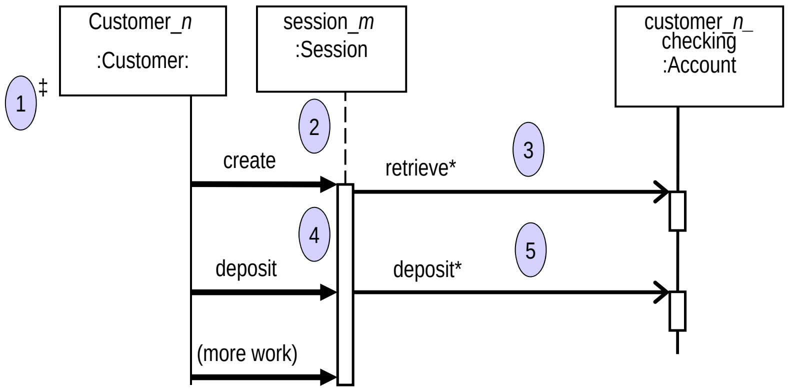
Platforms for Communicating Processors



Example of *Parallel Communicating Processes* Architecture

Requirement: Manage ATM traffic.

Architecture beginning with first session:

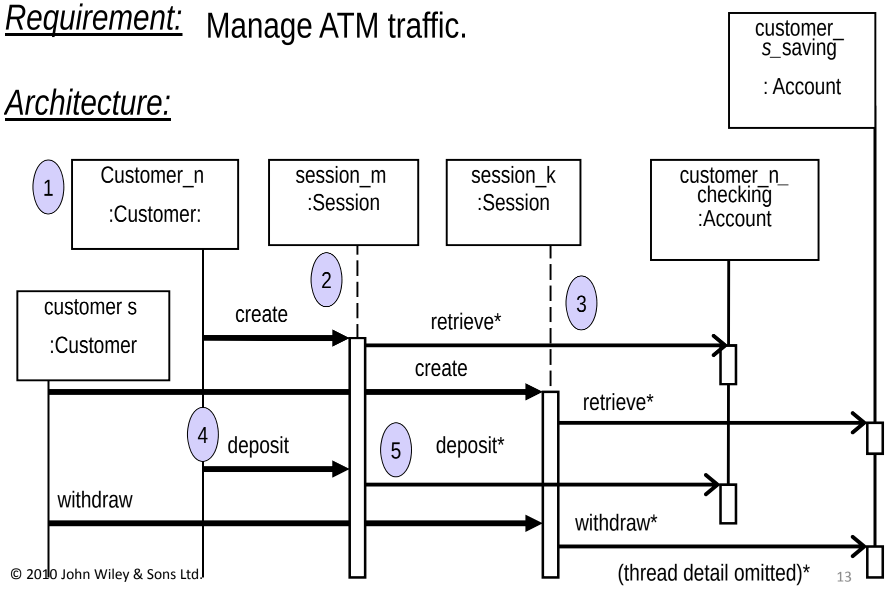


(thread detail omitted)*

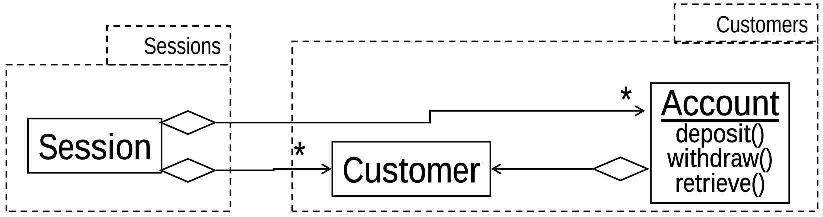
Example of *Parallel Communicating Processes* Architecture

Requirement: Manage ATM traffic.

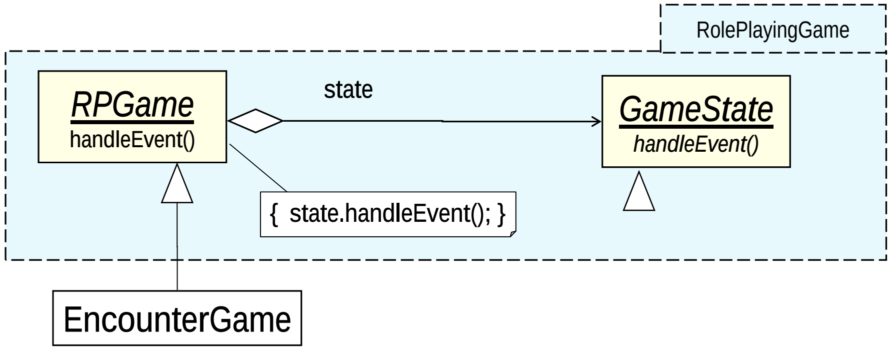
Architecture:



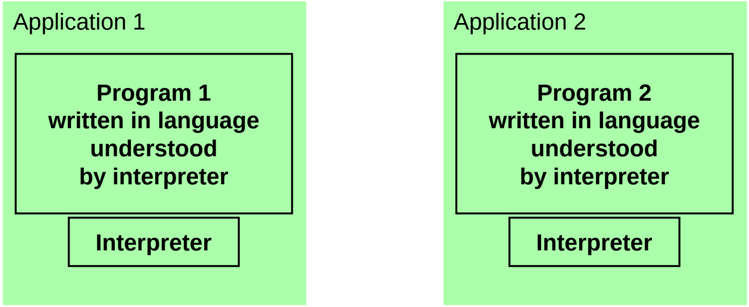
Class Model For Parallel Example



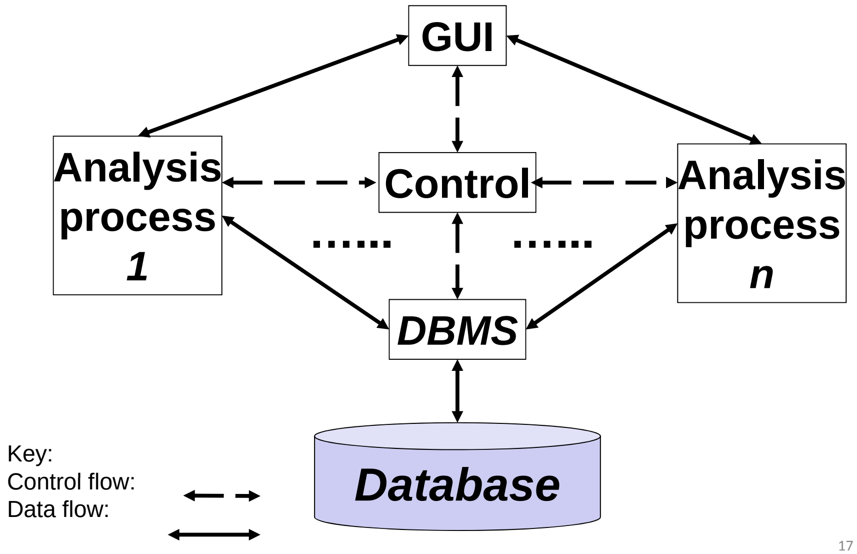
State Design Pattern Applied to *Encounter*



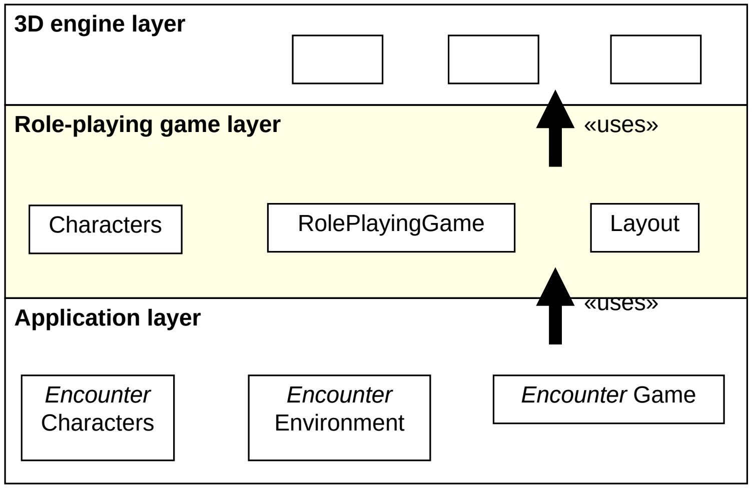
Virtual Machine Architectures:
Leveraging Interpreter to Facilitate Creation of Appliations



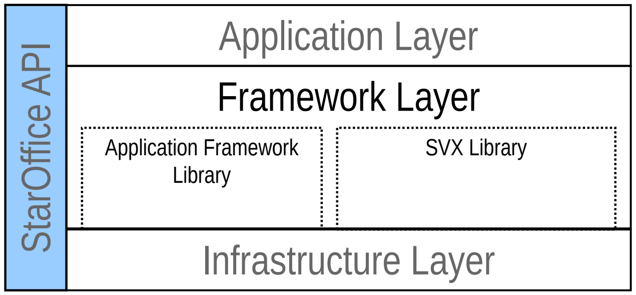
A Typical Repository Architecture



Layered Architecture

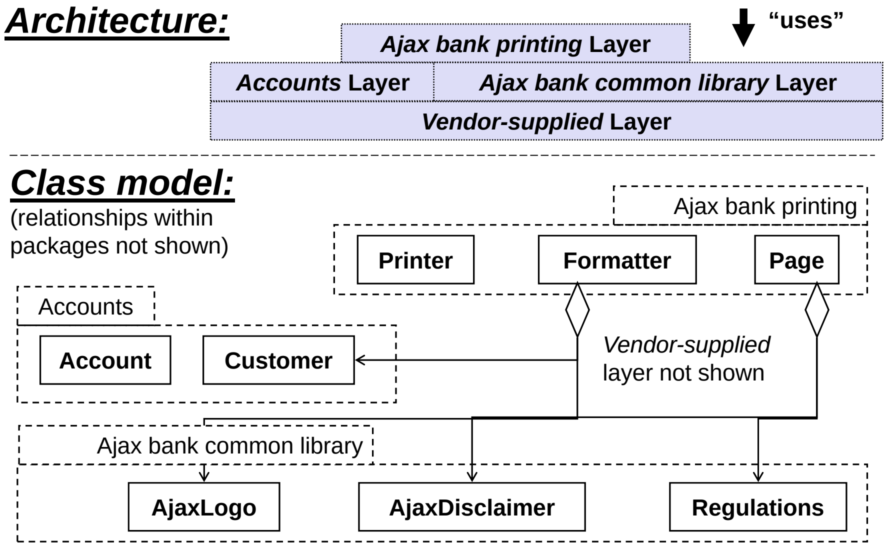


Framework Layer



From http://www.openoffice.org/white_papers/tech_overview/tech_overview.html#3

Layered Architecture Example Using Aggregation



Service-Oriented Architectures

Based on components that provide functionality according to an **interface** spec.

- – Principally via Web services
- – In the spirit of façade objects
- – Not necessarily OO

Example: An application concerning *orders*.

- – Wouldn't assume an *Order* class known to all
- – Instead: Define an *order* **schema**; reference when Web services involve orders

Service-Oriented Architectures *2 of 2*

- "Fire and forget"
  - – **Stateless** as much as possible
- **Extensible**
  - – Additional functionality easily added
- **Discoverable**
- **Account for Quality of Service**
  - – E.g., security

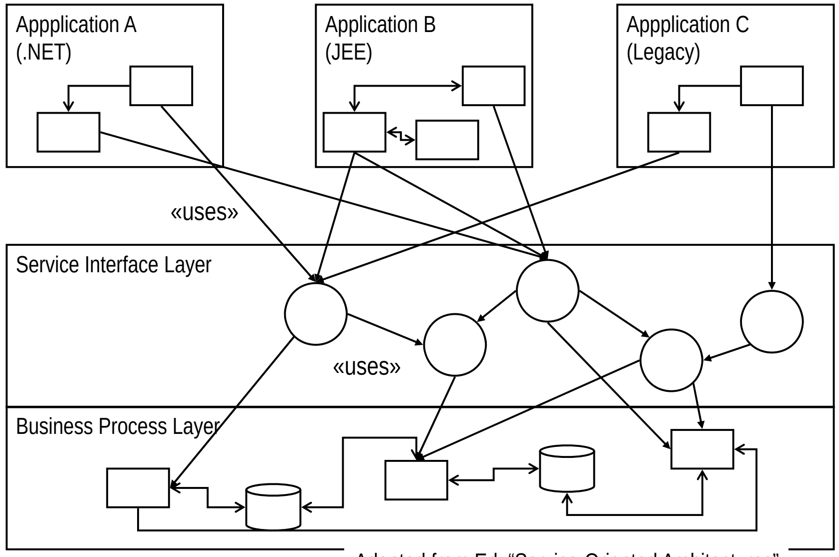
Many of these points are from *Service-Oriented Architecture by T. Erl*

Service-Oriented Architectures *3 of 3*

- "Fire and forget"
  - – **Stateless** as much as possible
- **Extensible**
  - – Additional functionality easily added
- **Discoverable**
- **Account for Quality of Service**
  - – E.g., security

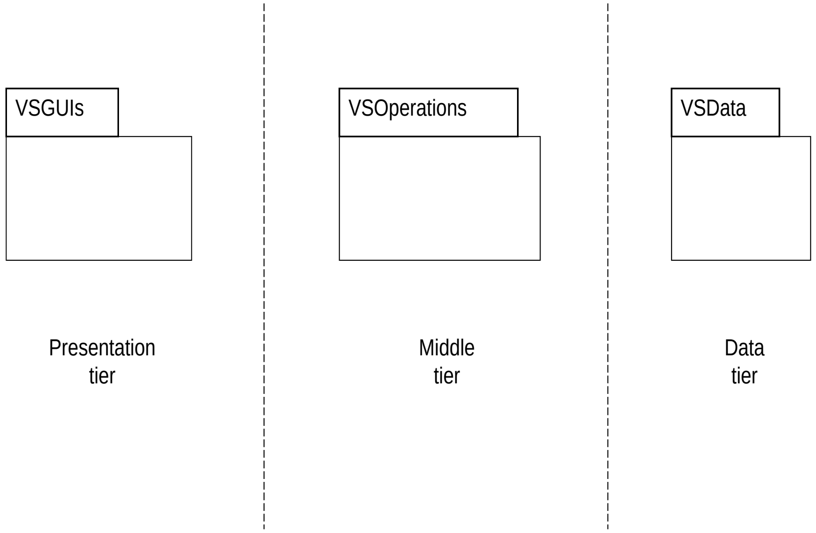
Many of these points are from *Service-Oriented Architecture by T. Erl*

Layering for Service-Oriented Architectures

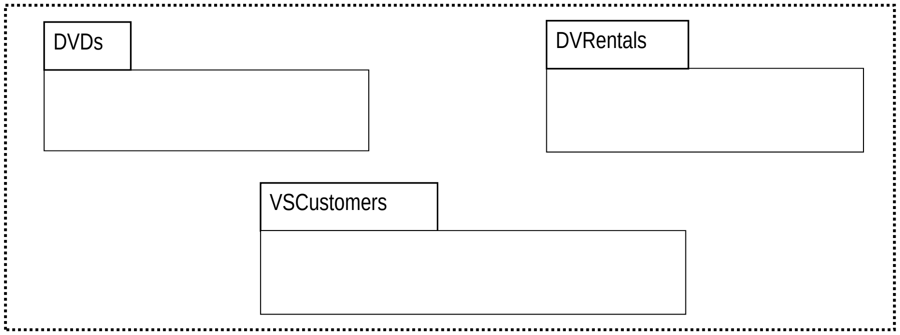


Adapted from Erl: "Service-Oriented Architectures"

3-Tier Architecture Aternative



Alternative Architecture for a Video Store Application



Comparing Architectures

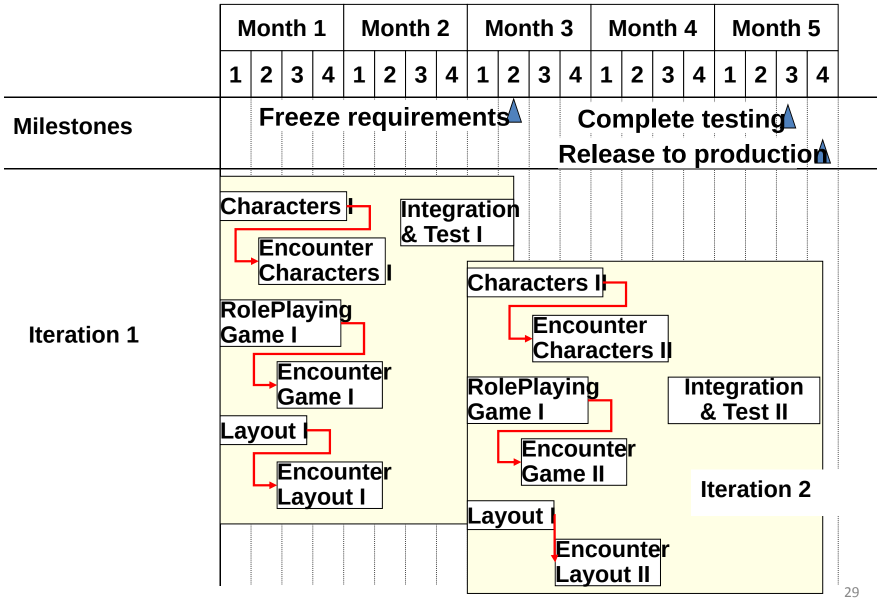
	Three-tier	Alternative
Understand-able?	Yes	Yes
Flexible?	Yes: GUI easy to change	Yes: Basic building blocks easy to identify.
Reusable?	Not very: Each layer is special to Video Store rentals.	Yes: Easy to generalize to generic rentals
Easy to construct?	Perhaps	Yes: Clear potential to use Façade.

IEEE 1016-1998 SDD Example Table of Contents

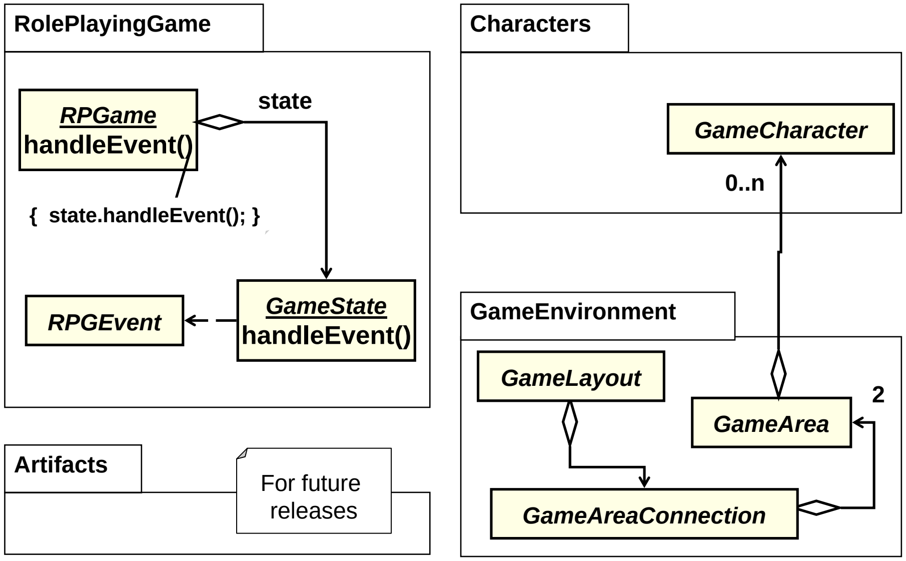
Architecture

1. Introduction
   - 1.1. Purpose
   - 1.2. Scope
   - 1.3. Definitions, acronyms & abbreviations
2. References
3. Decomposition description
   - 3.1. Module decomposition
     - 3.1.1 Module 1 description
     - 3.1.1 Module 2 description
   - 3.2 Concurrent process decomposition
     - 3.2.1 Process 1 description
     - 3.2.2 Process 2 description
   - 3.3 Data decomposition
     - 3.3.1 Data entry 1 description
     - 3.3.2 Data entry 2 description
4. Dependency description
   - 4.1 Inter-module dependencies
   - 4.2 Inter-process dependencies
   - 4.3 Data dependencies
5. Interface description
   - 5.1 Module interface
     - 5.1.1 Module 1 description
     - 5.1.2 Module 2 description
   - 5.2 Process interface
     - 5.2.1 Process 1 description
     - 5.2.2 Process 2 description
6. Detailed design
   - 6.1 Module detailed design
     - 6.1.1 Module 1 detail
     - 6.2.2 Module 2 detail
   - 6.2 Data detailed design
     - 6.2.1 Data entity 1 detail
     - 6.2.2 Data entity 2 detail

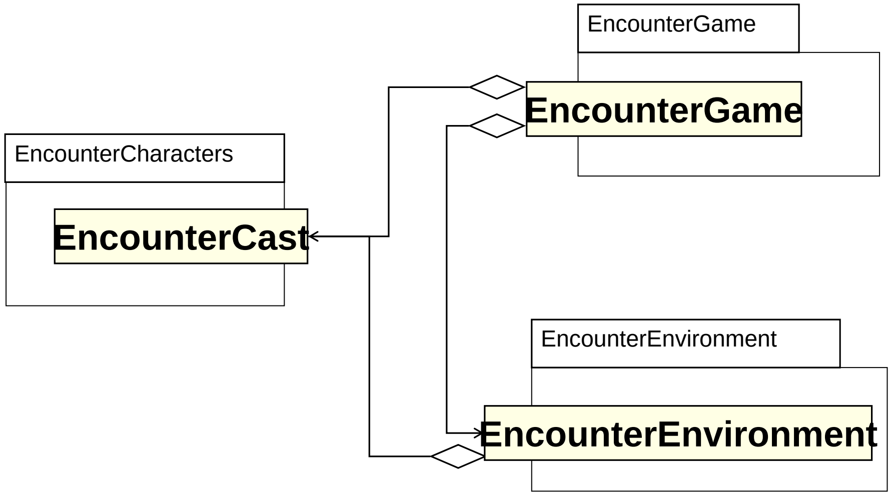
Schedule Following Architecture Selection



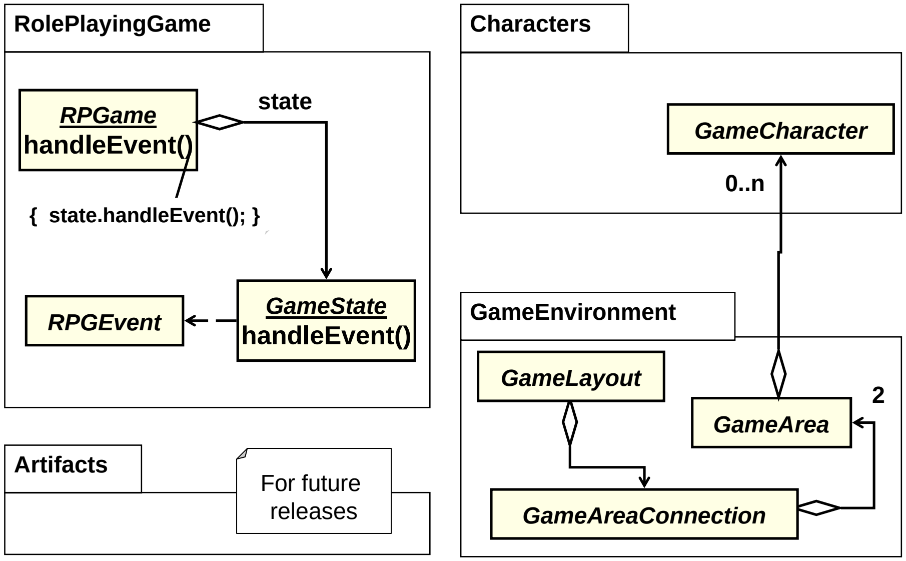
RPG Framework for Role-Playing Video Games



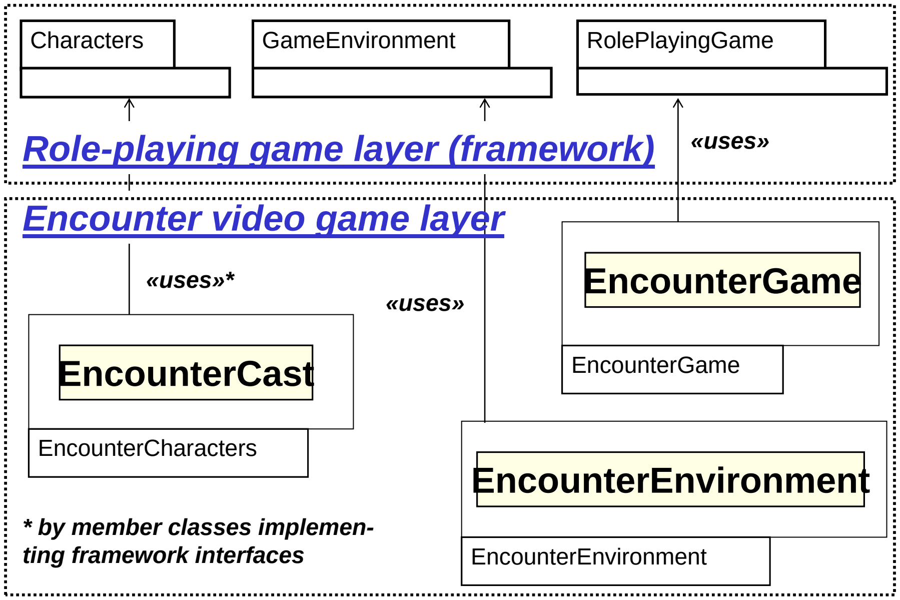
Architecture / Modularization of *Encounter*



RPG Framework for Role-Playing Video Games



FrameWork / Application Dependency



Characters

GameEnvironment

RolePlayingGame

Role-playing game layer (framework)

«uses»

Encounter video game layer

*«uses»**

«uses»

EncounterGame

EncounterGame

EncounterCast

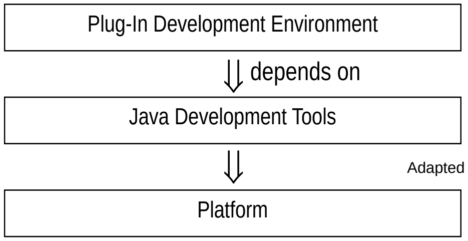
EncounterCharacters

EncounterEnvironment

EncounterEnvironment

*** by member classes implementing framework interfaces***

Architecture of Eclipse *1*: Overall



Plug-In Development Environment

⇓ depends on

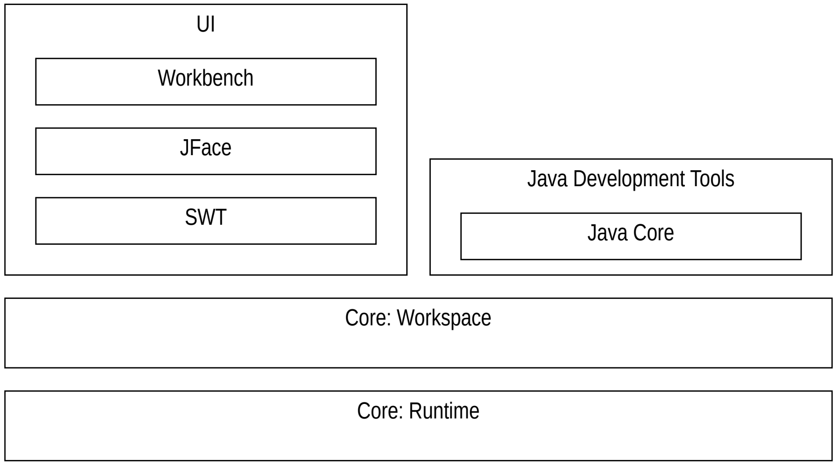
Java Development Tools

⇓

Platform

Adapted from “Contributing to Eclipse” p5

Architecture of Eclipse



UI

Workbench

JFace

SWT

Java Development Tools

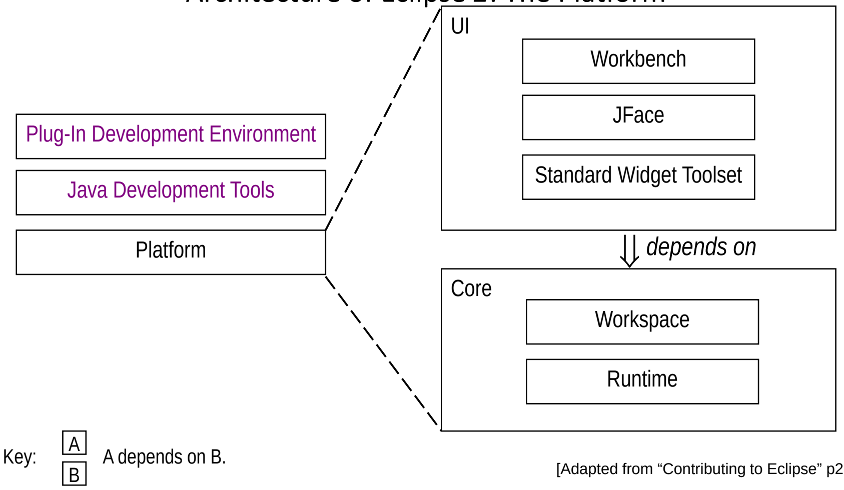
Java Core

Core: Workspace

Core: Runtime

[“Contributing to Eclipse p282]

Architecture of Eclipse *2*: The Platform



UI

Workbench

JFace

Standard Widget Toolset

Plug-In Development Environment

Java Development Tools

Platform

⇓ *depends on*

Core

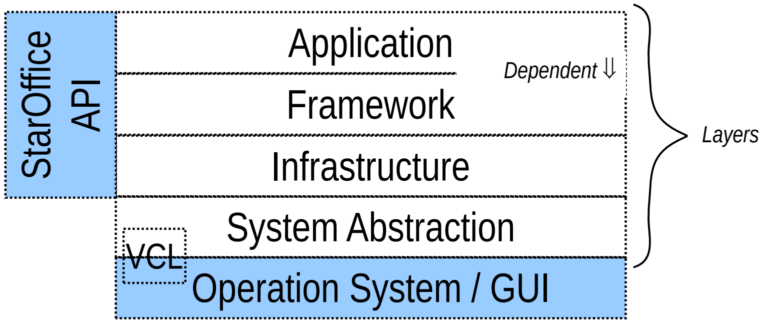
Workspace

Runtime

Key: A / B A depends on B.

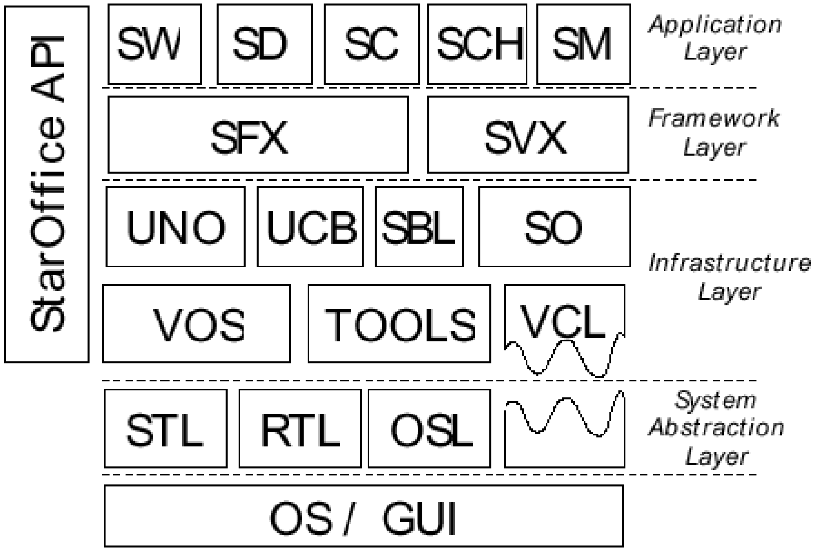
[Adapted from “Contributing to Eclipse” p283]

OpenOffice Architecture



Edited, from http://www.openoffice.org/white_papers/tech_overview/tech_overview.html#3

OpenOffice Architecture



From http://www.openoffice.org/white_papers/tech_overview/tech_overview.html#3