CSC 3380 Aymond

Section 1

2/10/2020

Project News

- Revibe project Slack is set up
  - One person from each team needs to set up the team channel
  - Each team member then joins their project channel
  - Post class-wide questions to the main channel
- Next Milestone: #2
  - Due Friday 2/21, 11PM
  - Upload to Moodle (1 upload for entire team)
  - Outline is in Project Kickoff Lecture Notes
  - All UML diagrams must be developed in EA

Object Oriented Design

- Source Control++
- The Design Process
- Software System Architecture
- **Architectural Styles**
- Object Oriented Design Principles

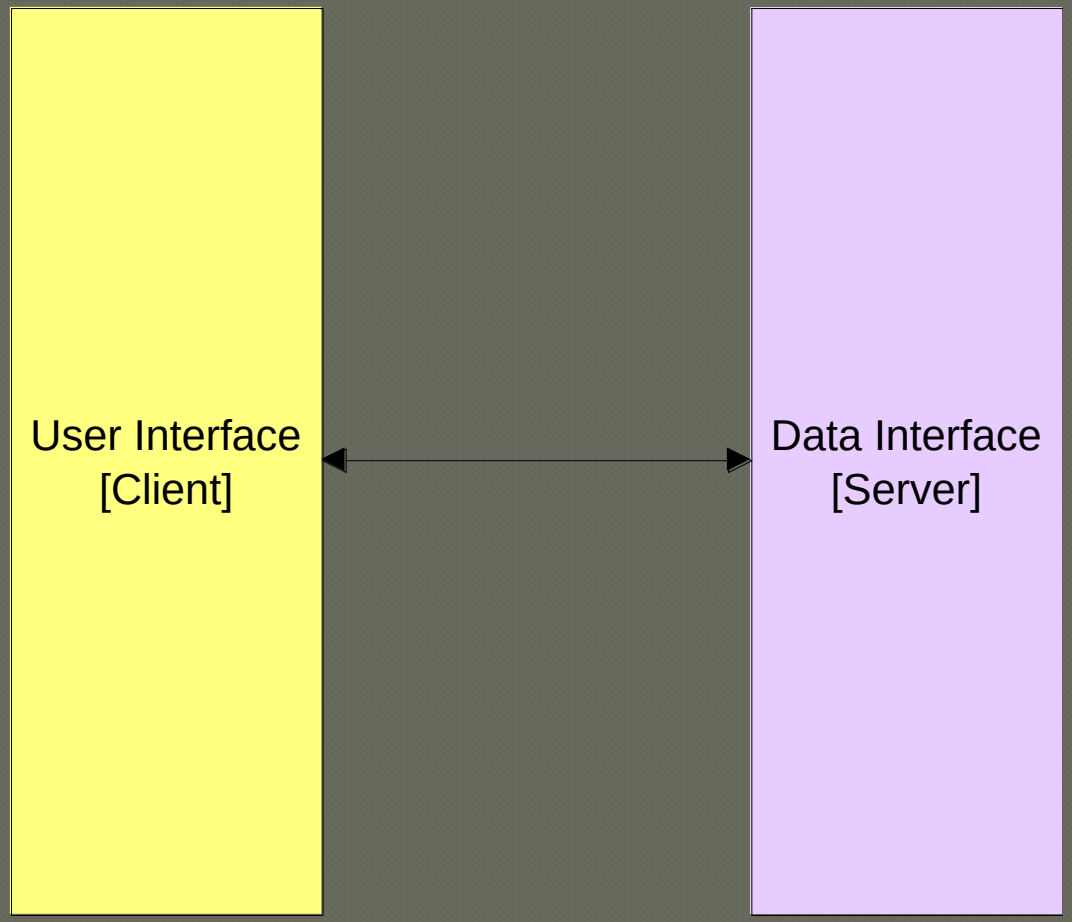
Architectural Styles

- **Client/Server**
- Data Centric
- Peer-to-Peer
- Pipe and Filter
- Model/View/Controller
- Publish/Subscribe

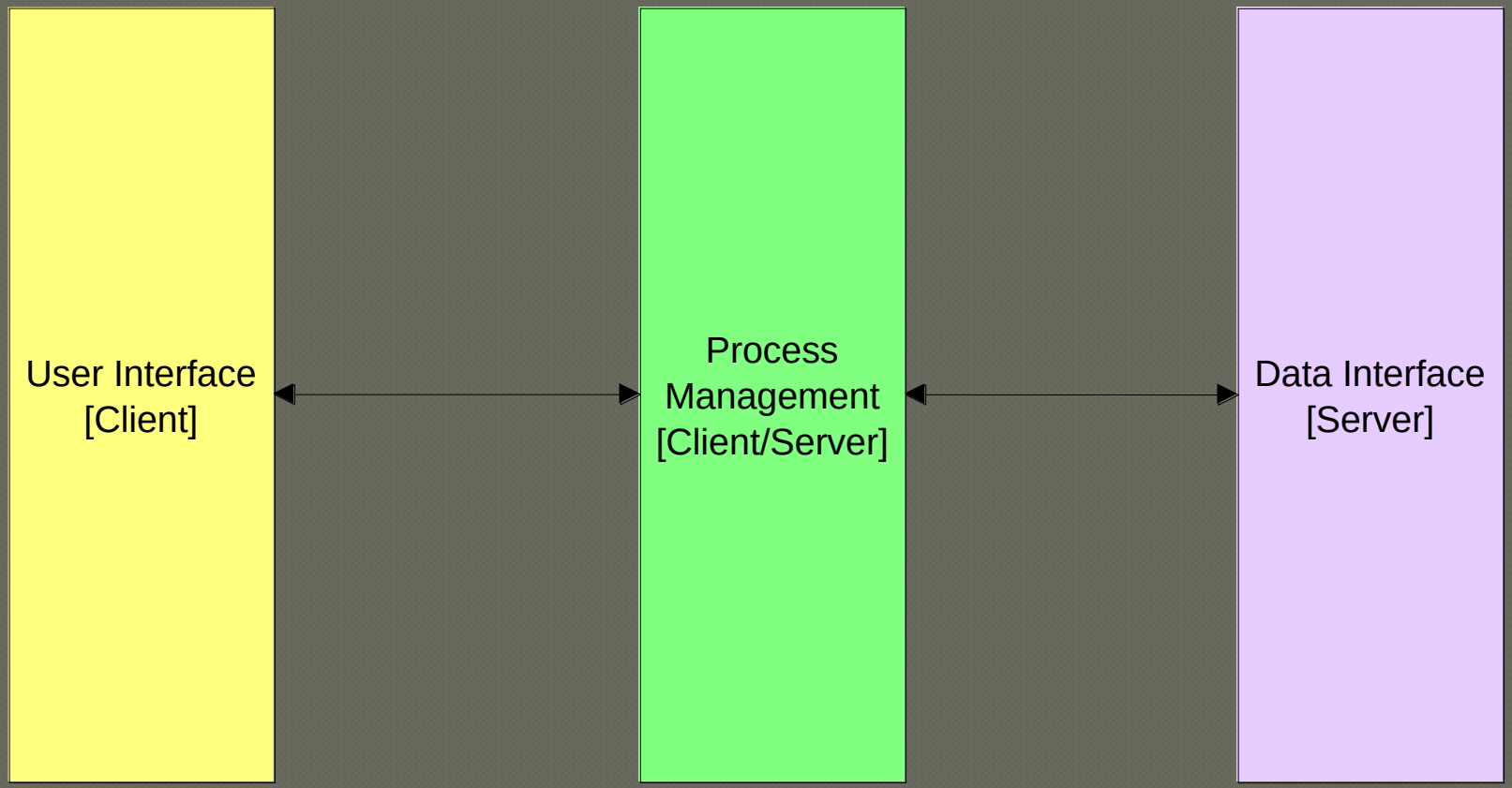
Client/Server

- A network architecture in which each computer or process on the network is either a client or a server.
  - Servers are processes dedicated to managing resources, such as disk drives (file servers), printers (print servers), network traffic (network servers), or computational services
    - The server "guards" access to the important resources that it manages
  - Clients are programs, which are run by users or specialized applications. Clients rely on servers for resources, such as files, devices, data, and computations (e.g., processing power)
    - Clients connect to the server

Client Server Abstraction: 2-tier Architecture

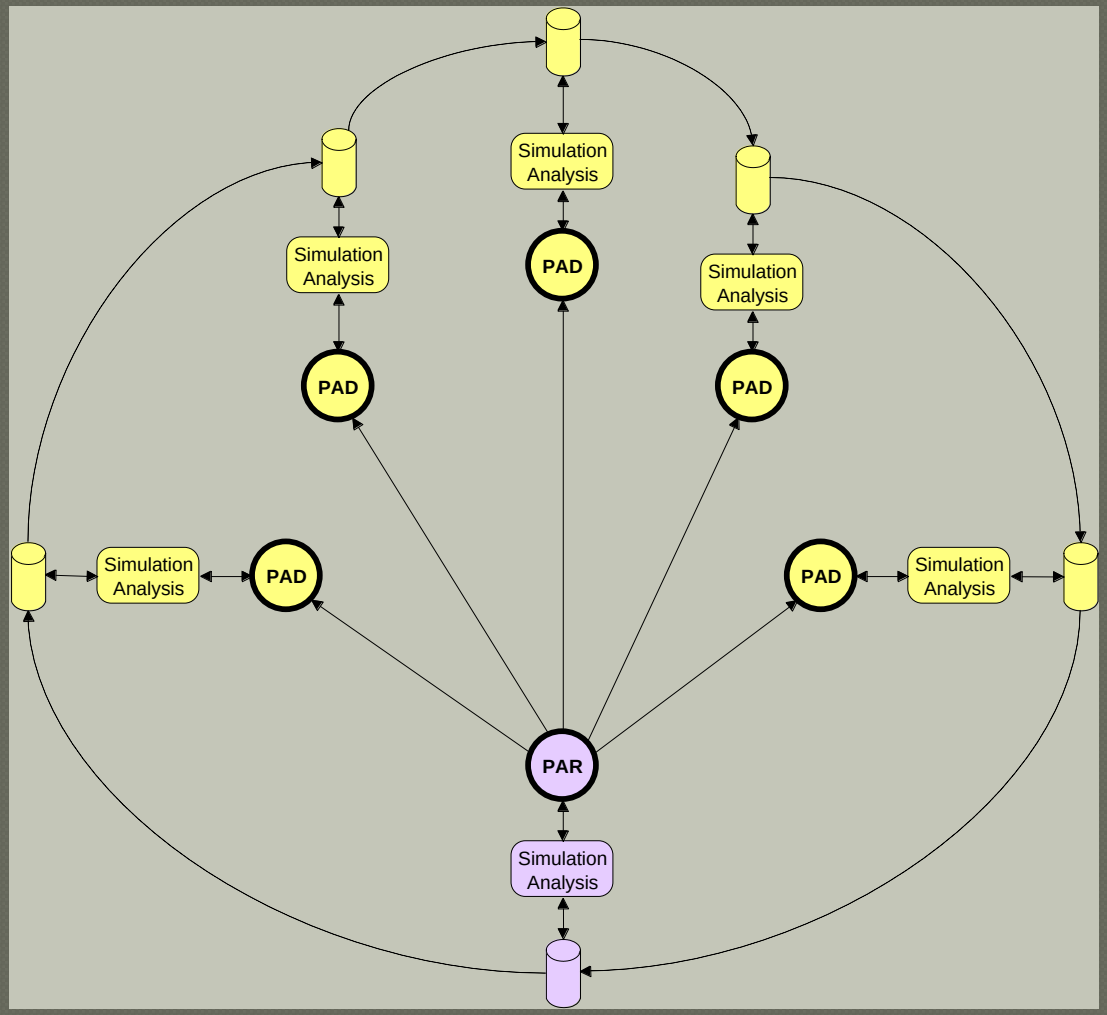


Client Server Abstraction: 3-tier Architecture



- Process Management acts as a server to the User Interface client
- It acts as a client to the Data Interface Server

Client/Server Example
US Army's PAR/PAD Generator



Other Examples of Client/Server

- Multiplayer videogames:
  - Clients absolutely cannot be trusted to change gamestate directly
  - Server validates actions and ensures rules are followed
  - Doesn't help with client side exploits (wallhacks)
- X11 windowing system
  - Allows a windowing application to be separated from a "view"
  - Streams drawing commands over network
  - Allows low-latency screen sharing, but still being replaced

A Note on Using Web Servers

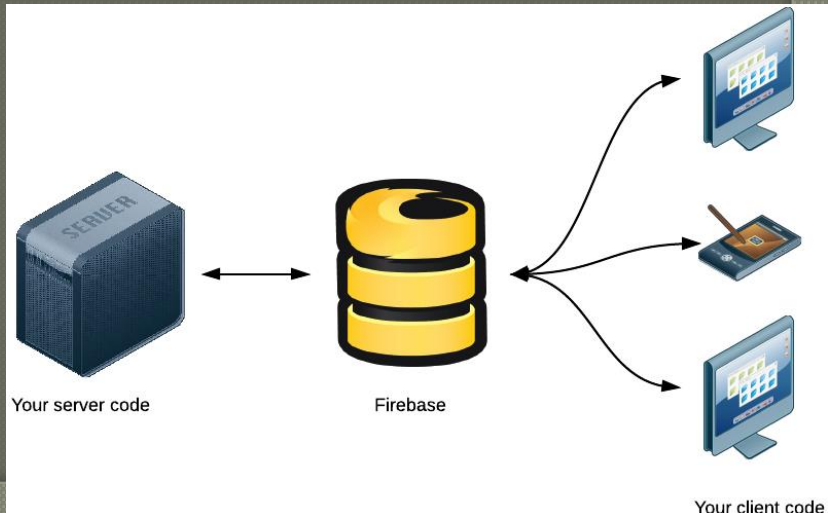
- If you plan on developing a web app, don't rewrite a webserver yourself
- The absolute easiest way to have a web-based server is for it to be a (fast) CGI application
- Install an http server on your host
  - Nginx is good one
  - Apache is widely supported
- Configure it to call your program
- Your server application will return either HTML or JSON

Architectural Styles

- Client/Server
- **Data Centric**
- Peer-to-Peer
- Pipe and Filter
- Model/View/Controller
- Publish/Subscribe

Data Centric vs Client/Server

- In the data centric architecture, both clients and servers modify the same database
  - Firebase apps use a data centric design
- This is different than a client/server model
  - In client/server, clients go through the server, and the server updates the database



Data Centric Benefits

- Lower latency than client/server
- Easy to scale number of servers (major benefit)
- Database handles authentication/security

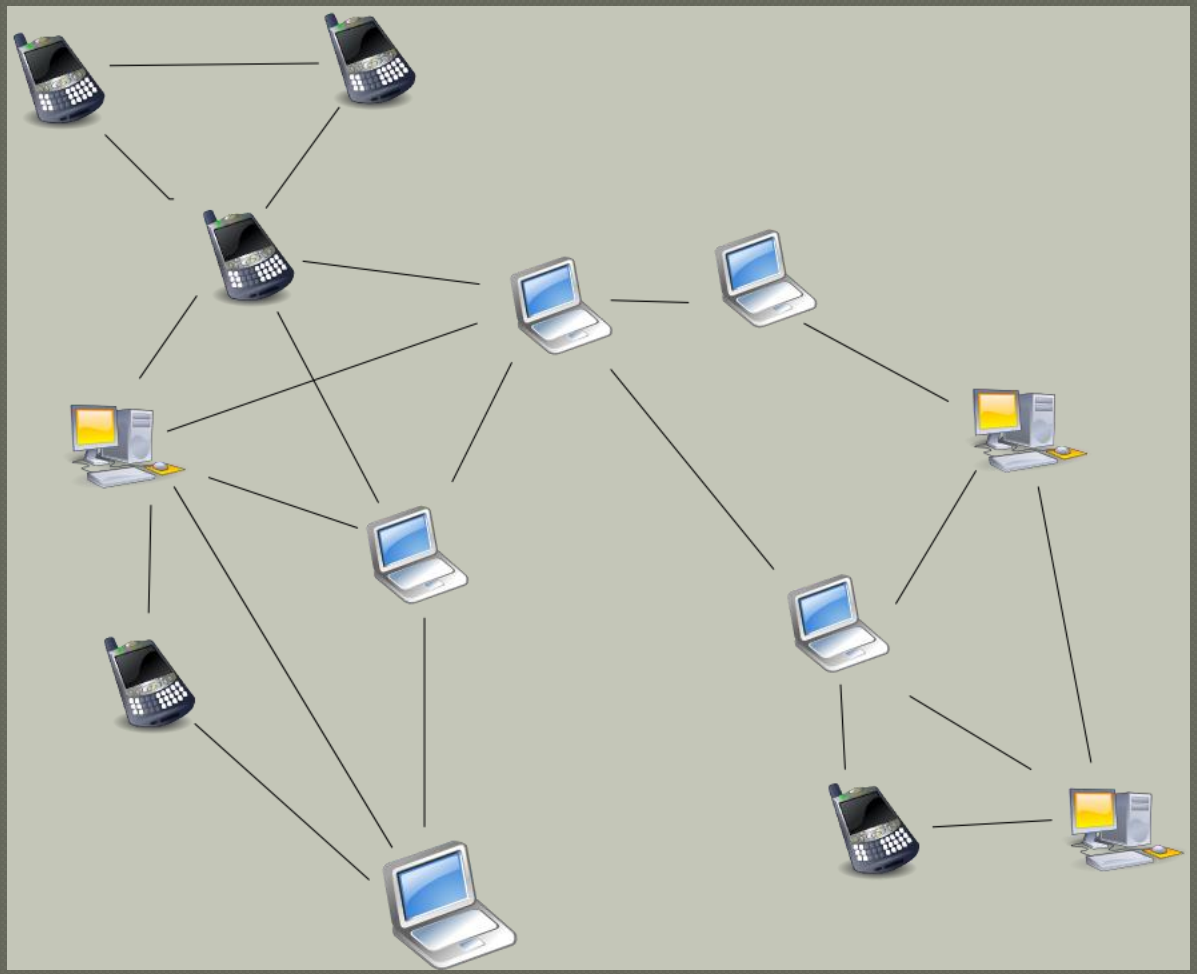
Data Centric Drawbacks

- Some operations are too complicated to allow clients to manage
- Breaks encapsulation
  - adds coupling between client and data
- Uglier communication interface
  - Servers pull requests from database instead of API
- Single point of failure
  - Attractive target for DDoS attacks

Architectural Styles

- Client/Server
- Data Centric
- **Peer-to-Peer**
- Pipe and Filter
- Model/View/Controller
- Publish/Subscribe

Peer to Peer Architecture



Peer to Peer Concept

- No central server, peers broadcast directly to each other
- Each client maintains private state
- Needs to manage:
  - Synchronization
  - Dropout
  - Security
  - Flooding
- Quite complicated

Peer to Peer Benefits

- Robust: no single point of failure
- Don't require central infrastructure
- Useful when communication range is limited (e.g., underwater drones)

Peer to Peer Drawbacks

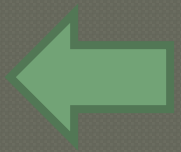
- Synchronization problems
  - What time is it?
- Consensus problems
  - What is the state?
- Byzantine attacks
  - A node becomes infected and provides wrong information to other peers, creating an unstable network

Peer to Peer Examples

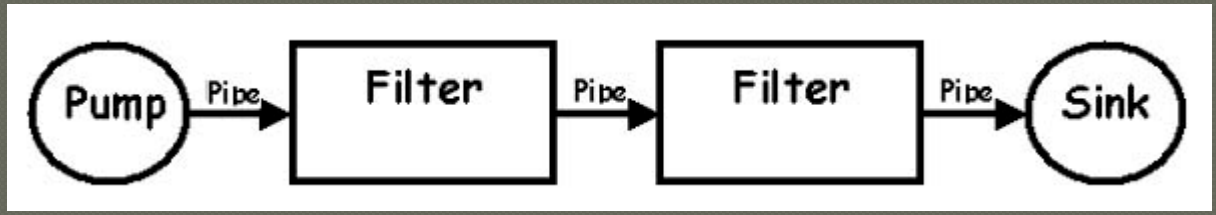
- Cryptocurrency
- Mesh networks
- Automotive networks
- Internet of Things (IOT)
- PC to PC local WiFi hotspot

Architectural Styles

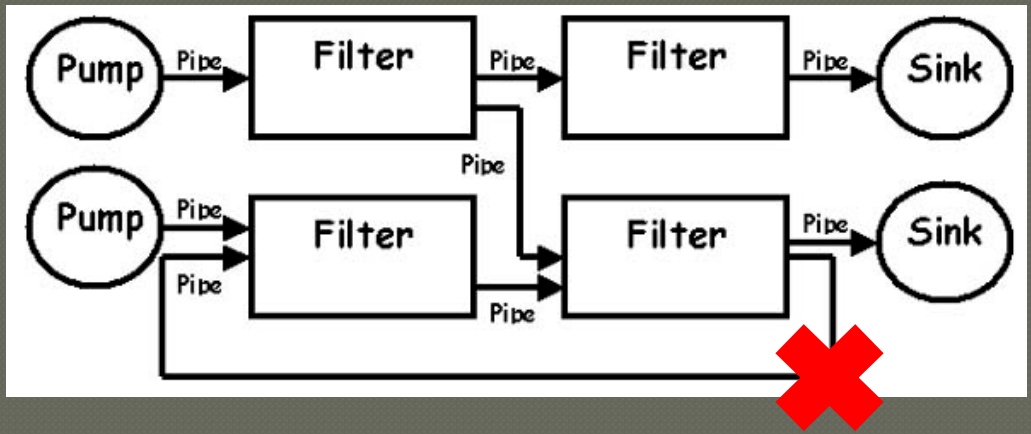
- Client/Server
- Data Centric
- Peer-to-Peer
- **Pipe and Filter**
- Model/View/Controller
- Publish/Subscribe



Pipe & Filter Architecture



- Data flows in one direction.
- Each component "transforms" data
- No loops, 2-way communication, feedback

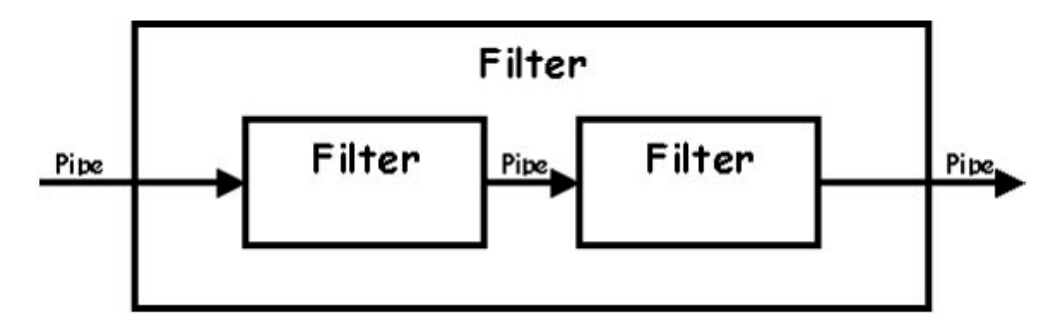


- Most important: no state (massive benefit)

http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html

Pipe & Filter Benefits

- Easy to understand
- Easy to break down problem



- Inherit Parallelization of Processing
- Highly flexible
- Very few bugs caused by state
- If you can use P&F, you probably should

http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html

Pipe & Filter Applications

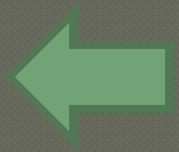
- Natural Language Processing
- Fundamental to the Unix philosophy:
  - Small programs that do one thing
  - Output and input are text
  - Piping programs together
    - `findmnt | grep … | nano`
    - `dmesg | sort | less`
- Particularly suited to functional programming ("components" are actually functions)
- Extremely common in mathematics and engineering

Pipe & Filter Drawbacks

- Implementation necessarily linear
  - Can't skip a step
  - Can't repeat a step

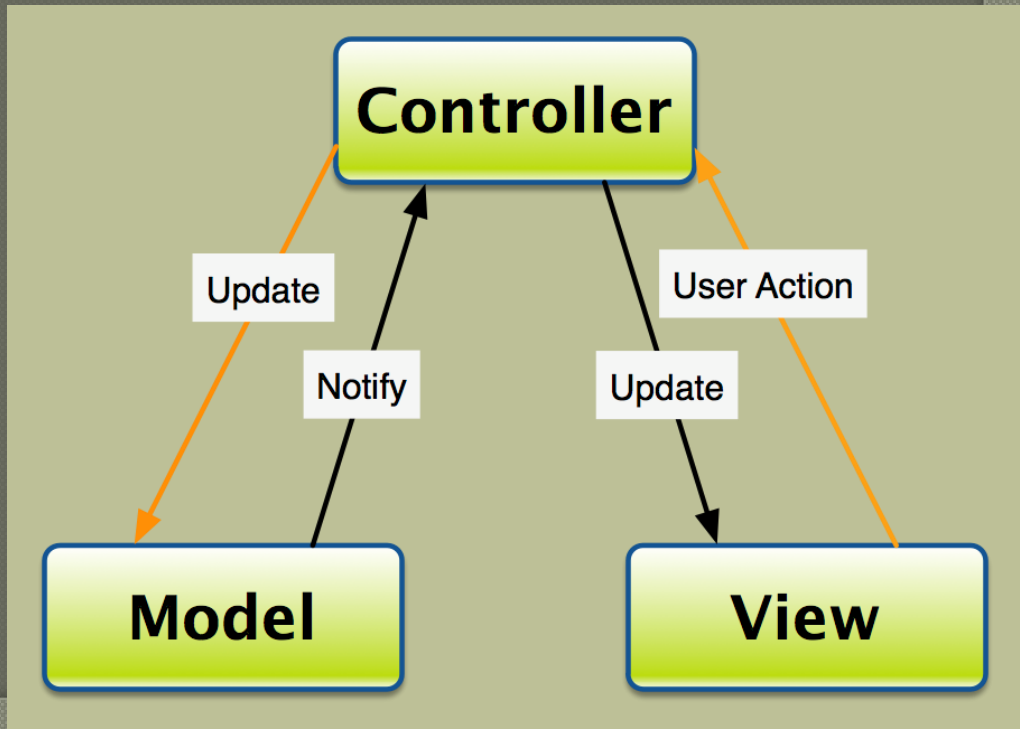
Architectural Styles

- Client/Server
- Data Centric
- Peer-to-Peer
- Pipe and Filter
- **Model/View/Controller**
- Publish/Subscribe



Model/View/Controller (MVC) Concepts

- The **model** represents the data
- The **view** is what the user sees: input/output
- The **controller** mediates between the model and the view and executes business logic (rules).



MVC Example

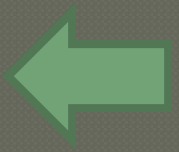
- Timesheet management:
  - The timesheet and employees are stored in the database (models)
  - The interface allows interaction with these objects (views)
  - The business logic makes sure changes are allowed (controller)

MVC Examples

- Extremely common for web apps with conventional business logic (no need for custom server)
- Every GUI app in the world
- Ruby-on-rails apps
- Video games in general (game state, game logic, interface)

Architectural Styles

- Client/Server
- Data Centric
- Peer-to-Peer
- Pipe and Filter
- Model/View/Controller
- **Publish/Subscribe**



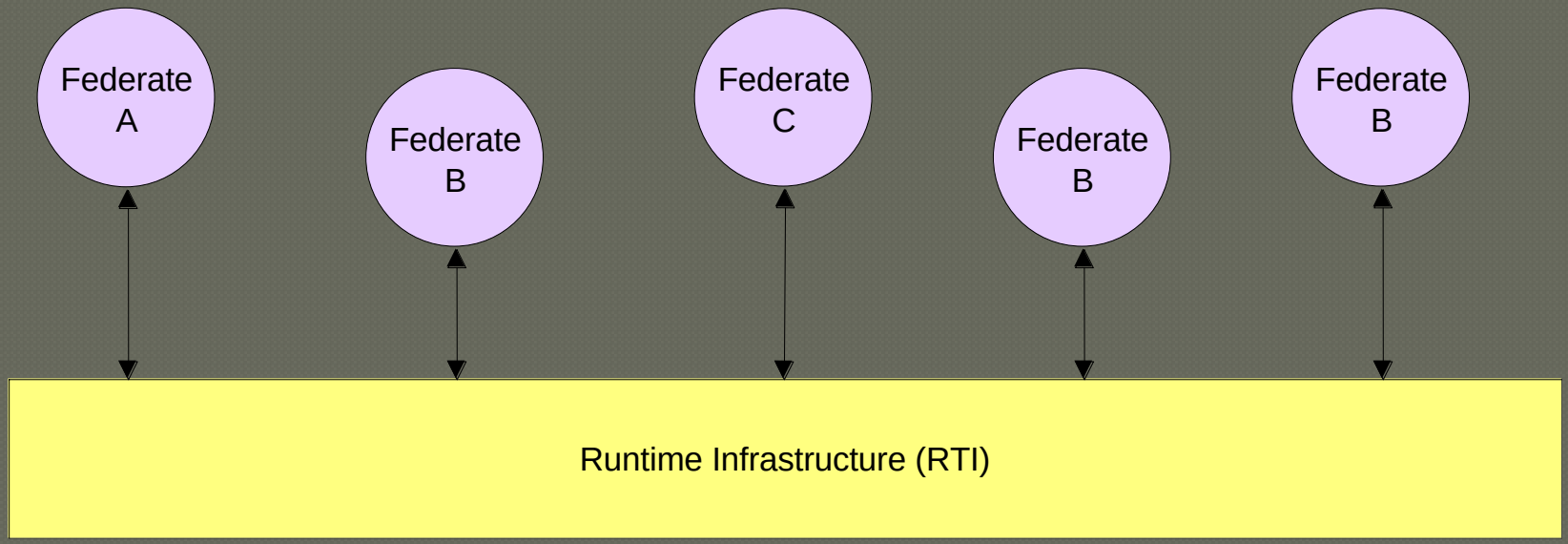
Publish/Subscribe Architecture

- Components do not communicate directly
- Publishers provide information to be used by any subscriber in the system
- A communication broker manages getting most appropriate information from a publisher to match the needs of a subscriber

High Level Architecture (HLA)

- The High Level Architecture (HLA) is a general purpose architecture for simulation reuse and interoperability
- HLA has merited the distinction of preferred architecture for simulation interoperability within the DoD

HLA Framework

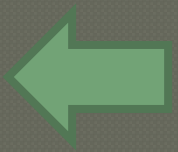


HLA Terms

- Federates
  - simulations, supporting utilities, or interfaces to live systems
- Runtime Infrastructure
  - a special-purpose distributed operating system
- Federation
  - sets of federates working together to support distributed applications
- Object Model Template (OMT)
  - Federates and federations are required to have an object model describing the entities represented in the simulations and the data to be exchanged across the federation
  - Federation Object Model (FOM)
  - Simulation Object Model (SOM)

Object Oriented Design

- Source Control++
- The Design Process
- Software System Architecture
- Architectural Styles
- **Object Oriented Design**



The OO Paradigm

- Abstraction
  - Hidden Data
    - Implementation of Abstract Data Type (ADT) is irrelevant
    - *** **Class members are not (NEVER) accessed directly** ***
      - No public class members
- Encapsulation
  - Data and methods on that data are bundled together
    - A class defines the data implementation, access to the data elements, and methods that act on the data
- Inheritance
  - A class can take on the properties of another class
    - Creates the is-a relationship between the base class and the superclass
- Polymorphism
  - Derived objects (those of a class inherited from another) can behave differently
    - Interface of inherited methods remain the same, but may function differently

The Five Principles of Class Design

- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

- Other OO Design Principles
  - YAGNI
  - Once & Only Once

http://wiki.c2.com/?PrinciplesOfObjectOrientedDesign

Single Responsibility Principle

- Each responsibility should be a separate class, because each responsibility is an axis of change
- A class should have one, and only one, reason to change

- If a change to the business rules causes a class to change, then a change to the database schema, GUI, report format, or any other segment of the system should not force that class to change

http://wiki.c2.com/?SingleResponsibilityPrinciple

Open/Closed Principle

- Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification
- Classes should be designed as if they will persist forever
- The motivation is to prevent the introduction of bugs
- Does this reduce flexibility?

http://wiki.c2.com/?OpenClosedPrinciple

The Liskov Substitution Principle

- “If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T." – Barbara Liskov, Data Abstraction and Hierarchy, SIGPLAN Notices, 23,5 (May, 1988).
- If you substitute an instance of the derived class in a call to a base class method, nothing should break that always works for the base class, as specified by the base class interface

http://wiki.c2.com/?LiskovSubstitutionPrinciple

The Interface Segregation Principle

- The dependency of one class to another one should depend on the smallest possible interface (http://www.objectmentor.com/resources/articles/isp.pdf)

http://wiki.c2.com/?InterfaceSegregationPrinciple

Dependence Inversion Principle

- A. High level modules should not depend upon low level modules. Both should depend upon abstractions.

- B. Abstractions should not depend upon details. Details should depend upon abstractions. (http://www.objectmentor.com/resources/articles/dip.pdf)

http://wiki.c2.com/?DependencyInversionPrinciple

The YAGNI Principle

- "You aren't gonna need it"
- Avoid developing unless you have to:
  - The cheapest code is the code you don't write
- If it's in the requirements you probably do need it

http://c2.com/xp/YouArentGonnaNeedIt.html

Once and Only Once

- We never want to duplicate code
- What if there's an error in the code?
  - Now you have to change it everywhere
  - There is no way to ensure that code is in sync
- **Change** is the only reason why we're doing any of this

Best Practices

- Separate what changes from what stays the same
  - If something stays the same, you won't break it
  - Keeping change limited reduces the amount of analysis
- Coupling vs Cohesion
  - Coupling is bad, cohesion is good.
  - Classes should work together in small, cohesive clusters
  - **You should have high cohesion within modules and low coupling between modules**
- Program to an interface, not an implementation
  - Depend on an interface where practical (instead of class)
  - Allows classes to be swapped out later
  - Even more pragmatic with duck-typing (next slide)

A Word on Duck Typing

- Comes from saying: "If it walks like a duck and quacks like a duck, it's a duck"
- Means that any object can be substituted for any other if it has the correct methods
- Class doesn't matter; don't need to know class in advance.
- Common in dynamic OO languages:
  - Javascript, Lua, Ruby, Python, etc.
- Doesn't exist in static languages, but structural typing similar
  - Go, O'Caml, Scala, etc.

Why is Duck Typing Relevant?

- Duck-typed languages often don't have "interfaces" or "abstract classes"
- Instead, functions take an untyped parameter and simply execute methods on it
- Semantic ambiguities are not discovered until runtime

- Recommendations:
  - Make sure to test for runtime errors
  - Draw base classes and interfaces in UML anyway to capture intended relationships

Unified Modeling Language (UML)

- **Introduction to UML**
- Architectural Modeling
- Structural Modeling
- Behavioral Modeling

Systems and Modeling

- The user of modeling has a rich history in all engineering disciplines
- The four basic principles of modeling
  - The choice of what models to create has a profound influence on how a problem is attached an how a solution is shaped
  - Every model may be expressed at different levels of precision
  - The best models are connected to reality
  - No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models

The Unified Modeling Language User Guide, by G. Booch, J. Rambaugh & I. Jacobson

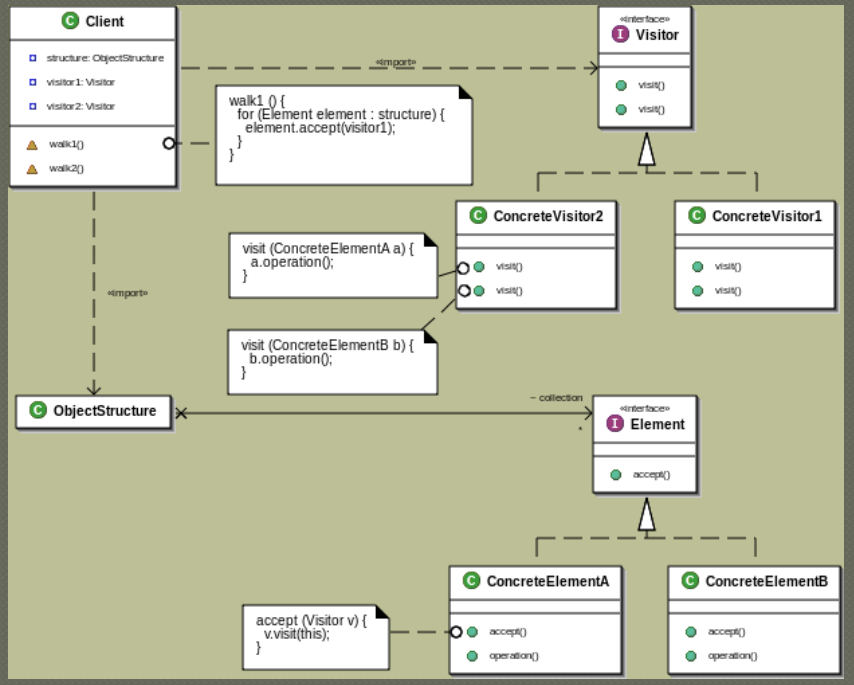
Object-Oriented Software System Architecture Views

- Complimentary and interlocking view:
  - Use case view
    - Exposing the requirements of the system
  - Design view
    - Capturing the vocabulary of the problem space and the solution space
  - Process view
    - Modeling the distribution of the system's processes and threads
  - Implementation view
    - Addressing the physical realization of the system
  - Deployment view
    - Focusing on system engineering issues
- Each of these views may have structural, as well as behavioral aspects
  - Together, these views represent the blueprints of software

The Unified Modeling Language User Guide, by G. Booch, J. Rambaugh & I. Jacobson

Unified Modeling Language (UML)

- Unified Modeling Language (UML) is a graphical **meta-language** for visualizing, specifying, and documenting software systems
  - UML is a language for visualizing
  - UML is a language for specifying
  - UML is a language for constructing
  - UML is a language for documenting



Unified Modeling Language (UML)

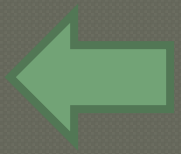
- Developers (the 3 amigos)
  - Grady Booch (Rational Software Corp)
  - James Rumbaugh (General Electric)
  - Ivar Jacobson (Objectory)
- Object Management Group (OMG)
  - An open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications
  - UML Standards
    - UML 1.0 (1995)
    - UML 1.x (1995)
    - UML 2.0 (2005), currently most widely used
    - UML 2.x, minor revisions

Unified Modeling Language (UML)

- Introduction to UML
- **Architectural Modeling**
- Structural Modeling
- Behavioral Modeling

Architectural Modeling

- **Component Diagrams**
  - Structure and connections of components
- Deployment Diagrams
  - Deployment of artifacts to nodes



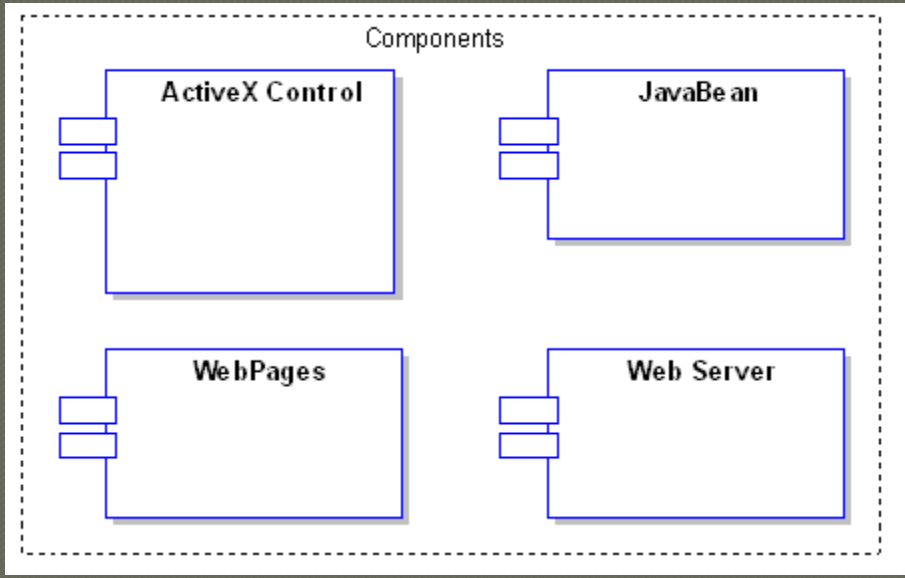
The Component Model

- The component model illustrates the software components that will be used to build the system.
- Components are high level aggregations of smaller software pieces, and provide a 'black box' building block approach to software construction.

https://sparxsystems.com/resources/tutorials/uml/component-model.html

The Component Model
Component Notation

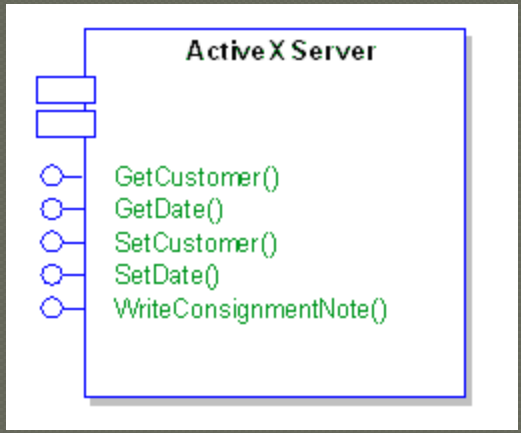
- A component may be something like an ActiveX control - either a user interface control or a business rules server.
- The component diagram shows the relationship between software components, their dependencies, communication, location and other conditions.
- Components are drawn as the following diagram illustrates:



https://sparxsystems.com/resources/tutorials/uml/component-model.html

The Component Diagram Interfaces

- Components may also expose interfaces
  - These are the visible entry points or services that a component is advertising and making available to other software components and classes
- Typically a component is made up of many internal classes and packages of classes
  - It may be assembled from a collection of smaller components



https://sparxsystems.com/resources/tutorials/uml/component-model.html

UML Components

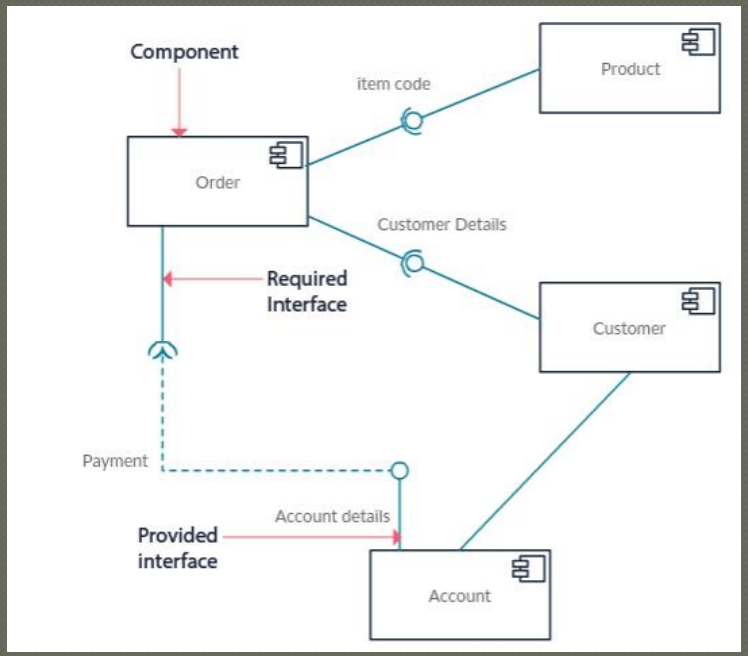
- The graphical representation of a component is a rectangle with tabs:

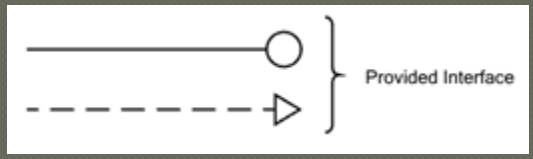
or

The Unified Modeling Language User Guide, by G. Booch, J. Rambaugh & I. Jacobson

Component Diagram

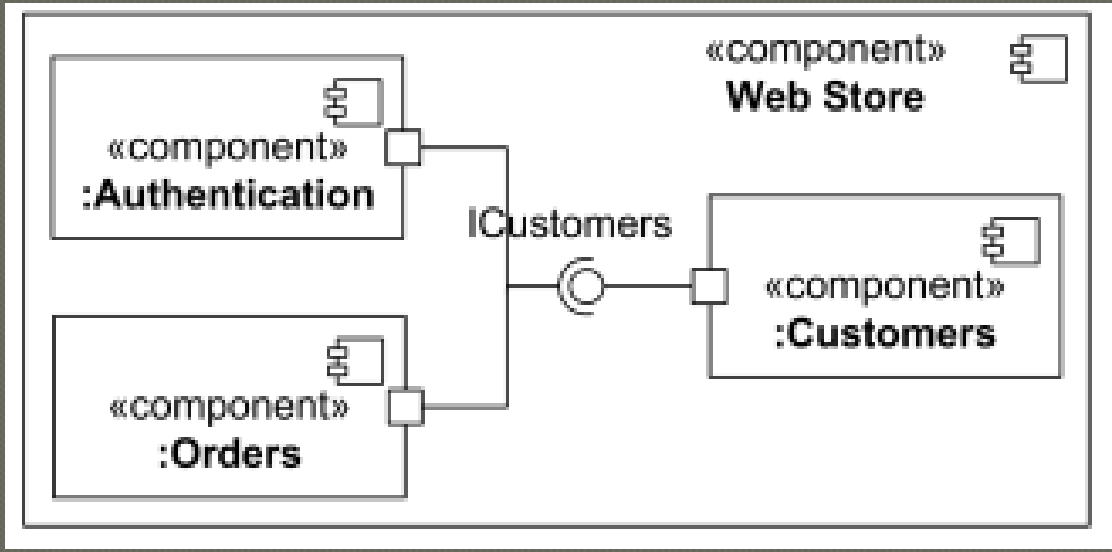
- Components communicate with each other using **interfaces**
- The interfaces are linked using connectors





The Unified Modeling Language User Guide, by G. Booch, J. Rambaugh & I. Jacobson
https://creately.com/blog/diagrams/uml-diagram-types-examples/

Sub-Components



https://www.uml-diagrams.org/component-diagrams-reference.html

The Component Diagram

- Requirements
  - Components may have requirements attached to indicate their contractual obligations - that is, what service they will provide in the model.
  - Requirements help document the functional behaviour of software elements.
- Constraints
  - Components may have constraints attached which indicate the environment in which they operate.
    - Pre-conditions specify what must be true before a component can perform some function;
    - post-conditions indicate what will be true after a component has done some work and
    - Invariants specify what must remain true for the duration of the components lifetime.

https://sparxsystems.com/resources/tutorials/uml/component-model.html

The Component Diagram

- Scenarios
  - Scenarios are textual/procedural descriptions of an object's actions over time and describe the way in which a component works.
  - Multiple scenarios may be created to describe the basic path (a perfect run through) as well as exceptions, errors and other conditions.
- Traceability
  - You may indicate traceability through realisation links.
  - A component may implement another model element (eg. a use case) or a component may be implemented by another element (eg. a package of classes).
  - By providing realisation links to and from components you can map the dependencies amongst model elements and the traceability from the initial requirements to the final implementation.

https://sparxsystems.com/resources/tutorials/uml/component-model.html

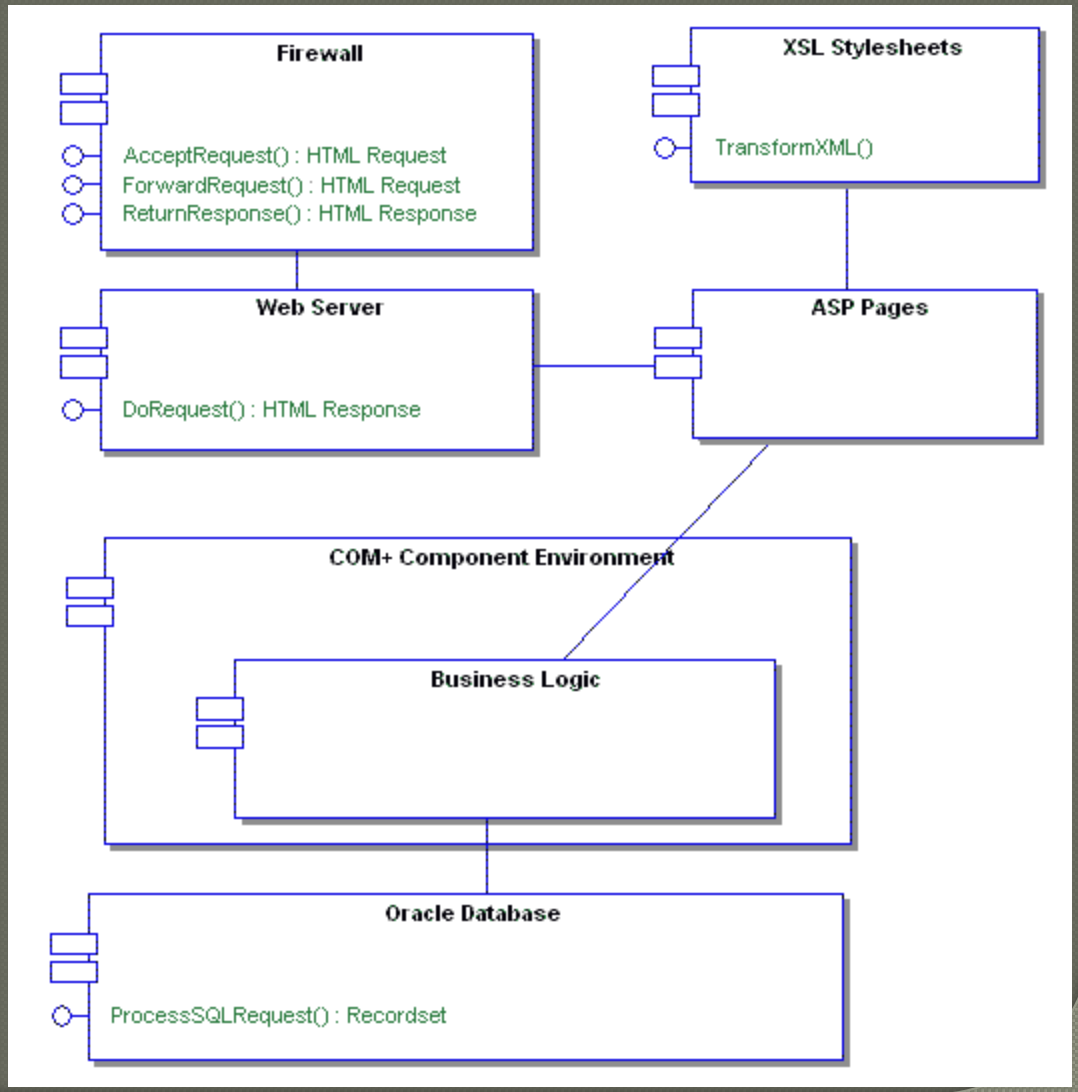
Component Diagram Example

- The following example shows how components may be linked to provide a conceptual/logical view of a systems construction.
- This example is concerned with the server and security elements of an on-line book store.
- It includes such elements as the web server, firewall, ASP pages & etc.

https://sparxsystems.com/resources/tutorials/uml/component-model.html

Component Diagram Example Server Components

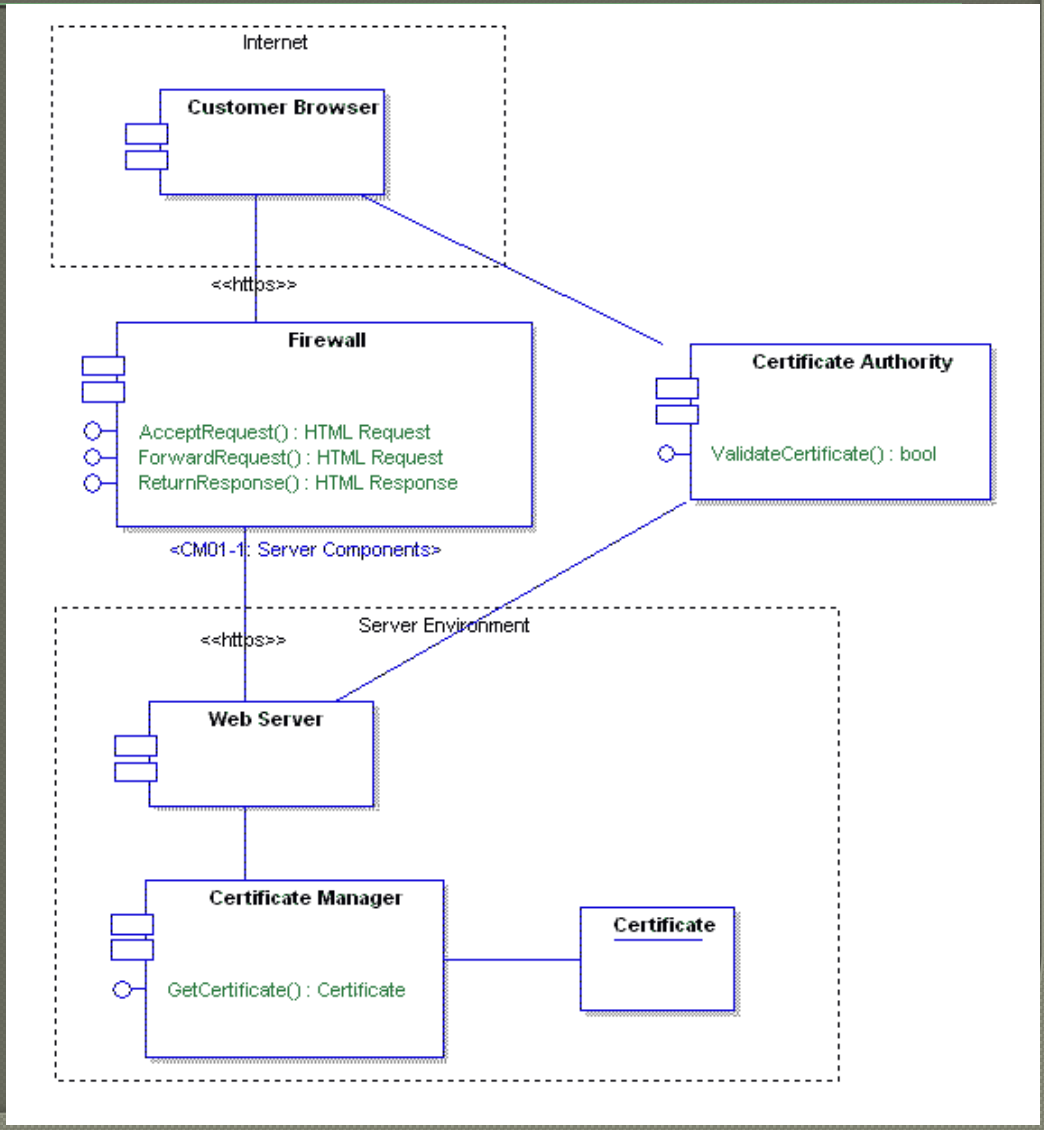
- This diagram illustrates the layout of the main server side components that will require building for an on-line book store.
- These components are a mixture of custom built and purchased items which will be assembled to provide the required functionality.



https://sparxsystems.com/resources/tutorials/uml/component-model.html

Component Diagram Example Security Components

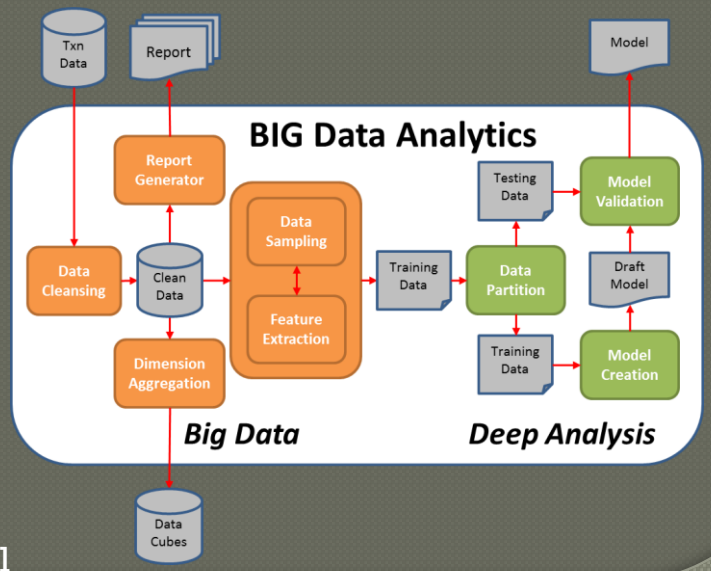
- The security components diagram shows how security software such as the Certificate Authority, Browser, Web server and other model elements work together to assure security provisions in the proposed system.



https://sparxsystems.com/resources/tutorials/uml/component-model.html

Homework #1

- Use Enterprise Architect to transform the first architecture that we looked at, into a UML component diagram
  - You will need to name the unnamed component
  - Due, 11PM 2/19; Do not wait until the last minute!!



http://horicky.blogspot.com/2012/08/big-data-analytics.html

Enterprise Architecture

- We will spend some time talking about EA on Monday, when we begin our UML discussion
- Instructions for access is posted to Moodle

- EA is installed on the 2324 PFT, 2326 PFT and 2317 PFT lab machines
- Students may enter 2326 PFT using the keypad at the back door

- Lab hours is posted to Moodle, but check lab doors for updated information

Lab Guidelines

- Be mindful of others in shared lab spaces
  - Occasional disruptions are unavoidable, but try to minimize disruptive activities and respond to reasonable complaints
  - If you cannot resolve an issue, contact your faculty advisor or the Systems Manager
- General Guidelines:
  - Keep your voice low
  - Take phone calls outside when possible
  - Keep in-lab meetings short and focused
  - TA's should usually meet students in room 2341 (not in the research lab)
- Lab Cleanliness:
  - Immediately discard food trash outside the lab after eating
  - Discard other trash regularly and organize occasionally
  - Clean spills using towels and water from the restrooms

Lab Guidelines

- Emergency Contacts:
  - For a medical or police emergency, call 911
    - You can call from any cell phone even without unlocking the phone.
    - If the phone has no service, go near a window.
  - For a maintenance emergency that threatens people, equipment or facilities, call Dr. Trammel at 985-232-5236
    - If there is no answer, call 225-578-2327 (any time of day).
- Systems Manager Contact Info:
  - Report non-emergency maintenance problems, abandoned desks or equipment, excessive trash, or other issues directly to the CSC Computer Manager
  - Dr. David Trammell
    - davidst@csc.lsu.edu
    - 985-232-5236

SPARX Systems Enterprise Architect

Thanks to SPARX Systems for LSU student use of Enterprise Architect for academic purposes

- **Include in all artifacts created with EA**

User Interface Guide
The EA Workspace

- The Enterprise Architect workspace is the interface through which you create and display your models
- The main workspace component is the central **Diagram View**, which is where you
  - create model diagrams,
  - view reports,
  - scroll through lists of model elements,
  - access the internet and
  - even edit and debug source code

https://www.sparxsystems.com/enterprise_architect_user_guide/10/getting_started/usingenterprisearchitect.html

User Interface Guide
The Main Menu

- The Main Menu provides access to
  - high-level functions related to the project life cycle
  - project and system administration functions;
- Throughout Enterprise Architect you can also access functions and operations using context menus

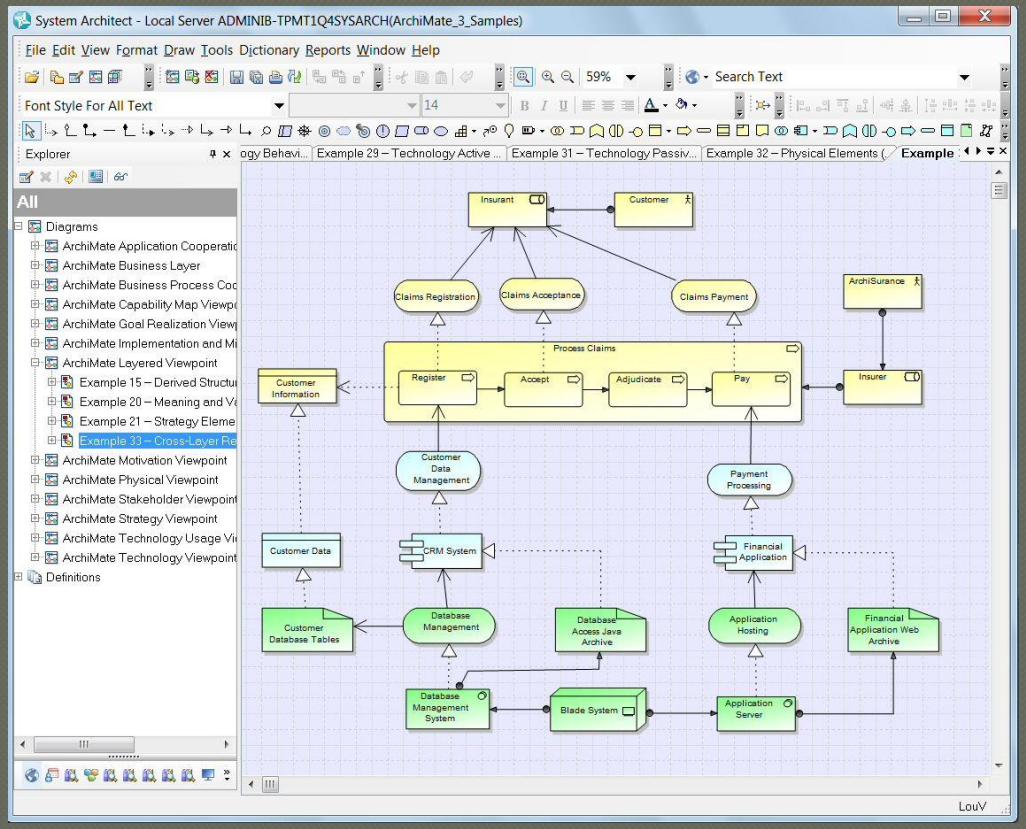
https://www.sparxsystems.com/enterprise_architect_user_guide/10/getting_started/usingenterprisearchitect.html

User Interface Guide Diagram Toolbox

- The Enterprise Architect Diagram Toolbox provides all the components and connectors that you use to create models using whatever diagrams are appropriate
- The Toolbox automatically matches the kind of diagram you have open, and whichever technology is currently active in your model

https://www.sparxsystems.com/enterprise_architect_user_guide/10/getting_started/usingenterprisearchitect.html

Component Diagram in EA

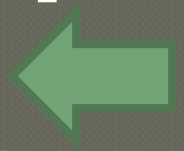


- Sparx Systems provides many tutorials and videos to help you get started

https://www.pinterest.com/pin/490610953150539728/

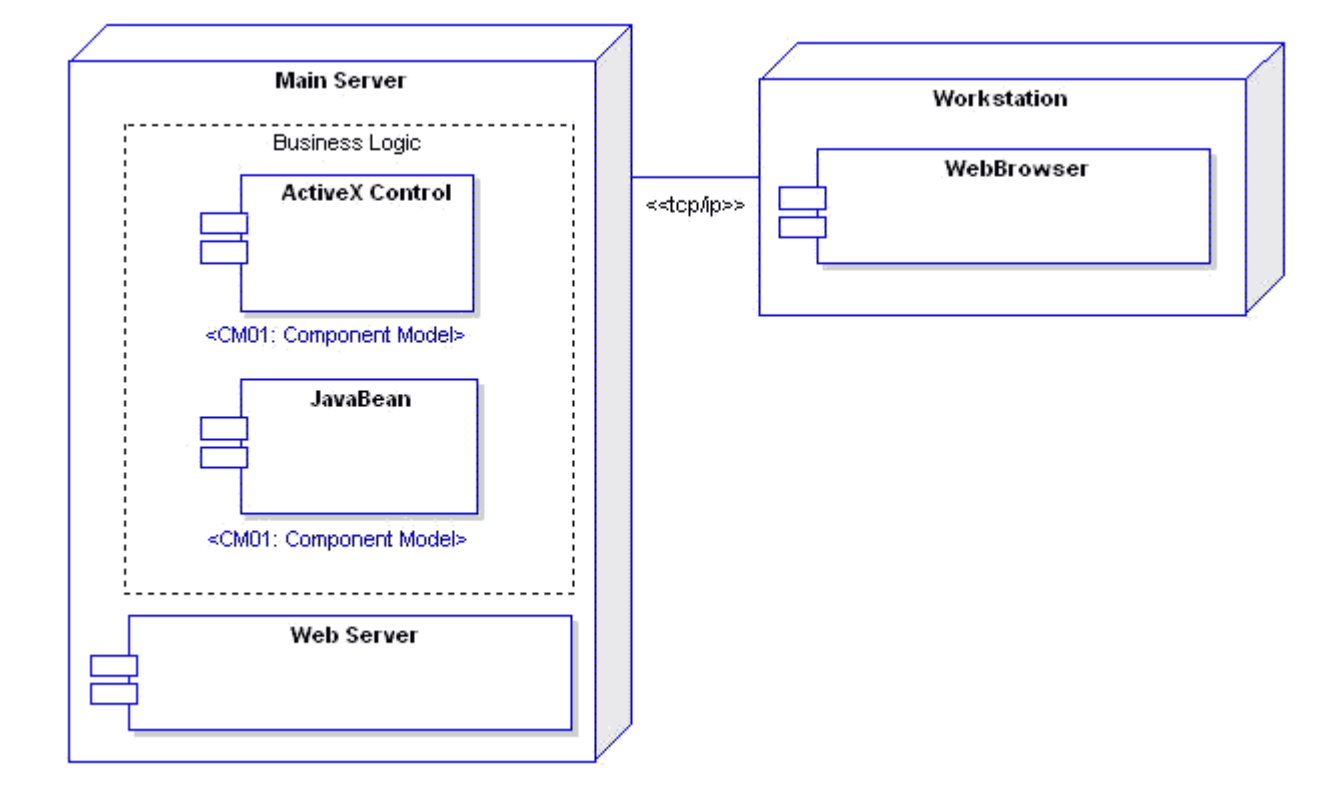
Architectural Modeling

- Component Diagrams
  - Structure and connections of components
- **Deployment Diagrams**
  - Deployment of artifacts to nodes



The Component Diagram: Components and Nodes

- A deployment diagram illustrates the physical deployment of the system into a production (or test) environment.
- It shows where components will be located, on what servers, machines or hardware. It may illustrate network links, LAN bandwidth & etc.



https://sparxsystems.com/resources/tutorials/uml/component-model.html