



XIX. Software Architectures

Software Architectures

UML Packages

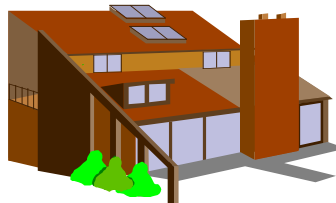
Client-Server vs Peer-to-Peer

Horizontal Layers and Vertical Partitions

3-Tier and 4-Tier Architectures

The Model-View-Controller Architecture

Broker Architectures for Distributed Systems



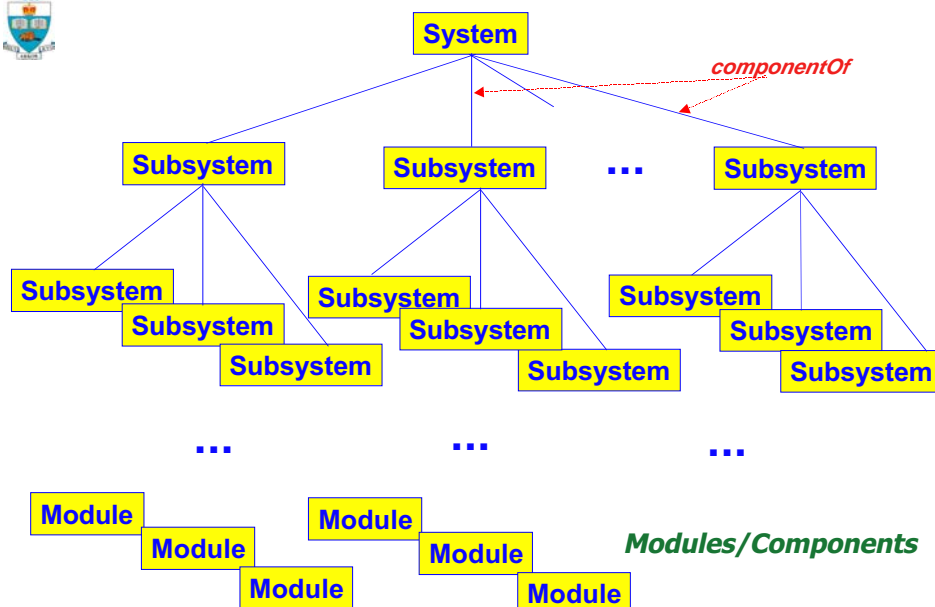
Software Architectures

- A software architecture defines the components of a software system and their inter-dependencies.
- For example, the ***client-server*** architecture consists of ***servers*** that support services, ***clients*** that use services.
- With such an architecture, I/O is placed on clients, running on PCs and workstations; data storage is assigned to a server, implemented in terms of a DBMS (e.g., DB2) and placed on a mainframe or mini. Consistency checking is located with the server, applications run on clients.
- ***Thick servers*** offer a lot of functionality, ***thin*** ones little.
- ***Thick clients*** have their own services, ***thin*** ones get almost everything from servers.



Subsystems

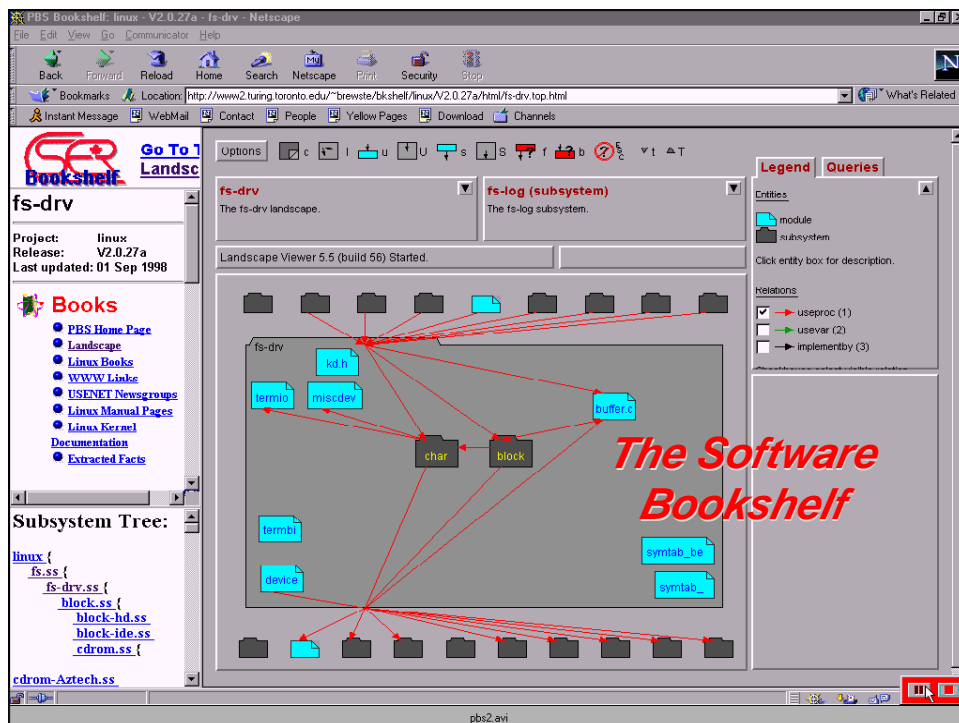
- A software **subsystem** is a component of a system or of another subsystem.
- **Modules** or **components** are atomic subsystems
- It's useful to subdivide software into subsystems
 - ✓ For better-managed software development;
 - ✓ For improved reuse (through components);
 - ✓ For improved portability (platform-specific code isolated to particular subsystems.)
 - ✓ For easier maintenance.
- Each subsystem has a well-defined interface with respect to the rest of the system.





Components and Connectors

- The architecture shown in the previous slide is one example of a software architecture where nodes represent subsystems/modules and the connectors represent component relationships.
- There are many others kinds of connectors that can be used, such as:
 - ✓ **Uses** -- one component uses data defined in another component;
 - ✓ **Calls** -- one component calls methods defined in another component;
 - ✓ **I/O** -- the output of one component is fed as input to another;





Architectural Styles

- It is useful to classify software architectures into classes of **architectural styles**. For example, the client-server architecture discussed earlier is an architectural style.
- There are many architectural styles, e.g., pipes and filters, object-orientation, event-based, layered, repository-based, client-server, three-tier,...others...
- We discuss here some architectures that relate to object-oriented information systems.

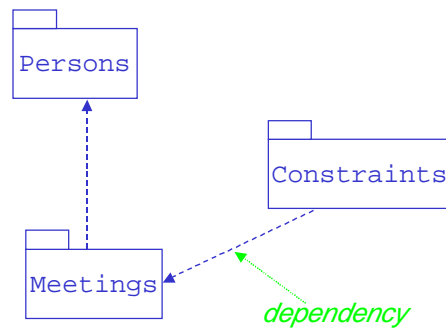


Packages

- A package in UML is a grouping of elements which:
 - ✓ May be packages (e.g., subsystems or modules);
 - ✓ May be classes.
- Each element of a software architecture (subsystem, module or class) is owned by a single package.
- There are many criteria for decomposing a software system into packages:
 - ✓ Ownership -- who is responsible for what;
 - ✓ Application -- e.g., a university dept model may be partitioned into staff, courses, programmes,...
 - ✓ Clusters of classes used together, e.g., course, course description, instructor, student,...



A Package Diagram



- A **dependency** means that if you change a class in one package (Meetings), you **may** have to change something in the other (Constraints).
- The concept is similar to compilation dependencies.
- It's desirable to minimize dependency cycles, if at all possible.

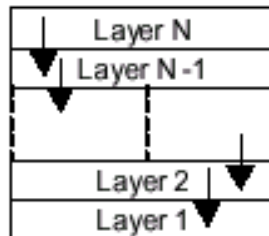


Decomposition into Subsystems

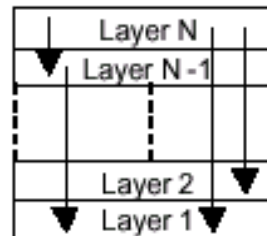
- A software system may be decomposed into **horizontal layers**, and/or **vertical partitions**.
- For a horizontal layer decomposition, each layer corresponds to one or more subsystems, and each layer uses services provided by the layers below it.
- Layered architectures have two forms:
 - ✓ **closed architecture** - each layer only uses services of the layer immediatebelow;
 - ✓ **open architecture** - a layer can use services from any lower layer.



Closed vs Open Layered Architecture



*Closed architecture—
messages may only be
sent to the adjacent
lower layer.*



*Open architecture—
messages can be sent
to any lower layer.*



Closed vs Open Layered Architectures

- Closed layered architectures -- Minimize dependencies between layers and reduce the impact of a change to the interface of any one layer.
- Open layered architectures:
 - ✓ Lead to more compact code, since the services of all lower layers can be accessed directly without the need for extra program code to pass messages through each intervening layer;
 - ✓ Break the encapsulation of layers, increase dependencies between layers and increase the complexity of changes to the system.



Client Server Architectures

- A client server architecture consists of **service consumers** (clients) and **service providers** (servers). Clients and servers may or may not be running on dedicated machines.
- Information exchange between clients and servers is done through messages.
- Server establishes connection with each client (possibly several), accepts messages from connected clients and responds to each.



Protocols for Communication

- Service requests and responses are accomplished through one of the following standard protocols:
 - ✓ **Remote Procedure Call (RPC)** -- invoke remote procedure, results sent; RPC is widely supported;
 - ✓ **Remote Data Access (RDA)** -- invoked procedure is a database query; supported by DBMS vendors;
 - ✓ **Queued Message Processing** -- requests queued.



Three-Tier Architectures

- Used widely in industry

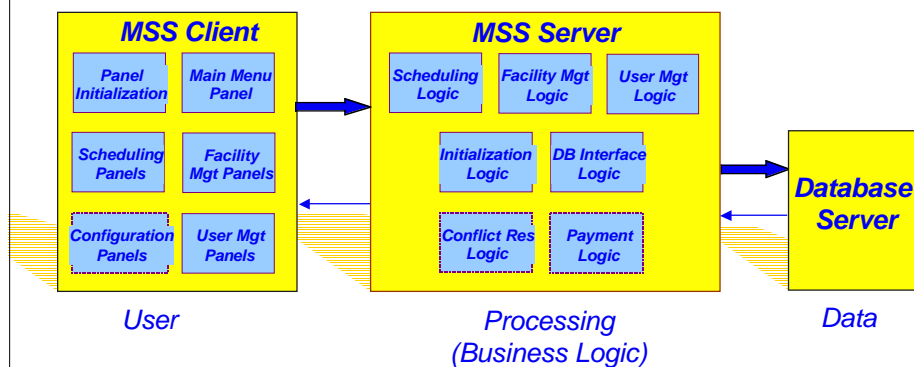


- Application layer may be placed with client (fat client) or the server (fat server), or split between them.
- For example, constraint checking may be done on the server side, other applications are run on the client side.



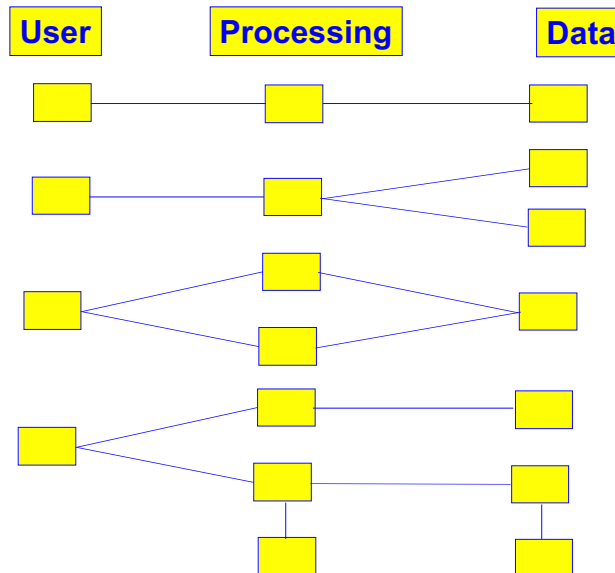
Example Three-Tier Architecture

- An architecture for a meeting scheduling system (MSS)





Many Possible Variations

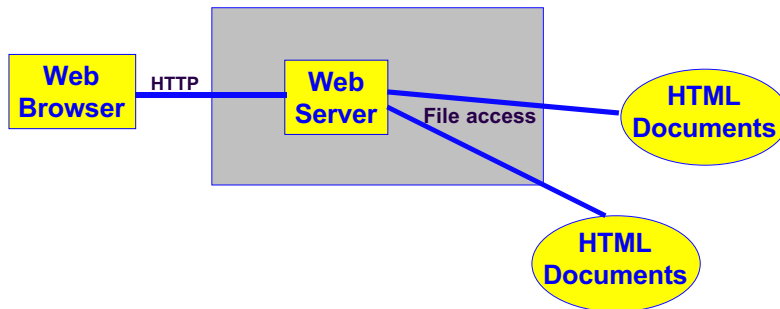


Web-Based Software Architectures

- These are client-server, based on WWW technologies.
- Elements of WWW technologies:
 - ✓ HTTP -- **HyperText Transfer Protocol**, used to transfer hypertext documents over the internet;
 - ✓ HTML -- **HyperText Markup Language**, used to define hypertext documents;
 - ✓ CGI -- **Common Gateway Interface** is a program (e.g., a unix shell script, or a perl script)
 - ✓ CGI scripts are programs that reside on a web server and are executed with a click to retrieve data, generate graphics etc.



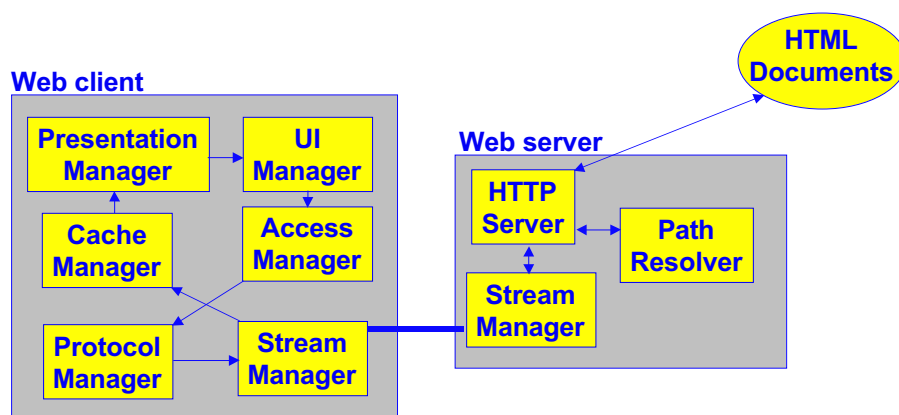
Static HTML-Based Architecture



- This architecture basically retrieves and displays HTML documents that reside on the web server site.



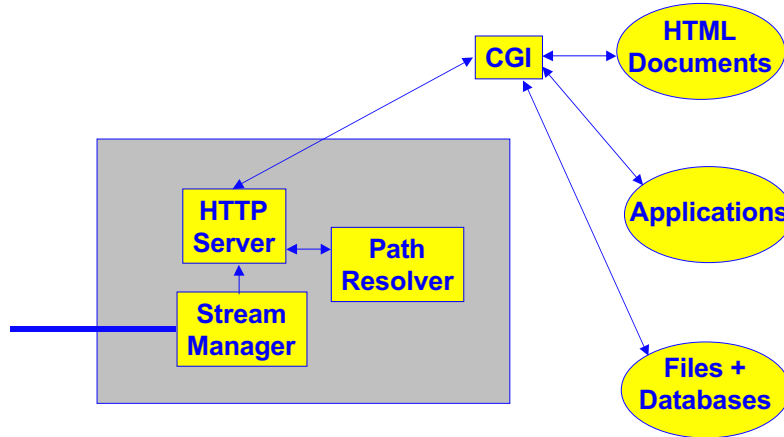
More Detailed Static Architecture



- Arrows indicate data and/or control flow.



Dynamic HTML-Based Architecture

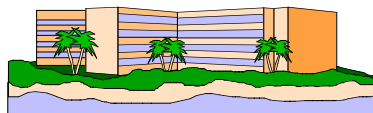


- The CGI gateway serves as demon which dispatches a request, dealt with by an application or a database server.



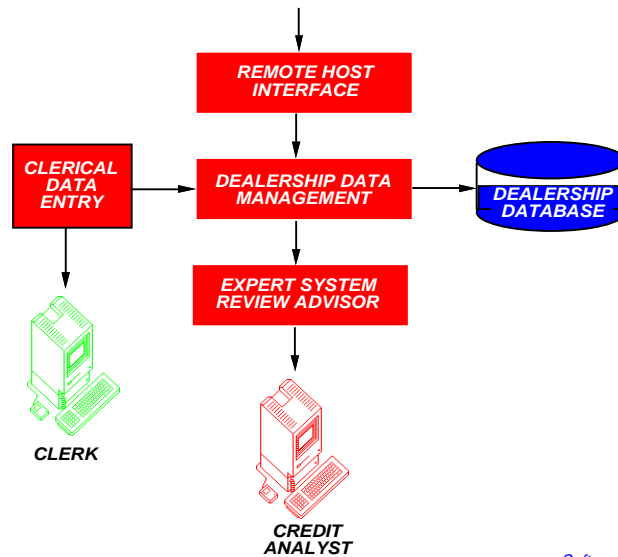
System Architecture Example

- ANALYST, General Motors Dealer Review Advisor.
- Assists credit analysts in 230 GM Acceptance Corporation branch offices analyzing dealership operations in order to decide on credit applications.
- Offers many benefits, including faster reviews, reduced training of personnel and consistency in decision-making.
- Uses an expert system, integrated into a vast, conventional data processing architecture.

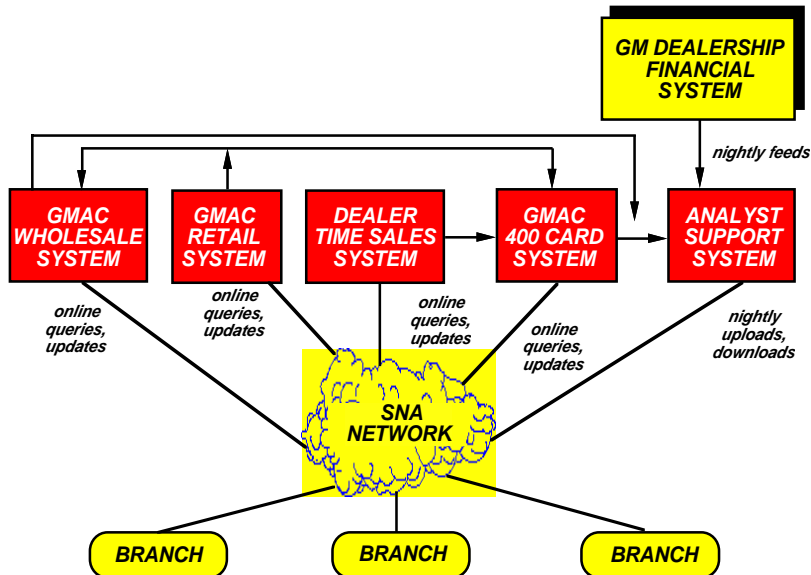




ANALYST Local Architecture

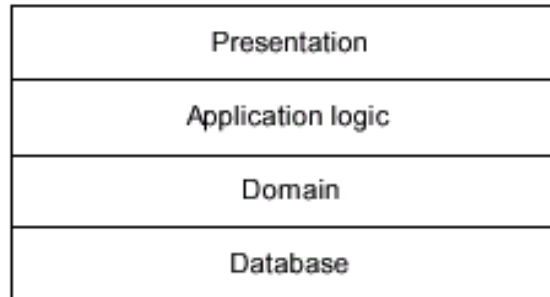


ANALYST Global Architecture





Four-Layer Architectures for Information Systems

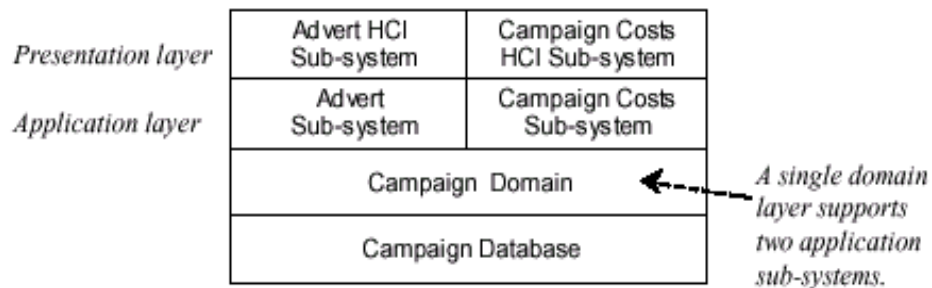


***This is a variation of the 3-tier architecture
we discussed earlier***



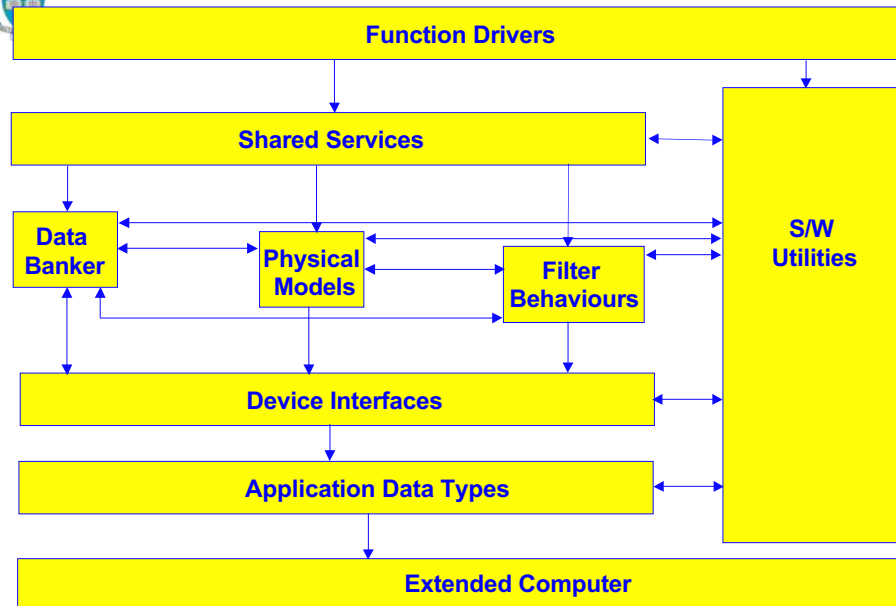
Vertical Partitioning

- Partition each layer into subsystems.
- Partitioning identifies ***weakly coupled*** subsystems within a layer.
- Each partition provides a self-contained service.





Architecture for the A-7E Aircraft

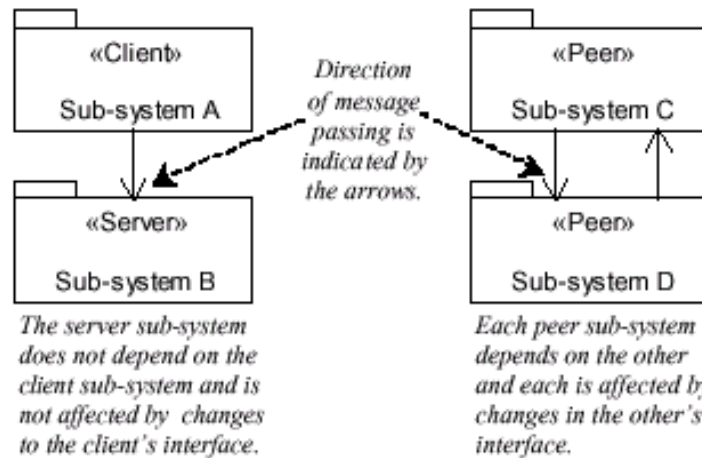


Notes on the A-7E Architecture

- This is a “uses” architecture.
- Modules in different components of the architecture:
 - ✓ Extended computer: virtual mem, parallelism, timer;
 - ✓ Device interfaces: air data, audible signal device, Doppler radar set,...;
 - ✓ Function driving: flight information display, panel, ground test,...;
 - ✓ Application data types: numeric, state transition,...;
 - ✓ Data banker: singular values, complex event,...;
 - ✓ Physical model: aircraft motion, earth characteristics, human factors;
 - ✓ Software utilities: powerup,....



Styles of Communication: Client-Server vs Peer-to-Peer



The Model View Controller (MVC) Architecture

- First used with Smalltalk but has since become widely used as an architecture for object-oriented software.
- Capable of supporting user requirements that are presented through differing interface styles.
- Aids modifiability and portability. In particular, allows one to change the functionality related to one class (e.g., *Courses*), without changing others (e.g., *DegreeProgrammes*.) Also, makes it easier to port a system to different I/O devices.
- This architecture is best suited for software systems where user interfaces play an important role.

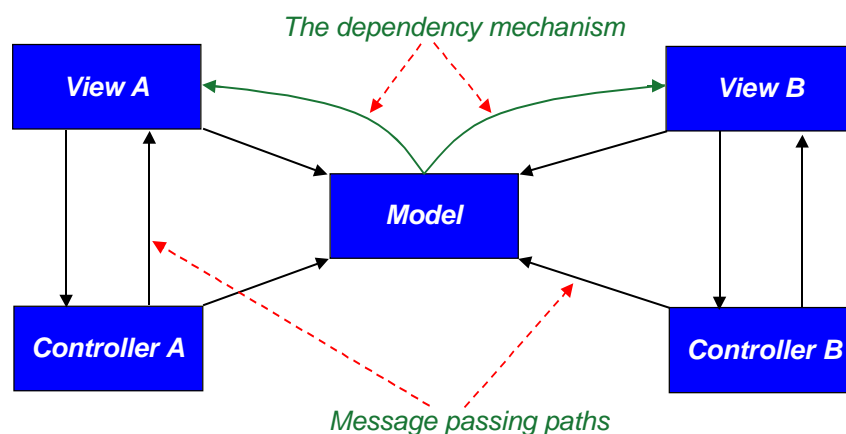


The MVC Architecture

- Consists of subsystems classified into one of:
 - ✓ **Model** -- provides main functionality of application, is aware of dependent view and controller components.
 - ✓ **View** -- supports a particular style and format of presentation (output) of information to the user: Retrieves data from model and updates its presentations when data has been changed in one of the other views; creates its own controller;
 - ✓ **Controller** -- accepts user input in the form of events that trigger execution of operations within the model; these may cause model changes, and may trigger updates in all views to keep them up to date.
- **Dependency Mechanism**: informs each view that the model data has changed, view must update itself.



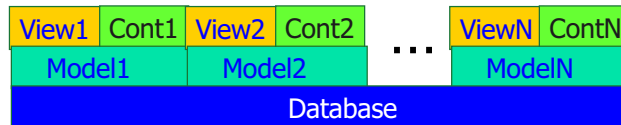
Model View Controller (MVC)



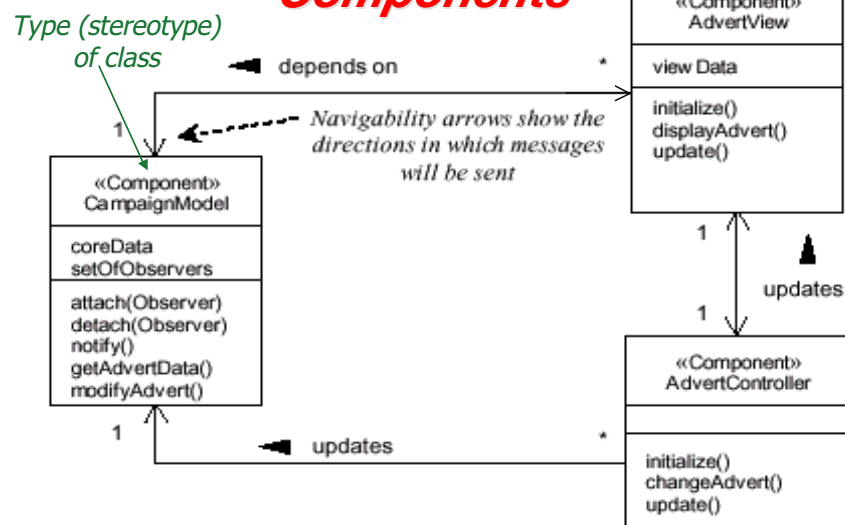


MVC as a Layered Architecture

- You can think of MVC architectures as a refinement of the presentation and application tiers of a 3-tier architecture.



Responsibilities of MVC Components



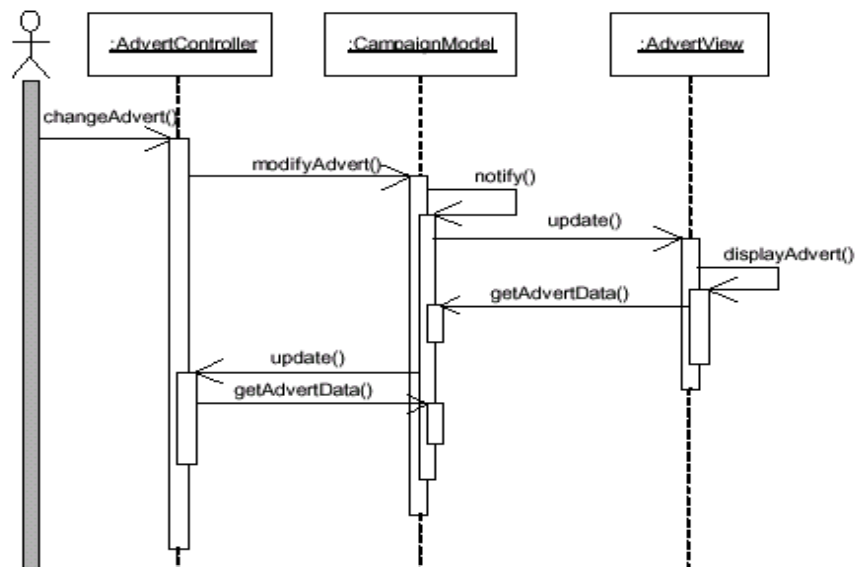


Notes on MVC

- The operation *update()* in *AdvertView* and *AdvertController* trigger these components to request data from *CampaignModel*, which has no knowledge of how this information will be used.
- The *attach()* and *detach()* operations allow views and controllers to be added to/removed from *setOfObservers*.
- The *notify()* operation of a model causes all associated views and controllers to be updated.



MVC Component Interaction



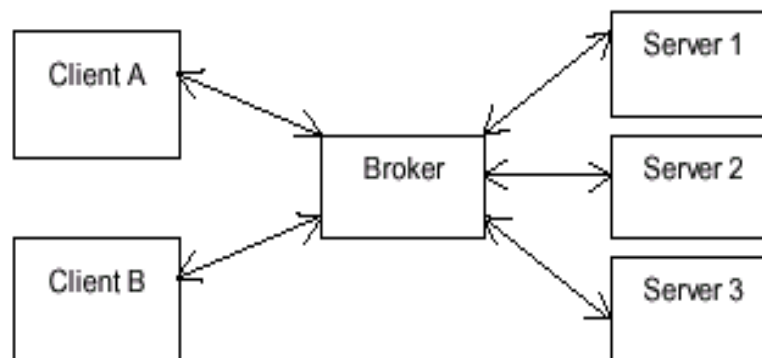


Broker Architectures for Distributed Systems

- A broker increases the flexibility of the system by decoupling the client and server components:
 - ✓ Each client sends its requests to the broker rather than communicating directly with the server component;
 - ✓ The broker then forwards the service request to an appropriate server.
- The client need not know where the server is located (it may be in local or remote computer.)
- Only the broker needs to know the location of the servers that it handles.

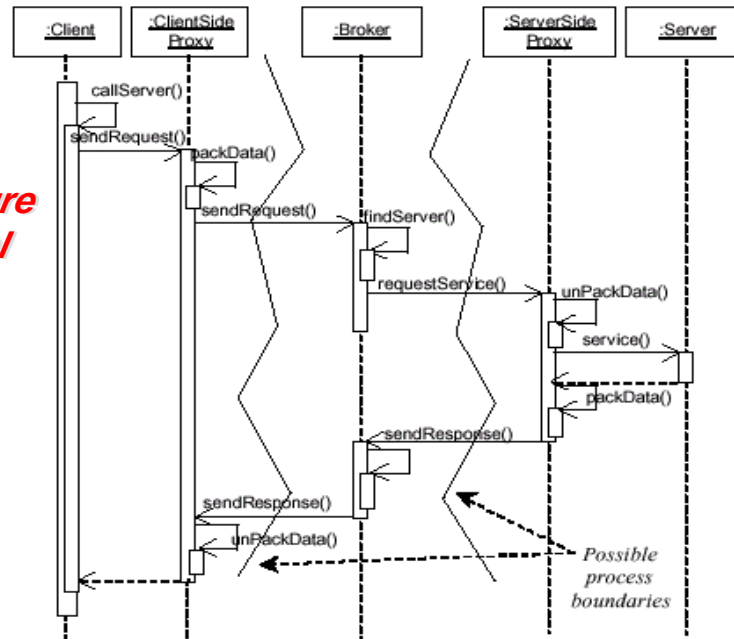


Simplified Broker Architecture





Broker Architecture for Local Server



Proxies

- Some classes (e.g., model classes like *Advert* and *Customer*) are “heavy-weight” in the sense that to create an instance, we need to access a database (...very expensive!).
- We would like to avoid creating instances of heavy-weight classes for as long as possible.
- A **proxy class** is associated to a heavy-weight class and has the same interface (allowable operations.)
- Proxy objects are created as needed and act like placeholders. When someone tries to operate on one (e.g., access one of its attributes), the corresponding heavy-weight object is created.



Threading and Concurrency

- Each independent flow of control can be modelled as an active object that represents a process or thread that can initiate control activity.
 - ✓ A **process** is a heavyweight flow (known to the operating systems itself) that can execute concurrently with other processes
 - ✓ A **thread** is a lightweight flow that can execute concurrently with other threads within the same process.
- Dynamic design identifies concurrent system parts:
 - ✓ Sequence diagrams imply sequential threads;
 - ✓ State/activity diagrams model concurrent execution.



Summary

- Architectural **software** design focuses on the main components of a software system and how they inter-relate.
- Architectural software design is an important phase of the software development process, and can -- literally -- make or break a development project.





Additional Readings

- [Booch99] Booch, G. Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide*. Chapter 22. Addison-Wesley.
- [Rumbaugh91] Rumbaugh, J et al. *Object-Oriented Modeling and Design*. Chapter 9, Prentice-Hall.