XVIII. Software Architectures

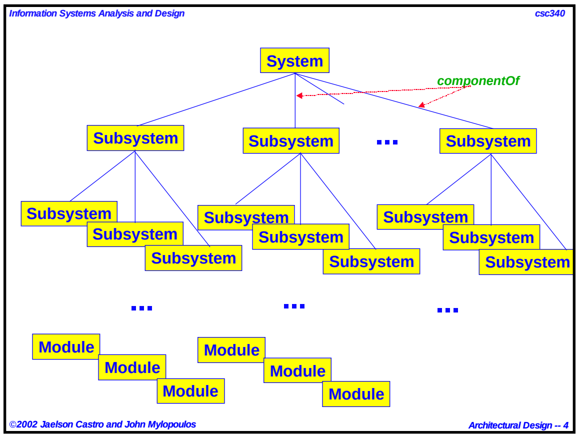
Software Architectures
Subsystems, Modules and Connectors
Pipes and Filters, Object-Oriented, Layered,
Event-Driven, Repository-Based Architectures
Client Server Architectures
Web-Based Software Architectures
Examples

Software Architectures

- A software architecture defines the components of a software system and how they use each other's functionality and data.
- An example of a software architecture is the **client-server** architecture. Such an architecture consists of **servers**, which support some kind of service, and **clients** which request and use server services.
- With a client-server architecture, an information system need not be seen as a monolithic program.
- Instead, input/output functions can be placed on clients, running on PCs and workstations;
- Data storage is treated as a server, implemented in terms of a DBMS such as DB2, Ingres, Sybase or Oracle and placed on a mainframe or mini
- Consistency checking is located with the server
- Applications are located with clients
- **Thick servers** offer a lot of functionality, **thin** ones little
- **Thick clients** have their own services, **thin** ones get almost everything from servers

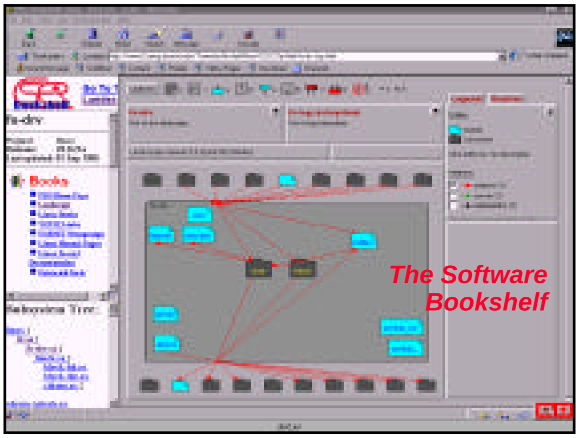
Subsystems

- A ***subsystem*** is a component of a system or of another subsystem.
- ***Modules*** are atomic subsystems (which are not further decomposed into subsystems.)
- It's useful to subdivide a software system into subsystems
  - ✓For better-managed software development;
  - ✓For improved reuse potential (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 the nodes represent subsystems or modules and the connectors between them describe "componentOf" 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;



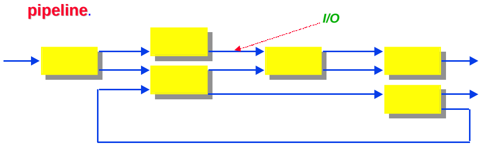
The Software Bookshelf

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.
- The styles we'll discuss below are as follows:
  - ✓ Pipes and filters;
  - ✓ Object-Orientation;
  - ✓ Event-Based
  - ✓ Layered;
  - ✓ Repository-Based;
  - ✓ Client-Server;
  - ✓ Three-Tier;
  - ✓ ...more...

Pipes and Filters

- Each component has **inputs** and **outputs**. A component reads streams of data on its inputs and produces data on its outputs, continuously as data are coming in.
- Components compute by performing local transformations on their inputs to produce their outputs and are termed **filters**. The connectors of components transmit the outputs of one component to the inputs of another and are termed **pipes**.
- Unix supports a linear pipe and filter architecture called **pipeline**.



Pipes and Filters: Strengths and Weaknesses

Strengths

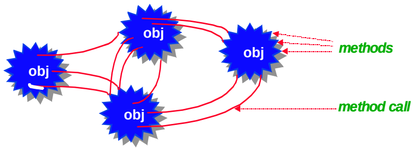
- Makes it easy to understand overall function of the system as a composition of filter functions
- Encourages reuse of filters
- Facilitates maintenance
- Facilitates deadlock and throughput analysis

Weaknesses

- Often leads to batch-type processing
- Not good for interactive applications where you often want to do incremental computations, e.g., incremental display updates
- Can't coordinate stream inputs
- Data transmission critical for system performance

Data Abstraction and Object-Orientation

- Data structures and their associated operations are **encapsulated** in an **abstract data type** (ADT) or **object**. The components of a system are instances of an ADT and they interact through procedure (or **method**) calls
- An object is responsible for preserving the integrity of its data structures and also these data structures are hidden from other objects.
- Objects may operate concurrently or not



Data Abstraction: Strengths and Weaknesses

Strengths

- Possible to change implementation of an object without affecting its clients
- Encourages decomposition of a problem into a number of interacting components/agents
- Encourages software reuse

Weaknesses

- For an object to interact with another, it must know its identity (not so for pipe&filter architectures)
- When the methods of an object change, so must all other objects that use this object

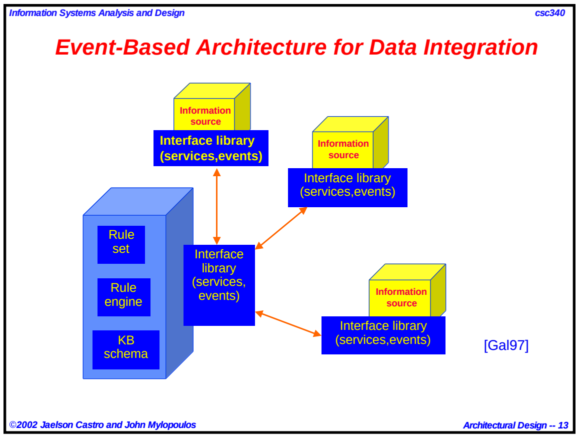
Client-Server Architecture a special case of the Data Abstraction Architecture

Event-Based Architectures

- Instead of invoking a procedure directly, a component can **announce** one or more **events** (such as arrival of data or execution of an operation)

  On <event> **if** <condition> **then** <action>

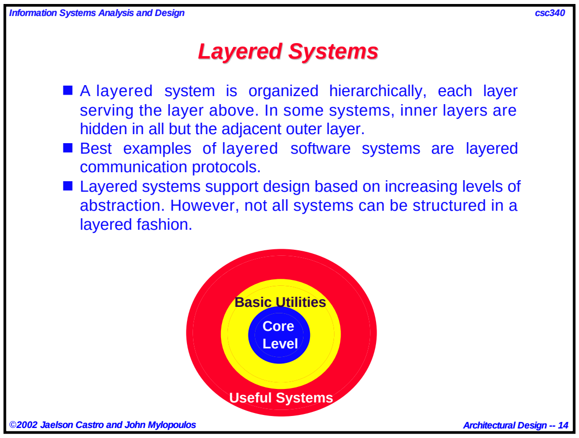
  On arrive(D) **if** D < a or D ≥ b **then** print("out of bounds")
- Such procedures are also called triggers, actors or event-condition-action rules
- An advantage of event-based invocation is that it encourages reuse; a component can be introduced in a system simply by registering it for the events of that system
- A drawback is that sometimes event-based systems become quite unpredictable and hard to control.



Event-Based Architecture for Data Integration

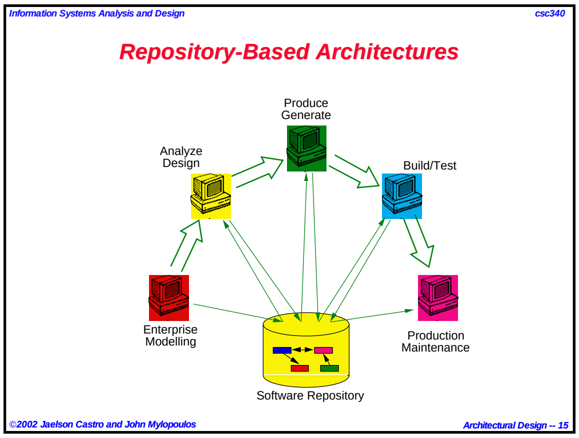
Information source
Interface library (services,events)
Information source
Interface library (services,events)
Rule set
Rule engine
KB schema
Interface library (services, events)
Information source
Interface library (services,events)

[Gal97]

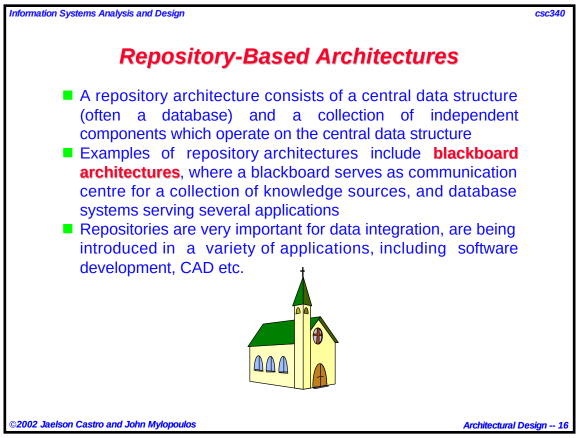


Layered Systems

- A layered system is organized hierarchically, each layer serving the layer above. In some systems, inner layers are hidden in all but the adjacent outer layer.
- Best examples of layered software systems are layered communication protocols.
- Layered systems support design based on increasing levels of abstraction. However, not all systems can be structured in a layered fashion.

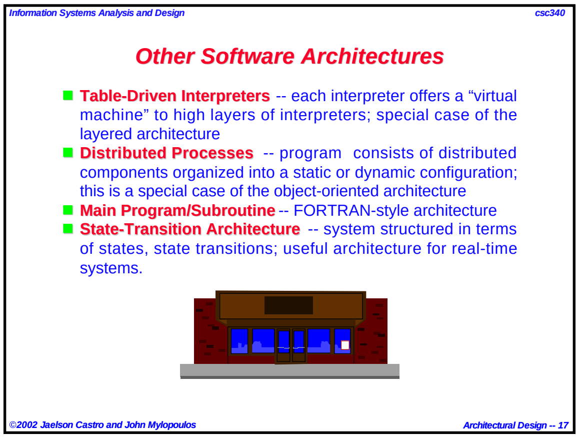


Repository-Based Architectures



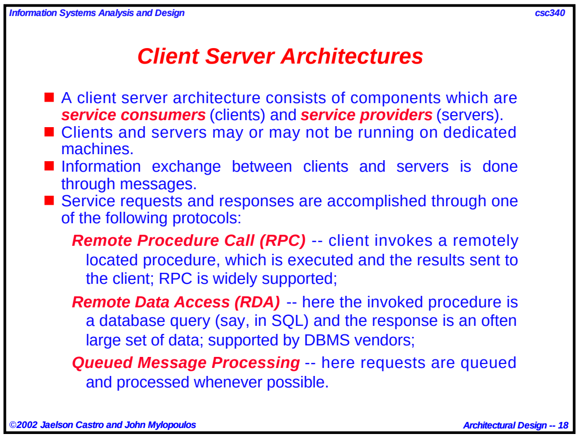
Repository-Based Architectures

- A repository architecture consists of a central data structure (often a database) and a collection of independent components which operate on the central data structure
- Examples of repository architectures include **blackboard architectures**, where a blackboard serves as communication centre for a collection of knowledge sources, and database systems serving several applications
- Repositories are very important for data integration, are being introduced in a variety of applications, including software development, CAD etc.



Other Software Architectures

- **Table-Driven Interpreters** -- each interpreter offers a “virtual machine” to high layers of interpreters; special case of the layered architecture
- **Distributed Processes** -- program consists of distributed components organized into a static or dynamic configuration; this is a special case of the object-oriented architecture
- **Main Program/Subroutine** -- FORTRAN-style architecture
- **State-Transition Architecture** -- system structured in terms of states, state transitions; useful architecture for real-time systems.



Client Server Architectures

- A client server architecture consists of components which are ***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.
- Service requests and responses are accomplished through one of the following protocols:

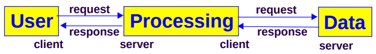
  Remote Procedure Call (RPC) -- client invokes a remotely located procedure, which is executed and the results sent to the client; RPC is widely supported;

  Remote Data Access (RDA) -- here the invoked procedure is a database query (say, in SQL) and the response is an often large set of data; supported by DBMS vendors;

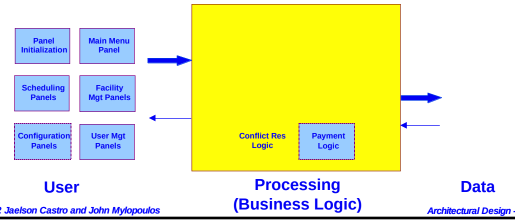
  Queued Message Processing -- here requests are queued and processed whenever possible.

Three-Tier Client Server Architectures

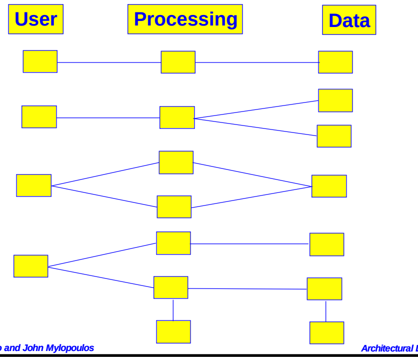
- Used widely in industry



E.g., architecture for a meeting scheduling system (MSS)



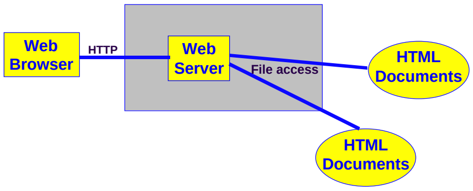
Many Possible Variations



Web-Based Software Architectures

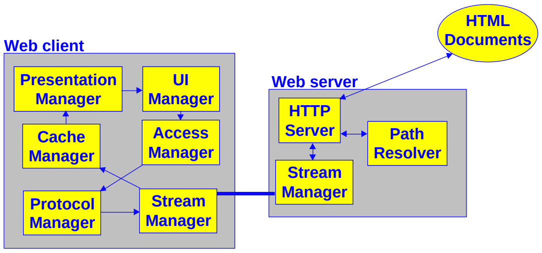
- These are client server too, but they are based on WWW technologies.
- Such architectures are becoming very popular because of static HTML-based applications, but also dynamic ones, such as those that involve Ecommerce.
  - HTTP -- HyperText transfer Protocol, used to transfer hypertext documents over 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



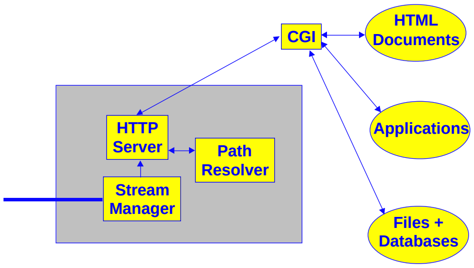
- This architecture basically retrieves and displays HTML documents that reside on the web server site.

More Detailed Static Architecture



- Arrows indicate data and/or control flow.

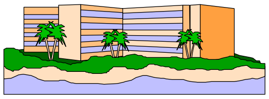
Dynamic HTML-Based Architecture



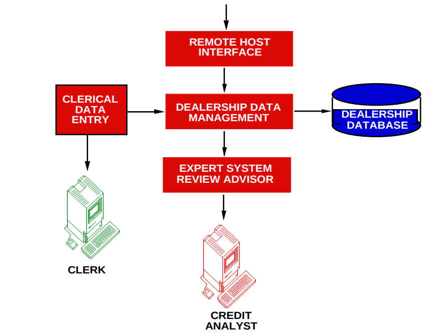
- The CGI gateway serves as deamon which dispatches a request, dealt with by an application or a database server.

Document Interchange Example: ANALYST (General Motors Dealer Review Advisor)

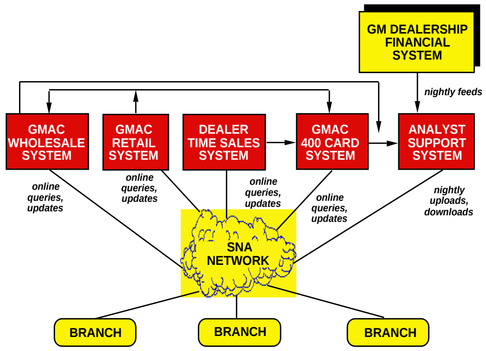
- Assists credit analysts in 230 GM Acceptance Corporation branch offices analyzing dealership operations in order to decide on credit applications.
- Offers many benefits, including faster reviews, reduced training of personnel and consistency in decision-making.
- Uses an expert system, integrated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



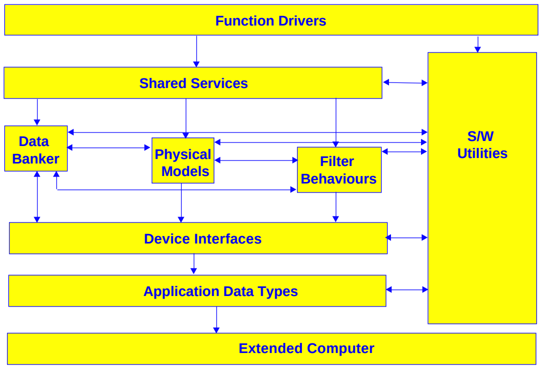
ANALYST Architecture at a GMAC Branch



ANALYST Architecture: Global GM Perspective



A Layered Architecture for the A-7E Aircraft



Notes on the A-7E Architecture

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

Summary

- Architectural system design addresses issues of hardware and software location, interconnectivity; also distribution of I/O processes, data stores and application processes
- Architectural system design is the most important step of system design and can, literally, make or break an information system development project

Additional Reading

[Architectures] http://www.pithecanthropus.com/~awg/browsing_library.html

[Bass98] Bass, L., Clements, P., Katzman, R., *Software Architecture in Practice*, Addison Wesley, 1998.

[Gal97] Gal, A. and Mylopoulos, J. "The CoopWARE Demo", http://www.cs.toronto.edu/~coopware, 1997.

[Garlan93] Garlan D. and Shaw, M., "An Introduction to Software Architectures", in *Advances in Software Engineering and Knowledge Engineering*, volume I, World Scientific, 1993.

[Umar97] Umar, A., *Application Reengineering*, Prentice Hall, 1997.

©2002 Jaelson Castro and John Mylopoulos