## The Component-and-Connector viewtype styles

- Elements, Relations and Properties for the Component-and-Connector viewtype
- Pipe-and-Filter Style
- Data-Centered Style
- Publish-Subscribe Style
- Client-Server Style
- Peer-to-Peer Style
- Communicating-Processes Style



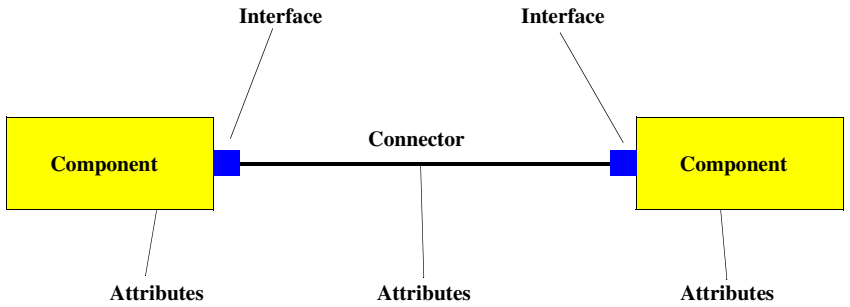
## Elements, Relations and Properties

- An architectural of a system includes
  - Components - elements
    where the computation is performed:
    objects, filters, databases
  - Connectors - relations
    performs the interaction among components:
    procedure calls, pipes, event broadcast
  - Attributes properties
    gives information for analysis and design:
    signatures, pre/post conditions
- An architectural style defines a family of architectures constrained by
  - Component/connector definitions
  - Topology rules
  - Semantic constraints



## Architectural Elements

Ref: Stephen H. Kaisler: Software Paradigms, Wiley, 2005.






## Components

- Basic building blocks
  - active computational entities in a system.
  - communicates with its environment through one or more ports or interfaces
- A port may be
  - a user interface,
  - a shared variable,
  - a procedure name that can be called by another component,
  - a set of events that can be emitted,
  - or some other mechanism

## Attributes

- Attributes of the component specify information for analysis and development of the software:
  - protocols
  - real-time constraints
  - behavioral constraints



## Connectors

- Connectors define the interaction between components.
- A connector
  - links the ports of two or more components
  - has a Role attached to it as an attribute that describes the behavior of the attached components, e.g. source and receiver in an unidirectional communication.
  - can have other attributes:
    - rates
    - capacities
    - latencies



## Interfaces

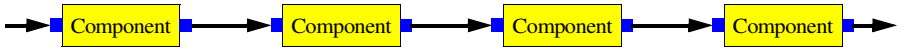
- A component
  - defines an interface through which a component links one component with another.
  - might have multiple interfaces.
- An interface
  - is associated with one and only one component
  - may connect to multiple interfaces in other components
  - may have attributes, e.g.
    - direction of communication
    - buffering capacity
    - protocols accepted



## Configuration

- A configuration (topology) is a connected graph of components and connectors describing architectural structure.
- The configuration must:
  - adhere to the rules for proper connectivity
  - satisfy concurrent and distributed properties
  - adhere to constraints such as design heuristics and style rules.
    e.g. end-to-end connectivity may not be longer than four components.
- Configuration is analyzed for:
  - performance
  - bottlenecks
  - concurrency
  - system reliability
  - security

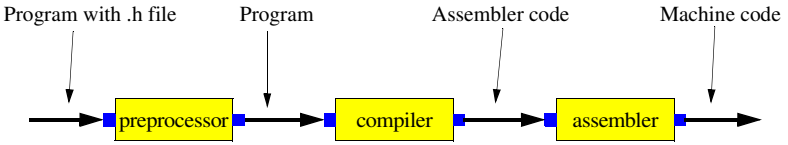
## Pipe-and-Filter Style




- Data flows from one process to the next, through a series of processes that do operations on the data.
- Each component is isolated from performance of upstream component,
  - Different speeds is taken care of by buffers.
    might lead to overflow error
- Components may be memoryless (no internal state preserved).
  Only memoryless components can be replaced at runtime.
- The availability of data controls the computation process.
- Data centered: **what** to compute (not how).
- Data flows from one process to the next, through a series of processes that do operations on the data.



## Pipe-and-Filter Example — A Compiler




- In UNIX this will be implemented as:

```
pp | cc | as defs.h program.c
```

where

- `pp` is the pre-processor
- `cc1` is the first step of the compiler - lexical analysis
- `cc2` is the second step of the compiler
- `as` is the assembler



## Pipes and Filters Systems

- data sets are processed in discrete, separable stages.
- each stage is represented by a component.
- each component operates on the entire data set and passes the results to next stage.
  the component act as a filter.
- connectors serves as links
  act as pipes



## Filters

- Basic activities:
  - enrich input data
    add data, e.g. from a data store or computed values
  - refine input data
    delete or sort data
  - transform input data
    e.g. from one type to another
- Two types of Filter:
  - Active Filter
    drives the data flow
  - Passive Filter
    driven by the dataflow

In a Pipes and Filters Architecture at least one of the filters must be active.
This active filter can be the environment, e.g. user input.

## Filters cont.

- If the input filter is active:
  - A push system
- If the output filter is active:
  - A pull system



## Pipes

- A Pipe
  - transfers data from one data source to one data sink
  - may implement a (bounded/unbounded) buffer
  - may connect
    - two threads of a single process
      - may contain references to shared language objects
    - two processes on a single host computer
      - may contain references to shared operating systems objects (e.g. files)
    - two processes on any host computer
      - distributed system



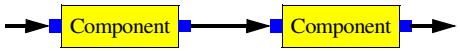
## Pipes and Filters — Properties

- Pure data-driven interaction.
- Each component has a set of inputs and a set of outputs.
- Data transmitted as a whole between Filters.
- Filters are independent programs that can be recombined freely to build family of systems.
- Each transformation step is completed before the next step starts.
- Filters ignore identity of other filters.

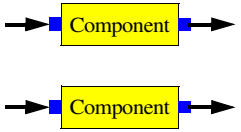


## Pipes and Filters — Composition Rules

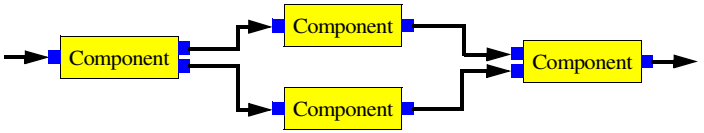
- Sequential Composition
  UNIX: F1 | F2




- Parallel Composition
  UNIX: F1 & F2

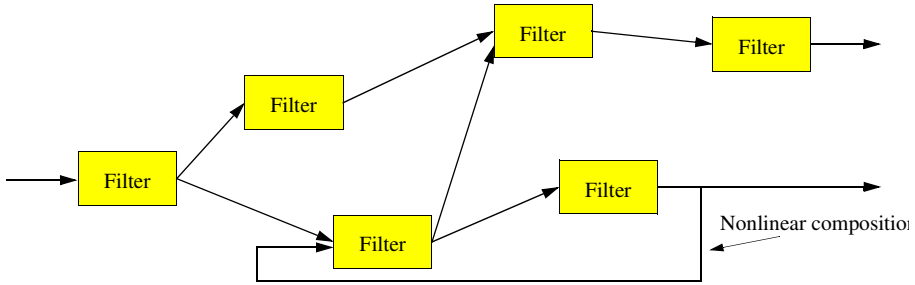



Tee and Join




- Restricted to Linear Composition (not all agree with this!)

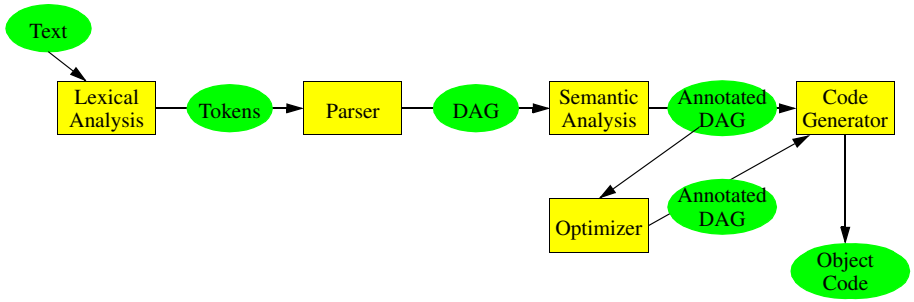
## Pipes and Filters — General Topology






## Pipes and Filters — Example

- A compiler





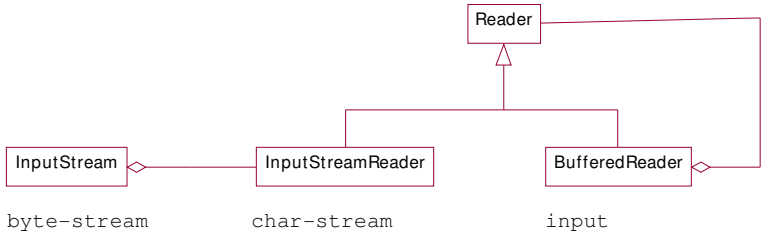

## Pipes and Filters — Other examples

- Radar Tracking System
- Excel spreadsheets
- Shell scripts
- In Java: The Decorator Design Pattern for files.



## Java: The Decorator Design Pattern for files

- Java I/O: Reading a file




```
BufferedReader input = new BufferedReader
                       (
                        new InputStreamReader(System.In)
                        );
```

## What are NOT Pipe-and-Filter Systems

- File-sharing systems
- CVS system
- Word processing



## Pipes and Filters Systems

### Advantages

- Easy to understand system input/output.
- Support for reuse
  Any two filters can be hooked together provided they agree on data transmission.
- Easy maintenance:
  - new filters can be added
  - old filters can be replaced
- Throughput and deadlock analysis possible.
- Natural support for concurrency.
  - Different programs.
  - No shared memory.



## Pipes and Filters Systems

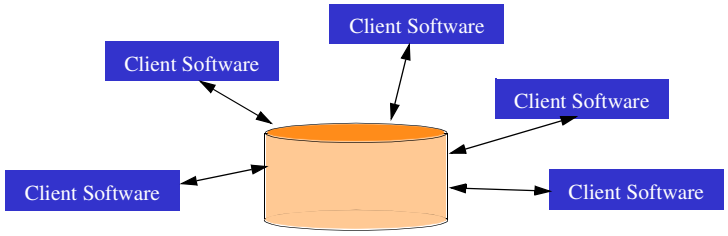
### Disadvantages

- Filter ordering crucial.
- Multiple I/O:
  - difficult to maintain correspondence between two separate but related streams.
  - difference in arrival data rate might impose a performance bottleneck.



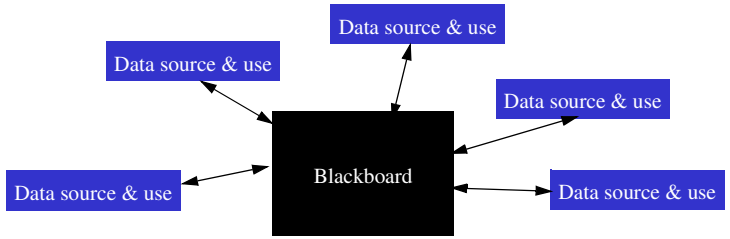
## Data-Centered Style

- Data.centered systems use a central data store to store all problem-related information.
  - Database
  - Blackboard
  - Repository
  - ...



## Blackboard System

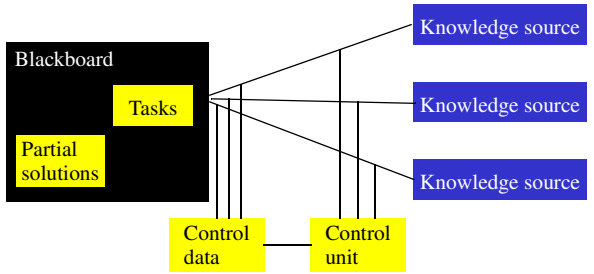
- A Blackboard is a database into which a process can insert, retrieve, or remove data.






## Blackboard System

- Allows multiple processes to communicate by reading and writing information and requests to a global data store.
- Each participating process has expert knowledge in its own field and some knowledge of how to solve the problem. A problem can not be solved by one process alone.
- Processes communicate strictly through the common blackboard whose content is visible to all processes.
- A control unit is responsible for selecting an appropriate process to solve it.






## Blackboard System

### Advantages

- Suitable when there are diverse sources of data.
- Suitable for physically distributed environments.
- Suitable for scheduling and postponement of tasks and decisions.
- Suitable for team problem-solving approaches.
  - Posting of problem subcomponents and partial results.

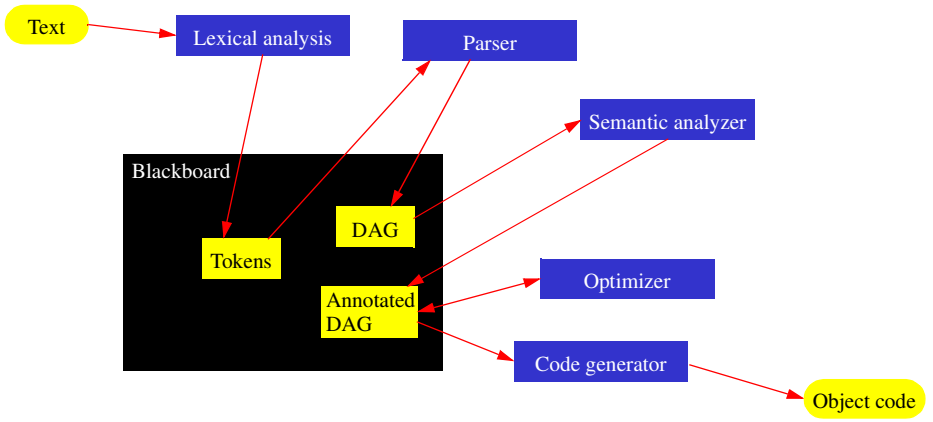
### Disadvantages

- Expensiveness.
- Difficult to determine partition of knowledge
- Control unit can be very complex.



## Blackboard System — Example

### Compiler




- Compare with the Pipe and Filter solution!

## Data Centered Style

**Advantages**

- Data Integrity: Data is entered at one place. No risk for duplicates.
- Design Reuse: Accurate, reliable data are available when needed.
- View-Generation: Alternate views of the data are facilitated by a single source of data.
- Process Flexibility: The data management process is not constrained to application usage or sequence.
- Data Interaction Independent of Application: Data can be accessed by the user through multiple applications.
- Scalability: The database can grow when needed.



## Publish-Subscribe Style

- Elements:
  - Component types: Independent C&C components that publish and/or subscribes to events.
  - Connector types: publish-subscribe
- An arbitrary number
  - call-back
- Example:
  - Observer design pattern

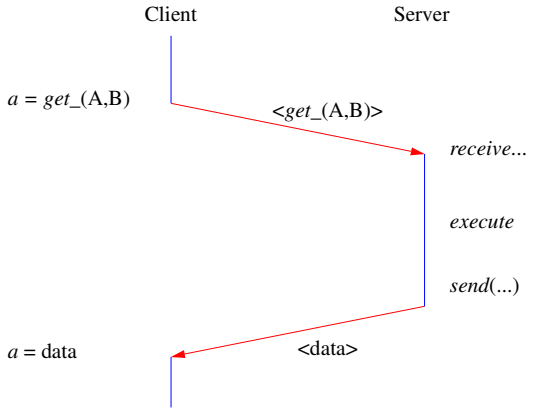


## Client-Server Style

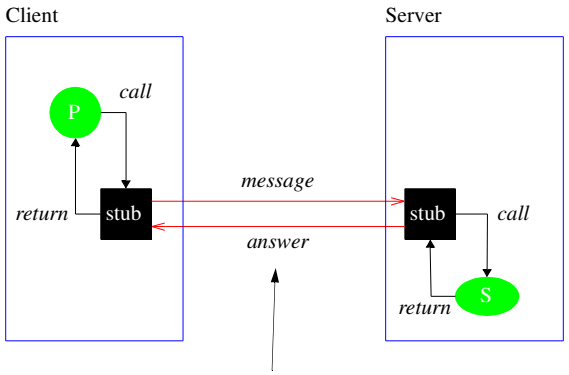
- A Server provides different services.
- A Client uses the services as (remote) subroutines from one ore more Servers.
- The Server might be a Client to another Server.
- A variant of the Main Program and Subroutines
  **but** the clients and servers
  - are implemented as separate entities (processes),
  - might be situated at different processors, and
  - can be implemented in another style
    (most often some variant of object oriented style).
- The mechanism for calling a server is called Remote Procedure Call, RPC.
  - This will require a protocol.



## Remote Procedure Call



## Remote Procedure Call

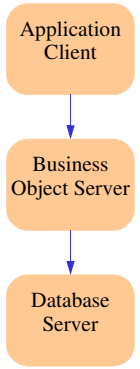



The data formats and special events (e.g. exceptions) must be standardized here



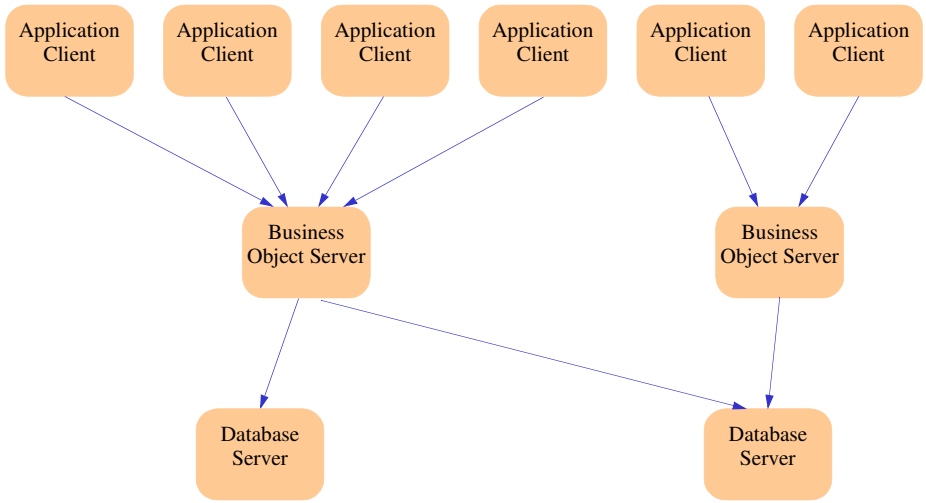
## Client-Server Systems

- 3-tier: Functionality divided into three logical partitions:
  - Application services
  - Business services
  - Data services






## Client-Server Systems (3-tier)






## Client-Server Systems

- Fat Client
  more functionality placed in the client.
- Fat Server
  more functionality placed in the server.
- There can be many versions of this:
  - all business services in the client
  - part of business services in the client
  - only GUI in client.

## Client-Server Systems

### Stateless vs. Stateful Servers

- Information about the Client state can be maintained in the Server.
- Stateful:
  Information about the Client state is maintained by the Server.
  - Might be useful if there is multiple messages sent to the server.
  - The information in messages can be reduced.
  - Security might be easier to implement.
  - But can lead to complex protocols and error handling.
- Stateless:
  Information about the Client state is not maintained by the Server.
  - The normal solution if there is only service requests that are independent of each other and only use a single message each.
  - Simple protocols and error handling must be performed by the client.
  - If it is used for multiple dependant requests the server must rely on the client that is doing right.
  - Security problem. e.g. NFS protocol.



## Client-Server Systems

### Issues

- When designing client-server systems the following must be considered:
  - Authentication:
    Verifying the identity of the client **and** the server.
  - Authorization:
    verifying the rights/privileges of the client.
  - Data security:
    Protect the data stored on the server.
  - Protection:
    Protect the server from malfunctioning clients.
  - Middleware:
    How to connect the clients to the server.

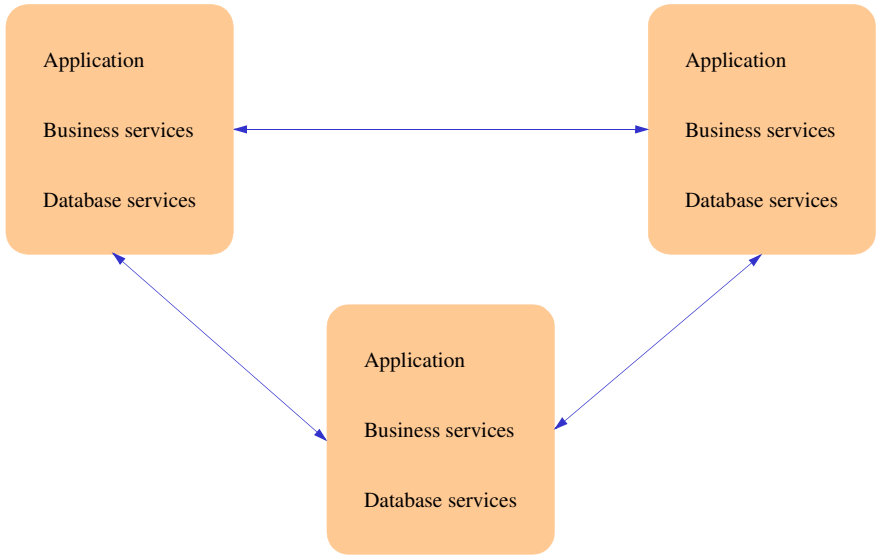


## Peer-to-Peer Style

- Peer-to-Peer Systems, P2P are
  - like Client-Server but all processes can be both clients and servers.
  - more flexible but also more complicated
    there might occur:
    - deadlock
    - starvation



## Peer-to-Peer Systems



## Communicating-Processes Style

- Generally communicating processes
- Distributed systems
- Concurrent operations.
- No common memory
  - Synchronization problems
- All big systems are more or less distributed!



## Distributed Systems

**Advantages**

- Computing power follows physical reality.
- Sharing of data and computing power.
- Modular extension of the system.
  - simple installation of units (step wise)
- Simpler support
- Increased availability:
  Can be used for achieving:
  - Fault Tolerance.
  - Reliability.
  - Performance - processor grid



## Distributed Systems

**Disadvantages**

- Expensive to develop
  - Synchronization problems
- Performs problems
  - Computer Communication delay