An Introduction to Software Architecture

David Garlan & Mary Shaw – 94

UNIVERSITY OF Waterloo

Motivation

Motivation

- An increase in (system) size and complexity
  - structural issues
  - communication (type, protocol)
  - synchronization
  - data access / manipulation
  - deployment
  - performance

Potential Solution

Architectural Principles

- Recognize common patterns
  - build new systems as variation on old systems
- Selecting the right architecture
  - crucial to success
- Making choices
- Representation
  - describes the high level properties

Architectural Style

Architecture

- *component*: represents computation
- *connectors*: facilitates component communication

Architectural Style/Configuration

- architecture = ‹components, connectors, constraints›

Visualization

- graph representation

Architectural Styles

Pipes and filters

Data abstraction

Implicit invocation

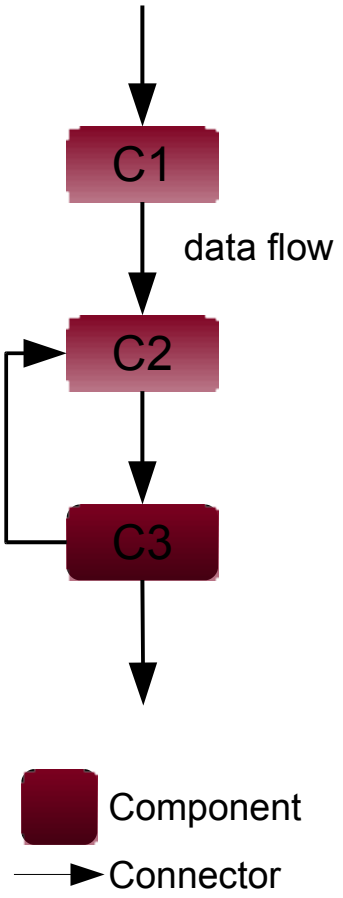
Layered systems

Repositories

Pipes & Filters

Overview

- Architectural pattern for stream processing
- A filter defines a processing/computation step
- Data flows through a sequential chain of filters
- A filter chain represents a system



University of Waterloo

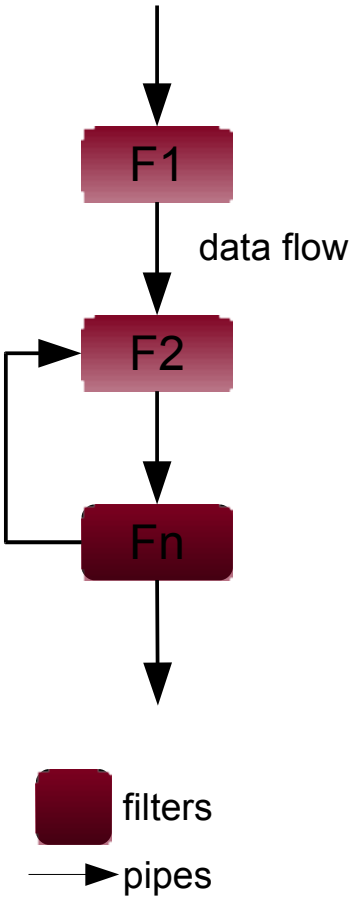
Pipes & Filters

Components (Filters)

- Set of inputs and outputs
- Input & output streams
- Local transformation
  - incremental output
- Known as filters

Connectors (Pipes)

- Facilitate data flow
- Known as pipes



Pipes & Filters

Invariants

- Independent entities
  - do not share state
  - have no knowledge of other filters
- Transformation
  - incremental
  - not dependent on order in the chain

Pipes & Filters

Specialization

- *Pipelines*: restricted to linear topology
- *Bounded pipes*: restricts the amount of data on a pipe
- *Typed pipes*: data on a pipe to be of an acceptable type

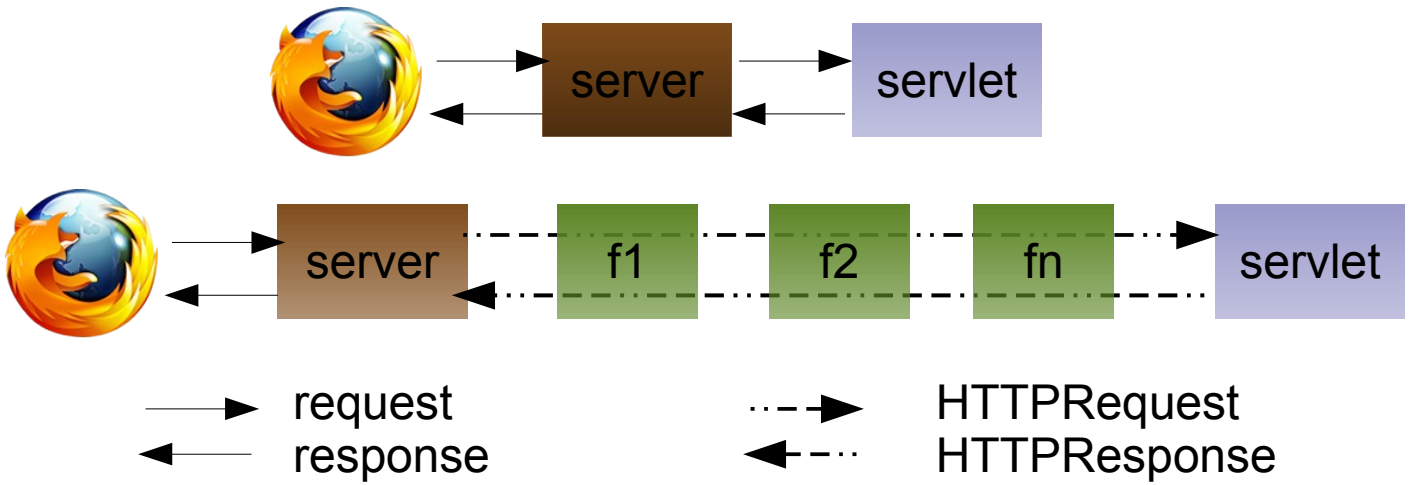
Question

- Can a filter process all of its input data as a single entity?

Pipes & Filters

Examples

- Unix shell programs
  - pipelines
  - cat file1 | sort | grep keyword
- JEE Servlet Filter (javax.servlet.Filter)
  - typed pipes

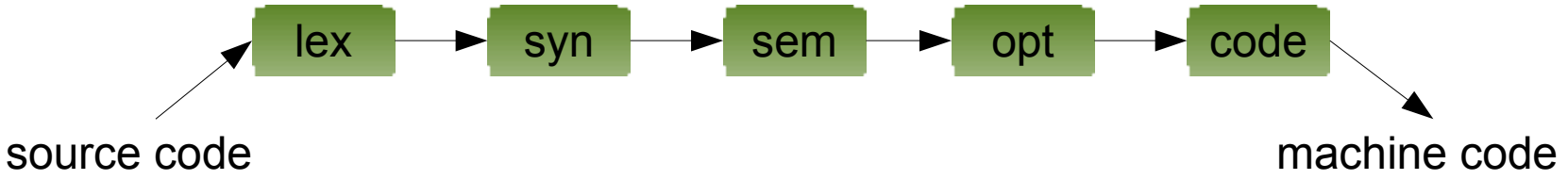


University of Waterloo

Pipes & Filters

Examples

- Compilers
  - More of a sequential batch architecture



Pipes & Filters

Advantages

- Simple composition
- Reuse
  - any two filters can be combined together
    - as long as they speak the same data language
- Prototyping
  - how many scripts make use of grep, awk, sed etc.
- Easy growth & evolution
  - Architectural evaluation for performance & bottlenecks
- Concurrency & parallelism

Pipes & Filters

Disadvantages

- Poor performance
  - each filter has to parse data
  - sharing global data is difficult
- Not appropriate for interaction
- Low fault tolerance threshold
  - What happens if a filter crashes
- Data transformation
  - to LCD to accommodate filters
- Increases complexity & computation

Data Abstraction

Object Oriented Organization (OOO)

- Encapsulation (data & operations)

Components

- Objects, modules

Connectors

- represent inter-object communication
  - synchronous or asynchronous

Data Abstraction

Key aspects

- Objects preserve their integrity
- no direct access
- Object representation is a private affair

Advantages

- Implementation changes with minimal global impact
- Decomposition
  - large system into a set of interacting objects
  - easy to manage & evolve

University of Waterloo

Data Abstraction

Disadvantages

- Interaction injects coupling
  - objects interact via public contract
  - what happens when the contract changes?
  - indirect coupling: A uses B, C uses B, then changes made by C on B are unexpected to A

Data Abstraction

Some thoughts

- Design by contract – interfaces
  - decouples inter-object dependencies
- Synchronization

What would happen if an object were to fail during an operation?

Implicit invocation

Event-based

- Components do not directly invoke other components
- Similar to *observer (GOF) design pattern*
  - implicit invocation architectural style has broader scope

Components

- Modules {event, callback | procedure}
  - objects, processes, distributed applications

Connectors

- Traditional method call
- Broadcast of events

Implicit invocation

Publish & Subscribe

- Components register for events
- Events are generated published
  - by different sources
  - to a centralized system
- Events are broadcast
  - via callback or procedure

Implicit invocation

Invariants

- Event generators do not know
  - about event consumers
  - functional impact on different components
- Broadcast ordering
  - components cannot make assumptions about ordered delivery

Implicit invocation

Examples

- News, fire alarms etc.
- MVC
- IDEs
- Database systems to
  - ensure consistency constraints
  - execute stored procedures
- User interface
  - Separation of data presentation from data management
- Enterprise application interaction

Implicit invocation

Advantages

- Minimal dependency and loose coupling
  - Components do not directly interact with each other
  - Components can be added or removed
- Highly reusable
  - Components can be replaced with newer components
    - without changing their interfaces
- Scalable
  - New components can simply register themselves

Implicit invocation

Disadvantages

- Loss of execution control
  - Who, when, what
- Data exchange
  - information has to be encapsulated within an event
  - shared repository
  - impact on
    - global system performance & resource management
- Event context
  - unpredictable side effects
  - how to debug such a problem?

University of Waterloo

Layered Systems

Organized hierarchy

- Each layer
  - provides a service to the layer above
  - acts as a client to the layer below

Components

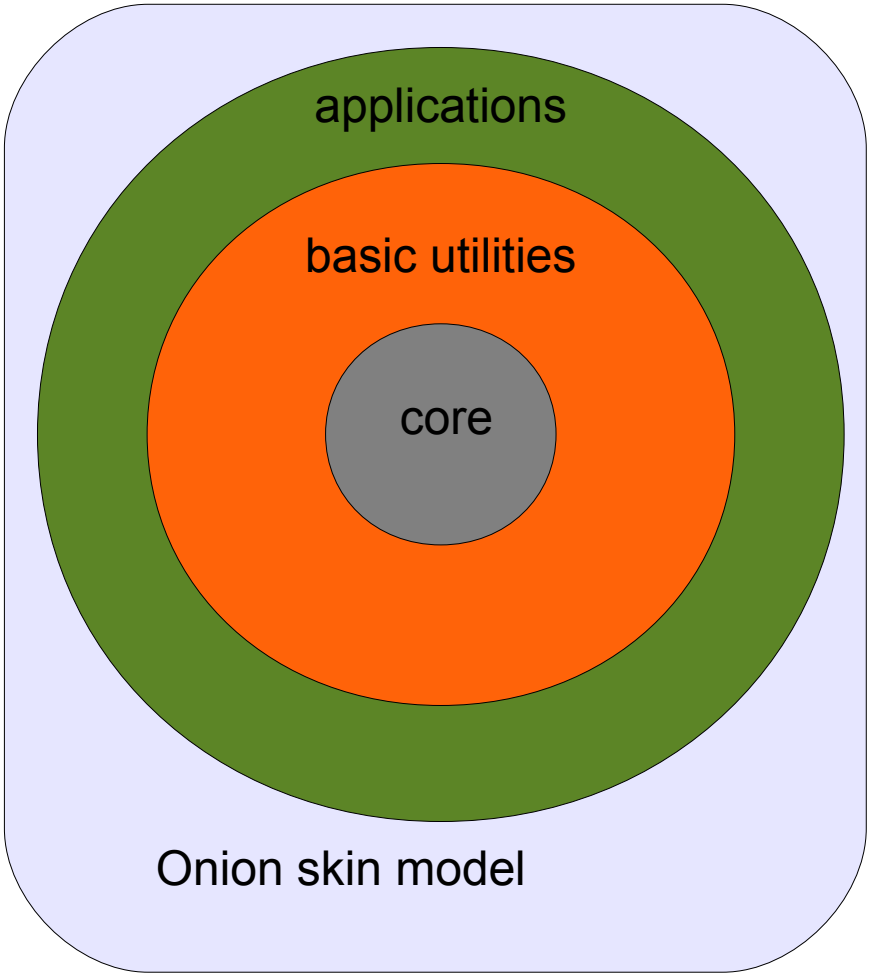
- Layers: composed of groups of subtasks
- API: Set of classes exposing an API layer

Connectors

- Communication protocols/interfaces
  - define the inter-layer interaction

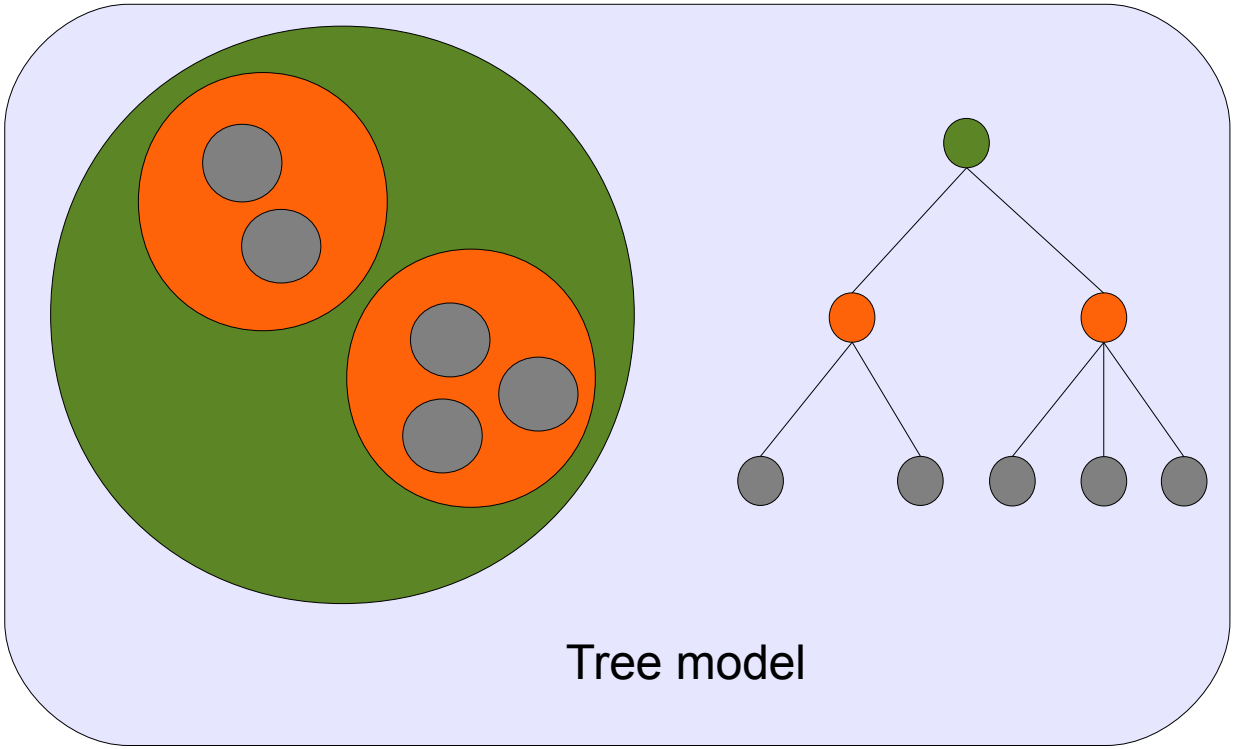
University of Waterloo

Layered Systems

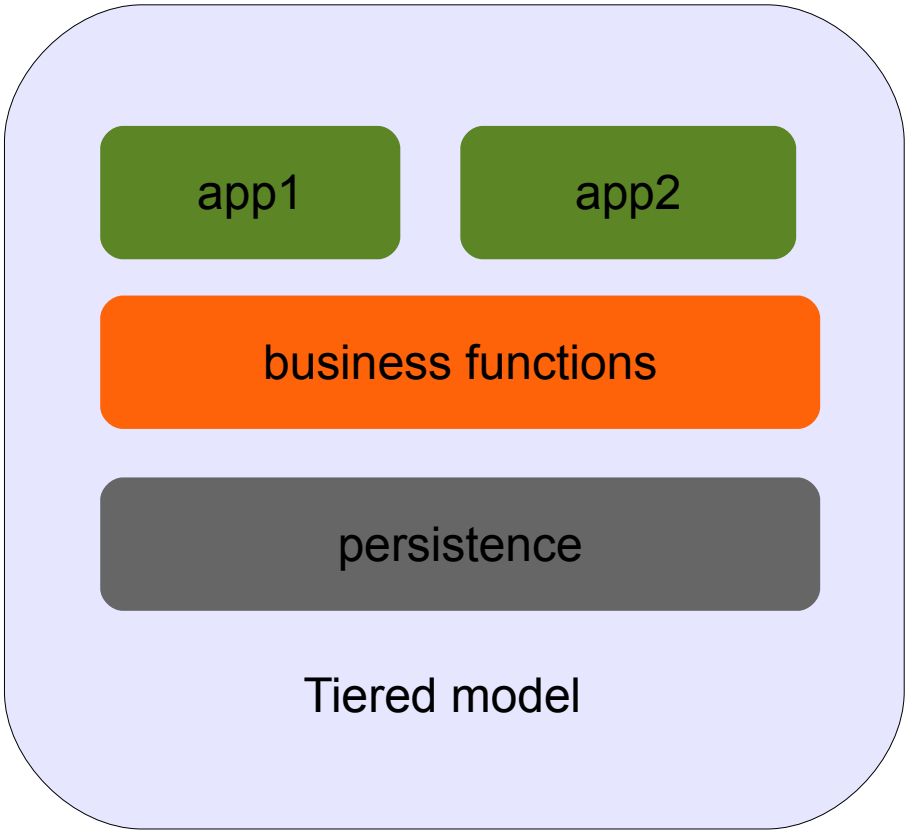


Onion skin model

Layered Systems



Layered Systems



Layered Systems

Invariants

- Limit layer interactions to adjacent layers only
  - Can be violated as follows:
    - A layer may use any layer below for service
- Much richer interaction when compared to pipeline
  - two way communication
- Layers must support the protocols of its upper and lower boundaries

Layered Systems

Advantages

- Increasing levels of abstraction
- Low coupling
  - easy to maintain
  - a layer only interacts with a layer above and a layer below
- Modular reuse
  - a layer can be replaced by another as long as the interface is not violated

Layered Systems

Disadvantages

- Not all systems can be layered
- Performance
  - May force the high level functions to be tightly coupled with low level implementation

Tiered Architecture

- Specialization for enterprise applications
  - tiers are generally physically separated

Repositories

Main idea

- Centralized source of information with many components

Components

- Type 1: central data-store component
  - represents system state/data
- Type 2: collection of data-use components
  - collection of independent components operate on the central data-store

Repositories

Connections

- Vary considerably
  - Active: Incoming streams of transactions trigger processes to act on data-store – *database*
  - Passive: current state of the data-store triggers processes – *blackboard*

Repositories

Advantages

- Efficient when dealing with large amounts of data
  - Known data schema
  - leads to ease of data sharing
  - centralized management
- Clients are loosely coupled

Repositories

Disadvantages

- Data model
  - is static, bounded by defined schema
  - resistant to change as many depend on it
  - evolution is expensive
- Difficult to distribute

Interpreter Style

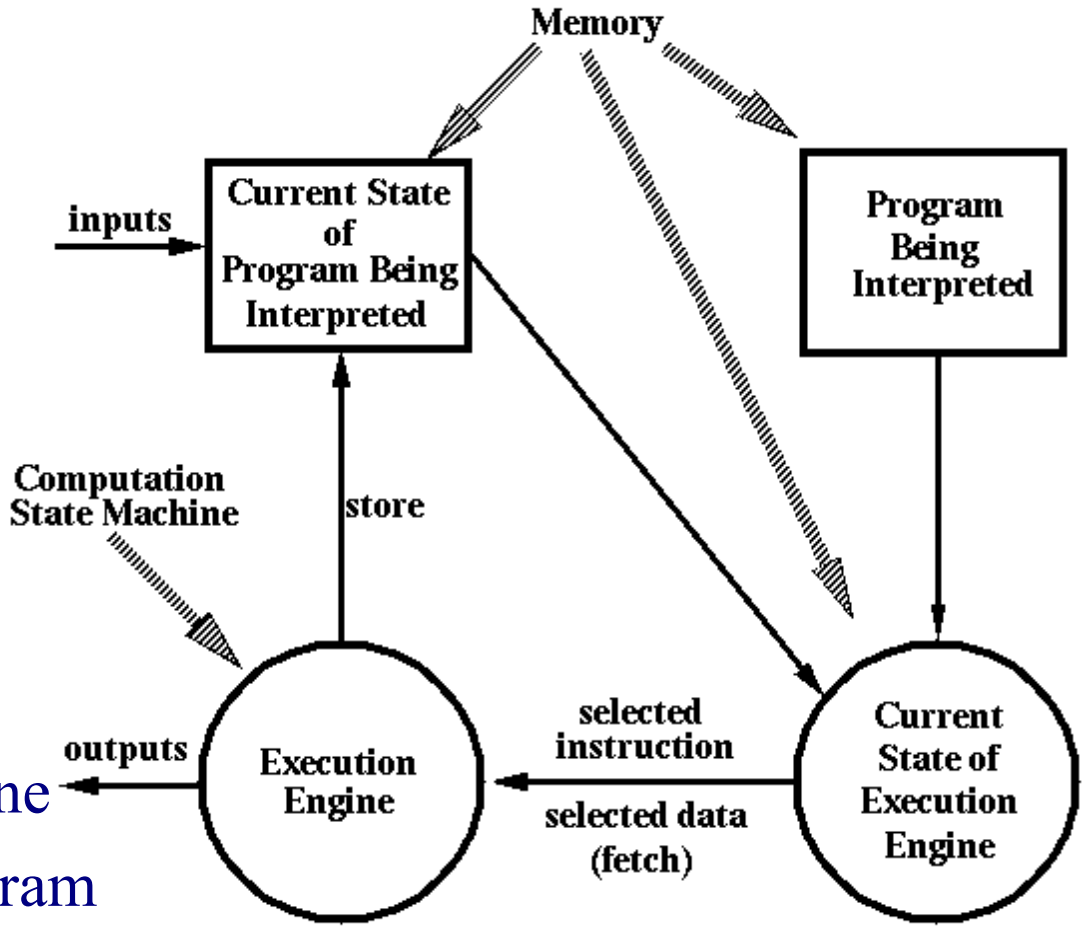
Main idea

- Bridge functionality
  - Suitable for applications in which the most appropriate language or machine for executing the solution is not directly available

Interpreter Style

Components

- interpretation engine
  - to do the work
- memory
  - contains the psuedo-code & state
- state
  - control state of the engine
  - current state of the program



Interpreter Style

Connectors

- procedure calls
- direct memory access

Examples

- Programming language compilers
  - Java, small talk
- Scripting languages
  - awk, Perl

Interpreter Style

Advantages

- Simulation of non-implemented parts
- Portability
  - across a variety of platforms

Disadvantages

- Performance
  - Computational complexity – slow execution