Software Architecture in Practice

Chapter 5: Architectural Styles - From Qualities to Architecture

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Sponsored by the U.S. Department of Defense

Lecture Objectives

This lecture will enable students to
- **define what is meant by "architectural style"**
- **list several examples of architectural styles and describe the key characteristics of each**
- **give examples of how the use of particular architectural styles helps achieve desired qualities**

Architectural Styles -1

An architectural style is a description of component types and a pattern of their runtime control and/or data transfer.

A style
- **is found repeatedly in practice**
- **is a package of design decisions**
- **has known properties that permit reuse**
- **describes a class of architectures**

Architectural Styles -2

A style is determined (described) by
- **a set of component types (e.g., data repository, process, object)**
- **a topological layout of these components**
- **a set of semantic constraints (e.g., a data repository is not allowed to change stored values)**
- **a set of interaction mechanisms (e.g., subroutine call, event, pipe)**

Architectural Styles -3

David Garlan and Mary Shaw have compiled a catalog of architectural styles [Shaw 95].

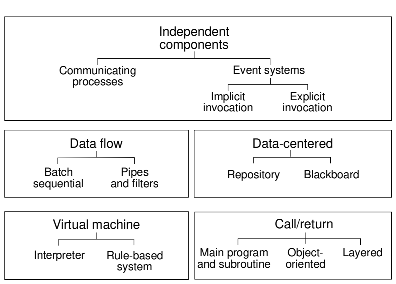
There is no complete list.

There is no unique, non-overlapping list.

Styles overlap.

Systems exhibit multiple styles at once.

Some Types of Architectural Styles -1



Some Types of Architectural Styles -2

The following styles will be discussed in detail in the next section of this lecture.

- **data-centered**
- **virtual machine**
- **call/return**

Other styles will be introduced through examples.

Data-Centered Style -1

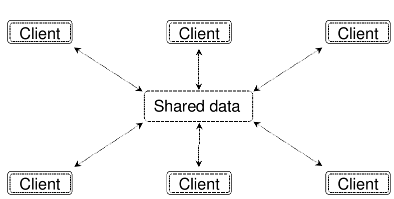
Goals

- **integrability**
- **scalability (new clients/data easily added)**

Examples

- **passive data store: repository style**
- **active data store: blackboard style**

Data-Centered Style -2

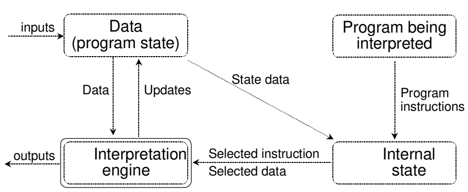


Virtual Machine Style -1

Goal: simulate non-native functionality for portability or prototyping

Examples
- **interpreters**
- **rule-based systems**
- **command language processors**

Virtual Machine Style -2

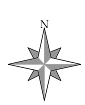


Call/Return Architectures

Dominant design style for 30 years

Goals: vary according to sub-styles

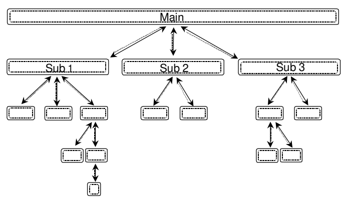
Sub-styles
- **main program/subroutine**
- **object-oriented**
- **layered hierarchies**



Main Program/Subroutine Style

Early goals: reuse, independent development

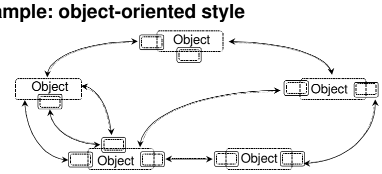
Example: hierarchical call/return style



Object-Oriented/Abstract Data Style

Goals
- **more natural modeling of real world**
- **reuse by increments**

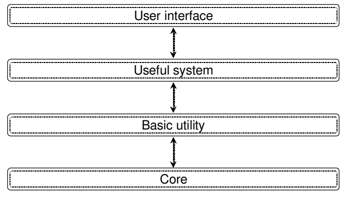
Example: object-oriented style



Layered Hierarchies

Goals: portability, reuse

Example: ISO Open Systems Interconnection reference model



Heterogeneous Systems

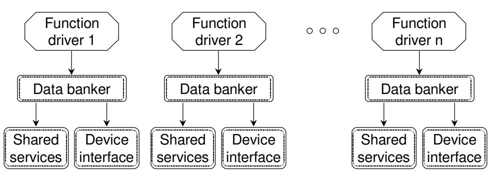
Few systems are constructed purely from a single style. Most are heterogeneous mixtures of styles.

There are three types of heterogeneous systems.
- **locational**
- **simultaneous**
- **hierarchical**

Locational Heterogeneity

Different styles in different parts of the system

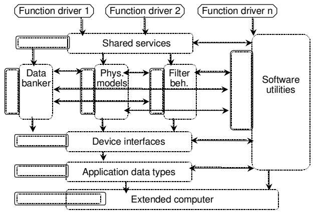
Example: A-7E used a cooperating process style in function drivers and call/return style elsewhere.



Simultaneous Heterogeneity

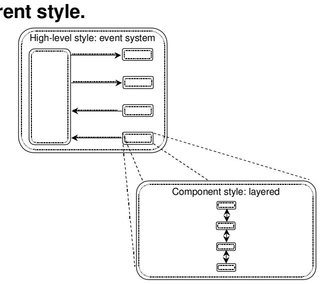
Different styles due to style overlap

Example: A-7E simultaneously used layered, cooperating process, and object-based styles.



Hierarchical Heterogeneity

A component of one style may itself be realized by a different style.



Using Styles To Achieve Qualities -1

We will look at a small historical example.

KeyWord-In-Context (KWIC) system
- **input: strings, each of which consists of several words.**
- **output: a sorted list of all orderings of each input string.**

Using Styles To Achieve Qualities -2

Example of KWIC operation
- **input**
  - **“Clouds are white.”**
  - **“Pittsburgh is beautiful.”**
- **output**
  - **are white clouds**
  - **beautiful pittsburgh is**
  - **clouds are white**
  - **is beautiful pittsburgh**
  - **pittsburgh is beautiful**
  - **white clouds are**

Styles Used with KWIC

KWIC with
- shared memory style
- abstract data types
- implicit invocation
- pipe-and-filter

KWIC with Shared Memory Style -1

Historical example: Shared memory style is the way that systems were built for performance reasons until the early 1970s.

Shared memory style is not normally used today due to concerns with other qualities. (Shared memory style does not easily scale up to large architectures.)

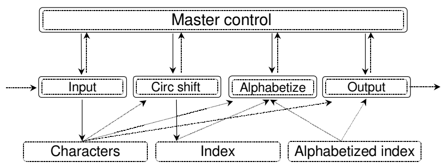
KWIC with Shared Memory Style -2

Master control calls
- Input, which inputs and stores characters
- CircShift, which reads characters/writes index
- Alphabetize, which alphabetizes index
- Output, which prints alphabetized index

Advantage: performance

Disadvantages: modifiability, reuse

KWIC with Shared Memory Style -3



KWIC with Abstract Data Types -1

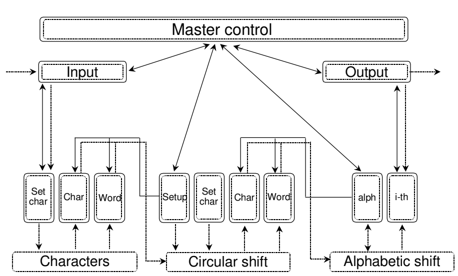
Data types are "protected" by access programs.

Data formats are proprietary to each module. Abstract data types hide whether alphabetization is done all at once, or on demand.

Advantage: modifiability to change existing functions or formats

Disadvantage: performance when accessing data

KWIC with Abstract Data Types -2



KWIC with Implicit Invocation - 1

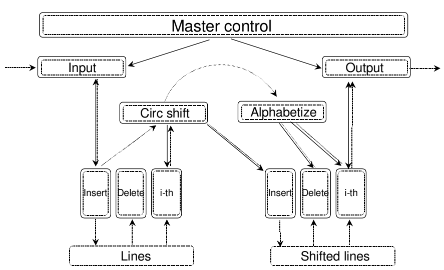
Calls to circular shift and alphabetizer are implicit, and are the result of inserting lines.

Adding a function is easy: register it against the event of interest, and it is automatically called.

Advantage: modifiability to add new functions

Disadvantages: tracking control flow is difficult; performance (implicit invocations will add overhead in tasking if there are many tasks); harder to analyze system for deadlocks

KWIC with Implicit Invocation - 2



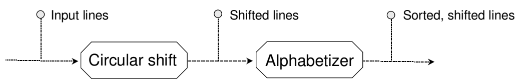
KWIC with Pipe-and-Filter - 1

Two functions do all the work.
- **one to produce the circular shifts**
- **one to alphabetize**

Advantage: functions can be reused; they are loosely coupled.

Disadvantage: this system will likely have poor performance.
- **Data is stored twice.**
- **Each filter is a separate process.**

KWIC with Pipe-and-Filter - 2



Carnegie Mellon University
Software Engineering Institute

Rough Comparison of KWIC Architectures

	Shared data	Abstract data type	Implicit invocation	Pipe and filter
Change in algorithm	-	-	+	+
Change in data representation	-	+	-	-
Change in function	-	-	+	+
Performance	+	+	-	-
Reuse	-	+	-	+

Carnegie Mellon University
Software Engineering Institute

Rules of Thumb for Choosing Styles

The goal of style catalogs is to develop a design handbook: "If your problem looks like x, use style y."

The practice is not that advanced yet. The best that we can do is offer rules of thumb.

Data-Flow Style

Use if

- **it makes sense to view your system as one that produces a well-defined, easily-identified output that is the direct result of sequentially transforming a well-defined, easily-identified input in a time-independent fashion**
- **integrability (in this case, resulting from relatively simple interfaces between components) is important**

Data-Flow Substyles

Dataflow network: **input and output occur as recurring series, with direct correlation between corresponding members of each series**

Pipeline, pipe and filter: **the computation involves transformations on continuous streams of data**

Closed loop control: **your system involves controlling continuing action, is embedded in a physical system, and is subject to unpredictable external perturbation so that preset algorithms go awry**

Call-and-Return Style

Use if the order of computation is fixed, and components can make no useful progress while awaiting the results of requests to other components

Call-and-Return Substyles -1

Object-oriented: overall modifiability and integrability (via careful attention to interfaces) are driving quality requirements

Abstract data types: many system data types whose representation is likely to change

Objects: many like modules, commonalties could be exploited through inheritance

Call-and-return-based client-server: modifiability with respect to the production of data and how it is consumed is important

Call-and-Return Substyles -2

Layered

- if the tasks in your system can be divided between those specific to the application and those generic to many applications but specific to the underlying computing platform
- if portability across computing platforms is important
- if you can use an already-developed computing infrastructure layer (operating system, network management package, etc.)

Independent Component Style

Use if

- your system runs on a multi-processor platform (or may do so in the future)
- your system can be structured as a set of loosely coupled components (meaning that one component can continue to make progress somewhat independently of the state of other components)
- performance tuning (by re-allocating work among processes) is important
- performance tuning (by re-allocating processes to processors) is important

Independent Component Substyles -1

Communicating processes: **message-passing is sufficient as an interaction mechanism**

Lightweight processes: **access to shared data is critical to meet performance goals**

Distributed objects: **the reasons for the object-oriented style and the interacting process style all apply**

Independent Component Substyles -2

Broadcast: **all of the components need to be synchronized from time to time, and/or availability is an important requirement**

Event systems: **you want to decouple the consumers of events from their signalers, or you want scalability in the form of adding processes that are triggered by events already detected/signaled in the system**

Data-Centered Style

Use if central issue is the storage, representation, management, and retrieval of a large amount of related long-lived data

Sub-styles
- ***transactional database/repository:*** **order of component execution is determined by a stream of incoming requests to access/update the data, and the data is highly structured**
- ***blackboard:*** **you want scalability in the form of adding consumers of data without changing the producers; or you want modifiability in the form of changing who produces and consumes which data**

Virtual Machine Style

Use if you have designed a computation, but have no fixed machine to run it on

Lecture Summary

Architectural styles can be described by the
- **nature of their components and connectors**
- **topography**
- **kind of qualities and reasoning they support**

Architectural styles have important influences or quality attributes.
- **The style(s) used for a system depends on which qualities are most desired.**
- **Styles provide a concise way of designing and describing a system.**

Discussion Questions - 1

1. **The last part of this lecture gives rules of thumb for style selection. Pick a few of the styles not**

   mentioned and see if you can write rules of thumb for choosing those styles.

2. **A large number of styles are designed to support the quality of modifiability. Give examples and styles of the different types of modification supported by them.**

Discussion Questions -2

3. **Styles, as they are commonly described, are the result of empirical observation, not a taxonomic organization from first principles. As a result, overlap is high: Objects can be cooperating processes that can be layered, and so on. Why is that? Hint: Think about what architectural structures from Lecture 2 are involved in the description of each style.**