Lecture 06:
Architecture Styles

Emad Shihab
Adapted from Ahmed E. Hassan and Spiros Mancoridis
What are Architectural Styles

An **Architectural Style** defines a family of systems in terms of a pattern of structural organization. It determines:

- the **vocabulary** of components and connectors that can be used in instances of that style
- a set of **constraints** on how they can be combined.
Why Architectural Styles

- Makes for an easy way to communicate among stakeholders
- Documentation of early design decisions
- Allow for the reuse and transfer of qualities to similar systems
Describing an Architectural Style

• The architecture of a specific system is a collection of:
  – computational components
  – description of the interactions between these components (connectors)
Describing an Architectural Style (Cont’d)

- Software architectures are represented as graphs where **nodes** represent components:
  - procedures
  - modules
  - processes
  - tools
  - databases

- and **edges** represent connectors:
  - procedure calls
  - event broadcasts
  - database queries
  - pipes
Repository Style

• Suitable for applications in which the central issue is establishing, augmenting, and maintaining a complex central body of information.

• Typically the information must be manipulated in a variety of ways. Often long-term persistence is required.
Repository Style (Cont’d)

- **Components:**
  - A central data structure representing the current state of the system.
  - A collection of independent components that operate on the central data structure.

- **Connectors:**
  - Typically procedure calls or direct memory accesses.
Repository Style (Cont’d)

Shared Data

Memory Access

Computation

Memory

Software Design (Software Architecture)
Repository Style Specializations

- Changes to the data structure trigger computations.
- Data structure in memory (persistent option).
- Data structure on disk.
- Concurrent computations and data accesses.
Repository Style Examples

- Information Systems
- Programming Environments
- Graphical Editors
- AI Knowledge Bases
- Reverse Engineering Systems
Repository Style Advantages

- **Efficient** way to store large amounts of data.
- **Sharing** model is published as the repository schema.
- **Centralized management**:  
  - backup  
  - security  
  - concurrency control
Repository Style Disadvantages

- Must agree on a data model a priori.
- Difficult to distribute data.
- Data evolution is expensive.
Pipe and Filter
Architectural Style

• Suitable for applications that require a defined series of independent computations to be performed on data.
• A component reads streams of data as input and produces streams of data as output.
Pipe and Filter
Architectural Style (Cont’d)

- **Components**: called filters, apply local transformations to their input streams and often do their computing incrementally so that output begins before all input is consumed.

- **Connectors**: called pipes, serve as conduits for the streams, transmitting outputs of one filter to inputs of another filter.
Pipe and Filter
Architectural Style (Cont’d)
Pipe and Filter Invariants

- Filters do not share state with other filters.
- Filters do not know the identity of their upstream or downstream filters.
Pipe and Filter Specializations

- **Pipelines**: Restricts topologies to linear sequences of filters.
- **Batch Sequential**: A degenerate case of a pipeline architecture where each filter processes all of its input data before producing any output.
Pipe and Filter Examples

- **Unix Shell Scripts:** Provides a notation for connecting Unix processes via pipes.
  - `cat file | grep Erroll | wc -l`

- **Traditional Compilers:** Compilation phases are pipelined, though the phases are not always incremental. The phases in the pipeline include:
  - *lexical analysis + parsing + semantic analysis + code generation*
Pipe and Filter Advantages

- **Easy to understand** the overall input/output behavior of a system as a simple composition of the behaviors of the individual filters.
- **They support reuse**, since any two filters can be hooked together, provided they agree on the data that is being transmitted between them.
Pipe and Filter
Advantages (Cont’d)

- Systems can be **easily maintained and enhanced**, since new filters can be added to existing systems and old filters can be replaced by improved ones.
- They permit certain kinds of **specialized analysis**, such as throughput and deadlock analysis.
- The naturally **support concurrent execution**.
Pipe and Filter Disadvantages

- Not good choice for **interactive systems**, because of their transformational character.
- Excessive parsing and unparsing leads to **loss of performance** and **increased complexity** in writing the filters themselves.
Case Study:
Architecture of a Compiler

• The architecture of a system can change in response to improvements in technology.
• This can be seen in the way we think about compilers.
Early Compiler Architectures

• In the 1970s, compilation was regarded as a sequential (batch sequential or pipeline) process:
Early Compiler Architectures

Most compilers create a separate symbol table during lexical analysis and used or updated it during subsequent passes.
Later, in the mid 1980s, increasing attention turned to the intermediate representation of the program during compilation.
Hybrid Compiler Architectures

- The new view accommodates various tools (e.g., syntax-directed editors) that operate on the internal representation rather than the textual form of a program.
- Architectural shift to a repository style, with elements of the pipeline style, since the order of execution of the processes is still predetermined.
Hybrid Compiler Architectures

Lex → Syn → Sem → Opt → Cgen

Attributed Parse Tree
Symbol Table

Edit → Flow

Software Design (Software Architecture)
References