Architectural Styles and Patterns 1

David Nandigam

Overview

- Architectural styles and patterns
  - Data flow
  - Call-and-return
  - Interacting processes
  - Data-oriented repository
  - Data-sharing
  - Hierarchical
  - Process control systems
  - Table-driven
  - Other
- Heterogeneous architectures

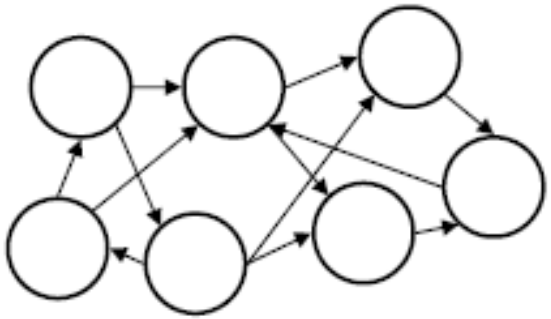
Data Flow

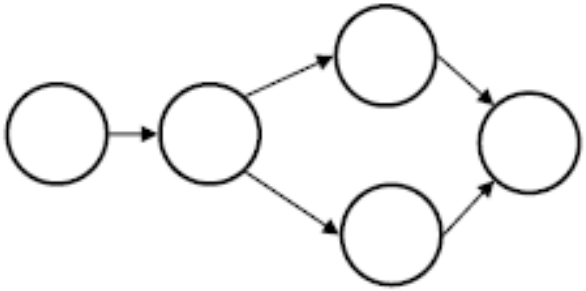
- A data flow system is one in which:
  - The availability of data controls the computation
  - The structure of the design is determined by the orderly motion of data from component to component
  - The pattern of data flow is explicit
  - This is the only form of communication between components
- There are variety of variations on this general theme:
  - How control is exerted (e.g., push versus pull)
  - Degree of concurrency between processes
  - Topology

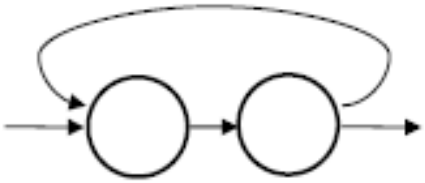
Data Flow

- Components: Data Flow Components
  - Interfaces are input ports and output ports
  - Input ports read data; output ports write data
  - Computational model: read data from input ports, compute, write data to output ports
- Connectors: Data Streams
  - Uni-directional
    - Usually asynchronous, buffered
  - Interfaces are reader and writer roles
  - Computational model: transport data from writer roles to reader roles
- Systems
  - Arbitrary graphs
  - Computational model: functional composition

Patterns of Data Flow







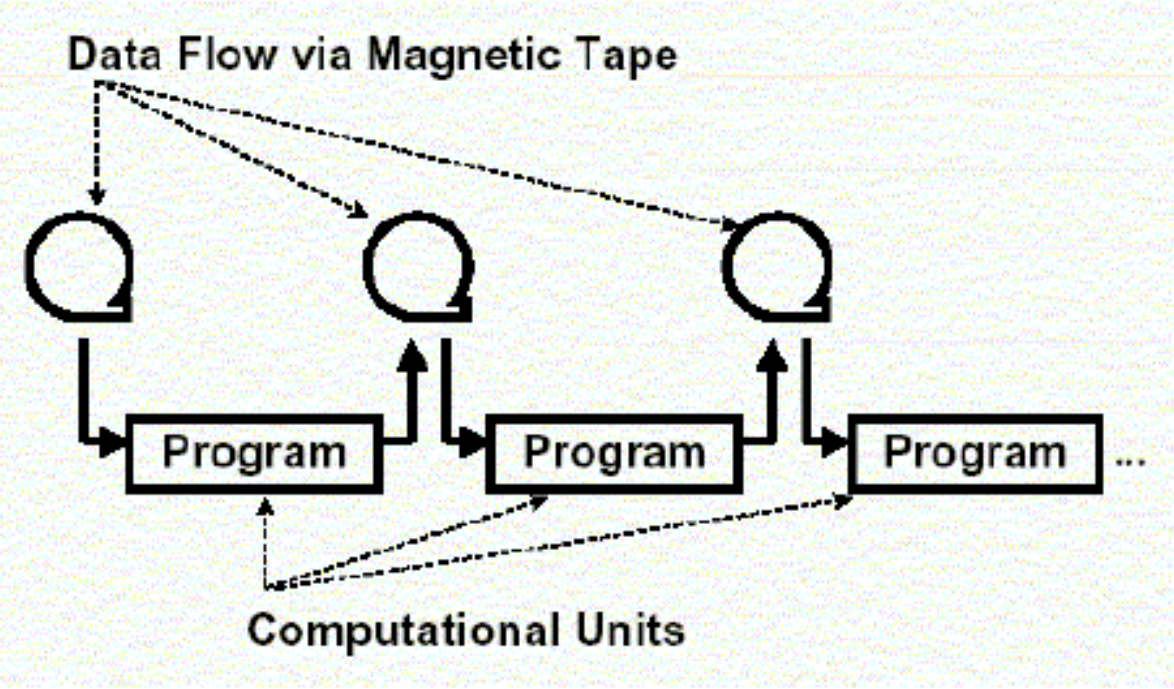
- Data can flow in arbitrary patterns
- Primarily we are interested in linear data flow patterns
- ...or in simple, constrained cyclical patterns...

Kinds of Data Flow Architectures

- Batch sequential
- Dataflow network (pipes&filters)
  - acyclic, fanout, pipeline, Unix, etc.
- Closed loop control

Characteristics of Batch Sequential Systems

- Components (processing steps) are independent programs
- Connectors are some type of media - traditionally magnetic tape
- Each step runs to completion before the next step begins



Pipes and Filters

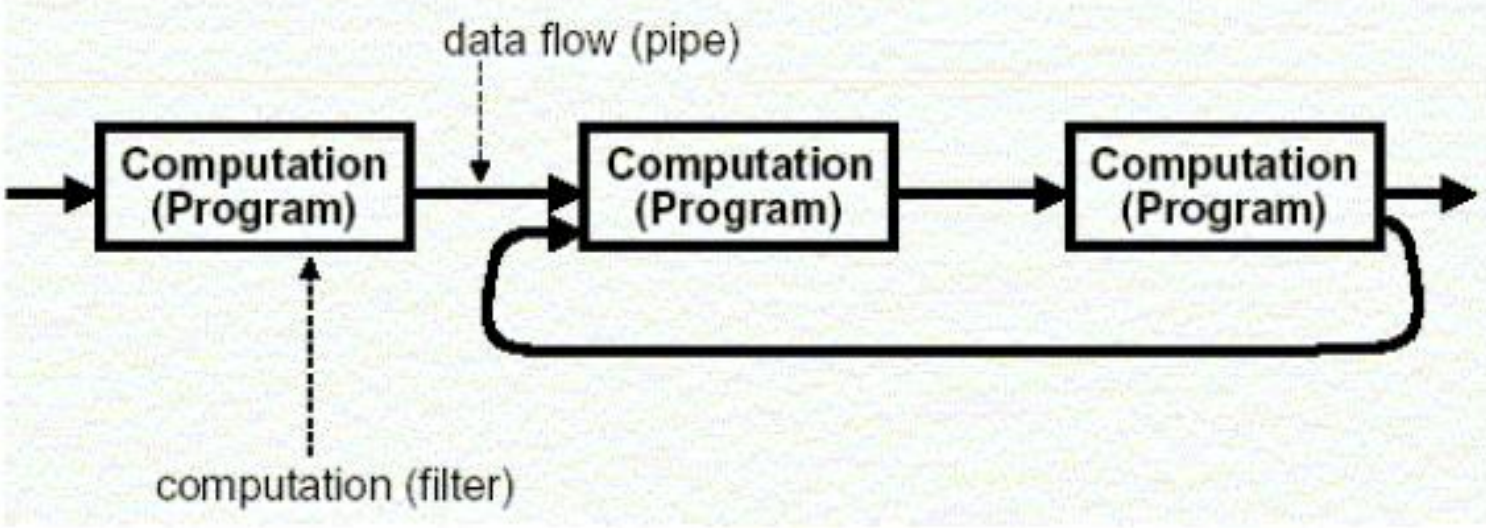
- Architectural pattern [style] providing a structure for systems that process a stream of data
- Each processing step is encapsulated in a filter component
- Data is passed through pipes between adjacent filters
- Recombining filters allows you to build families of related systems

Pipes and Filters

- Components (Filters)
  - Read streams of data on input producing streams of data on output
  - Local incremental transformation to input stream (e.g., filter, enrich, change representation, etc.)
  - Data is processed as it arrives, not gathered then processed
  - Output usually begins before input is consumed
- Connectors (Pipes)
  - Conduits for streams, e.g., first-in-first-out buffer
  - Transmit outputs from one filter to input of other

Pipes and Filters

- Compared to the batch-sequential style, data in the pipe&filter style is processed *incrementally*
- The tape of the batch sequential system, morphed into a language and operating system construct



Pipes and Filters

- Invariants
  - Filters must be independent, no shared state
  - Filters don't know upstream or downstream filter identity
  - Correctness of output from network must not depend on order in which individual filters provide their incremental processing
- Common specialisations
  - Pipelines: linear sequence of filters
  - Bounded and typed pipes …

Pipe and Filter: Strengths

- Easy to understand - overall behaviour is a simple composition of behaviour of individual filters.
- Reuse - any two filters can be connected if they agree on that data format that is transmitted.
- Ease of maintenance - filters can be added or replaced.
- Prototyping e.g. Unix shell scripts are famously powerful and flexible, using filters such as sed and awk.
- Architecture supports formal analysis - throughput and deadlock detection.
- Potential for parallelism - filters implemented as separate tasks, consuming and producing data incrementally.

Pipe and Filter: Weaknesses

- Can degenerate to 'batch processing' - filter processes all of its data before passing on (rather than incrementally).
- Often poor inherent performance – e.g., parsing/unparsing in each filter & sharing global data is expensive or limiting.
- Can be difficult to design incremental filters.
- Not appropriate for interactive applications - doesn't split into sequential stages. POSA book has specific styles for interactive systems, one of which is Model-View-Controller.
- Synchronisation of streams will constrain architecture.
- Error handling is Achilles heel e.g. filter has consumed three quarters of its input and produced half its output and some intermediate filter crashes! Restart or resynchronize pipeline.
- Implementation may force lowest common denominator on data transmission e.g. Unix scripts everything is ASCII.

Pipe-and-Filter vs. Batch Sequential

- Both decompose the task into a fixed sequence of computations (components) interacting only through data passed from one to another

Batch Sequential	Pipe-and-Filter
• course grained • high latency • external access to input • no concurrency • non-interactive	• fine grained • results starts processing • localized input • concurrency possible • interactive awkward but possible