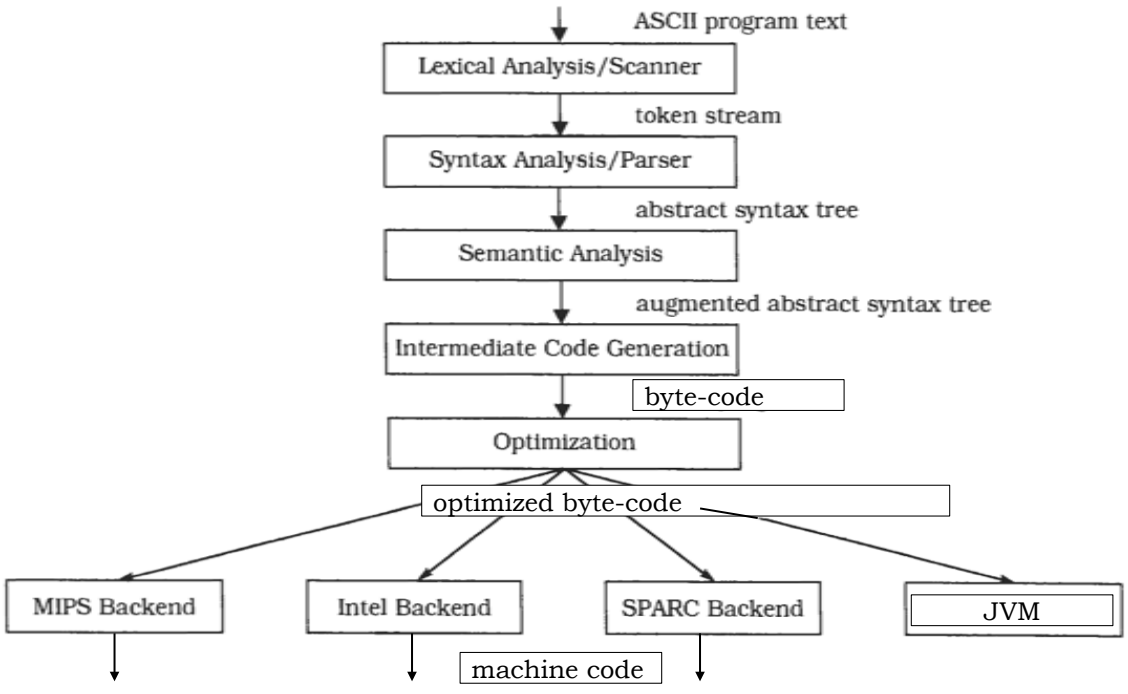
Pipes and Filters Pattern

Pipes and Filters Introduction

- The Pipes and Filters architectural pattern provides 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.

Example

- Suppose we want to develop a new compiler for the Java programming language.
  - We want the compiler to produce either optimized byte-code to run on a JVM, or optimized machine code to run natively on various platforms.
  - Similar to the GNU GCJ compiler.
- A backend will translate byte-code into the machine instructions of a specific processor for best performance.
- Conceptually, translation from Java to byte-code consists of the phases lexical analysis, syntax analysis, semantic analysis, byte-code generation, byte-code optimization, and optionally machine-code generation.
- Each stage has well-defined input and output data.



Problem

- How to provide a design that is suitable for processing data streams.

Context

- Some applications process streams of data.
  - Input data streams are transformed stepwise into output data streams.
- However, using common and familiar request/ response semantics for structuring such types of application is typically impractical.
- Instead we must specify an appropriate data flow model for them.

Context (2)

- Modeling a data-flow-driven application raises some non-trivial developmental and operational challenges.
- First, the parts of the application should correspond to discrete and distinguishable actions on the data flow.
- Second, some usage scenarios require explicit access to intermediate yet meaningful results.
- Third, the chosen data flow model should allow applications to read, process, and write data streams incrementally rather than wholesale and sequentially so that throughput is maximized.
- Lastly, long-duration processing activities must not become a performance bottleneck.

Forces

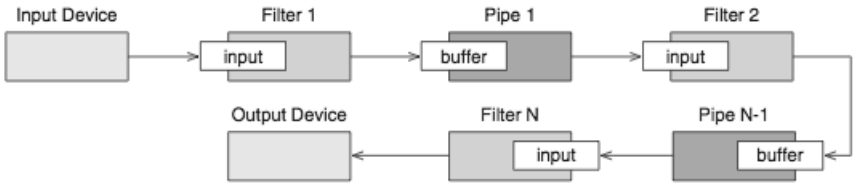
- Future system enhancements should be possible by exchanging processing steps or by recombination of steps, even by users.
- Small processing steps are easier to reuse in different contexts than large components.
- Non-adjacent processing steps do not share information.
- Different sources of input data exist, such as a network connection or a hardware sensor providing temperature readings, for example.
- It should be possible to present or store final results in various ways.
- Explicit storage of intermediate results for further processing in files clutters directories and is error-prone, if done by users.
- You may not want to rule out multi-processing the steps, for example running them in parallel or quasi-parallel.

Solution

- The Pipes and Filters architectural pattern divides the task of a system into several sequential processing steps.
- These steps are connected by the data flow through the system. The output data of a step is the input to the subsequent step.
- Each processing step is implemented by a filter component.
- A filter consumes and delivers data incrementally, in contrast to consuming all its input before producing any output, to achieve low latency and enable real parallel processing.

Solution (2)

- The input to the system is provided by a data source such as a text file. The output flows into a data sink such as a file, terminal, animation program and so on.
- The data source, the filters and the data sink are connected sequentially by pipes that buffer data.
- Each pipe implements the data flow between adjacent processing steps.
- The sequence of filters combined by pipes is called a processing pipeline.



Solution (3)

- In a single-process arrangement, pipes are typically implemented as queues.
- In a distributed arrangement, pipes are realized as some form of *messaging* infrastructure that passes data streams between remote filters.
- If a filter performs a long-duration activity, consider integrating multiple parallel instances of the filter into the processing chain.
  - Such a configuration can further increase system performance and throughput, as some filter instances can start processing new data streams while others are processing previous data streams.

Filter Details

- Filter components are the processing units of the pipeline.
- A filter *enriches* data by computing and adding information, *refines* data by concentrating or extracting information, and *transforms* data by delivering the data in some other representation. A concrete filter implementation may combine any of these three basic principles.
- An active filter starts processing on its own as a separate program or thread. Most commonly, the filter is active in a loop, pulling its input from and pushing its output down the pipeline.
- A passive filter component is activated by being called either as a function (pull) or as a procedure (push).

Pipe Details

- Pipes denote the connections between filters, between the data source and the first filter, and between the last filter and the data sink.
- If two active components are joined, the pipe synchronizes them. This synchronization is done with a first-in-first-out buffer (queue).
- If activity is controlled by one of the adjacent filters, the pipe can be implemented by a direct call from one filter to the next. Direct calls make filter recombination harder, however.

Variations

- The single-input single-output filter specification of the Pipes and Filters pattern can be varied to allow filters with more than one input and/or more than one output.
- Processing can then be set up a s a directed graph that can even contain feedback loops.
  - The design of such a system, especially one with feedback loops, requires a solid foundation to explain and understand the complete calculation.
  - A rigorous theoretical analysis and specification using formal methods are appropriate, to prove that the system terminates and produces the desired result.
- If we restrict ourselves to simple directed acyclic graphs, however, it is still possible to build useful systems.

Pipes and Filters Benefits

- A *Pipes and Filters* architecture decouples different data processing steps so that they can evolve independently of one another and support an incremental data processing approach.
- Pipes and Filters provides for flexibility by exchanging or reordering the processing steps.
- Recombining filters allows you to build families of related systems using existing processing components.

Pipes and Filters Liabilities

- Applicability
  - Whether a separation into processing steps is feasible strongly depends on the application domain and the problem to be solved. For example, an interactive, event-driven system does not split into sequential stages.