Pipes and Filters **

When transforming a DOMAIN MODEL (182) into a technical software architecture . . .

. . . we must sometimes provide a design that is suitable for processing data streams.

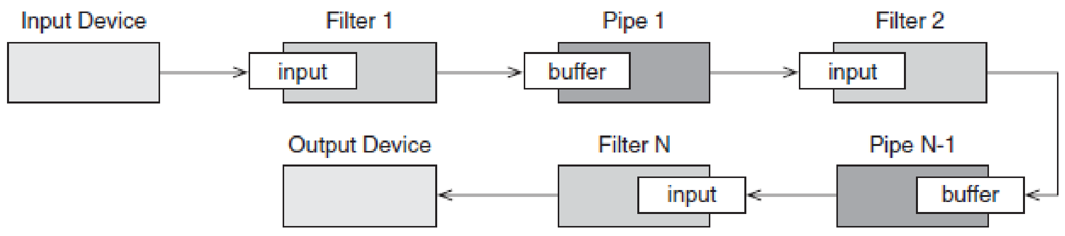
♦♦♦

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.

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. Last but not least, long-duration processing activities must not become a performance bottleneck.

Therefore:

Divide the application's task into several self-contained data processing steps and connect these steps to a data processing pipeline via intermediate data buffers.



Implement each processing step as a separate filter component that consumes and delivers data incrementally, and chain the filters such that they model the application's main data flow. In the data processing pipeline, data that is produced by one filter is consumed by its subsequent filters. Adjacent filters are decoupled using pipes that buffer data exchanged between the filters.

♦♦♦

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. Within a PIPES AND FILTERS architecture, filters are the units of domainspecific computation. Each filter can be implemented as a DOMAIN OBJECT (208) that represents a specific, self-contained data processing step. Filters with a concurrent DOMAIN OBJECT implementation enable incremental and concurrent data processing, which increases the performance and throughput of a PIPES AND FILTERS arrangement. 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.
Pipes are the medium of data exchange and coordination within a PIPES AND FILTERS architecture. Each pipe is a DOMAIN OBJECT that implements a policy for buffering and passing data along the filter chain:

data producing filters write data into a pipe, while data consuming filters receive their input from a pipe. The integration of pipes decouples adjacent filters so that the filters can operate independently of one another, which maximizes their individual operational performance. In a single-process PIPES AND FILTERS arrangement, pipes are typically implemented as queues. Pipes with a concurrent DOMAIN OBJECT implementation enable incremental and concurrent data processing, as do concurrent filters. In a distributed arrangement, pipes are realized as some form of MESSAGING (221) infrastructure that passes data streams between remote filters. Pipes that are implemented as a DOMAIN OBJECT shield filters from a knowledge of their specific implementation, which also allows transparent swapping of implementation forms. Such a design supports a flexible (re-)deployment of filters in a distributed PIPES AND FILTERS arrangement. MESSAGES (420) help to encapsulate the data streams that are passed along the pipes.