Software Design and Architectures
SE-2 / SE426 / CS446 / ECE426

Lecture 6 : Software Architectures: Dataflow Systems

Dataflow software systems are modeled after flow-based and queue-based systems in other fields of engineering, in which material or product passes through successive processing and transformation stations on the way between entering and leaving a process. In dataflow-based software, the material is information and the processing stations are filters or transforms. "A dataflow network … operates on a large, continuously-available data stream." [Shaw & Clements, Boxology]

According to Somerville, dataflow is not a architectural decision at all, but a modular decomposition decision. In other words, he considers that dataflow design is a detailed design step. We have seen dataflow regions of what are basically repository systems. Now we'll look at the design of what are really pure dataflow software systems.

Delisle and Garlan's Oscilloscope
In 1990 N Delisle and D Garlan published a formal description of the software of a digital oscillioscope which described the process of making a picture (a *trace*) of a signal from the various inputs and adjustable parameters of a modern oscilloscope unit.

The complexities are due to

- the many views (e.g. periodicity, form in a small interval, difference from another signal) which are of interest to an osc. user.
- wide rage of incoming signals and noise, needing hardware and software filters to adjust
- other nonsidplay needs like measurement, store and retrieve forms, arithmetic on forms, interaction with desktop computer, etc.

The data being passed between these dataflow bubbles are

signals, which are synchronized voltage measurements
waveforms, which are data associating time to voltage within a fixed time range
traces, which are horizontal-to-vertical position datasets (for display).
trigger events, which are messages indicating that a sample should be taken

The *couple* bubble performs an initial filter of a signal by scaling according to the average direct-current component. This feature is choosable by parameter, as in some cases (not known to me) the scaling is not required. Hence the bubble has two inputs and one output.

The *acquire* bubble turns a signal into a waveform by sampling and saving a time slice of the signal. It is done during an interval which is selected dynamically by a *trigger* and chosen by a user parameter (as *delay* and *duration*).

The *W->T* bubble turns a waveform into a trace by aligning the start time of the waveform with the horizontal origin and the ground voltage with the vertical origin. Then user parameters for scaling the voltage vertically and time horizontally are applied to produce a trace.

The *clip* bubble turns a trace into a displayable trace by clipping to the display screen parameters.

The *select channel* bubble uses a user input to distinguish between two (or more!) input signals for observing to detect channels. Note the reuse of the *couple* bubble in this pipeline.

The *detect trigger* bubble adjusts its behaviour according to user inputs (*e.g.* slope+voltage) to indicate an event on the (selected) signal which calls for the beginning of acquisition.

It should be possible to imagine trace or waveform data being spooled off to repositories (not in this diagram) or being subject to arithmetic filters.

This technique of building a software system out of filters (processors) and pipes (communication paths) enables concentration on the logic of each subfunction. Mathematically it may be considered as a simple composition of functions.

Web Servers.
This section is based on a paper by A Hassan and R Holt.

A web server is the front of a web site or sites interface to the web.

In the reference architecture presented here, data enters and leaves in the form of HTTP protocol interaction. This is a dataflow architecture in which the bubbles ar e

The *reception* bubble waits for browser requests, parses them, and translates requests to the internal format chosen by the architects for the server as a whole. This bubble also interacts with the browser to learn its capabilities, using this information to annotate the parsed format.

The *request analyser* translates URLs in the request into local file names (where possible) and in some cases performs automatic correction of requests (*e.g.* mistyped URLs).

The *access control* subsystem translates a locally-formatted request to workflow which handles security, for example, by passing information back through to the browser for requesting passwords. In some cases this will result in the internally formatted request being completely changed to some kind of error page.

The *resources handler* subsystem determines the type of resource required, such as a static file, or the initiation of a servlet or script.

A *transaction log* subsystem (not shown) records the requests.

In the diagram of Apache in particular, one may see that this architecture is adapted in two ways: access control is divided into a smaller pipeline of authentication and authorization; and request handling is similarly divided into two subsystems. The authors suggest this is due to the need for open distrfibuted development.

In the diagram of AOL Server, which has a goal of serving databases to the web (using TCL), there is an additional subsystem, the *database interface* for piping data from the databases into the web environment.

The *communication driver* bubble abstracts various communication protocols including HTTP to the rest of the system internally.

The *daemon core* bubble translates incoming requests to a standard internal format.

The *perm* bubble validates permissions and authorization (it is *access control*).

The *URL handle* bubble translates requests into responses.

Communication Styles
These arise from the various ways in which data can be communicated. This section can be interpreted as a high-level discussion of construction techniques: but for something as fundamental as communication, the choice of construction technique will have an big impact on the architecture. You

can't build skyscrapers out of framing lumber and plywood.

Pure Batch

Continuous Streams and Processes
with complete processing
without complete processing

Process Control

Variations
These arise from constraining the topology (connectedness) of the data flow graphs.

No feedback loop

Pipeline

Fan-Out-Only

References
Shaw and Garlan 2.2, 3.1.4, 3.2, 6.2
N Delisle and D Garlan. A formal specification of an oscilloscope. IEEE Software, Sep 1990. See http://www-2.cs.cmu.edu/afs/cs/project/able/ftp/NDDG90/NDDG90.IEEE.pdf
A E Hassan and R C Holt. A Reference Architecture for Web Servers. WCRE 2000. See http://plg.uwaterloo.ca/~aeehassa/home/pubs/wcre2000.pdf
Somerville