Software Architecture

Lecture 2
Data Flow Systems

João Pedro Sousa
George Mason University

last class

- Software Architecture is about representing the structure of a system from different angles: views

- views help manage the complexity of describing the architecture

- viewtypes determine the kinds of things a view talks about
  - three primary viewtypes: module, C&C, allocation

- some styles occur frequently within each viewtype
  - module: decomposition, generalization, layered, ...
  - C&C: pipe & filter, client-server, pub-sub...
  - allocation: deployment, work assignment...
today's outline

- data flow styles
  - process control
  - batch-sequential
  - pipe & filter
    - case study: Tektronix

- lab 1

- pipe & filter sub-styles & implementation

Acknowledgment
some of the material presented in this course is adapted from 17655, taught to the MSE at CMU by David Garlan and Tony Lattanze

in data flow styles

data flow dominates structure and control

- a data flow system is one in which:
  - the structure of the design is determined by the motion of data from component to component
  - the availability of data controls the computation
  - the pattern of data flow is explicit
  - this is the only form of communication between components

- variations on this theme
  - how control is exerted (e.g., push versus pull)
  - degree of concurrency between processes
  - granularity of computation
  - topological restrictions (e.g., pipeline)
data flow styles
assume:

- **connectors** are data streams
  - interfaces are reader and writer roles
  - transport data from writer roles to reader roles
  - unidirectional (usually asynchronous, buffered)

- **components** do not know the identity of upstream/downstream producers/consumers
  - read data from input ports
  - compute
  - write data to output ports

---

in data flow styles
availability of data controls the computation

- any component that has input may process it
- overall data transformation is the "functional composition" of individual transformations

\[ s \xrightarrow{f} g \xrightarrow{h} ? \]

\[ h(g(f(s))) \]
data flow styles
structure is an arbitrary graph

in general, data can flow in arbitrary patterns

however, frequently data flows in "stages"

or in simple, highly constrained cyclic structures

three major data flow styles

- process control
  - looping structure to control environment variables

- batch sequential
  - sequential processing steps, run to completion
  - typical of early MIS applications

- pipe & filter
  - incremental transformation of streams
  - Unix pipes and Yahoo pipes are special cases (sub-styles)
example of process control

open-loop control system

Ambient Air in

Heated Air out

Gas Valve

Chimney

example of process control

closed-loop control system

Thermostat

Chimney

Ambient Air in

Heated Air out

Gas Line
process control notions

- open-loop system: process variables not used to control the system
- closed-loop system: process variables used to control the system
  - controlled variable: goal  
    (ex: air temperature inside the house)
  - set Point: value for the controlled variable
  - input variable: what the system can measure  
    (ex: temperature of the outside air coming into the furnace)
  - manipulated variables: what the system can affect 
    (ex: turning the furnace on or off)
- feedback control  
  controlled variable is measured and taken into account
- feed-forward control  
  process variables other than c.v. are taken into account

architecture of closed-loop process control

```
inputs  
set point value  
control task  
manipulated values  
Process to Control  
controlled value  
feedback  
```

legend
- computation
- data flow
example architecture
heating closed-loop control

outside temp

desired temp

control task

on/off

furnace

temp inside the house

example architecture
very simplified avionics

pilot inputs:
trim
roll
pitch
yaw

altitude
air speed

control task

aileron ctl. process

elevator ctl. process

rudder ctl. process

A to D conv

servo value

legend
computation
data flow

+/- digital positive or negative increment

~ analog signal
three major data flow styles

- **process control**
  - looping structure to control environment variables

- **batch sequential**
  - sequential processing steps, run to completion
  - typical of early MIS applications

- **pipe & filter**
  - incremental transformation of streams
  - Unix pipes and Yahoo pipes are special cases (sub-styles)

---

batch sequential assumes:

- **connectors**
  - data is transmitted as a whole between steps

- **components**
  - processing steps are independent programs
  - each step runs to completion before the next step starts
examples of batch sequential systems

- classical data processing
  - payroll computations
  - IRS tax return computations

- early software development environments

three major data flow styles

- **process control**
  - looping structure to control environment variables

- **batch sequential**
  - sequential processing steps, run to completion
  - typical of early MIS applications

- **pipe & filter**
  - incremental transformation of streams
  - Unix pipes and Yahoo pipes are special cases (sub-styles)
pipe & filter
assumes:
- connectors, called pipes:
  - move data from a filter output to a filter input
  - one-way, order-preserving, data-preserving
  - system action is mediated by data delivery

- components, called filters:
  - incrementally transform the input data to output data
    - enrich data by computation and adding information
    - refine by distilling data or removing irrelevant data
    - transform data by changing its representation
  - operate independently/concurrently among each other
    - no external context in processing streams
    - no state preservation between instantiations
    - no knowledge of upstream/downstream filters
    - topology determines the overall computation, not relative speed/CPU allocation of filters

example of pipe & filter system
telemetry data collection

Telemetry System

Sensors (a few or a few thousand)
- PRESSURE 1
- PRESSURE 2
- TEMP 1
- TEMP 2
- ...
- ETC

Airborne System

Recorder

Ground System

Computer Displays

- frame collection
- time tag frame
- major frame decommutation
- minor frame decommutation
- measurement units
- apply coefficients
- display data
- record data
special case of the pipe & filter style

**Unix pipes**

- **pipes**: buffered streams supported by OS
  - assume ASCII character streams
  - can treat files as well as filters as data sources and sinks
    - the good news: anything can connect to anything
    - the bad news: everything must be encoded in ASCII, if not, it must be "translated" before piping

- **filters**: Unix processes
  - built-in ports: stdin, stdout, stderr
  - filters by default transform stdin to stdout
example of Unix pipes system

```
cat /etc/passwd | grep "joe" | sort > junk
```

does this stretch any assumption of the "pure" pipe & filter style?

special case of the pipe & filter style

**Yahoo pipes**

- web application for authoring *Yahoo pipes*
- *Yahoo pipe* is a data mashup
  - application hosted by Yahoo
  - combine, filter, and present data from different web sources
outline

- data flow styles
  - process control
  - batch-sequential
  - pipe & filter
    - case study: Tektronix
- lab 1
- pipe & filter sub-styles & implementation

major manufacturer of oscilloscopes

Tektronix

signals ➔ oscilloscope ➔ traces and measurements
Tektronix had a problem (real)

- increasing complexity of instrumentation systems
  - increasing role of software in products that were once largely hardware
  - raised expectations of users
- separate development cultures
  - similar products developed by different divisions
  - little sharing
- build-from-scratch methods
  - new hardware or new UI => new software
- products hard to develop and evolve
  - increasingly serious bugs due to concurrency
- excessive time-to-market (~ 4-5 years)

Tektronix challenge

- build next generation instrumentation systems
- allow reuse between product divisions
- sophisticated products in response to user desires
- consistent user interface
  - across multiple hardware platforms
- multiple user interfaces for same platform
  - (vertical markets)
- reduce time to market
Tektronix tried
O-O approach with mixed results

created class hierarchy

- waveform
  - max-min wvfm
  - x-y wvfm
  - accumulate wvfm

**waveform**
- w: time→voltage
- max: →voltage
- min: →voltage
- invert: ...
- add: ...

**result:**
- hundreds of classes
- hard to organize hierarchy
- doesn’t relate to structure of system (oscilloscope)

next, Tektronix tried
layered approach also with mixed results

- Hardware
- Digitization
- Visualization
- User interface

**result:**
- designers found layer boundaries
  - hard to enforce and unrealistic
third, Tektronix tried pipe & filter with better results

result:
- clearly reflects design organization and features
- not clear how to model user input

fourth, Tektronix tried extending p & f with parameterized filters

result:
- clearly reflects design organization and features
- models user input
- not directly useful for implementors
fifth, Tektronix extended p & f with colored pipes

result:
- clearly reflects design organization and features
- models user input
- describes the nature of data and performance constraints on each pipe

software architecture had an impact on Tektronix business

- architectural style was used as basis for the next generation of oscilloscope products
- led to highly successful framework, in which time-to-market was cut substantially
- reliability of products was improved
- user interface was extensible
- new thrust of research/development collaborations
- led to new frameworks beyond oscilloscopes
take 5

outline

- data flow styles
  - process control
  - batch-sequential
  - pipe & filter
    - case study: Tektronix

- lab 1
  - recover views from the code: module, C&C

- pipe & filter sub-styles & implementation
lab 1: pipe & filter system
build avionics instrumentation systems

- data comes in from airplane sensors

<table>
<thead>
<tr>
<th>ID</th>
<th>Data Descriptions and Units</th>
<th>Type</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Time: number of milliseconds since the Epoch (00:00:00 GMT on January 1, 1970)</td>
<td>long int</td>
<td>8</td>
</tr>
<tr>
<td>01</td>
<td>Velocity: airspeed of the vehicle, measured in knots per hour</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>02</td>
<td>Altitude: vehicle’s distance from the average surface of oceans, measured in feet</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>03</td>
<td>Pressure: atmospheric pressure external to the vehicle, measured in PSI</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>04</td>
<td>Temperature: temperature of the vehicle’s hull, measured in degrees Fahrenheit</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>05</td>
<td>Pitch: angle of the nose of the vehicle, if positive, the vehicle is climbing</td>
<td>double</td>
<td>8</td>
</tr>
</tbody>
</table>

- framed as

```
0000  Time  0001  Velocity  ...  n  data
0000  Time  0001  Velocity  ...  n  data
```

lab 1: pipe & filter system
build avionics instrumentation systems

- system A: reads flight data and
  - converts Temp to Celsius
  - converts altitude to meters
  - removes other fields

- system B: same plus
  - includes pressure data
  - removes pressure outliers > 80psi or <50 psi
  - and replaces them by interpolated values (avg of previous and next)

- system C: merges streams from two sets of sensors
  - output frames are sorted by time
lab 1: pipe & filter system
based on a code framework

remember:
- module and C&C viewtypes show different aspects

module view

C&C view

lab 1

module view

Object

Thread

Plumber

OutputStream

InputStream

FilterFramework

SourceFilter

MiddleFilter

SinkFilter

Legend
Java lib
app class
extends (is a)
uses
why is the code written like this?
...a little history

- the Greeks had the following number system:
  - separate letters for numbers between 1 and 9
    (for example, Ι was nine)
  - separate letters for multiples of ten between 10 and 90
    (for example, Π was sixty)
  - separate letters for multiples of one hundred between 100 and 900
    (for example, Τ was 300)
  - so 309 was written 'ΤΙΘ'
- the positional system arrived in 500 AD
  - imagine someone used to the Greek system
    being annoyed with having to write extra digits

why is the positional system successful?
outline

- data flow styles
  - process control
  - batch-sequential
  - pipe & filter
    - case study: Tektronix

- lab 1

- pipe & filter sub-styles
  & implementation

pipe & filter sub-styles & implementation

system level

- topological constraints
  - some styles insist on a pipeline, or no cycles
    - that is, some styles are hierarchical, some aren’t

- strategies for creation of elements
  - centralized versus distributed
  - static versus dynamic
  - declarative versus operational
    (some systems have both)
pipe & filter sub-styles & implementation
components

- concurrency
  - separate processes or single address space

- constraints on ports
  - e.g., Unix components typically have 3 ports: stdin, stdout, stderr
  - e.g., some styles require single input and single output port

- computational model
  - e.g., how do you deal with multiple inputs?

---

pipe & filter sub-styles & implementation
connectors

- finite versus infinite buffering
- dealing with end of data
  - can a writer terminate the flow of data?
  - can a reader choose not to consume data on a pipe?
  - what can a reader or a writer do after data has been terminated?
- does data have different types?
- how many kinds of pipes are there? (colored pipes)
pipe & filter sub-styles & implementation

example: Unix pipes

two kinds of specification
- shell (ex:)
  `inpipe > capitalize | rem_white_space | compress > outfile`
- programmatically:
  use stdio library to create pipes and hook them up

topology
- linear if from shell
- may have cycles if done programmatically

creation
- static if from shell
- dynamic if done programmatically

filters
- concurrency: separate processes
- ports: three for shell filters
- computational model:
  filter chooses which ports to read and write
  (if you guess wrong deadlock may result)

pipes
- supported by operating system
- buffered (e.g., 2K bytes)
- special end-of-file marker
- reader can close its end of a pipe at any time
- must be aware of whether input is coming from a file or a filter
- data has one type only: raw bytes
references for implementation strategies

- book on Java threads
  - *Concurrent Programming in Java: Design Principles and Patterns* (2nd Edition) by Doug Lea

- online tutorial on Java threads

- Design Patterns
  - book by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

in summary

select a data flow style when:

- task is dominated by the availability of data
- data can be moved predictably from process to process

pipe-and-filter architectures are good choices for many data flow applications because

- they permit reuse and reconfiguration of filters
- generally easy to reason about
- reduce system testing
- may allow incremental AND parallel processing

there may be a performance penalty when implementing data flow styles over a single process
in summary
styles are rarely usable in simple pure form

- one technique is to specialize styles
  - styles become more constrained, domain-specific
  - trade generality (expressiveness) for power (analytic capability)
  - we saw this today in the examples of data flow styles

in summary
there are many points on the spectrum of specialization

- we looked at a few example today (stars) but will be looking at lots more during the course