Stencil
A Declarative System for Visualizing Dynamic Data

Joseph A. Cottam
CREST – Indiana University
Objective

Construct a system for working with dynamic data. In that systems, provide abstractions for analysis, abstractions for representation and an efficient runtime. Make this system accessible for integration.
Reading a Stencil Program

stream flowers(sepalL, petalW, sepalW, petalL, species, obs)

layer FlowerPlot
guide
  legend[X: 0, Y: 100] from FILL_COLOR
  axis[guideLabel: "Petal Length"] from Y
  axis[guideLabel: "Petal Width"] from X
from flowers
  ID: obs
  X:* Scale[0,100](petalW)
  Y:* Scale[0,100](petalL)
  FILL_COLOR: BrewerColors(species)
    -> SetAlpha(50,BrewerColors)

REGISTRATION: "CENTER"
SHAPE : "CROSS"
Dynamic Binding

stream flowers(sepalL, petalW, sepalW, petalL, species, obs)

layer FlowerPlot
guide
  legend[X: 0, Y: 100] from FILL_COLOR
  axis[guideLabel: "Petal Length"] from Y
  axis[guideLabel: "Petal Width"] from X
from flowers
  ID: obs
    X:* Scale[0,100](petalW)
    Y:* Scale[0,100](petalL)
  FILL_COLOR: BrewerColors(species)
    -> SetAlpha(50,BrewerColors)

  REGISTRATION: "CENTER"
  SHAPE : "CROSS"
**Dynamic Binding**

```plaintext
layer FlowerPlot
from flowers
  X:* Scale[0,100](petalW)

layer FlowerPlot
from flowers
  X: Scale.map(petalW)
  #data: petalL
from #Render
  X: Scale.query(#data.0)
```

#Render is the stream that controls rendering
Guide Creation

stream flowers(sepalL, petalW, sepalW, petalL, species, obs)

layer FlowerPlot

guide

legend[X: 0, Y: 100] from FILL_COLOR

axis[guideLabel: "Petal Length"] from Y

axis[guideLabel: "Petal Width"] from X

from flowers

ID: obs
X: * Scale[0,100](petalW)
Y: * Scale[0,100](petalL)
FILL_COLOR: BrewerColors(species)

-> SetAlpha(50,BrewerColors)

REGISTRATION: "CENTER"
SHAPE : "CROSS"
1  stream survey(fruit)
2
3  layer plot
4  guide
5  axis from X
6  from survey
7    ID: Count()
8    X:* Rank(fruit) -> Mult(5, Rank)
9    /* X:* Index(fruit) -> Mult(5, index)*/
10   /* X:* Index(fruit) -> Mult(10, index)*/
11   Y: Count(fruit) -> Mult(5, Count)
12   REGISTRATION: "CENTER"
13   FILL.COLOR: Color{150,150,255}
# Counterpart

## Scale between 0 and 1

<table>
<thead>
<tr>
<th>Scale method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale.map(0)</td>
<td>0</td>
</tr>
<tr>
<td>Scale.map(10)</td>
<td>1</td>
</tr>
<tr>
<td>Scale.map(9)</td>
<td>.9</td>
</tr>
<tr>
<td>Scale.map(10)</td>
<td>1</td>
</tr>
<tr>
<td>Scale.query(.8)</td>
<td>.8</td>
</tr>
<tr>
<td>Scale.map(20)</td>
<td>1</td>
</tr>
<tr>
<td>Scale.query(10)</td>
<td>.5</td>
</tr>
<tr>
<td>Scale.map(10)</td>
<td>.5</td>
</tr>
<tr>
<td>Scale.query(100)</td>
<td>-1</td>
</tr>
<tr>
<td>Scale.query(-1)</td>
<td>-1</td>
</tr>
</tbody>
</table>

## Count occurrences

<table>
<thead>
<tr>
<th>Count method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count.map(&quot;page&quot;)</td>
<td>1</td>
</tr>
<tr>
<td>Count.map(&quot;page&quot;)</td>
<td>2</td>
</tr>
<tr>
<td>Count.map(&quot;page&quot;)</td>
<td>3</td>
</tr>
<tr>
<td>Count.query(&quot;page&quot;)</td>
<td>3</td>
</tr>
<tr>
<td>Count.query(&quot;page&quot;)</td>
<td>3</td>
</tr>
<tr>
<td>Count.map(&quot;pic&quot;)</td>
<td>1</td>
</tr>
<tr>
<td>Count.map(&quot;pic&quot;)</td>
<td>1</td>
</tr>
<tr>
<td>Count.map(&quot;pic&quot;)</td>
<td>2</td>
</tr>
<tr>
<td>Count.query(&quot;pic&quot;)</td>
<td>2</td>
</tr>
<tr>
<td>Count.query(&quot;grape&quot;)</td>
<td>-1</td>
</tr>
</tbody>
</table>
Task Parallelism Interleaving

Abstract Activity Interleaving

Classical Architecture

Concurrent Architecture

Stencil Activity Interleaving

Phase Key
- Red: Analysis
- Green: Prerender
- Blue: Render
- Black: Lock

Elapsed Milliseconds
Task Runtime

Runtime (ms) vs. Data points Loaded

Render Timing

- 8000 ms
- Stencil (1n)
Current Projects

• Simplified Runtime
  – Multiple runtimes

• In-situ visualization
  – Directly Shared data
  – Code generation (reducing abstraction costs)

• Simplify custom operator integration