Layout and Animation Language

Why Would I Walk Through Mud?

Ras Bodik, Thibaud Hottelier, James Ide, Doug Kimelman (IBM), Kimmo Kuusilinna (Nokia), Per Ljung (Nokia)
NY Times
How Fancy Layout Are Created Today

Pick a canned layout from ProtoViz (DSL for vis.)
  – Limited to the library.
  – Non-programmers cannot define their own.

If programmer, must write own layout engine.
  – May takes days => can’t quickly try layout ideas.
  – ~10x more code if using Python/JavaScript
Our Language Mud

We choose **declarative programming** because

- Empower designers
  - Designers know the “what”, but not the “how”
- Naturally maps on human thought process for layout/visualization if well designed
- Eventually, programming by demonstration

Our compiler does **synthesis**. Why not use an off-the-shelf constraint solver? Ex: Cassowary[Badros], SMT solvers, Prolog.

- Performance
  - We compile down to tree passes. Linear Time.
  - No search and backtrack.
TreeMap of Financial Industry

NY Times
TreeMap on the Drawing Board

Designer’s View

• Area is market capitalization

• Companies stacked vertically (or horizontally)

• Parent exactly encompasses children
Mud Hello World

1. Document is a tree.
2. We place local constraints

\[ C.w = B.w = A.w / 2 \]

In TreeMap, we have two building blocks
- H, the horizontal divider
- V, the vertical divider
Specification of V

Let’s write the spec in English and then translate it progressively into Mud

English Specification:
1. V is a rectangle with some style.
2. V area is divided vertically among its children.
3. V’s children are stacked on top of each other.
4. V area is proportional to the sum its children’s capitalization.
The Three Constraints

```
trait VDiv(h, w) {  // vertical division
    h = children[0].h + children[1].h
    w = children[0].w = children[1].w
}
trait VStack() {  // vertical stacking
    children.left = 0
    children[0].top = 0
    children[0].h = children[1].top
}
trait TreeMap(h, w, cap) {  // area =~ cap
    SCALE * cap = h * w
    cap = children[0].cap + children[1].cap
}
```
V Is a Composition of Trait

Let’s declare two new building blocks:

```plaintext
let V with RelCoord, BasicBoxStyle, VDiv, VStack, TreeMap
let H with RelCoord, BasicBoxStyle, HDiv, HStack, TreeMap
```

“Trait”: Composable unit of behaviour
Are we done?

Tool: “your treemap is under-constrained”
   – There are distinct ways to lay it out:

   Fix: Root.h = 640
Alternative fix: set the aspect ratio.

Benefits of our semantics:
   – Show possible solutions ==> Designer-friendly debugging
   – Unique solution ==> predictable layout.
Prototyping with Mud

Mud flexibility allows designers to experiment.

Example: Let’s make treemap fixed size!
  – At Root node: $h = 640$, $w = 320$
  – Tool tells us to make the scaling factor a variable

Mud compiler produces the new layout engine
  – New engine requires four, rather than three passes
  – The extra pass computes the right scaling factor.
Animations

How to add animation?

ie, transform the tilemap from Jan to Feb layout?

1. Interpolate Jan-to-Feb capitalization data, obtaining new capitalization for each frame
2. Update the document tree with this data
3. Rerun the layout engine (recompute layout)
Radial Layout

Polar coordinates in Mud

```ruby
trait Polar (x, y, ω, radius) {
  x = parent.x + radius * cos(ω)
  y = parent.y + radius * sin(ω)
}
```
Under the Hood

Specifications
Bi-directional constraints
Declarative/relational

Directional constraints
Operational/functional

Attribute Grammar

Layout Engine

TreeMap  Table  SideBar
Building Blocks

Dependencies

AG compiler [Leo & Adam]

Tree traversals
Ongoing Work

The rubber meets the mud:

- Data visualization: this summer @ Nokia Lab
- GUIs, documents will be next
- Learn how designers would use and debug Mud

Come see the demos at poster session

- Vertical integration of almost entire browser stack
- Give us your ideas for data vis of personal data
Summary

• Declarative programming for designers (data visualization, GUIs, documents).

• Fast layout for big data and small battery. No search, no fixed-point. Instead, linear time, parallel.

• Constraints compiler based on two-step synthesis. Local constraints to functions. Functions to global solver.
That is all folks