Efficient Composition Styles

Franz Achermann, Nathanael Schärli
Software Composition Group
University of Berne
www.iam.unibe.ch/~scg
Overview

• Introduction
  – What is a composition language
  – How do we represent components
  – Provided required services
  – Establishing a connection: call, multiplex, co-routines

• Composition style
  – Generic abstractions

• Problem
  – This gives a performance penalty
  – Example

• Solution: Partial evaluation
  – Separate Side effects
  – Inline Sandbox and root to enable substitution.

• Discussion.
The dream of legos

• What is a composition language
  – How do we represent components?
  – Provided and required services
  – Instances and factories

Comp. A
Bricks and Mortar

• How can we connect?
  – Call Services
    • e.g. one provided to one required service
  – Events
    • e.g. one required to multiple provided services
      – asynchronous, same time …
  – Composition mismatches
Low-level composition

- one
- many
  - blocking
  - concurrent

- fixed - reconfigurable
- default: do nothing
- may add state
  - cache invocations
  - synchronize
Piccola, a pure composition language

- Immutable forms for flexibility
  - Datastructure extensibility
  - Environments
- Functional abstractions expressivity
- Channels and agents concurrency

+ best practice from scripting languages
**Piccola composition layers**

Piccola provides the mechanism needed to define both styles and scripts

<table>
<thead>
<tr>
<th>Applications</th>
<th>components + scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compositional Styles</td>
<td>streams, events, GUI composition</td>
</tr>
<tr>
<td>Standard libraries</td>
<td>basic coordination abstractions, basic object model</td>
</tr>
<tr>
<td>Piccola syntax layer</td>
<td>services, operator syntax, namespaces, built-in types</td>
</tr>
<tr>
<td>Composition primitives</td>
<td>agents, channels, forms</td>
</tr>
</tbody>
</table>
Piccola in a Nutshell

A form consists of a sequence of bindings

```
helloForm =
    Label = "hello world"
    do: println Label
```

A script composes components by invoking services

```
makeFrame
    Title = "Demo"
    makeButton(helloForm) ? Action(helloForm)
```

High level operators hide details of object wiring...
Reuse Composition

• Composition Abstractions
  – Wrappers and connectors
  – Composition Style
  
  – connect by hand… or
  
  F1  |  F2  |  F3  |  asStream(print)
The problem

- External objects almost always have to be adapted.
- Example:
  - adapt core interfaces for numbers, strings, collections from Java and Squeak to Piccola.
  - The wrapper is invoked whenever a boolean passes the language bridge (e.g. 5 times):
    - (a == 1) & (c == d).not()
Wrap Numbers in JPiccola

`asB X: dynamic.wrapper.asBoolean X
dynamic.wrapper.asBoolean this:
    peer = this.peer
    &_& y: asB boolop.and(val = this, vall = y)
    |_| y: asB boolop.or(val = this, vall = y)
select C: boolop.select(val = this, vall = C)
|_: asB boolop.not(val = this)
_==_ y: asB this.equals(val = y)
_&&_ LB: boolop.select(
    val = this
    vall = (true = LB, false: asB this))()
_||_| LB: boolop.select(
    val = this
    vall = (true: asB this, false = LB)))()`
Partial Evaluation

• But… Side effects:
  – Separate side effects from the residual part
    • E.g.: \( a = \text{ch}.\text{send}(\text{asBoolean}(5 == x)) \)

• Inline Sandbox and root to enable substitution
  – sandbox captures the namespace:
    \[
    \text{root} = (\text{root}, \text{load } x), \; \text{openFrame}()
    \]
    \[
    \text{`y1} = \text{load } x, \; (\text{root}, \text{y1}).\text{openFrame}()
    \]
Optimized MakeButton

'y1 =
  'y2 = Host.class("java.awt.Button")
  'y3 = y2.new()
  'y4 = y3.setLabel(val = "Demo")
  'y5 = Host.class("pi.pi...ActionListener")
  'y6 = y5.new(actionPerformed: println "hello world")
  'y7 = y3.addActionListener(val = y6)
  root
'y8 =
  'y9 = Host.class("java.awt.Frame")
  'y10 = y9.new()
  'y11 = y10.setTitle(val = "Demo")
  'y12 = y10.add(val=y1.y3,type="java.awt.Component")
  ... add window listener, pack(), show()

Title = "Demo"
frame = y8.y10
Goals...

• Optimize inter-language wrappers

• Other areas:
  – IDE
  – compile wrappers
  – error detection
  – type inference, required interfaces
Questions...