AOKell

OW Architecture Meeting
Fractal Workshop
June 21, 2005

L. Seinturier, N. Pessemier, L. Duchien
T. Coupaye
Plan

1. Research topics
2. AOKell
3. Conclusion
4. Perspectives
Research Topics

Aspects & components
Joint work with France Telecom R&D (T. Coupaye)

- Component: modularize vertical functionalities
- Aspect: modularize crosscutting functionalities
- Need both (at the component, object and architecture levels)

Current experiences around Fractal
- Aspects in the model: FAC -> Nicolas Pessemier
- Aspects for implementing the Fractal specifications: AOKell
Techniques for implementing Fractal specifications

- Julia: mixins & bytecode engineering (ASM)
- AOKell: aspects for implementing controllers

Expected benefits

- Easier to develop, debug, maintain
- Better integration with IDEs
- Reducing the development time for writing new controllers
- Reducing the learning curve
Requirements for controllers
- Feature injection (e.g. Binding controller interface)
- Interception (e.g. LifeCycle)

Our proposal
- 1 controller = 1 aspect
  - Feature injection = inter-type declaration
  - Interception = code advising
- Controller part of a Fractal component = aspects weaving

- AspectJ
  - « reference » aspect weaver
  - Compile-time weaving (perf ++)
  - Load-time weaving (and run-time in the near future)
  - Tooling, IDE & debugger integration
Aspects implement crosscutting concerns

Best practice in AOP

- Aspects implement the integration logic
- Aspects delegates treatments
Solution

- 1 aspect per controller type
- 7 aspects
  - BC, LC, NC, CC, SC, Factory, Component

- Each aspect delegates the controller logic to a j.i.Object
- Pointcut definition based on a « type system » for controller
Controller « type system »
Controller « type system »
Cont’d
public aspect ANameController {

    private NameController FlatType. _nc;

    public String FlatType. getFcName() {
        return _nc.getFcName();
    }

    public void FlatType. setFcName(String arg0) {
        _nc.setFcName(arg0);
    }

    public NameController FlatType. getFcNameController() { return _nc; }
    public void FlatType. setFcNameController(NameController nc) { _nc=nc; }
}
public aspect ALifeCycleController {

    private LifeCycleController FlatType._lc;

    public String FlatType.getFcState() { return _lc.getFcState(); }
    public void FlatType.startFc() throws IllegalLifeCycleException { _lc.startFc(); }
    public void FlatType.stopFc() throws IllegalLifeCycleException { _lc.stopFc(); }

    pointcut methodsUnderLifecycleControl( FlatType advised ):
        execution( * FlatType+.*(..) ) && target(advised) &&
        ! controllerMethodsExecution() && ! jlObjectMethodsExecution();

    before(FlatType advised) : methodsUnderLifecycleControl(advised) {
        if( advised.getFcState().equals(LifeCycleController.STOPPED) ) {
            throw new RuntimeException("Components must be started before accepting method calls");
        }
    }
}
Full implementation of the Fractal specifications

- API, ADL, template, ...

Fractal/Julia junit conformance tests: 117 ok / total: 131
(14 failed: specific to Julia, or issues in interpreting the specs)

Performances similar to those of Julia

- AOKell: 646 ms
- Julia (optim none): 679 ms
- Julia (optim mergeControllersInterceptorsAndContent): 552 ms

Further work: implementing the same optimization levels as Julia
Project overview
.jar size: 189KB (154 + 35 for aspejrt.jar) (Julia 2.2 180KB)

Applications tested with AOKell
- hw API, ADL, templates, …
- Fractal RMI
- Fractal Explorer
- cache-controller

Further works
- other app (GoTM, Speedo, …)
- Dream controllers
Conclusion

- The experience worked
- Solution similar (perf, size) to Julia’s

Future works
- Experiment the run time weaving features of Aspect (already done with load-time)
- Improve the writing of controller parts hidden dependencies (eg LifeCycle -> Content)
  - Aspectize it (already done by Julia with mix-in)
  - Provide a component oriented implementation
Perspectives
CBSD for controller parts

Expected benefits

➢ ... the same as for component based applications
An architectural description for each controller part types (12)
<definition name="aokell.lib.membrane.primitive.Primitive" 
    extends="LifeCycleControllerType, BindingControllerType, ComponentControllerType, 
    NameControllerType, SuperControllerType" >

    <component name="ComponentController" 
        definition="aokell.lib.control.component.PrimitiveComponentController" />
    <component name="NC" definition="aokell.lib.control.name.NameController" />
    <component name="LC" definition="aokell.lib.control.lifecycle.NonCompositeLifeCycleController" />
    <component name="BC" definition="aokell.lib.control.binding.PrimitiveBindingController" />
    <component name="SC" definition="aokell.lib.control.superc.SuperController" />

    <binding client="this./sComponent" server="ComponentController./sComponent" />
    <binding client="this./sName" server="NC./sName" />
    <binding client="this./sLifecycle" server="LC./sLifecycle" />
    <binding client="this./sBinding" server="BC./sBinding" />
    <binding client="this./sSuper" server="SC./sSuper" />

    <binding client="BC./cComponent" server="ComponentController./sComponent" />
    <binding client="LC./cBinding" server="BC./sBinding" />
    <binding client="LC./cComponent" server="ComponentController./sComponent" />

    <attributes signature="aokell.lib.membrane.MembraneAttributeItf" />

</definition>
Component factory

- Creates an instance of the component
- Creates an instance of the controller part
  (from the ADL description – static Java backend)
Perspectives
CBSD for controller parts

Problem

- Controllers must themselves be controlled
- Interface signature clashes between interface signature of
  - The base level
  - The control level

E.g. binding controller and the BindingController interface

- 1 for managing component bindings
- 1 for managing bindings between controllers

Solution

- Naming conventions on interface names ( // prefix)
- Do we need a 3rd level to control the controller part?
Conclusion

- Feasible
- Existing version of AOKell
  - 12 ADL description of membrane
  - Performances JACBenchmark
    additional cost: +20% (can certainly be reduced)