

Aspects pour l'adaptabilité des applications à base de composants

L. Seinturier

JTE AOP-Systèmes
16 septembre 2005 - Lille



Plan

1. Contexte
2. Fractal
3. AOKell
4. Conclusion

Lionel Seinturier

S2 - 21/11/2003

1. Contexte

Ingénierie du système/intergiciel (*middleware*)

- Objet

Besoins

- configuration
- déploiement
- *packaging*
- assemblage
- dynamique
- gestion de la complexité

2 tendances

- composant
- aspect

Lionel Seinturier

S3 - 21/11/2003

1. Contexte

Aspects pour système/intergiciel

- +sieurs *frameworks* (JBoss AOP, AspectWerkz)
- exemple fondateur Tomcat
- +sieurs *frameworks* visent appli c/s 3 tiers (Spring AOP, JAC)
- + grosses applications aspects
 - expérience autour IBM Websphere [Colyer04]
 - aspectisation ORB CORBA [Zhang04]
 - serveur Web JaWS [Kulesza00]
- gestion de la complexité
- modularité
- gestion de la dynamique

Lionel Seinturier

S4 - 21/11/2003

1. Contexte

Composants pour système/intergiciel

- entités + réutilisables (+ haut niv abstraction que objet)
- connectables (ADL)
- prog par assemblage
- + systématique + vérifiable
- séparation fonctionnel – non fonctionnel

1. Contexte

Composants pour système/intergiciel

- composants applicatifs
 - conteneurs EJB, CCM
 - conteneurs ouverts et extensibles
CCM [Vadet04], EJB [Bartorello01]
- composant dans le *middleware*
 - composants avec un niveau de contrôle
Fractal [Bruneton04], OpenCOM [Clarke01], K-Component [Dowling01]



2. Fractal

Fractal



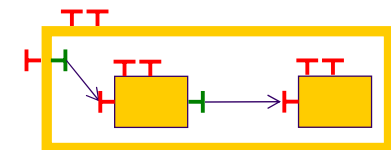
- modèle de composants ObjectWeb (FT R&D – INRIA)
- composants à grain fin (wrt CCM, EJB, .Net/COM+)
- proche modèle classe/interface
- 1 modèle
- différentes implémentations (selon perf., besoins, langages, ...)
 - 1 en C (Think), 3 en Java (Julia, ProActive, AOKell), 1 en Smalltalk (FracTalk), 1 en .Net (FractNet)
- applications Fractal : *middleware*
 - JOnAS à la carte [Abdelatif05]
 - AAA serveur agent mobile / JORAM [Dream]
 - GOTM, Speedo, Comanche, ...

2. Fractal

Fractal

Un composant Fractal - 2 niveaux

- contenu (classe implémentation)
- membrane (niveau méta de contrôle)
- modèle hiérarchique
- autorisant le partage
- introspectable et dynamique
- API de manipulation de composants
- ADL XML de description d'assemblages



2. Fractal

Fractal

Un modèle de composants

- léger
 - hiérarchique
 - généraliste
 - implémentation de réf. Julia performante
- mais niveau de contrôle difficilement adaptable, extensible

3. AOKell

Goal

- customize/extend/adapt the control level (membrane)
 - with AOP
 - with components
- component adaptable to env. with ≠ constraints

Expected benefits

- Easier to develop, debug, maintain
- Better integration with IDEs
- Reducing the development time for writing new controllers
- Reducing the learning curve

3.1 Aspects

Requirements for controllers

- Feature injection (e.g. Binding controller interface)
- Interception (e.g. LifeCycle)

Our proposal

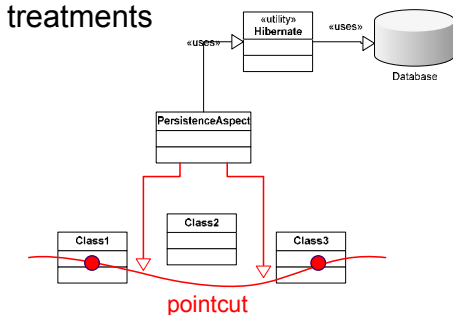
- 1 controller = 1 aspect
 - Feature injection = inter-type declaration
 - Interception = code advising
 - Controller part of a Fractal component = aspects weaving
- AspectJ
 - « reference » aspect weaver
 - Compile-time weaving (perf ++)
 - Load-time weaving (and run-time in the near future)
 - Tooling, IDE & debugger integration

3.1 Aspects

Aspects implement crosscutting concerns

Best practice in AOP

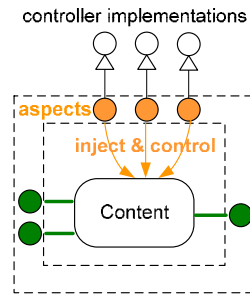
- Aspects implement the integration logic
- Aspects delegates treatments



3.1 Aspects

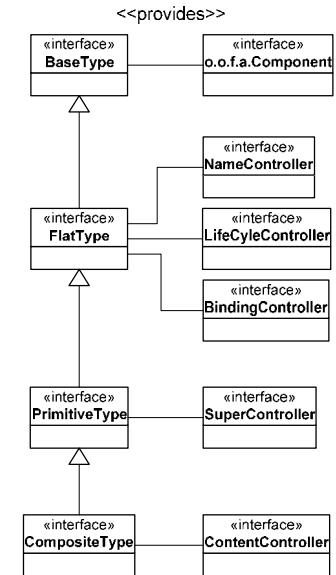
Solution

- 1 aspect per controller type
- 7 aspects
 - BC, LC, NC, CC, SC, Factory, Component
- Each aspect delegates the controller logic to a j.l.Object
- Pointcut definition based on a « type system » for controller



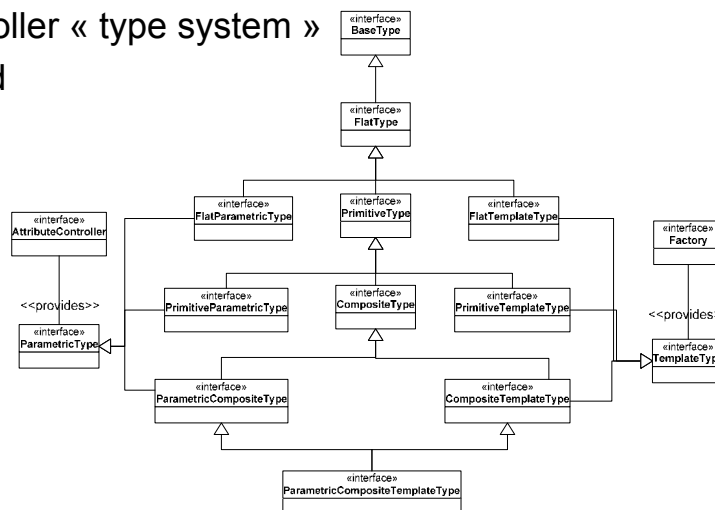
3.1 Aspects

Controller « type system »



3.1 Aspects

Controller « type system »
Cont'd



3.1 Aspects

```
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; }
}
```

AspectJ
Inter-type declarations

Object implementation
of the name controller

3.1 Aspects

```
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() && ! jObjectMethodsExecution();

    before(FlatType advised) : methodsUnderLifecycleControl(advised) {

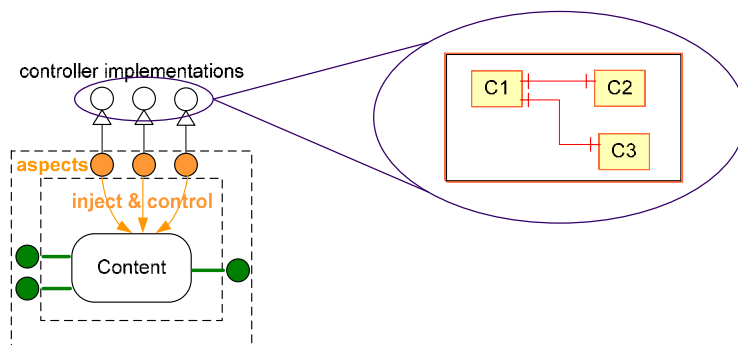
        if( advised.getFcState().equals(LifeCycleController.STOPPED) ) {
            throw new RuntimeException("Components must be started before
                accepting method calls");
        }
    }
}
```

3.2 Components

Goal

- customize/extend/adapt the control level (membrane)
 - with AOP
 - with components
- component adaptable to env. with ≠ constraints

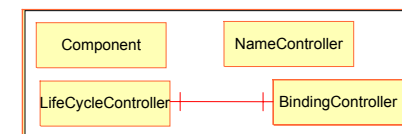
3.2 Components



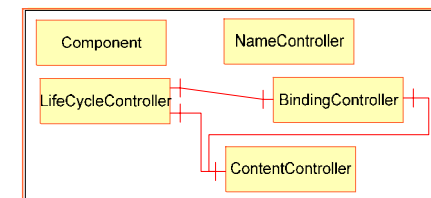
3.2 Components

An architectural description for each controller part types (12)

primitive



composite



3.2 Components

```

<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" />

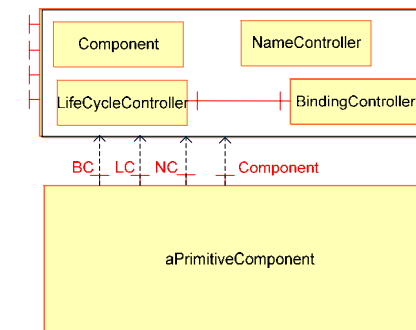
  <attributes signature="aokell.lib.membrane.MembraneAttributeIrf" />
</definition>

```

3.2 Components

Component factory

- Creates an instance of the component
- Creates an instance of the controller part



4. Conclusion

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

JACBenchmark

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

Further work: implementing the same optimization levels as Julia

4. Conclusion

Project overview

.jar size: 189KB (154 + 35 for aspectjrt.jar) (Julia 2.2 180KB)

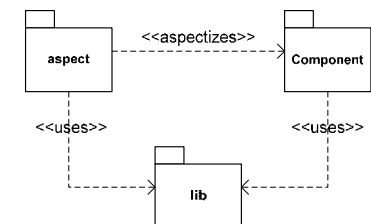
Applications tested with AOKell

- hw API, ADL, templates, ...
- Fractal RMI
- Fractal Explorer

- cache-controller

Further works

- other app (GoTM, Speedo, ...)
- Dream controllers



4. Conclusion

- Solution similar (perf, size) to the reference implementation (Julia)
- Component model for middleware
- with a level of control
 - engineered with aspects and components