Towards a Reference Model for Implementing the Fractal Specifications for Java and the .NET Platform

Fractal Workshop
ECOOP 2006

L. Seinturier – N. Pessemier
INRIA & Univ. of Lille, France

C. Escoffier – D. Donsez
LSR & Univ. of Grenoble, France

This work is partially funded by France Telecom under the external research contract #46131097

Plan

Introduction
Platform architecture
Implementations
Conclusion and future directions

Assumption: basic knowledge of the Fractal component model
1. Introduction

- Several existing implementation of the Fractal Specifications
  - Java (Julia, ProActive), C (Think), SmallTalk (FracTalk), C++ (Plasma)

- These platforms share the Fractal API
  - provide a compile-time compatibility of application components

- Extending these platforms (e.g. with new controllers)
  - so far a matter of understanding the internals of the platform

- Goal: provide a common ground for platform developers
  - Experiment: a Java and .NET implementation of the Specifications

2. Platform Architecture

The role of a platform
- provide an implementation for the Fractal API
- provide a way of implementing control membranes

- membranes are implemented as a set of controllers

Controllers perform
- code injection  ⇒ adding functionalities to components
- code interception  ⇒ modifying the behavior of existing functionalities

⇒ The membrane acts as a container for components

Issue for platform implementors
- how to engineer these containers
2. Platform Architecture

Example: Julia
- mixin
- bytecode engineering (ASM)

Back to the basics

Controllers perform
- code injection ⇒ adding functionalities to components
- code interception ⇒ modifying the behavior of existing functionalities

Candidate technologies
- generation and transformation
  - code or bytecode
  - compile-time or load-time or run-time
- MOP
- AOP
2. Platform Architecture

Aspect-Oriented Programming [Kiczales 97]

Our proposal
- 3 level architecture
- 1 aspect per controller

controllers
implementation

supervises & controls

aspects

component
implementation

Control logic

Glue logic

Application logic
delegates to

Example: AspectJ
- inter-type declaration
- pointcut and advice

public aspect ALifeCycleController {
    private LifeCycleController LCType._lc;

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

    pointcut methodsUnderLifecycleControl( LCType advised ): execution( * LCType+.*(..) ) && target(advised) && ! controllerMethodsExecution() && ! jlObjectMethodsExecution();
    before(LCType advised) : methodsUnderLifecycleControl(advised) {
        if( advised.getFcState().equals(LifeCycleController.STOPPED) ) {
            throw new RuntimeException("Components must be started before accepting method calls");
        }
    }
}
3. Implementations

Instantiating the 3-layer architecture

Implementations

<table>
<thead>
<tr>
<th>Java (AOKell)</th>
<th>.NET (FractNet)</th>
</tr>
</thead>
</table>

Control logic
Application logic
Glue logic

Component implementation

Delegates to
Supervises & controls

Aspects

3. Implementations

Code structure

<table>
<thead>
<tr>
<th>glue</th>
<th>component</th>
</tr>
</thead>
<tbody>
<tr>
<td>AspectJ / AspectDNG</td>
<td>Bootstrap component</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AOKell</th>
<th>FractNet</th>
<th>% (base of source code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>glue</td>
<td>AspectJ</td>
<td>12%</td>
</tr>
<tr>
<td></td>
<td>or Spoon</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AspectDNG</td>
<td></td>
</tr>
<tr>
<td>component</td>
<td>Java</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td>tools</td>
<td>83%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5%</td>
</tr>
</tbody>
</table>
3. Implementations

Control logic
- written in Java
- javac or J# compiler

2 versions
- pure OO controllers
- componentized membranes
  - control components
  - control membranes
  - Fractal API and ADL

Glue logic
- J2SE: written with AspectJ
- .NET: written with AspectDNG
  - input: MSIL assembly (outputed by any .NET language compiler)
  - output: a woven MSIL assembly
4. Conclusion

A common ground for 2 versions of the Fractal platform

- **Java: AOKell**
  - [fractal.objectweb.org](http://fractal.objectweb.org)
  - solution comparable (perf., code size) to Julia
  - fully compatible (JUnit tests passed)
  - able to run existing applications
    (comanche, GoTM, Fractal Explorer, …)
  - can be compiled for ≠ targets: J2SE, J2ME CDC, J2ME CLDC

- **.NET: FractNet**
  - [http://www-adele.imag.fr/fractnet/](http://www-adele.imag.fr/fractnet/)
  - a first step towards the .NET world
  - pending work
    - Fractal ADL, unit testing

4. Future directions

Evolution of the glue logic

- moving from AspectJ to Spoon  
  - [http://spoon.gforge.inria.fr](http://spoon.gforge.inria.fr)
  - Java source-to-source transformer

- rationale
  - performance (weaving and weaved code)
  - towards a CT convergence of AOKell & Julia
    - reusing Julia controller with AOKell
    - a Spoon version of the Julia mixin algorithm
4. Future directions

Issue: Julia-AOKell interoperability

- controller interoperability
- component interoperability

- controller interoperability
- both Julia and AOKell define their own Controller interface
- candidate for the Fractal API v3? for a new API (so-called SPI)?

- component interoperability
- e.g. an heterogeneous assembly with Julia and AOKell components
- issue: « internal » API extending the Fractal API
  - LifeCycleCoordinator extends LifeCycleController
  - SuperControllerNotifier extends SuperController
  - Template extends Factory
  - ComponentInterface extends Interface
  - ContentBindingController (new interface)

- stick to this API to provide Fractal component interoperability
- candidate for the Fractal API v3? for a new API (so-called SPI)?