

Engineering Component Frameworks for Middleware Platforms

Lionel Seinturier
INRIA ADAM

Adapt@Lille – 03/13/2008

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE



CENTRE DE RECHERCHE LILLE - NORD EUROPE



A word of presentation

Lionel Seinturier

- Pr @ U. Lille – INRIA – ADAM team
- Research & teaching activities: middleware, OOP, components, aspects

Past project

- JAC : AO middleware platform

<http://jac.ow2.org>



Current projects

- AOKell : AO component framework <http://fractal.ow2.org/aokell>
- CFs for middleware engineering



Plan

1. Context
2. Background
3. Components & aspects: AOKell
4. Components & Service Oriented Middleware Platform
5. Conclusion



1. Context

In the past (90s): software engineering for middleware « à la CORBA »

- object

New requirements

- configuration
- deployment
- packaging
- assembling
- dynamicity
- management of interaction and dependencies

Present directions

- component, aspect, MDE, reflexivity



1. Context

Component vs object for system/middleware

- reusability
- higher abstraction level
- which can be assembled (ADL)
- separation between functional and non functional concerns



1. Context

Software component

- [McIlroy 68]
- 30 years later: Sun EJB, OMG CCM, MS .NET/COM+, ...
- [Szyperski 02]: 11 definitions +/- ≡

A component is a unit of composition with **contractually specified interfaces** and **context dependencies** only. A software component can be **deployed** independently and is subject to **composition** by third parties. [Szyperski 97]



1. Context

Numerous component frameworks (CF)

- 20+
- many build on top of Java (but also C/C++)
- EJB, Java Beans, CCM, COM+, JMX, OSGi, SCA, CCA, SF
- Fractal, K-Component, Comet, Kilim, OpenCOM, FuseJ, Jiazzi, SOFA, ArticBeans, PECOS, Draco, Wcomp, Rubus, Koala, PACC-Pin, OLAN, Newton, COSMOS, Java/A, HK2
- Bonobo, Carbon, Plexus, Spring
- analysis/design: UML2



1. Context

Numerous component frameworks lead to

Different terminologies

- component, bean, bundle
- interface/binding, port/connector, facet, sink, source
- required/provided, client/server, export/import, service/reference
- container, membrane, technical services, controller
- framework, application servers

High variability in properties attached to these notions

- Fractal : component, interface, binding, client/server
- CCM : component, facet, port, sink, source
- UML 2 : component, fragment, port, interface
- OSGi : bundle, imported/exported package, service/reference

Same term with different meaning across component frameworks

- notions need to be qualified (« connector in the sense of... »)
- equivalences not always easy to define

1. Context

Classifying CFs for system/middleware

Application

- EJB, CCM, .NET/COM+, SCA, Spring

Application

Middleware

- Fractal, JMX, OpenCOM, OSGi

Middleware

Component-based middleware for component-based applications

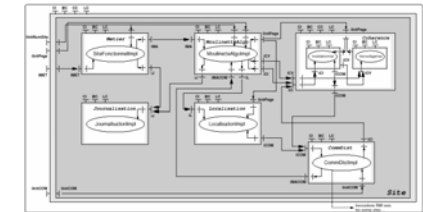
- Fractal-EJB [Abdellatif 05]
- OSGi-EJB [Desertot 06]

1. Context

Software Architecture

A software architecture of a program or computing system is the structure or **structures** of the system, which comprise **software components**, the externally visible **properties** of those components, and the **relationships** among them. [Bass 98]

- language: ADL
- many XML-based
- survey: [Medvidovic 00]



1. Context

Complementarity

- architecture : top-down (architecture build from components)
- components : bottom-up (assembled with other components)

Component frameworks for middleware

- share (more or less) the same concepts
- but
 - target different domains and execution contexts
 - with various degrees of technical (non-fonctional) services
 - technical services are seldom adaptable (hard coded in the CF)

Problem statement

- **tailor the technical services to let the CF be adapted**

2. Background

Fractal component model

[Bruneton, CBSE 2004 & SPE 2006]

- ObjectWeb consortium for open-source middleware
- component, interface (provided/required), binding
- hierarchical with sharing
- dynamically introspectable and reconfigurable
- XML-based architecture description language
- focusing on the development of middleware building blocks
 - e.g. Joram JMS server, JOnAS J2EE server, Speedo JDO, GoTM transaction monitor, Comanche HTTP server, Petals JBI ESB
- <http://fractal.ow2.org>



2 dimensions

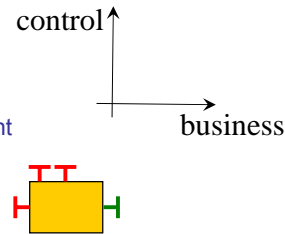
- business
- control

2. Background

Fractal component model

Control dimension

- the non functional properties of a component
- defined in a so-called control membrane
- composed of a set of controllers
- accessible via control interfaces



Programming language independant

- Java (Julia, ProActive), SmallTalk (FracTalk), C (Think), C++ (Plasma), .NET (FracNET)

Limitation

- control dimension can not be extended without modifying the CF

2. Background

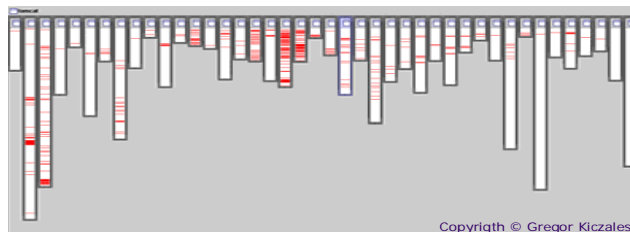
2 point of views on Fractal

- a component framework
 - a programming model for component-based applications
 - an API
 - some tools (runtime, ADL, console, libraries, ...)
- a framework for component frameworks
 - general principles (control interface, controller, interceptor, membrane)
 - to build CFs (aka personalities)
 - with ≠ programming model and/or ≠ non functional properties for e.g. SCA, OSGi, JBI, Fractal, ...

2. Background

Aspect Oriented Programming

- software engineering modularization technique [Kiczales 97]
- reduce code scattering and code tangling
- notion of an aspect which modularizes a crosscutting feature
- compile-time and run-time aspect weavers
- many available weavers: AspectJ, JBoss AOP, JAC, ...



Copyright © Gregor Kiczales

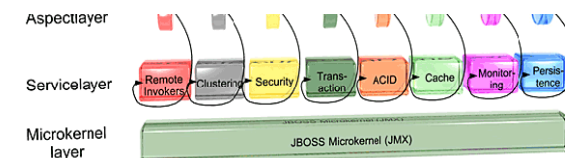
2. Background

Existing approaches based on components and aspects

e.g. JBoss J2EE application server & the JBoss AOP framework

Limitations

- services are still programmed as objects (not components)
- coarse-grain services (« à la EJB »)



Plan

1. Context
2. Background
- 3. Components & aspects: AOKell**
4. Components & Service Oriented Middleware Platform
5. Conclusion

3. AOKell

AOKell: our contribution for opening the Fractal CF

Existing controllers (technical services) in the Fractal CF

- binding controller : communication chanel with other components
- attribute controller : configurable properties of a component
- lifecycle controller : policy for starting/stopping services
- content, super : sub/super component management
- name : naming
- component : component identity and type
- template : component instantiation

Note: many semantic variations exist around these functionalities (local, distributed, ...)

3. AOKell

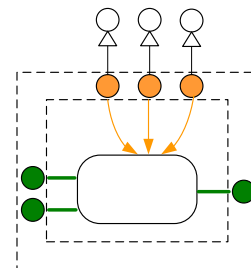
AOKell: our contribution for opening the Fractal CF

Existing controllers (technical services) in the Fractal CF

- binding
- attribute
- lifecycle
- content, super
- name
- component
- template

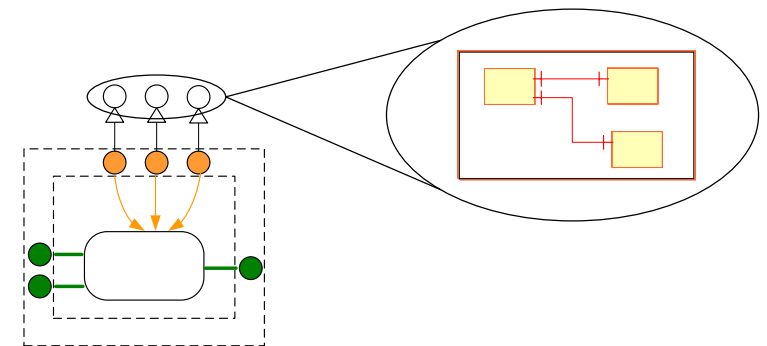
1 AspectJ aspect per controller

- glues the control dimension with the application dimension
- delegates to the controller implementation



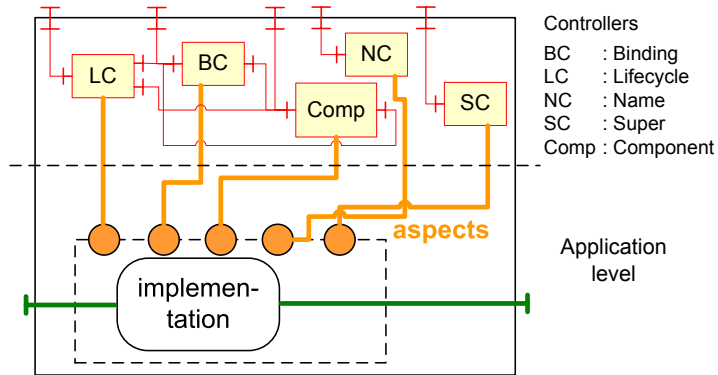
3. AOKell

The implementation of the control can benefit from components too



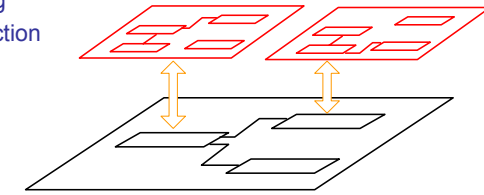
3. AOKell

The implementation of the control can benefit from components too



3. AOKell

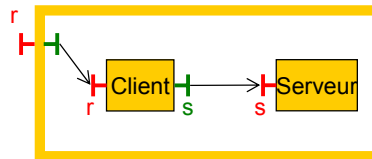
- control component
 - regular Fractal component providing a non functional property for an application level component
- membranes as assemblies of control components
- aspects to glue control components with application level components
 - code advising
 - code introduction



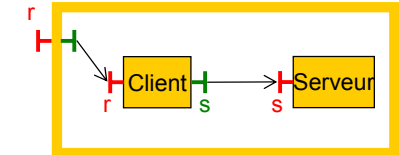
3. AOKell

Note: the same concept and tools (API, ADL) are used at the application and control levels

Example: ADL



3. AOKell

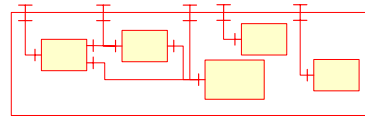


```

<definition name="HelloWorld">
  <interface name="r" role="server" signature="Runnable" />
  <component name="Client">
    <interface name="r" role="server" signature="Runnable" />
    <interface name="s" role="client" signature="Service" />
    <content class="ClientImpl" />
  </component>
  <component name="Server">
    <interface name="s" role="server" signature="Service" />
    <content class="ServerImpl" />
  </component>
  <binding client="this.r" server="client.r" />
  <binding client="client.s" server="server.s" />
</definition>
    
```



3. AOKell



```

<definition name="org.objectweb.fractal.okell.lib.membrane.primitive.Primitive"
<!-- Type of provided interfaces skipped for clarity sake -->

<component name="Comp" definition="PrimitiveComponentController" />
<component name="NC" definition="NameController" />
<component name="LC" definition="NonCompositeLifeCycleController" />
<component name="BC" definition="PrimitiveBindingController" />
<component name="SC" definition="SuperController" />

<!-- Delegation bindings of provided interfaces skipped for clarity sake -->
<binding client="Comp//binding-controller" server="BC//binding-controller" />
<binding client="BC//component" server="Comp//component" />
<binding client="LC//binding-controller" server="BC//binding-controller" />
<binding client="LC//component" server="Comp//component" />

<controller desc="mComposite" />
</definition>

```



3. AOKell

Fractal bootstrap component



Instantiating a component

- creating the content
- instantiating the composite control component
- gluing them together with aspects (one per control component)

Issues: controlling the control components?

- control component control themselves (meta-circularity)?
- ad-hoc implementation of the (meta)-control ⇒ chosen solution



3. AOKell

Applications

Control membranes for

- discrete event simulation [INRIA Mascotte]
- adaptation of legacy components for numerical computation [INRIA PARIS]
- RTSJ [A. Pišek PhD Thesis – ISORC08]
- active components for soft real time [F. Loiret PhD Thesis]



3. AOKell

Performance evaluation

- application level service invocations: unchanged
- control level service invocations
 - costlier but more adaptable
 - assumed to be a lot less frequent than app. level service invocations
- memory footprint
 - larger than in other version of the Fractal CF
 - in progress work
 - different implementation strategies for the control membranes
 - inlining



3. AOKell

Short conclusion on AOKell

- allows adapting the technical services provided by a CFs
- use component (so called control component) and assemblies to implement technical services
- use aspect to glue control components with application level components

Future work

- complex forms of control architectures spanning many components
- use the approach for coarser-grained component containers (e.g. SCA, JBI)

L. Seinturier, N. Pessemier, L. Duchien, T. Coupaye. *A Component Model Engineered with Components and Aspects*. CBSE'06. LNCS 4063, pp. 139-153. June 2006.



Plan

1. Context
2. Background
3. Components & aspects: AOKell
4. Components & Service Oriented Middleware Platform
5. Conclusion



4. Components and SOA

Current distributed (CORBA, .NET, Java EE, etc.)
often complex, rigid and monolithic

SOA trend (Service Oriented Architecture)

identified requirements

- well-defined interfaces with a business related semantics
- **standardized** communications protocols
- flexible (re)composition of services to promote software flexibility

"the vision" [Consortium Open SOA – www.osoa.org]

- *A service is an abstraction that encapsulates a software function*
- *Developers build services, use services and develop solutions that aggregate services*
- *Composition of services into integrated solutions is a key activity*



4. Components and SOA

SOA

architectural principles

- weak coupling
 - *Components integrate with other components without needing to know how other components are implemented*
- flexibility
 - *Components can easily be replaced by other components*
- services
 - *Services can be easily invoked either synchronously or asynchronously*
- composition
 - *Composition of solutions clearly described*
- productivity
 - *Easier to integrate components to form composite application*

often organized around Web Services (but not necessarily)

- SOAP (HTTP, XML), WS-*, BPEL orchestration

4. Components and SOA

SCA : a component model for SOA

Initiative: IBM, Oracle, BEA, SAP, Sun, TIBCO, ...

Goal: to structure SOA architectures

1st specification: 12/2005, v1 03/2007

Implementations: Apache Tuscany, Newton, Fabric3

Assembly model

- how to define structure of composite applications

Component implementations specifications

- how to write business services in particular languages
- Java, C++, PHP, Spring, BPEL, EJB SLSB, ...

Binding specifications

- how to access services
- Web services, JMS, RMI IOP, REST, ...

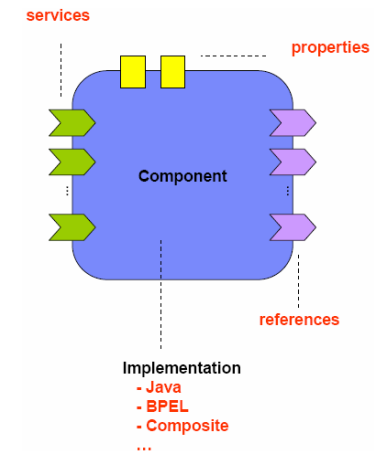
Policy framework

- how to add infrastructure services
- security, transactions, reliable messaging, ...

4. Components and SOA

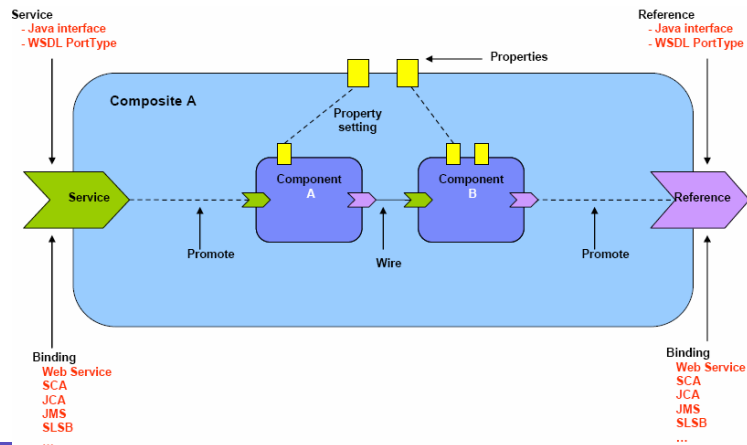
SCA Component

service/reference
property
implementation



4. Components and SOA

SCA Assembly



4. Components and SOA

SCA Assembly

service/reference

- type
 - interface definition language (IDL) : Java or WSDL
- *binding* : in relation with a (remote) communication protocol
 - Web Service, RMI-IOP, JMS, intra-JVM local call

wire

- reference to service
- service/service or reference/reference : so called promotion link



4. Components and SOA

SCA

yet another component model

- a way to revisit
 - remote communication with distributed entities
 - services for middleware platform
 - component framework concepts
- focus on the idea that services and implementations are decoupled

originality: independence between interfaces and communication protocols

post CORBA, .NET technology

- yet another experiment towards the quest for the universal middleware to define communication for new applications and legacy code

similar to CORBA, .NET

- emphasis on the programming model (*software engineering*) for applications rather than on the architecture of the underlying middleware platform (*product dependant*)



4. Components and SOA

Programming model

- Assembly model
 - ADL basé XML
- Component implementations
 - 1 set of rules per programming language
 - Java : uses Java 5 annotations

```
@Service(AccountService.class)
```

```
public class AccountServiceImpl implements AccountService {
    @Reference public AccountDataService accountDataService;
    @Reference public StockQuoteService stockQuoteService;
    @Property public String currency;
    public AccountReport getAccountReport(String s) { ... }
}
```



4. Components and SOA

OSOA 1.0 API

SCA Service Component Architecture Java Common Annotations and APIs

RequestContext, ComponentContext

ServiceReference, CallableReference

Conversation

@Reference, @Property

@Scope

- @Init, @Destroy, @EagerInit
- @Conversational, @EndsConversation, @ConversationID, @ConversationAttributes

@OneWay

@Context, @ComponentName

@Callback

@Constructor



4. Components and SOA

SCA personality vs Fractal personality

dependency injection

- SCA : @Reference @Constructor
- Fractal : BindingController

property injection

- SCA : @Property
- Fractal : AttributeController

component identity

- SCA : @Context + ComponentContext
- Fractal : Component

component name

- SCA : @ComponentName
- Fractal : NameController



4. Components and SOA

SCA personality vs Fractal personality

component initialization

- SCA : ctor + @Init + @EagerInit
- Fractal : ctor

component destruction

- SCA : @Destroy
- Fractal : *missing*

stub for accessing components

- SCA : ServiceReference - CallableReference
- Fractal : Interface (+ serializable)

component instantiation policy

- SCA : @Scope per request, per client, singleton, stateless
- Fractal : singleton



4. Components and SOA

SCA personality vs Fractal personality

asynchronous method invocation

- SCA : @OneWay
- Fractal : see Dream

callback

- SCA : @Callback + CallableReference
- Fractal : *missing*

conversation management

- SCA : @Conversation, @ConversationAttributes, @ConversationId, @EndsConversation
- Fractal : *missing*



4. Components and SOA

Membrane which provides a control semantics to achieve a SCA personality for a component

4 controllers + 1 interceptor

4 controllers

SCAComponent : component identity

SCAContentController : component instantiation policy

SCALifeCycleController : component initialization (@EagerInit)

SCABindingController : component bindings



4. Components and SOA

SCAComponent : component identity
dedicated interface (ComponentContext) and implementation

«interface»
ComponentContext
+getURI()
+cast()
+getService()
+getServiceReference()
+getProperty()
+createSelfReference()
+getRequestContext()

SCAContentController : component
instantiation policy
dedicated implementation,
private interface (no need to export it)

«interface»
SCAContentController
+getFcContent() : Object
+releaseFcContent(in content : Object)

SCALifeCycleController : component initialization (@EagerInit)
same interface as Fractal LC, dedicated implementation

SCABindingController : component bindings
same interface as Fractal BC, dedicated implementation

4. Components and SOA

interceptor: related to component instantiation policy

scaPrimitive membrane type

- 4 SCA-specific controllers
- + Fractal/Julia controllers: Component, NC, SC

a scaPrimitive component is a Fractal primitive component which can be assembled with other Fractal components

4. Components and SOA

Controllers

SCACmpCtx : sca-component-controller

SCACC: sca-content-controller

SCAPC: sca-property-controller

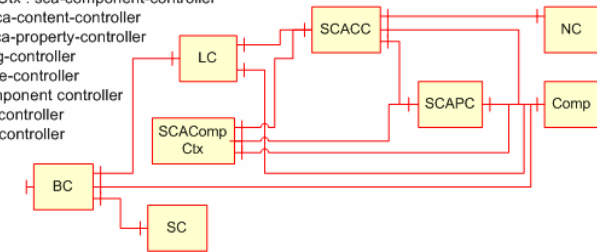
BC: binding-controller

LC: lifecycle-controller

Comp: component controller

SC: super-controller

NC: name-controller



5. Conclusion

- don't re-invent yet another component model (tools, API, frmwk)
- think in terms of adaptability of
 - the component model
 - the technical services which are implied by this model
- towards the notion of a profile for component frameworks
 - much like the notion of a profile for UML metamodeling
 - being able to derive differents personalities of components
- towards a better interoperability between component frameworks