Webinar

Fractal in Java

Lionel Seinturier

University of Lille – LIFL & INRIA – ADAM

http://fractal.ow2.org/java

fractal@ow2.org

April 21, 2009

The Fractal Component Model

From the previous Webinar…

- FT R&D, INRIA
- *open source*
- http://fractal.ow2.org

- main target domains: system and middleware
- general enough for any other application domain
- fine-grained (wrt EJB or CCM), close to classes
- lightweight
- independent from programming languages

- homogeneus view of layers (OS, *middleware*, services, applications)
  - Fractal *everywhere*
- in order to unify and facilitate
  - design, development, deployment, management

Outline

1. Developing with Fractal in Java
   - 1.1 Fraclet
   - 1.2 Fractal ADL
   - 1.3 Fractal API
2. Platforms
   - 2.1 Julia
   - 2.2 AOKell
3. Conclusion

1. Developing with Fractal in Java

3 complementary tools

- Fraclet
  - annotation based programming model
- Fractal ADL
  - XML-based architecture description language (ADL)
- Fractal API
  - Java API for dynamically
    - introspecting
    - reconfiguring

1.1 Fraclet

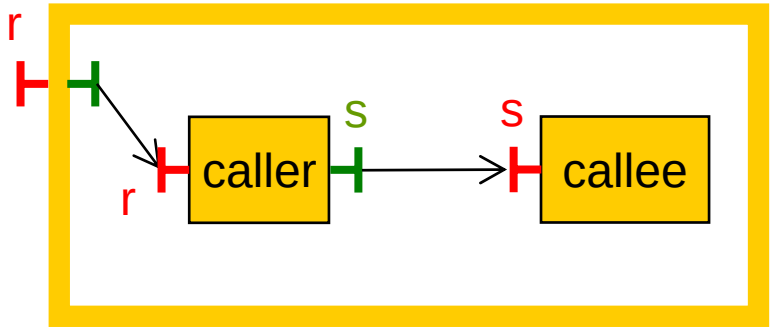
Main annotations

- annotation based programming model (Java 5 or XDocLet)
- @Component
  - apply on the implementation class of a component
  - 2 optional parameters
    - name : component name
    - provides : services provided by the component
- @Requires
  - apply on a field : reference to the required service (of type T)
    - field of type T for singleton (1-1) references
    - field of type Map<String,T> for multiple (1-n) (COLLECTION) references
  - 3 optional parameters
    - name : component interface name
    - cardinality : SINGLETON (default) or COLLECTION
    - contingency : MANDATORY (default) or OPTIONAL

1.1 Fraclet

Example: Hello World in Fractal

- root composite component with 2 sub-components
- sub-component callee providing an interface
  - named s
  - of type `interface Service { void print(String msg); }`
- sub-component caller providing an interface
  - named r
  - of type `java.lang.Runnable` (*de facto* Fractal convention)
  - delegated at the level of the composite component
- caller requires the service provided by callee



1.1 Fraclet

Hello World in Fractal – The Callee Component

```
package hw;

@Component(
   provides=
      @Interface(name="s",signature=Service.class) )
public class CalleeImpl implements Service {
   public void print( String msg ) {
      System.out.println(msg);
   }
}
```

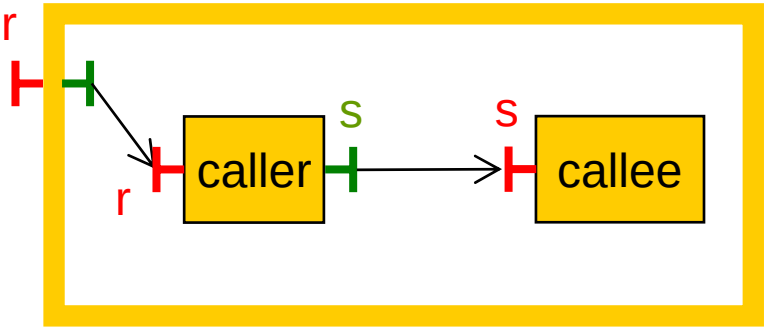
1.1 Fraclet

Hello World in Fractal – The Caller Component

```
package hw;

@Component(
   provides=
      @Interface(name="r",signature=Runnable.class) )
public class CallerImpl implements Runnable {

   @Requires(name="s")
   private Service service;

   public void run() {
      service.print("Hello world!");
} }
```

1.1 Fraclet

Hello World in Fractal – Assembling

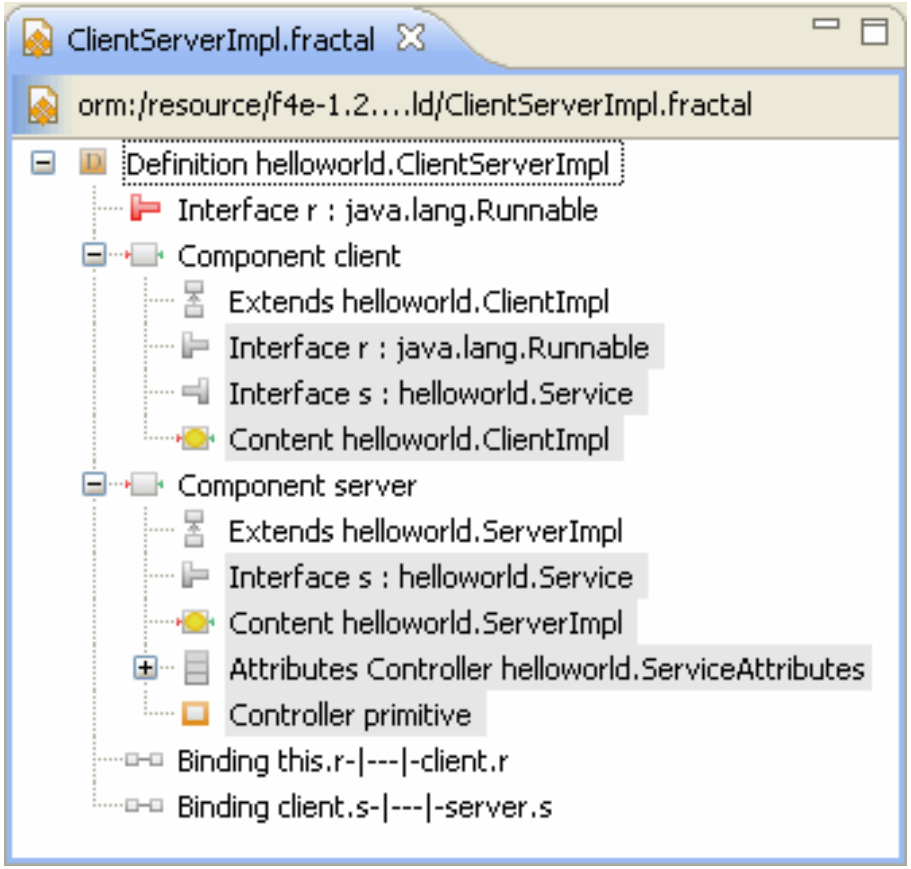


Fractal ADL

```
<definition name="hw.HelloWorld">

  <interface name="r" role="server"
             signature="java.lang.Runnable" />

  <component name="caller" definition="hw.CallerImpl" />

  <component name="callee" definition="hw.CalleeImpl" />

  <binding client="this.r" server="caller.r" />
  <binding client="caller.s" server="callee.s" />

</definition>
```

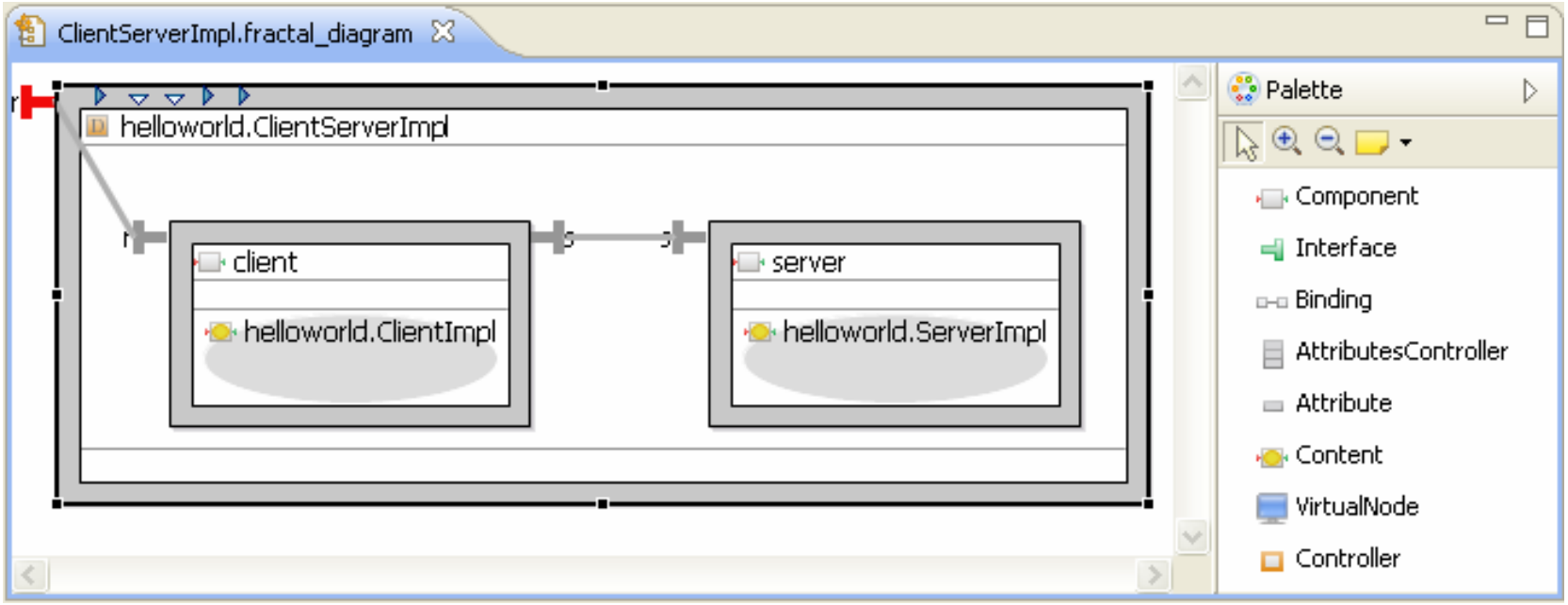
Hello World in Fractal – Running

- Command line
- Eclipse
  - F4E
  - Fractal project
  - component programming
  - ADL editing
  - http://fractal.ow2.org/f4e



Hello World in Fractal – Running

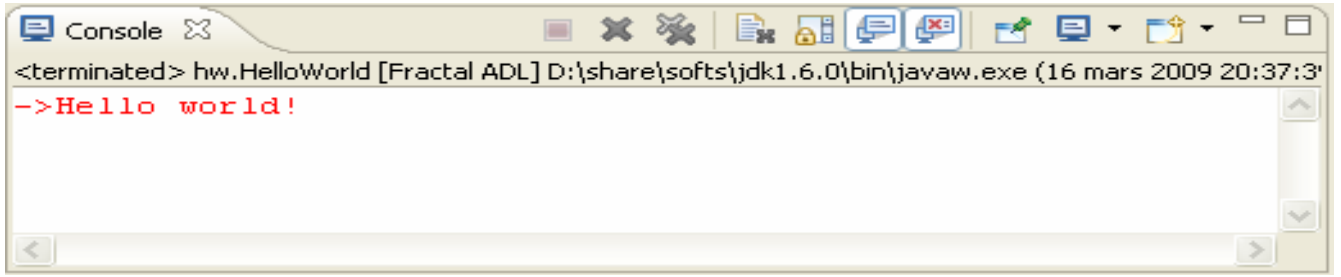
- GMF based graphical editor



1.1 Fraclet

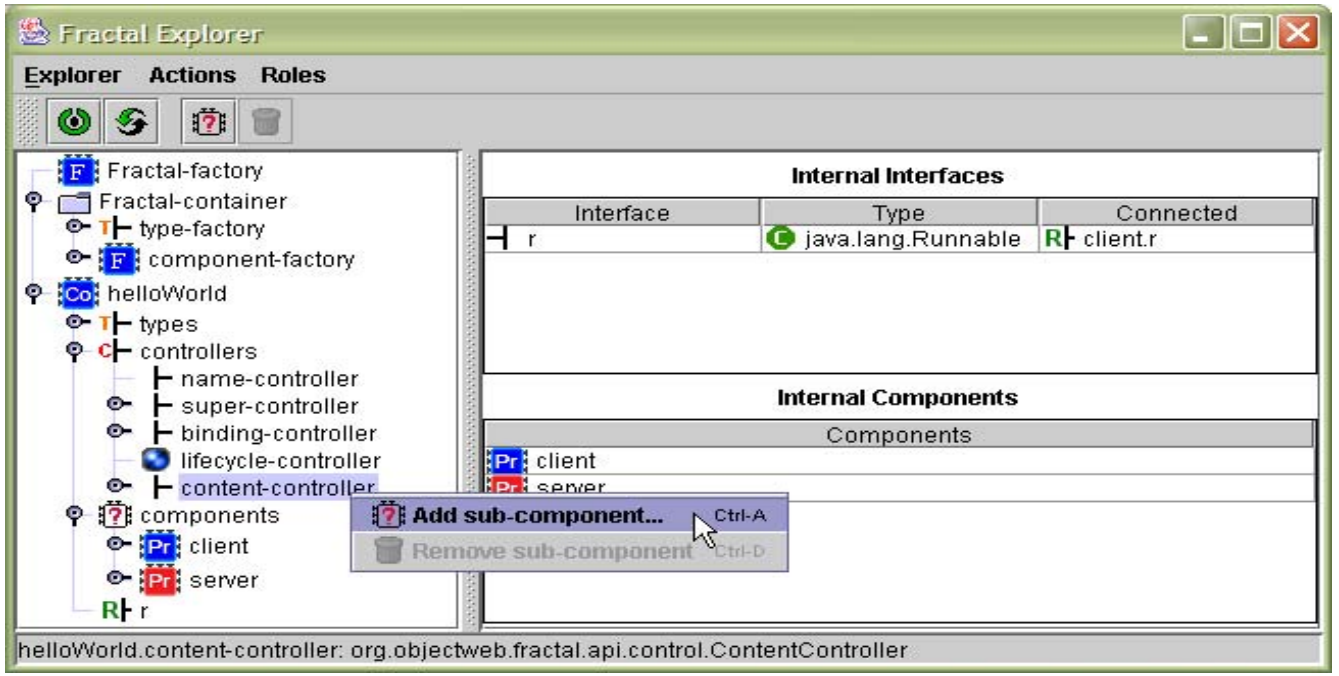
Hello World in Fractal – Running

Console



Fractal Explorer

- runtime tree view
- start/stop
- launch
- introspect
- reconfigure



1.1 Fraclet

In a nutshell

- component implementation in Java
- annotation for component related metadata
- assembling with Fractal ADL
- execution

1.1 Fraclet

Other Fraclet annotations

- @Interface : provided interface
- @Attribute : component property
- @Lifecycle : lifecycle callbacks
- @Controller : control interface reference injection

- @Node : virtual node for distributed programming
- @Membrane : controller descriptor

http://fractal.ow2.org/fraclet for more details

Outline

1.2 Fractal ADL

XML-based language for defining & configuring
a Fractal component-based system

Basic DTD for defining

- interface
- component (composite and primitive)
- binding

Describe the **initial** architecture of a Fractal component-based system

Toolchain

- to parse assembly files
- to instantiate the corresponding component-based system
- extensible
  - the DTD and the toolchain can be extended with new tags & processing components

1.2 Fractal ADL

Basic notions

- root definition
- interface
- component

Advanced notions

- extended definition
- component sharing
- parameterized definition
- toolchain

1.2 Fractal ADL

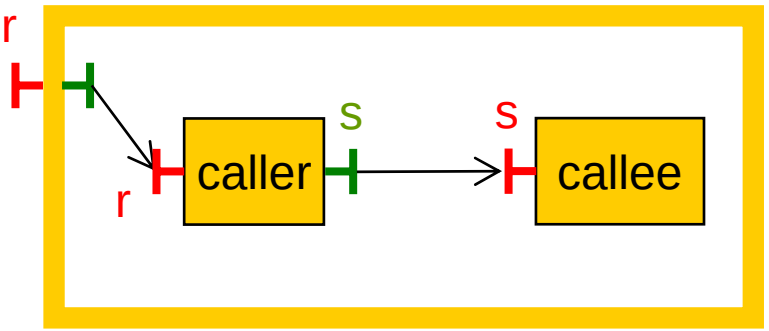
Root definition

- XML file with the .fractal extension
- Tag <definition> to define a top-level component for the system
  - 0 or n <interface> for defining interfaces
  - 0 or n <component> for defining sub-components
    - primitive or composite inserted in the top-level component
    - each component can be
      - defined inlined in the file
      - defined in a external file
  - 0 or n <binding> for defining bindings between sub-components

1.2 Fractal ADL

Hello World in Fractal – Revisited

Fractal ADL




```
<definition name="hw.HelloWorld">
```

```
<interface name="r" role="server"
           signature="java.lang.Runnable" />
```

External definition in file CallerImpl.fractal

```
<component name="caller" definition="hw.CallerImpl" />
<component name="callee" definition="hw.CalleeImpl" />
```

External definition in file CalleeImpl.fractal

```
<binding client="this.r" server="caller.r" />
<binding client="callee.s" server="callee.s" />
```

```
</definition>
```

1.2 Fractal ADL

Interface

<interface

name = "r"	interface name
role = "server"	server (provided) or client (required)
signature = "java.lang.Runnable"	Java type of the interface
cardinality = "singleton"	singleton (default) or collection
contingency = "mandatory"	mandatory (default) or optional

/>

```
<!ELEMENT interface EMPTY >
<!ATTLIST interface
  name CDATA #REQUIRED
  role (client | server) #IMPLIED
  signature CDATA #IMPLIED
  contingency (mandatory | optional) #IMPLIED
  cardinality (singleton | collection) #IMPLIED >
```

1.2 Fractal ADL

Component

- defined in an external .fractal file
- defined inlined
  - interfaces
  - content
    - primitive Java implementation class
    - composite sub-components + bindings
  - attribute(s)
    - components can export properties
    - via an interface named attribute-controller
    - the interface provides setter/getter for managing properties
    - Fractal ADL provides tags for configuring their **initial** value
  - controller descriptor
    - the membrane type associated with the component

1.2 Fractal ADL

Component

- Example 1: primitive component

```
<component name="caller">
  <interface name="r" role="server"
             signature="java.lang.Runnable" />
  <interface name="s" role="client" signature="hw.Service" />
  <content desc="hw.CallerImpl" />
  <attributes signature="hw.ServiceAttributes">
    <attribute name="header" value="->" />
    <attribute name="count" value="1" />
  </attributes>
</component>
```

1.2 Fractal ADL

Component

- Example 2: composite component

```
<definition name="HelloWorld">
  <interface name="r" role="server"
             signature="java.lang.Runnable" />
  <component name="caller" definition="hw.CallerImpl" />
  <component name="callee" definition="hw.CalleeImpl" />
  <binding client="this.r" server="caller.r" />
  <binding client="caller.s" server="callee.s" />
</definition>
```

1.2 Fractal ADL

Binding

- a reference between a required interface and a provided interface

```
<binding client="this.r" server="caller.r" />
```

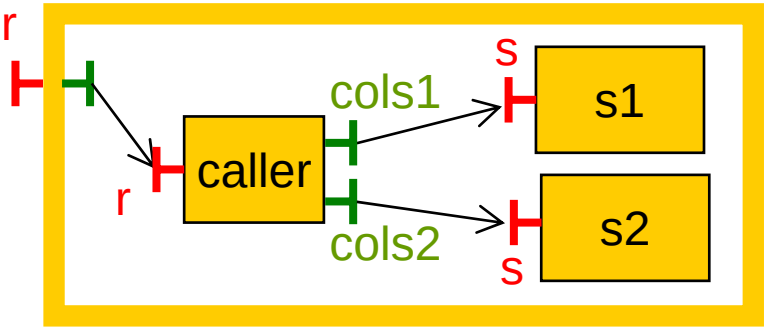
  - client : the source (required) interface
  - server : the target (provided) interface
  - syntax
    - *"componentName.interfaceName"*
    - componentName can be *this* (current component)

<!ELEMENT binding EMPTY >

<!ATTLIST binding client CDATA #REQUIRED server CDATA #REQUIRED >

1.2 Fractal ADL

A (slightly) more complex example with collection interfaces




```
<definition name="hw.HelloWorld">
  <interface name="r" role="server"
             signature="java.lang.Runnable" />
  <component name="caller">
    <interface name="r" role="server" signature="java.lang.Runnable" />
    <interface name="cols" role="client"
               cardinality="collection" signature="hw.Service" />
    <content desc="ClientImpl" />
  </component>
  <component name="s1"> ... </component>
  <component name="s2"> ... </component>
  <binding client="this.r" server="caller.r" />
  <binding client="caller.cols1" server="s1.s" />
  <binding client="caller.cols2" server="s2.s" />
</definition>
```

1.2 Fractal ADL

Basic notions

- root definition
- interface
- component

Advanced notions

- extended definition
- component sharing
- parameterized definition
- toolchain

1.2 Fractal ADL

Advanced Notion – Extended Definition

- reuse and extend existing definitions

Good practise

- separate the definition of a component type from its implementation

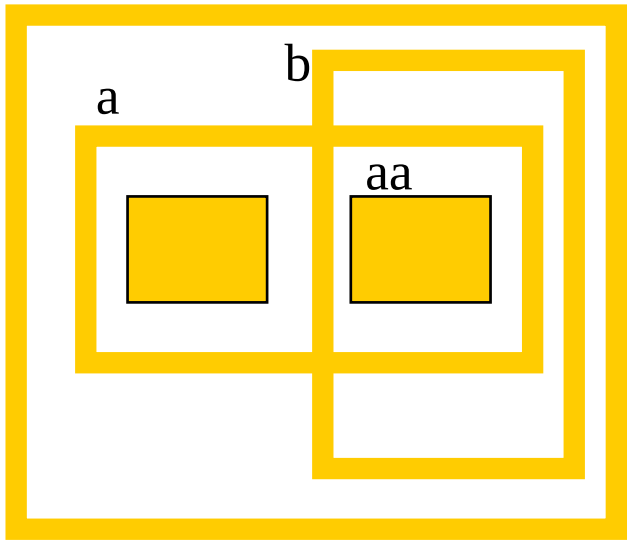
```
<definition name="hw.CallerType">
  <interface name="r" role="server" signature="j.l.Runnable" />
  <interface name="s" role="client" signature="hw.Service" />
</definition>

<definition name="hw.Caller" extends="hw.CallerType" >
  <content class="hw.CallerImpl" />
</definition>
```

1.2 Fractal ADL

Advanced Notion – Component Sharing

- a component with several parent components
- first definition
- the next ones reference the first one

```
<definition name="foo">

  <component name="a">
    <component ... />
    <component name="aa"> ... </component>
  </component>

  <component name="b">
    <component name="aa" definition="a/aa" />
  </component>

</definition>
```



1.2 Fractal ADL

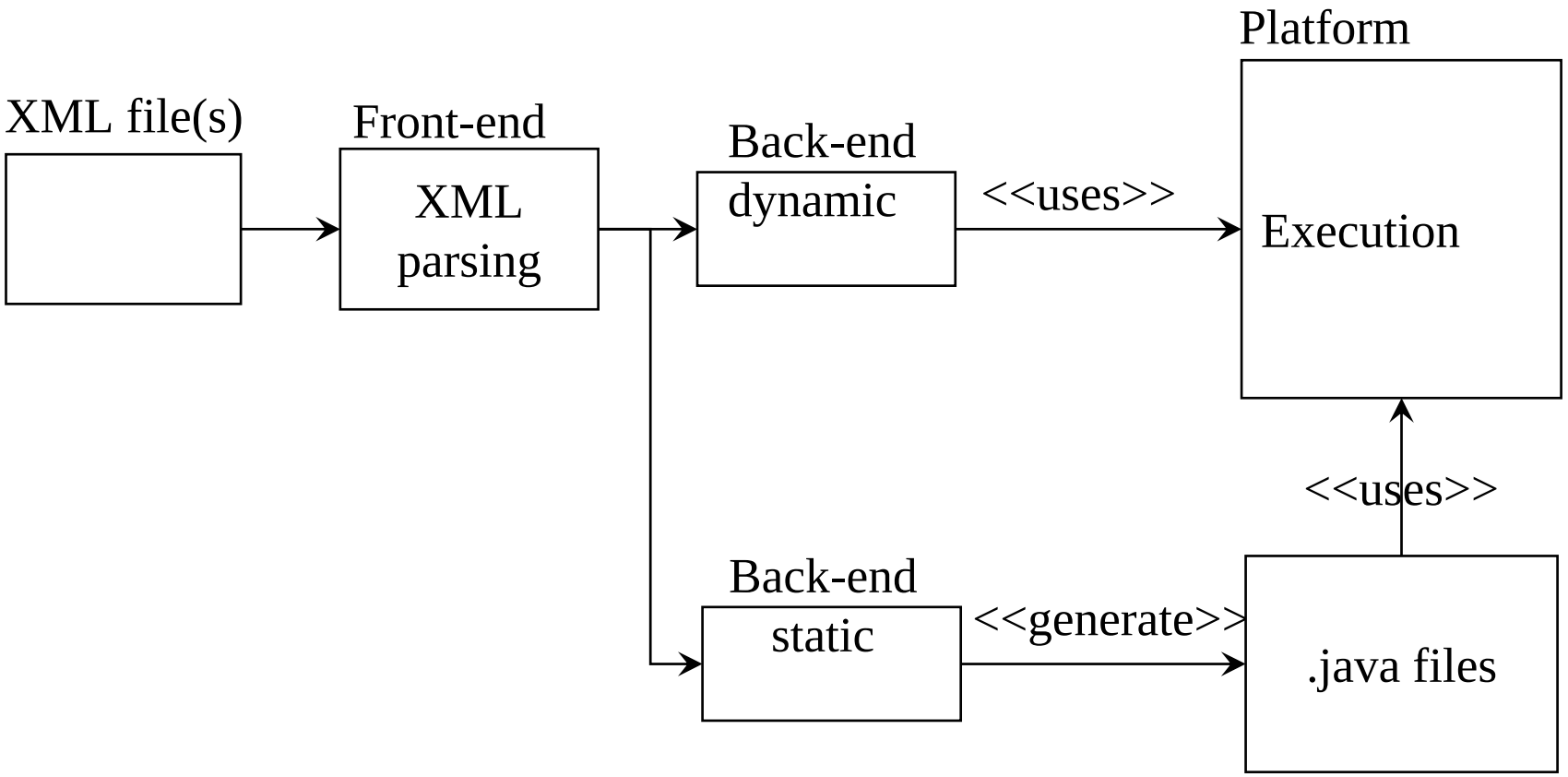
Advanced Notion – Parameterized Definition

- arguments may be declared when defining an architecture
- used in the definition with ${…}

```
<definition name="hw.Client" arguments="itfname,impl" >

  <interface name="r" role="server" signature="..." />
  <interface name="s" role="client" signature="${itfname}"
   />

  <content class="${impl}" />
</definition>
```

1.2 Fractal ADL

Advanced Notion – Toolchain



Outline

1.3 Fractal API

Fractal is a dynamic component model

- components and assemblies are runtime entities
- applications can be dynamically reconfigured

Introspection and modification

- binding controller
- component
  - introspection
    - hierarchy: content controller and super controller (parent)
    - component: interface discovery
  - modification
    - dynamic instantiation
    - hierarchy: content controller and super controller (parent)
- API Fractal

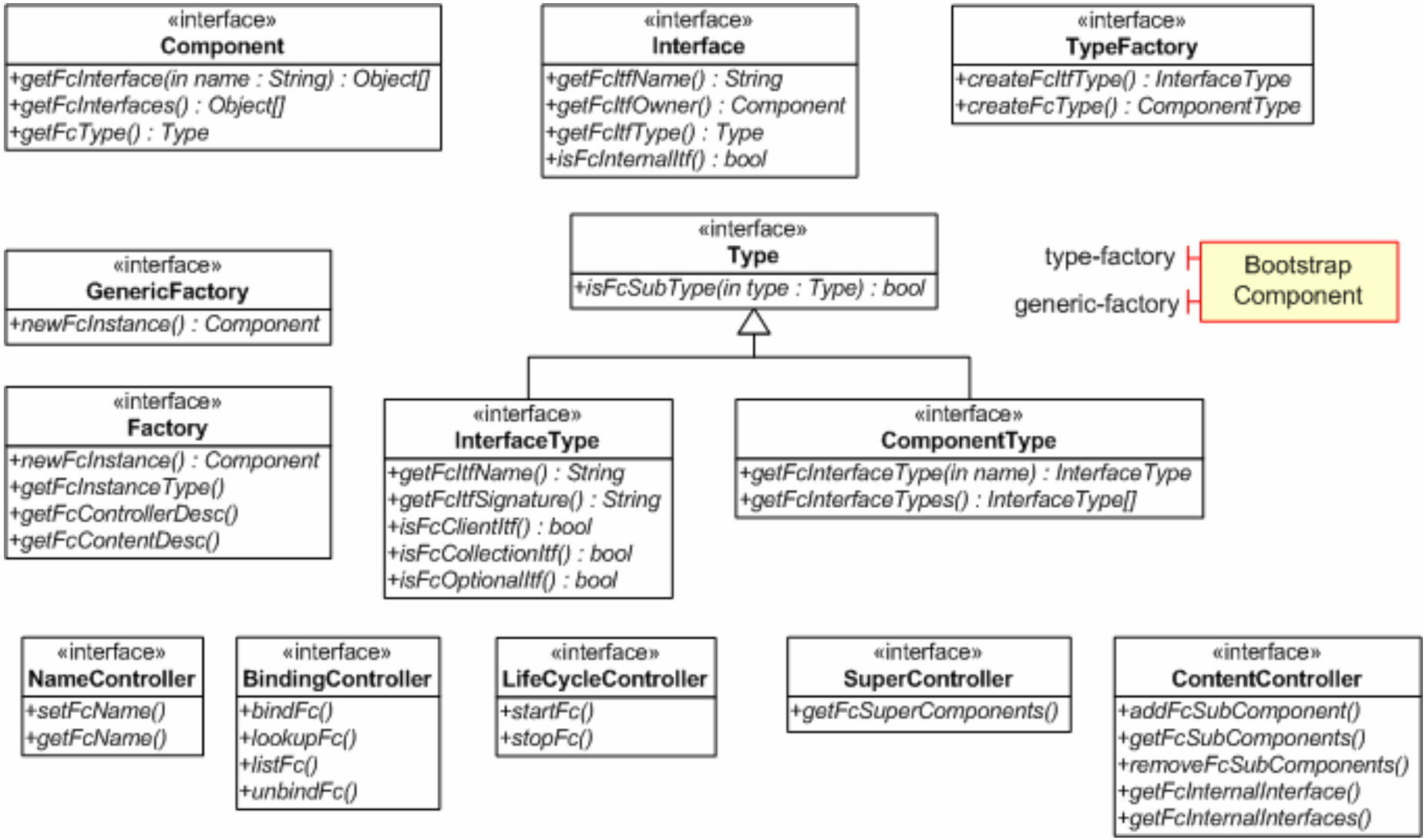
1.3 Fractal API

- lightweight (16 interfaces, <40 methods)

Illustration: Setting up the Hello World Example with the API

1. Create interface & component types
2. Instantiate components
3. Assemble components
    1. Create hierarchies
    2. Bind components
4. Start the application

1.3 Fractal API

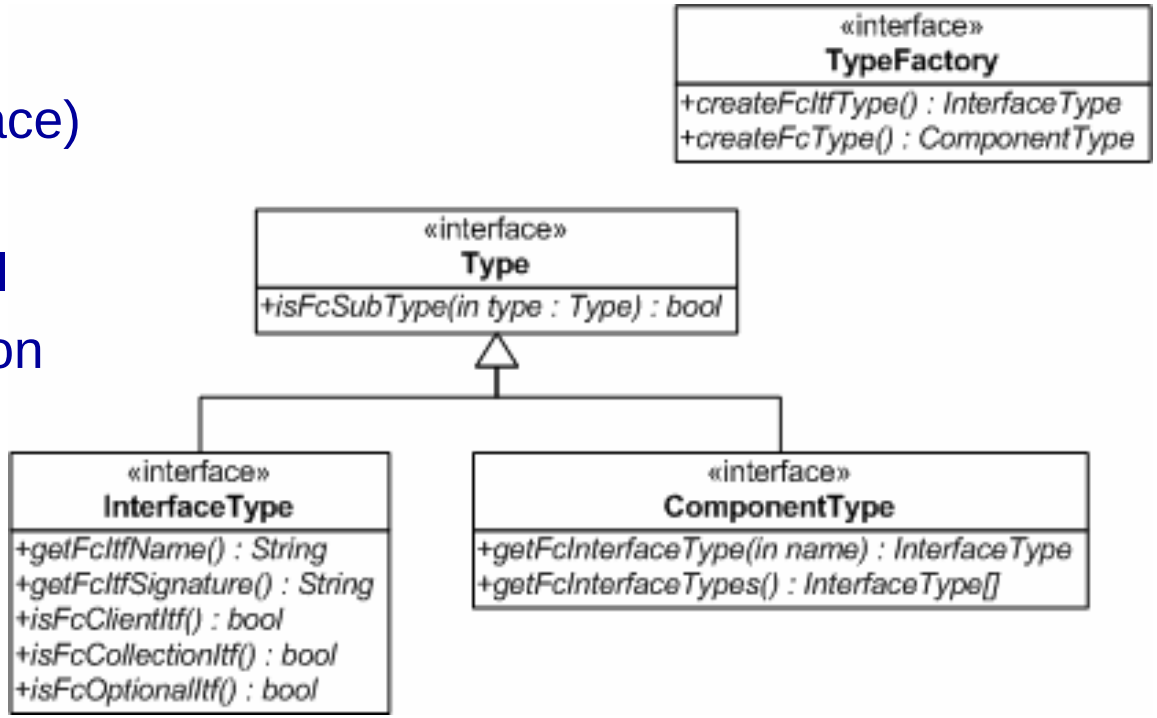


1.3 Fractal API

1. Create interface & component types

Interface type

- name
- signature (Java interface)
- 1 bool : true = client
- 1 bool : true = optional
- 1 bool : true = collection



Component type

- array of interface types

1.3 Fractal API

1. Create interface & component types

- Retrieve the TypeFactory

```
Component boot = Fractal.getBootstrapComponent();
TypeFactory tf = Fractal.getTypeFactory(boot);
GenericFactory cf = Fractal.getGenericFactory(boot);
```

- Create a component type for the root composite

```
ComponentType rootType = tf.createFcType(new InterfaceType[] {
  tf.createFcItfType(
     "r",                       // name
     "java.lang.Runnable",      // Java signature
     false,                     // server (provided)
     false,                     // mandatory
     false)                     // singleton
});
```

1.3 Fractal API

1. Create interface & component types

- Create component type for the caller and calle components

```
ComponentType callerType = tf.createFcType(new InterfaceType[] {
    tf.createFcItfType(
        "r", "java.lang.Runnable", false, false, false),
    tf.createFcItfType(
        "s", "hw.Service", true, false, false)
});

ComponentType calleeType = tf.createFcType(new InterfaceType[] {
    tf.createFcItfType("s", "hw.Service", false, false, false)
});
```

1.3 Fractal API

2. Instantiate components

- Instantiate the root composite

```
Component rootComp = cf.newFcInstance(
    rootType,           // component type
    "composite",        // membrane
    null);              // implementation
```

- Instantiate the primitive components

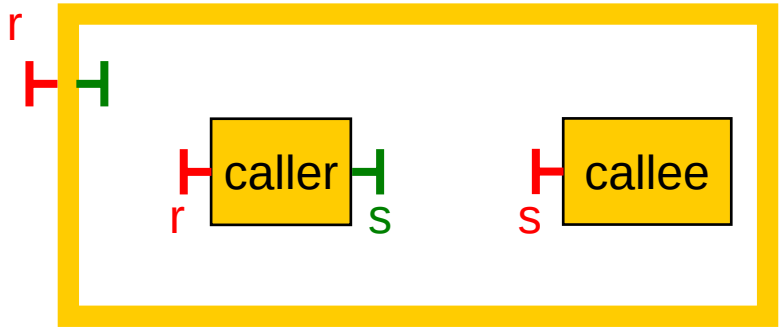

```
Component callerComp =
    cf.newFcInstance(callerType, "primitive", "hw.CallerImpl");
Component calleeComp =
    cf.newFcInstance(calleeType, "primitive", "hw.CalleeImpl");
```

1.3 Fractal API

3. Assemble components

- Create hierarchies
  - insert the Caller & Callee component in the root composite

```
Fractal.getContentController(rootComp).
    addFcSubComponent(callerComp);
Fractal.getContentController(rootComp).
    addFcSubComponent(calleeComp);
```

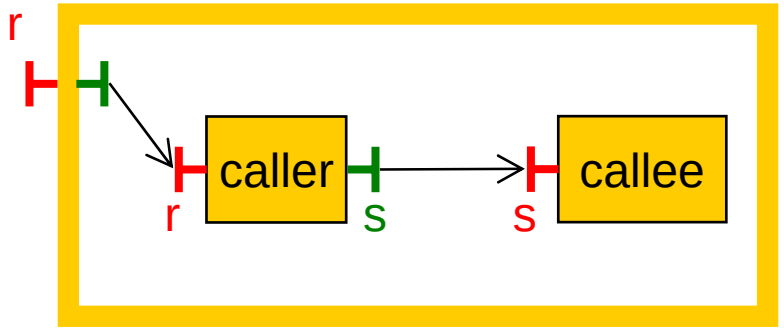


1.3 Fractal API

3. Assemble components

- Create bindings

```
Fractal.getBindingController(rootComp).bindFc(
   "r", callerComp.getFcInterface("r"));

Fractal.getBindingController(rootComp).bindFc(
   "s", calleeComp.getFcInterface("s"));
```



1.3 Fractal API

4. Start the application

- Start the root composite
  - recursively start sub-components
- Invoke the provided interface on the root composite

```
Fractal.getLifeCycleController(rootComp).startFc();
((Runnable)rootComp.getFcInterface("r")).run();
```

1.3 Fractal API

Conclusion on Fractal API

- intuitive
- no complexity
- powerful
- base for other tools (Fraclet, Fractal ADL, etc.)

- verbose
- slightly underspecified
  - e.g. getFcInterface(), lookupFc() return Object (not Interface)
  - membrane descriptor are Object
- ➢ leave the API open to extensions to the model

Outline

1. Developing with Fractal in Java
   - 1.1 Fraclet
   - 1.2 Fractal ADL
   - 1.3 Fractal API
2. Platforms
   - 2.1 Julia
   - 2.2 AOKell
3. Conclusion

2. Platforms

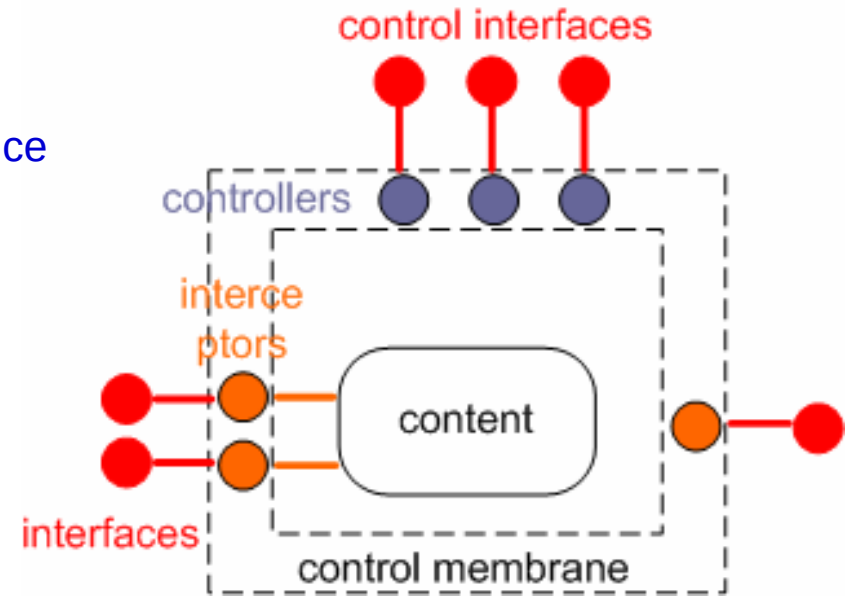
- several platforms
  - 3 for Java
    - Julia — reference implementation
    - AOKell — aspect-oriented implementation
    - ProActive — implementation for grid computing
  - 3 for C (Think, Cecilia, MNF)
  - 1 for C++ (Plasma)
  - 1 for SmallTalk (FracTalk)
  - 1 for .NET (FractNet)

- several platforms for different needs

2. Platforms

Concepts associated with platforms

- control membrane
  - hosting infrastructure for a component
  - provide non-functional services
  - implements the component semantics
  - aka container in other component framework (e.g. EJB)
  - composed of a set of controllers
- controller
  - provide a particular non functional service
- control interface
  - the service provided by a controller
- interceptors
  - filter communications
- content
  - the business code of the component



Outline

1. Developing with Fractal in Java
   - 1.1 Fraclet
   - 1.2 Fractal ADL
   - 1.3 Fractal API
2. Platforms
   - 2.1 Julia
   - 2.2 AOKell
3. Conclusion

2.1 Julia

- reference implementation of the Fractal component model
- http://fractal.ow2.org/julia

- demonstrate adequation/feasibility of the specifications
- extensible framework for programming control membranes
- designed with the following objectives
  - minimize memory footprint of the control membranes
  - minimize runtime overhead of the control membranes

2.1 Julia

Modular organization of the platform

1. fractal-api : Fractal API
2. julia-runtime : Julia internal API
3. julia-asm : bytecode generation framework
   - based on the ASM bytecode engineering library
   - implements a mixin-based programming model
4. julia-mixins
   - a set of mixins implementing the default execution semantics
   - other mixins may be provided for other execution semantics

- (slight) mismatch around the term Julia
  - framework: julia-runtime + julia-asm
  - default execution semantics: julia-mixins

2.1 Julia

Control membrane programming

- membrane descriptor
  - text file(s) (e.g. julia.cfg) loaded by the framework which configures
    - control interfaces
    - controller implementations (set of mixins)
    - interceptor
    - optimization level

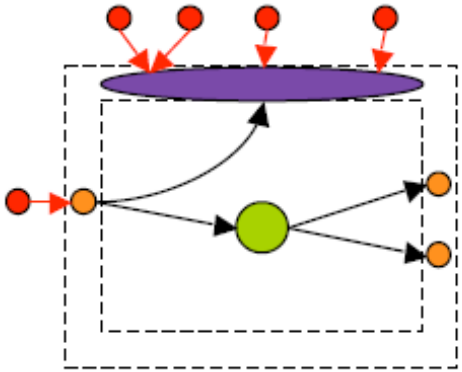
  for each membrane type (primitive, composite, etc.)

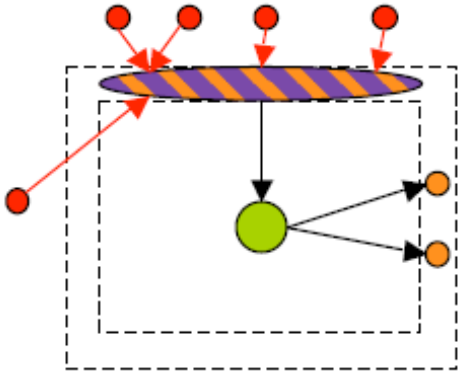
- mixin-based programming model
  - [Bracha, OOPSLA 1990]
  - replacement for multiple inheritance
  - foster reuse when developing controllers
  - enable optimizations

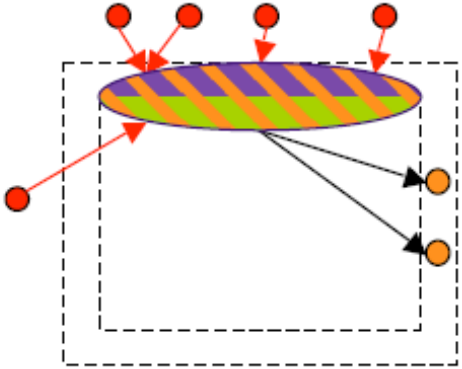
2.1 Julia

Intra-component optimization

- optimize memory footprint for a component
- various merge levels
  - controller (left-side)
  - + interceptors
  - + content



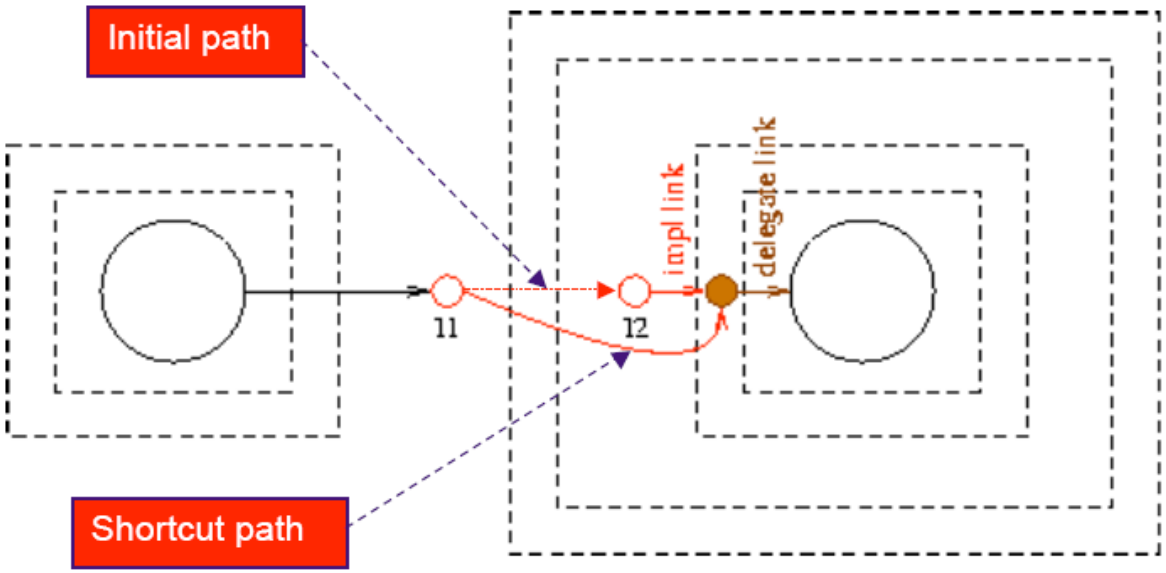




2.1 Julia

Inter-components optimization

- optimize runtime overhead
- compute shortcuts when several bindings are defined



Outline

1. Developing with Fractal in Java
   - 1.1 Fraclet
   - 1.2 Fractal ADL
   - 1.3 Fractal API
2. Platforms
   - 2.1 Julia
   - 2.2 AOKell
3. Conclusion

2.2 AOKell

- implementation of the Fractal Specifications
- aspect-based framework for engineering the control level

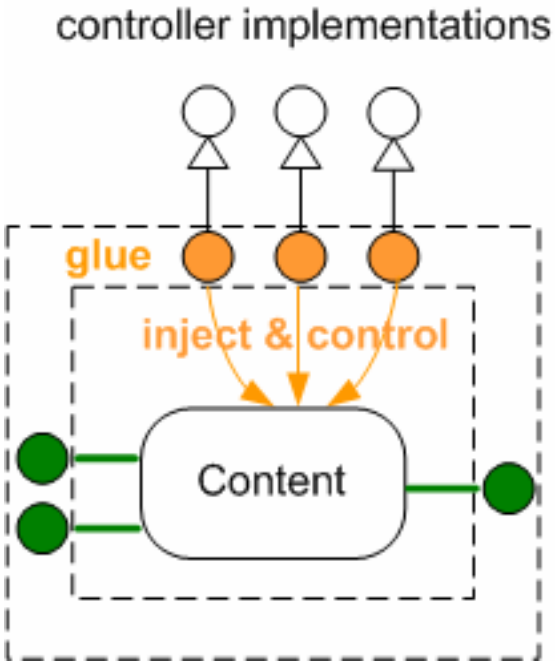
Expected benefits

- easier to develop, debug, maintain new controllers
- better integration with IDEs
- reducing the development time for writing new controllers
- reducing the learning curve

2.2 AOKell

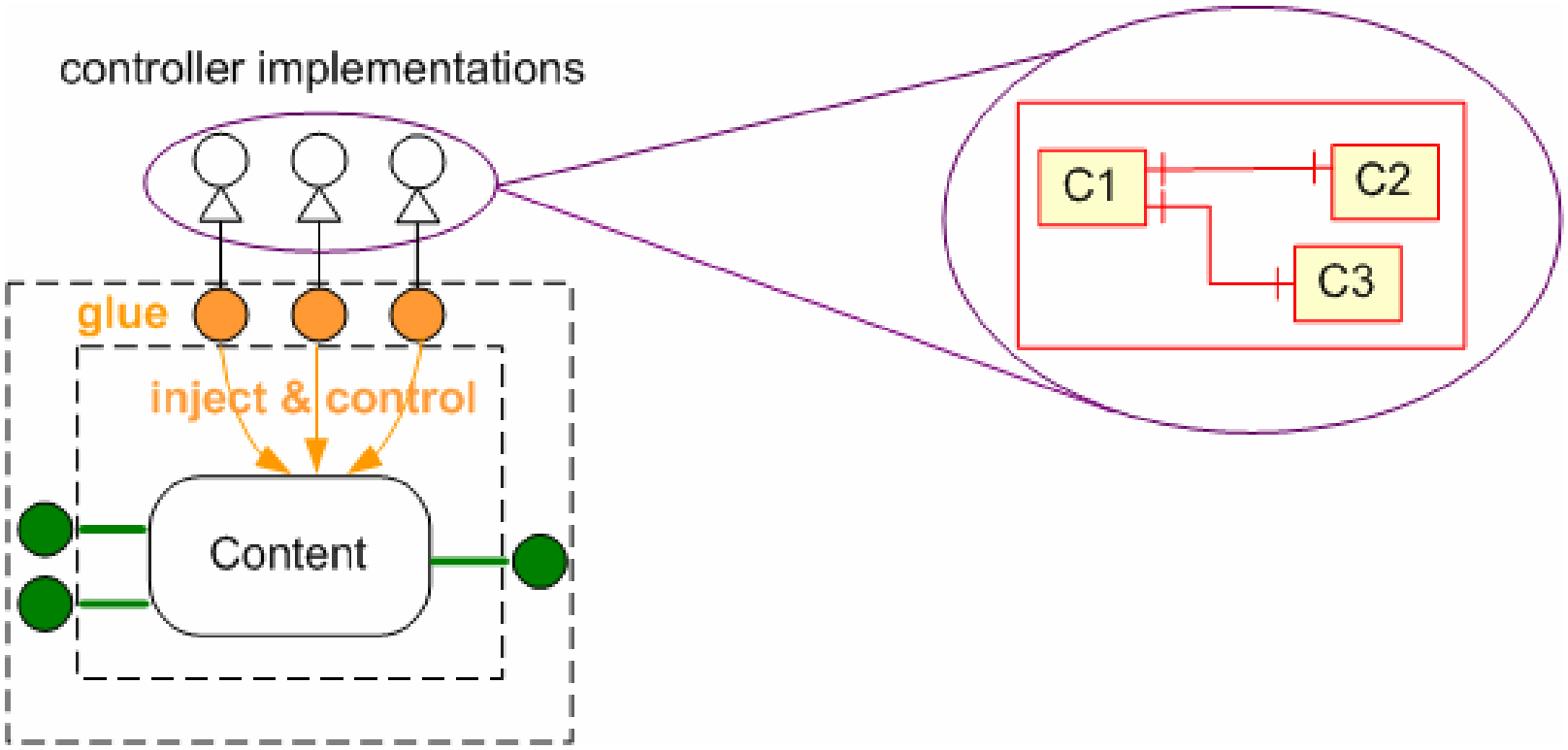
AOKell: a contribution for opening the Fractal CF [CBSE 2006]

- Existing controllers (technical services) in the Fractal CF
  - binding
  - attribute
  - lifecycle
  - content, super
  - name
  - component interfaces & type
  - template
- 1 AspectJ aspect per controller
  - glues the control dimension with the application dimension
  - delegates to the controller implementation



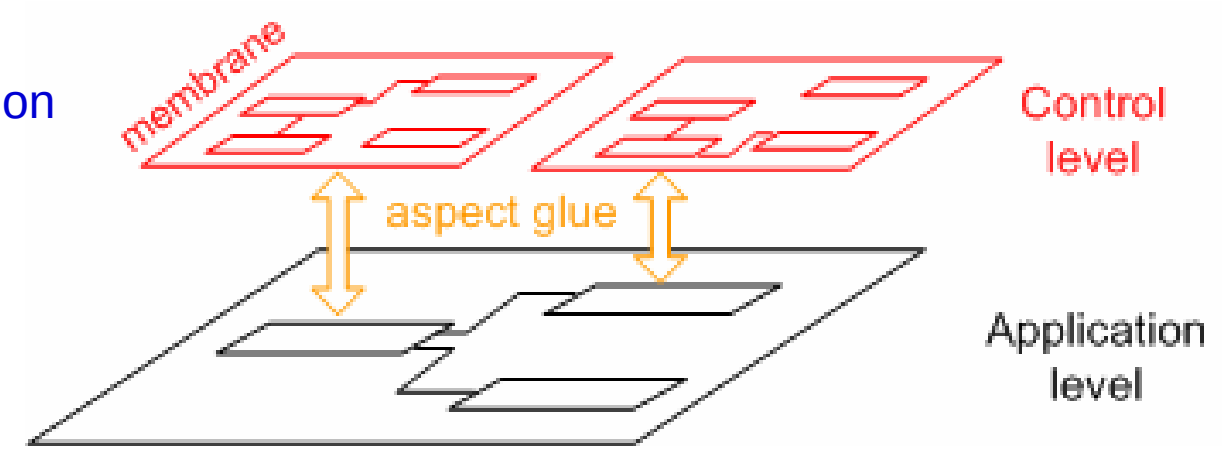
2.2 AOKell

The implementation of the control can benefit from components too

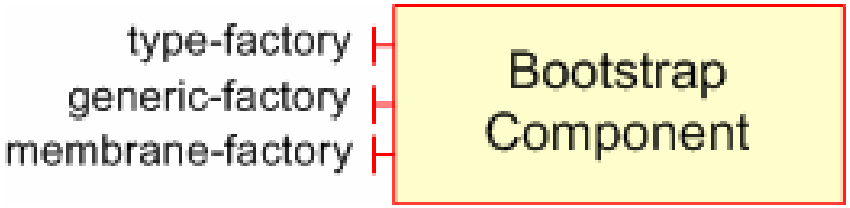


2.2 AOKell

- 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



2.2 AOKell



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

Conclusion

- uniform approach (ADL + comp.) for business and technical layers

Future work

- complex form of control architectures which span different components

Outline

1. Developing with Fractal in Java
   - 1.1 Fraclet
   - 1.2 Fractal ADL
   - 1.3 Fractal API
2. Platforms
   - 2.1 Julia
   - 2.2 AOKell
3. Conclusion

3. Conclusion

Fractal component model

- introspectable, dynamic
- fine-grained component (close to a class)
  - hierarchical approach to foster decomposition into sub-systems
- programming language independent
- structuring applications
  - at design & implementation time
  - at runtime
- tooling (Eclipse F4E, FractalExplorer, etc.)

3. Conclusion

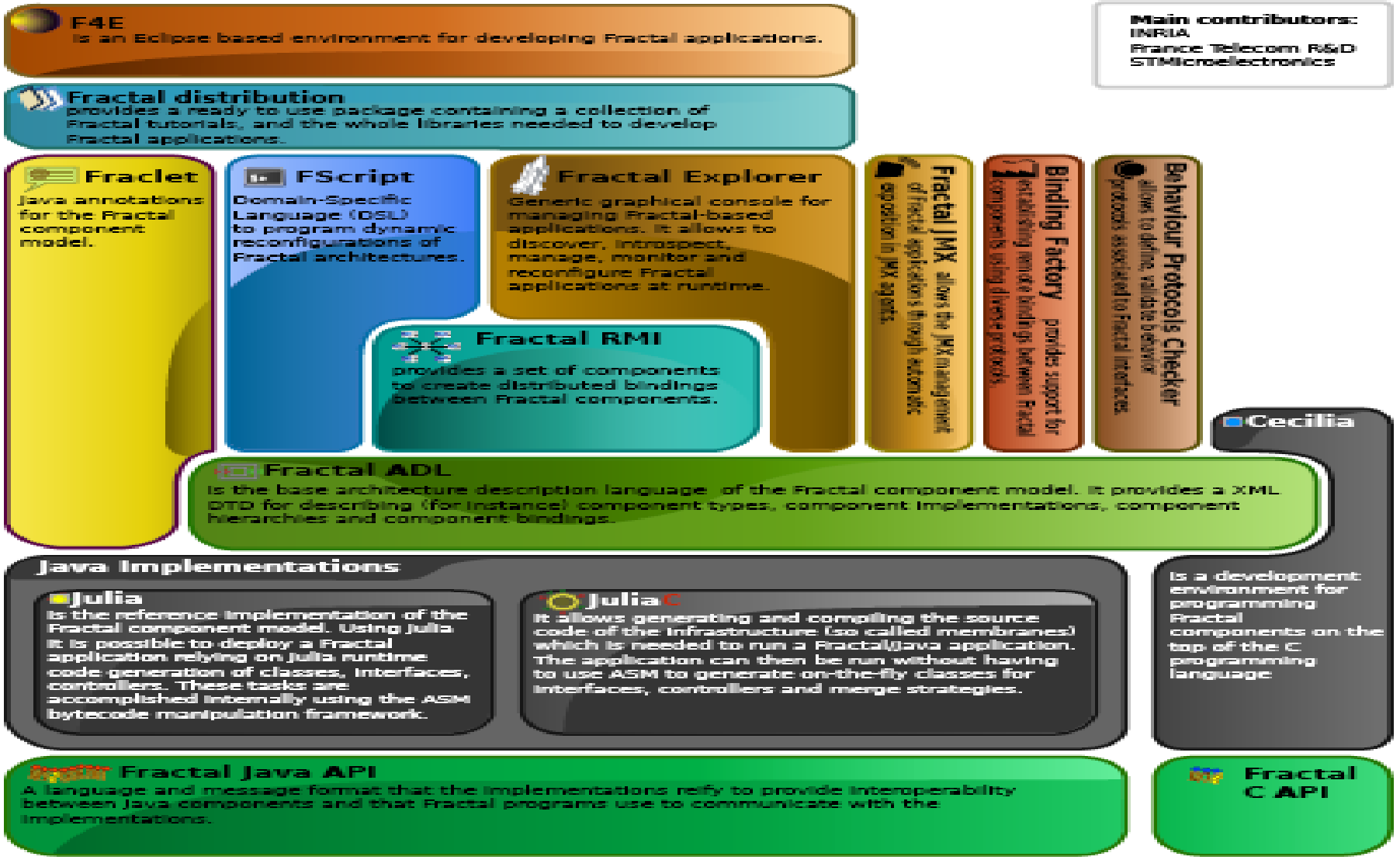
Fractal component model

Many other tools/work not addressed by this presentation

- remote communication (Fractal RMI)
- administration (Fractal JMX)
- Think/Cecilia & embedded systems
- component libraries
  - DREAM, CLIF, Speedo, Perseus, GoTM, …
- binding factory
- compiling Fractal applications (Juliac)

➢ large ecosystem

3. Conclusion



3. Conclusion

- R&D activities and Tools
  - Formal models and calculi (INRIA, Verimag)
  - Configuration (Fractal/Think ADL - FT, INRIA, STM), navigation/query (EMN, FT)
  - Dynamic reconfiguration (FT, INRIA)
  - Management - Fractal JMX (FT)
  - Packaging, deployment (INRIA, LSR, Valoria)
  - Security, isolation (FT)
  - Correctness: structural integrity (FT), behavioural contracts based on assertions (ConFract - I3S, FT), behavior protocols (Charles U., FT), temporal logic (Fractal TLO - FT), automata (INRIA), test (Valoria)
  - QoS management (Plasma - INRIA, Qinna - FT)
  - Self-adaptation, autonomic computing (Jade - INRIA, Safran - EMN, FT)
  - Components & aspects (FAC, Julius, AOKell - INRIA, FT)
  - Components & transactions (Jironde - INRIA)
- Some operational usages
  - Jonathan, Jabyce, Dream, Perseus, Speedo, JOnAS (persistence), GoTM, CLIF…
- Dissemination in industry (FT, STM, Nokia), universities including teaching (Grenoble, Chambéry, Nantes…), conferences (JC, LMO, SC, Euromicro…)

3. Conclusion

Special issue of the Annals of Telecommunications on Component-based architecture: The Fractal initiative. January-February 2009.

Fractal Specifications
E. Bruneton, T. Coupaye, J.-B. Stefani.
The Fractal Component Model.
http://fractal.ow2.org/specification/index.html

Fractal & Julia
E. Bruneton, T. Coupaye, M. Leclerc, V. Quéma, J.-B. Stefani.
The Fractal Component Model and its Support in Java.
Software Practise and Experience. 36(11-12):1257-1284. 2006.

AOKell
L. Seinturier, N. Pessemier, L. Duchien, T. Coupaye.
A Component Model Engineered with Components and Aspects.
9th Intl. Symp. on Component-Based Software Engineering (CBSE).
LNCS 4063, pp. 139-153. June 2006.

Component Based Software Development
C. Szyperski.
Component Software – Beyond Object-Oriented Programming.
Addison-Wesley, 2nd edition, 2002.

Thank you for your attention

http://fractal.ow2.org/java

fractal@ow2.org

Questions?