Software Components with Fractal: Reflective Software Architecture and Runtime Reconfiguration

Martin Monperrus

version of Dec 28, 2012

[Take-away] Fractal provides you with

- Automated dependency injection (no setter, constructor)
- Safe assembly
- Safe reconfiguraton

Sources

The fractal component model and its support in java:
http://www.cs.colostate.edu/saxs/se/FractalComponent.pdf

Fractal Specification:

http://fractal.ow2.org/specification/fractal-specification.pdf

Fractal API:

http://fractal.ow2.org/current/doc/javadoc/fractal/index.html

Acknowledgments: Thanks a lot to Lionel Seinturier and Eric Bruneton for helping me in preparing the lecture

Where is the architecture?

Hidden!

No difference between short-lived objects and architectural objects

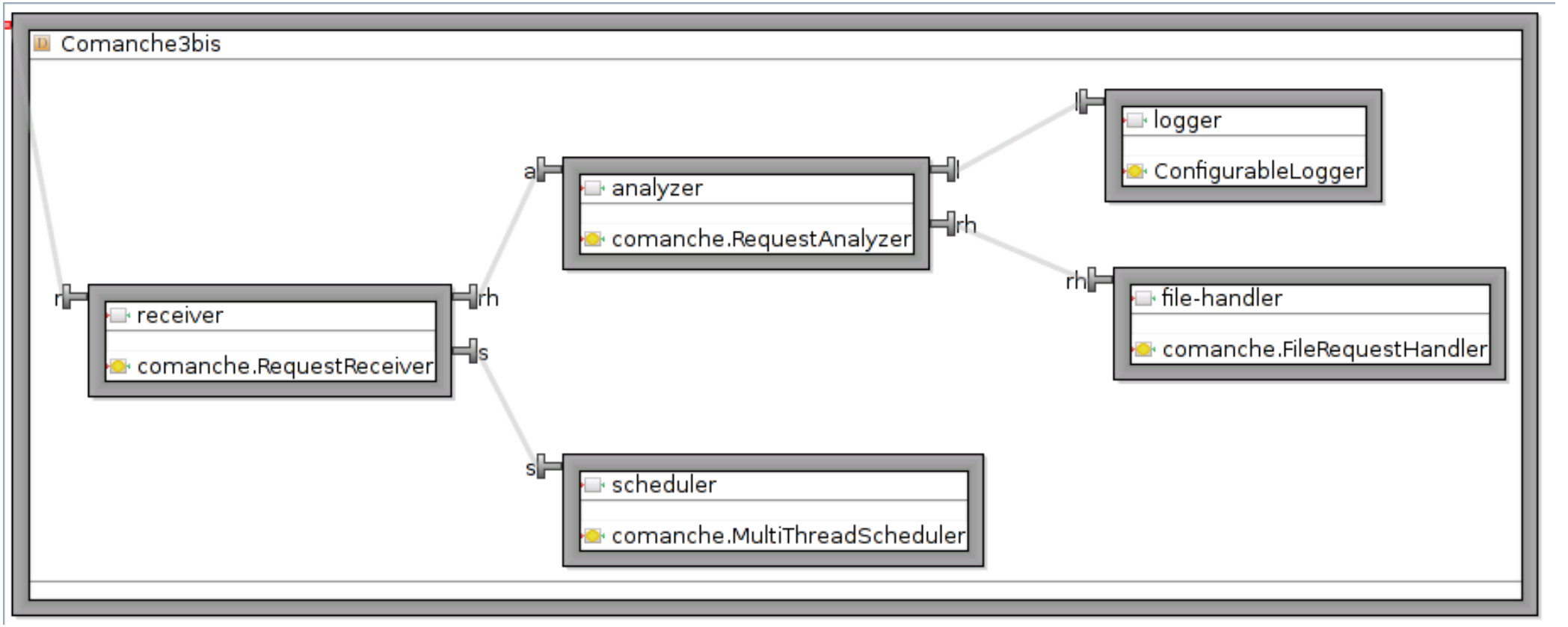
But there is a kind of architecture

```
public class HTTPServer {
  public static void main (String[] args) {
    RequestReceiver rr = new RequestReceiver();
    RequestAnalyzer ra = new RequestAnalyzer();
    RequestDispatcher rd = new RequestDispatcher();
    FileRequestHandler frh = new FileRequestHandler();
    ErrorRequestHandler erh = new ErrorRequestHandler();
    Scheduler s = new MultiThreadScheduler();
    Logger l = new BasicLogger();
    rr.analyzer= ra;
    rr.scheduler =  s;
    ra.handleRequest = rd;
    ra.logger = l;
    rd.handler.add(frh);
    rd.handler.add(erh);
    rr.run();
  }
}
```

Assembling the application is error-prone

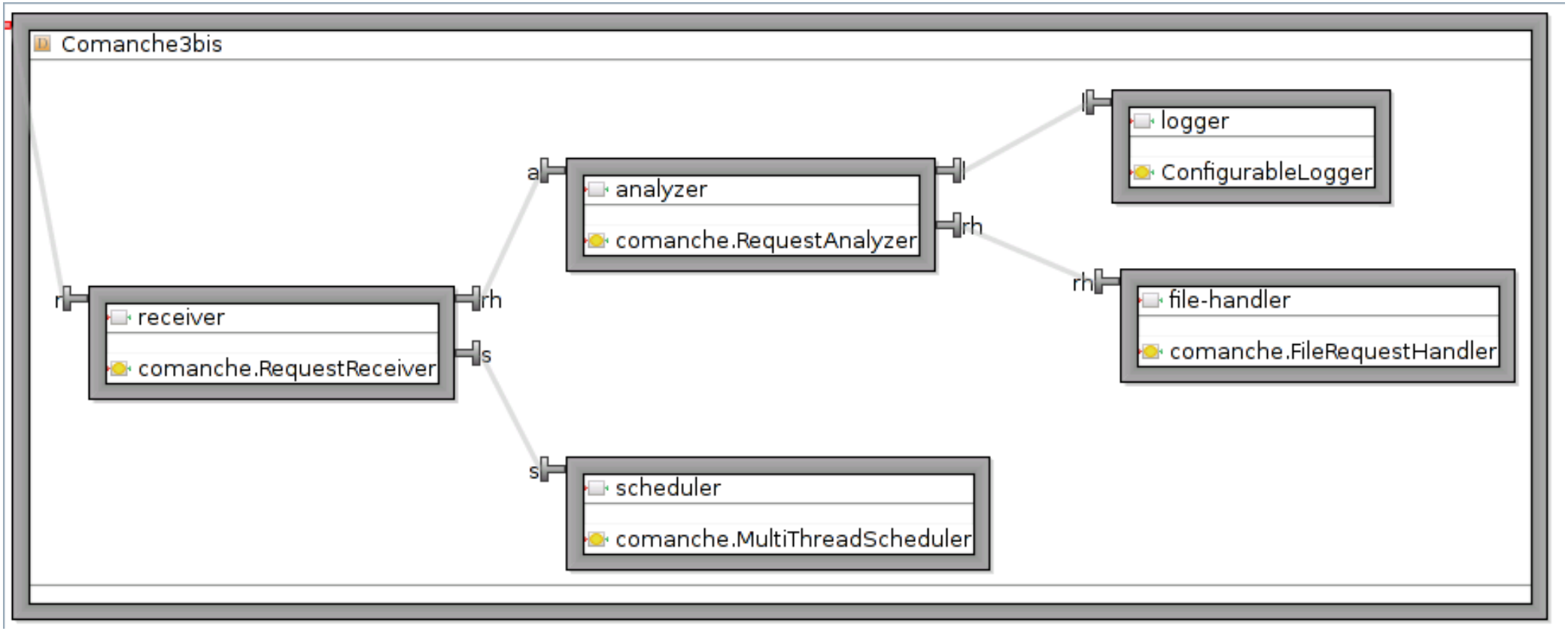
Reconfiguring the application at your own risk.

"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." [Software architecture in practice, Len Bass, Paul Clements, Rick Kazman, 1998]



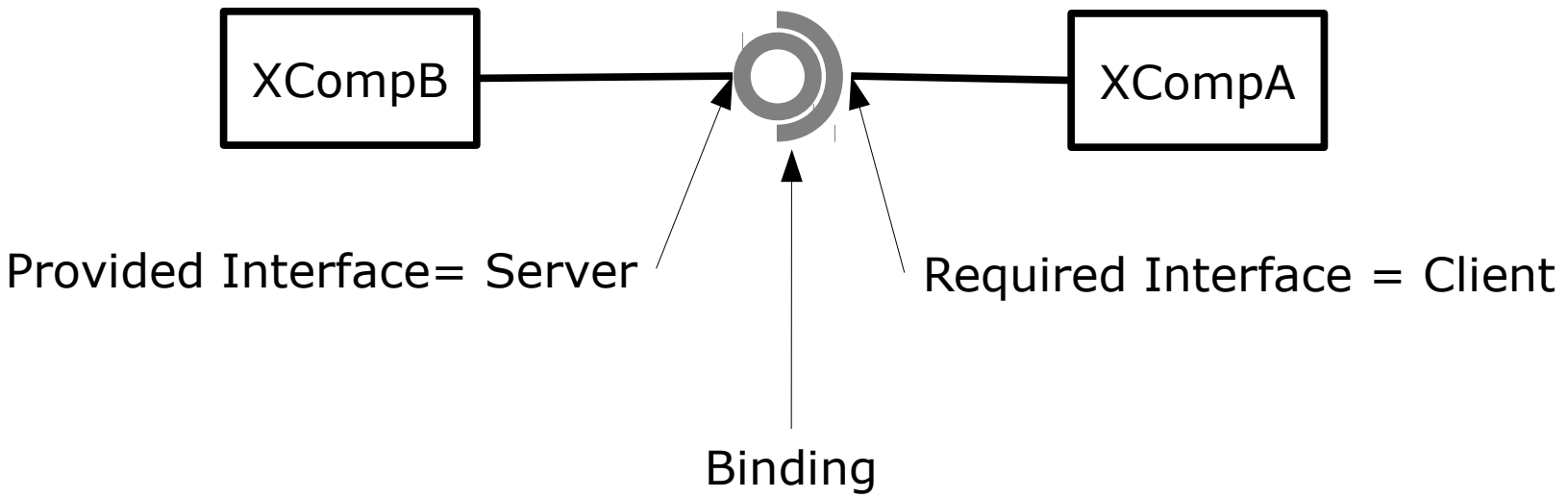
Architecture Description Language

An **Architecture Description Language** is a notation in which architecture models can be expressed.



The simplest Fractal assembly ("example.fractal")

```
<definition name="XApp">
 <component name="a" definition="XCompA">
  <interface name="Capability-client"
          role="client" signature="java.lang.Runnable"/>
  <content class="XCompA"/>
 </component>
 <component name="b">
  <interface name="Capability-server"
          role="server" signature="java.lang.Runnable"/>
  <content class="XCompB"/>
 </component>
 <binding client="a.Capability-client" server="b.Capability-server"/>
</definition>
```



The simplest Fractal assembly

```
<definition name="XApp">
 <component name="a" definition="XCompA">
  <interface name="Capability-client"
          role="client" signature="java.lang.Runnable"/>
  <content class="XCompA"/>
 </component>
 <component name="b">
  <interface name="Capability-server"
          role="server" signature="java.lang.Runnable"/>
  <content class="XCompB"/>
 </component>
 <binding client="a.Capability-client" server="b.Capability-server"/>
</definition>
```

```
public class XCompA
    implements BindingController
{

    private Runnable field;

    ...
}
```

```
public class XCompB
implements Runnable { }
```

To bind components 4 methods to add, nothing to modify.

```
import org.objectweb.fractal.api.control.BindingController;
public class XCompA implements  BindingController {
  private Runnable field;
  private Logger l;

  public String[] listFc () { return new String[] { "Capability-client"}; }

  public Object lookupFc (String itfName) {
    if (itfName.equals("Capability-client")) { return field; }
    throw new NoSuchInterfaceException(itfName);
  }

  public void bindFc (String itfName, Object itfValue) {
    if (itfName.equals("Capability-client")) { field = (Runnable)itfValue; }
    throw new NoSuchInterfaceException(itfName);
  }

  public void unbindFc (String itfName) {
    if (itfName.equals("Capability-client")) { field = null; }
    throw new NoSuchInterfaceException(itfName);
  }
```

- bindFc/unbindFC are generic getter and setters
- lookupFc and listFC are elements of reflection
- implementation very systematic and easy

Runing a Fractal application

The command-line way:

```
$ java -cp .... .Dfractal.provider=org.objectweb.fractal.julia.Julia
org.objectweb.fractal.adl.Launcher.main -fractal example
```

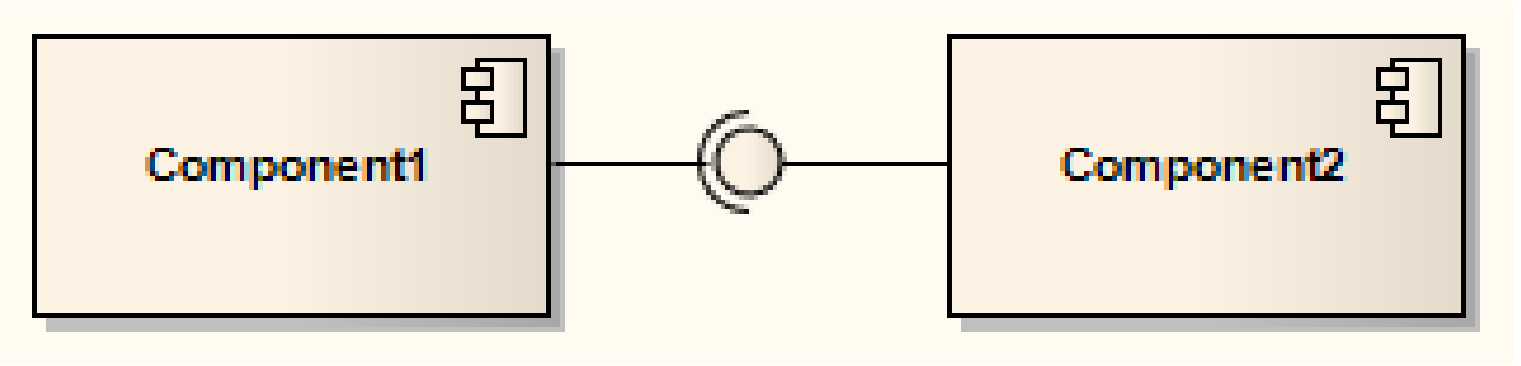
The programmatic way:

```
public static void main(String[] args) throws Exception {
// select the implementation of the component model
System.setProperty("fractal.provider", "org.objectweb.fractal.julia.Julia");
Factory f = FactoryFactory.getFactory(FactoryFactory.FRACTAL_BACKEND);
Component comp = (Component)f.newComponent("example", null);
Fractal.getLifeCycleController(comp).startFc();
((Runnable)comp.getFcInterface("Runnable")).run();
}
```

Fractal Core Insights

class A implements I1 { I2 field; } class B implements I2

- I1 is a provided interface: we can plug an A into a field
- B is a required interface: field can not be null
- Same as hardware: male (implements)/female (field), interface



The Fractal Component Model

A Fractal component defines:

- a set of interfaces
- a scope (not all components can be bound together, contrary to objects)
- an implementation class

A Fractal interface defines:

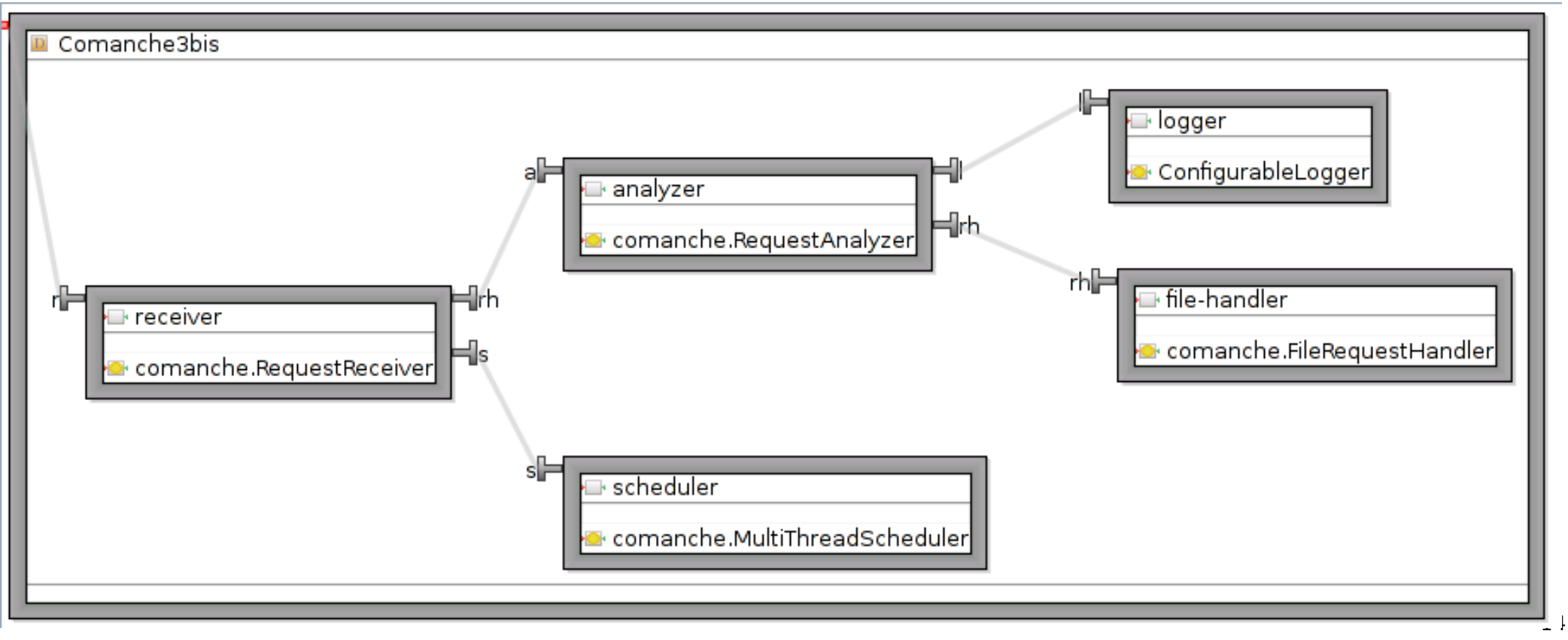
- a signature
- a role
  - provided interface ("role=server")
  - required interface ("role=client")
- a cardinality
  - singleton (default)
  - collection (cardinality="collection")
- a contingency
  - mandatory (default)
  - optional (contingency="optional")

The framework handles the semantics of the assembly

Architecture Description Language

Advantages:

- Declarative definition of component assemblies
- No compilation required
- Semantic checking of the properties of the assembly:
  - e.g. mandatory interfaces
  - e.g. scoping rules



Element of semantics

```
<!-- abstract component definition -->
<definition name="DispatcherType">
  <interface name="h" role="client" signature="RequestHandler"
cardinality="collection" contingency="optional"/>
</definition>
```

```
<!-- concrete, instanciable component
-->
<definition name="Dispatcher1"
extends="DispatcherType">
  <content
class="comanche.Dispatcher1"/>
</definition>

public class Dispatcher1  {}
```

Incorrect: if the component type has a client interface (field to be set), the implementation class must implement BindingController (generic setter)

?

```
<definition name="Dispatcher2"
extends="DispatcherType">
  <content
class="comanche.Dispatcher2"/>
</definition>

public class Dispatcher2 implements
BindingController {
}
```

Correct

Element of semantics

```
<definition name="DispatcherType">
  <interface name="rh" role="server" signature="Capability"/>
  <interface name="h" role="client" signature="RequestHandler"
cardinality="collection" contingency="optional"/>
</definition>
```

```
<definition name="Dispatcher1"
extends="DispatcherType">
  <content
class="comanche.Dispatcher1"/>
</definition>

public class Dispatcher1 implements
BindingController {
RequestHandler obj;
}
```

Incorrect: the implementation class must implement all "server" interfaces (Capability is missing)

?

```
<definition name="Dispatcher2"
extends="DispatcherType">
  <content
class="comanche.Dispatcher2"/>
</definition>

public class Dispatcher2 implements
Capability, BindingController {
RequestHandler obj;
}
```

Correct

Element of semantics

```
<definition name="ComponentA">
  <interface name="rh" role="server" signature="Capability"/>
  <content class="comanche.ComponentA"/>
</definition>
<definition name="ComponentB">
  <interface name="h" role="client" signature="Capability"
cardinality="collection" contingency="optional"/>
  <content class="comanche.ComponentB"/>
</definition>
```

```
<definition name="Application">
<component name="a"
definition="ComponentA"/>
<component name="b"
definition="ComponentB"/>
<binding client="a.rh"
server="b.h"/>
</definition>
```

Incorrect: A binding must complies with client/server specification (a.rh is server not client)

?

```
<definition name="Application">
<component name="a"
definition="ComponentA"/>
<component name="b"
definition="ComponentB"/>
<binding client="b.h"
server="a.rh"/>
</definition>
```

Correct

Element of semantics

```
<definition name="ComponentA">
  <interface name="rh" role="server" signature="Capability"
contingency="optional"/>
  <content class="comanche.ComponentA"/>
</definition>
<definition name="ComponentB">
  <interface name="h" role="client" signature="Capability"
cardinality="collection" contingency="optional"/>
  <content class="comanche.ComponentB"/>
</definition>
```

```
<definition name="Application">
<component name="a"
definition="ComponentA"/>
<component name="b"
definition="ComponentB"/>
<binding client="b.h"
server="a.rh"/>
</definition>
```

Correct

?

```
<definition name="Application">
<component name="a"
definition="ComponentA"/>
<component name="b"
definition="ComponentB"/>
<binding client="b.h"
server="a.rh"/>
<binding client="b.h"
server="a.rh"/>
</definition>
```

Incorrect: 2 bindings on the same interface a.rh but ComponentA.rh is not a collection

Element of semantics

```
<definition name="ComponentA">
  <interface name="rh" role="server" signature="Capability"/>
  <content class="comanche.ComponentA"/>
</definition>
<definition name="ComponentB">
  <interface name="h" role="client" signature="Capability"
cardinality="collection"/>
  <content class="comanche.ComponentB"/>
</definition>
```

```
<definition name="Application">
<component name="a"
definition="ComponentA"/>
<component name="b"
definition="ComponentB"/>
</definition>
```

Incorrect: Interface rh is not optional

?

```
<definition name="Application">
<component name="a"
definition="ComponentA"/>
<component name="b"
definition="ComponentB"/>
<binding client="b.h"
server="a.rh"/>
</definition>
```

Correct

Element of semantics

```
<definition name="ApplicationType">
  <component name="composite">
    <component name="a">
      <interface name="rh" role="server" signature="Capability"
contingency="optional"/>
      <content class="comanche.ComponentA"/>
    </component>
  </component>
  <component name="b">
    <interface name="h" role="client" signature="Capability"
cardinality="collection" contingency="optional"/>
    <content class="comanche.ComponentB"/>
  </component>

</definition>
```

```
<definition name="Application"
extends="ApplicationType">
<binding client="b.h"
server="a.rh"/>
</definition>
```

Incorrect: Components are scoped. a and b are not in the same scope

?

```
<definition name="Application"
extends="ApplicationType">
<component name="c" extends >
<content
class="comanche.ComponentB"/>
</component>
</definition>
```

Correct

Example of runtime errors ...

when Fractal/Julia checks the implementation of interfaces, e.g.:

```
The implementation class does not implement all the server
interfaces of the component
```

when Fractal/Julia checks the correctness of the bindings, e.g.

```
Multiple bindings from the same interface
```

```
Mandatory client interface 'analyzer.rh is not bound
```

```
Trying to invoke a method on a client interface, or on a
server interface whose complementary interface is not
bound. (no content class)
```

Fractal: so far

- Designing the architecture:
  - Interfaces (it forces you to think in terms of interfaces)
  - Component types (set of required/provided interfaces)
  - Component assemblies (concrete classes + concrete bindings)
  - Work division

- Advantage: the architecture is explicit at compile time (files .fractal) and at runtime (complete introspection)
- And:
  - Fractal/Julia is LGPL and can be used in industrial settings
  - Fractal/Julia can be used on low memory devices (e.g. embedded systems).

More complex binding controller

```
import org.objectweb.fractal.api.control.BindingController;
public class RequestAnalyzer implements RequestHandler, BindingController {
  private RequestHandler rh;
  private Logger l;

  public String[] listFc () { return new String[] { "l", "rh" }; }

  public Object lookupFc (String itfName) {
    if (itfName.equals("l")) { return l; }
    else if (itfName.equals("rh")) { return rh; }
    else return null;
  }

  public void bindFc (String itfName, Object itfValue) {
    if (itfName.equals("l")) { l = (Logger)itfValue; }
    else if (itfName.equals("rh")) { rh = (RequestHandler)itfValue; }
  }

  public void unbindFc (String itfName) {
    if (itfName.equals("l")) { l = null; }
    else if (itfName.equals("rh")) { rh = null; }
  }
```

Controllers

Controller interfaces are at the same level as application interfaces

Application interfaces:

```
((Logger)comp.getFcInterface("logger")).log()
```

Controller interfaces:

```
controler = ((BindingController)comp.getFcInterface("binding-
controller"))
controler = Fractal.getBindingController(comp);
controler = Fractal.getLifeCycleController(comp);
....
```

Note that the same interface is:

```
Logger logger =(Logger)comp.getFcInterface("logger");
Interface itf =(Interface)comp.getFcInterface("logger");// same
object, mixin
```

Introspection

Component Architecture at runtime:

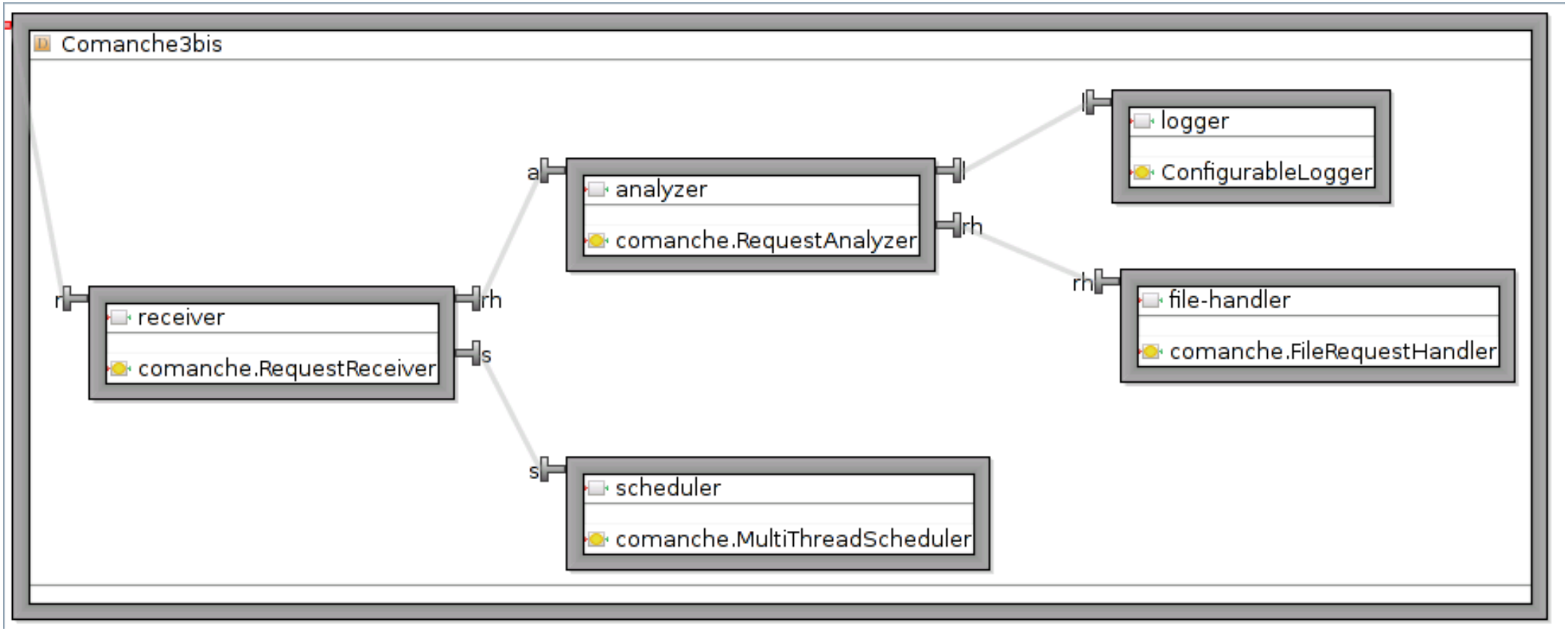
```
public static void print(Component comp, int tabs) throws Exception {
for (int i=1;i<=tabs;i++) System.out.print(" ");
System.out.println(Fractal.getNameController((Component)comp).getFcName());

for (Object o:comp.getFcInterfaces()) {
for (int i=1;i<=tabs;i++) System.out.print(" ");
  System.out.print(" - implements ");
    System.out.println(((org.objectweb.fractal.api.Interface)o).getFcItfName());}

try {
for (Object o:Fractal.getContentController(comp).getFcSubComponents()) {
  print((Component)o,tabs+1);}
} catch( NoSuchInterfaceException e){/* nothing to do */}
}
```

```
Server:
 - implements name-controller (org.objectweb.fractal.api.control.NameController)
 - implements binding-controller (org.objectweb.fractal.api.control.BindingController)
 - implements component (org.objectweb.fractal.api.Component)
 - implements content-controller (org.objectweb.fractal.api.control.ContentController)
 - implements lifecycle-controller
(org.objectweb.fractal.julia.control.lifecycle.LifeCycleCoordinator)
 - implements super-controller
(org.objectweb.fractal.julia.control.content.SuperControllerNotifier)
```

Introspection + Intercession = Reflection



The architecture can be known at runtime.
The architecture can be built at runtime.
The architecture can be ** reconfigured** at runtime (reflection).

Reconfiguration

- Plain Java objects are not safe for reconfiguration
- The main strength of Fractal is reconfiguration
  - at runtime
  - depending on the application state
  - semantic (architecture constraints are always checked)
- Reconfiguration consists of:
  - Rebinding components
  - Creating components
  - Removing components

```
// what if other objects call
methods on removed components?
public class HTTPServer {
  RequestReceiver rr;
  RequestAnalyzer ra;
  RequestDispatcher rd;
  ErrorRequestHandler erh;

public void reconfigure() {
   rd.handler.remove(erh);
   // scheduling cuts here!
    rd.handler.add(new
CoolErrorHandler());
  }
}
```

Automated Management of Lifecycle

```
public static void main(String[] args) throws Exception {
System.setProperty("fractal.provider", "org.objectweb.fractal.julia.Julia");
final Factory f = FactoryFactory.getFactory(FactoryFactory.FRACTAL_BACKEND);
final Component comp = (Component)f.newComponent("Comanche3bis", null);
  new Thread(new Runnable() {
    public void run () {try {
      Thread.sleep(10000);
      Fractal.getLifeCycleController(comp).startFc();
} catch (Exception e) {
  e.printStackTrace();
}}}).start();

// The method run of the interface foo  is actually executed 10 sec. later,
when the component is started
((Runnable)comp.getFcInterface("Runnable")).run();
}
```

Starting and stopping components is essential to keep a correct state during reconfiguration. Fractal/Julia handles the list of waiting calls.

A simple reconfiguration: changing the logger

```
// stopping the system
Fractal.getLifeCycleController(comp).stopFc();

// searching for the analyzer
Component analyzer=null;
for (Object o:Fractal.getContentController(comp).getFcSubComponents()) {
    if
(Fractal.getNameController((org.objectweb.fractal.api.Component)o).getFcName().equals("an
alyzer"))
        analyzer= (org.objectweb.fractal.api.Component)o;
}

// creating the new component
Component newLogger = (Component)f.newComponent("StackTraceLogger", null);

// adding it to validate containment
Fractal.getContentController(comp).addFcSubComponent(newLogger);

// binding it
BindingController bc = Fractal.getBindingController(analyzer);
bc.unbindFc("l");
bc.bindFc("l", newLogger.getFcInterface("l"));

// restarting the component
Fractal.getLifeCycleController(comp).startFc();
```

Exercice #1: what if?

```
// stopping the system
Fractal.getLifeCycleController(comp).stopFc();

// searching for the analyzer
Component analyzer=null;
for (Object o:Fractal.getContentController(comp).getFcSubComponents()) {
    if
(Fractal.getNameController((org.objectweb.fractal.api.Component)o).getFcName().equals("an
alyzer"))
        analyzer= (org.objectweb.fractal.api.Component)o;
}

// creating the new component
Component newLogger = (Component)f.newComponent("StackTraceLogger", null);

// adding it to validate containment
Fractal.getContentController(comp).addFcSubComponent(newLogger);

// binding it
BindingController bc = Fractal.getBindingController(analyzer);
bc.unbindFc("l");
bc.bindFc("l", newLogger.getFcInterface("l"));

// restarting the component
Fractal.getLifeCycleController(comp).startFc();
```

Exercice #2: what if?

```
// stopping the system
Fractal.getLifeCycleController(comp).stopFc();

// searching for the analyzer
Component analyzer=null;
for (Object o:Fractal.getContentController(comp).getFcSubComponents()) {
    if
(Fractal.getNameController((org.objectweb.fractal.api.Component)o).getFcName().equals("an
alyzer"))
        analyzer= (org.objectweb.fractal.api.Component)o;
}

// creating the new component
Component newLogger = (Component)f.newComponent("StackTraceLogger", null);

// adding it to validate containment
Fractal.getContentController(comp).addFcSubComponent(newLogger);

// binding it
BindingController bc = Fractal.getBindingController(analyzer);
bc.unbindFc("l");
bc.bindFc("l", newLogger.getFcInterface("l"));

// restarting the component
Fractal.getLifeCycleController(comp).startFc();
```

Exercice #3: what if?

```
// stopping the system
Fractal.getLifeCycleController(comp).stopFc();

// searching for the analyzer
Component analyzer=null;
for (Object o:Fractal.getContentController(comp).getFcSubComponents()) {
    if
(Fractal.getNameController((org.objectweb.fractal.api.Component)o).getFcName().equals("an
alyzer"))
        analyzer= (org.objectweb.fractal.api.Component)o;
}

// creating the new component
Component newLogger = (Component)f.newComponent("StackTraceLogger", null);

// adding it to validate containment
Fractal.getContentController(comp).addFcSubComponent(newLogger);

// binding it
BindingController bc = Fractal.getBindingController(analyzer);
bc.unbindFc("l");
bc.bindFc("l", newLogger.getFcInterface("l"));

// restarting the component
Fractal.getLifeCycleController(comp).startFc();
```

Exercice #4: what if?

```
// stopping the system
Fractal.getLifeCycleController(comp).stopFc();

// searching for the analyzer
Component analyzer=null;
for (Object o:Fractal.getContentController(comp).getFcSubComponents()) {
    if
(Fractal.getNameController((org.objectweb.fractal.api.Component)o).getFcName().equals("an
alyzer"))
        analyzer= (org.objectweb.fractal.api.Component)o;
}

// creating the new component
Component newLogger = (Component)f.newComponent("StackTraceLogger", null);

// adding it to validate containment
Fractal.getContentController(comp).addFcSubComponent(newLogger);

// binding it
BindingController bc = Fractal.getBindingController(analyzer);
bc.unbindFc("l");
bc.bindFc("l", newLogger.getFcInterface("l"));

// restarting the component
Fractal.getLifeCycleController(comp).startFc();
```

Correctness of reconfiguration

~~// Fractal.getLifeCycleController(comp).stopFc();~~

```
IllegalLifeCycleException: The component is not stopped (component
= /Comanche3bis/analyzer)
```

~~// Fractal.getContentController(comp).addFcSubComponent(newLogger);~~

```
IllegalBindingException: Not a local binding (client interface =
/Comanche3bis/analyzer.l, server interface = /StackTraceLogger.l)

IllegalContentException: Would create non local bindings (super
component = /Comanche3bis, sub component = /Comanche3bis/logger)
```

~~// bc.unbindFc("l");~~

```
IllegalBindingException: Already bound (client interface =
/Comanche3bis/analyzer.l)
```

~~//bc.bindFc("l", newLogger.getFcInterface("l"));~~

```
IllegalBindingException: Mandatory client interface unbound (client
interface = /Comanche3bis/analyzer.l)
```

Fractal/Julia ensures that the reconfiguration is correct.

Summary

- Fractal provides you with:
  - declarative software architecture
    - multiplicity, optional, polarity (client/server)
  - reflection on the components (introspection and intercession)
  - a safe lifecycle management infrastructure
- Fractal/Julia is LGPL and can be used in industrial settings
- Fractal/Julia can be used on low memory devices (e.g. embedded systems)