Software Components

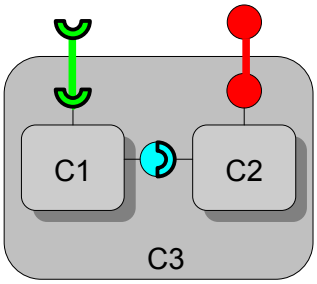
Christian Perez
LIP/INRIA
2009-2010

Content

- Overview
- Fractal
- CCM (CORBA Component Model)

Software Component

- Black boxes
- Interaction with the environment
  - Well defined interfaces
  - A set of typed ports
- Building an application
  - Assembling components
  - Either dynamic or static ADL
- Implementation
  - External model (C++, Fortran, Java, etc …)
  - An assembly (composite)



Forms of interactions

- Explicit message
  - Data (event)/Message/Document
- Implicit message
  - RPC/RMI/GridRPC/Service invocation
- Data sharing
- Workflow/dataflow
- Skeleton/template
- Chemical

Communication kind
- *Synchronous/Asynchronous*
  - *Deferred/Future*

Entity Kind
- *Sequential/Parallel*
- *Known/unknown*

Fractal

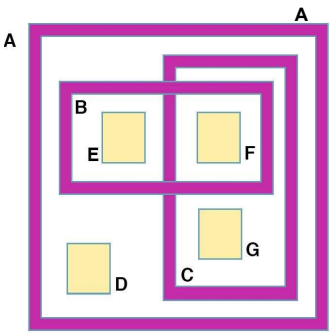
http://fractal.ow2.org

Overview

- General model defined by France Télécom R&D and INRIA (2002)
- Features
  - Few restrictions
  - Self-* capacity
    - observation,
    - control,
    - reconfiguration.
  - Open and adaptable :
    - Extra-functional services can be personalized

Fractal Component

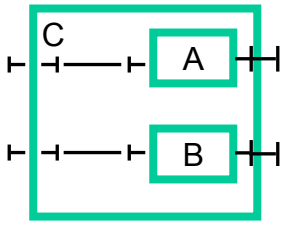
- Hierarchical model
  - Primitive components
  - Composite
- Component can be shared
- Two parts:
  - Contents
    - set of (functional) components
  - Membrane
    - Interfaces and controllers

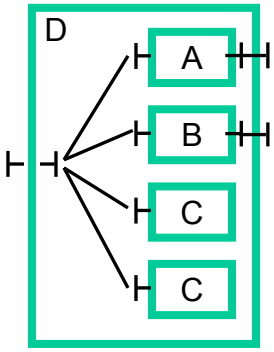


Fractal Components

- Primitive
  - Any language (OO)

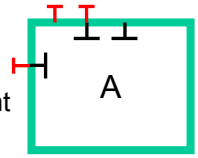
- Composite
  - Internal interfaces
  - Internal bindings
  - Imbrications



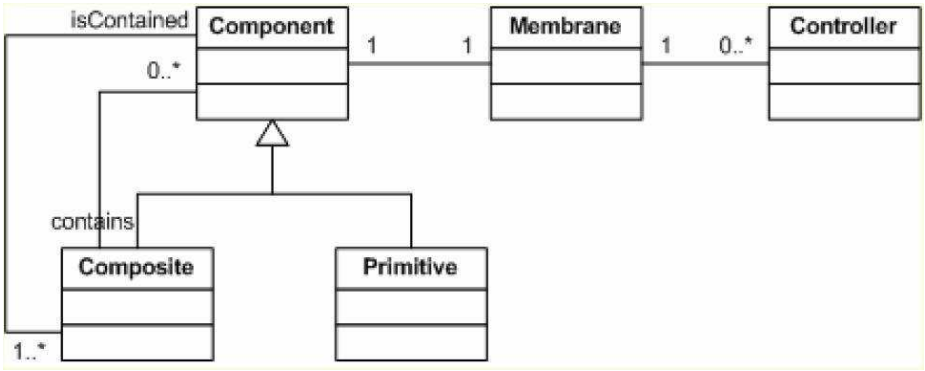


Fractal Component

- Functional interfaces
  - Access points (I/O)
  - Definition
    - Name, Signature (type), client/server, mandatory/optional, simple/multiple
- Controllers
  - Optional
  - Reside in the membrane
    - Used to be object, now can also be component
  - Accessible through a control interface
  - Examples
    - Introspection, binding, etc.



Fractal Component



Examples of Interfaces

```
interface Component {
   any[] getFcInterfaces ();
   any   getFcInterface (string itfName) …;
   Type  getFcType ();
}
interface Type {
   boolean isFcSubTypeOf (Type t);
}

interface Interface {
   string    getFcItfName ();
   Type      getFcItfType ();
   Component getFcItfOwner ();
   boolean   isFcInternalItf ();
}

interface GenericFactory {
   Component newFcInstance (Type t, any controllerDesc,
                            any contentDesc) …;
}
```

Examples of Controllers

```
interface AttributeController { }

interface BindingController {
   string[] listFc ();
   any      lookupFc (string clientItfName) …;
   void     bindFc (string clientItfName, any serverItf) …;
   void     unbindFc (string clientItfName)…;
}

interface ContentController {
   any[]       getFcInternalInterfaces ();
   any         getFcInternalInterface (string itfName)…;
   Component[] getFcSubComponents ();
   void        addFcSubComponent (Component c) …;
   void        removeFcSubComponent (Component c) …;
}
```

Level of conformance

	C	I	CT, IT	AC, BC, CC, LC	F	T
0						
0.1				X		
1	X					
1.1	X			X		
2	X					
2.1	X			X		
3	X	x				
3.1	X	X		X		
3.2	X	X		X	X	
3.3	x	X		X	x	x

C: Component
AC: Attribute
F: Factory

I: Interface
BC: Binding
T: Template

CT: ComponentType
CC: Content

IT: InterfaceType
LC: LifeCycle

Dynamically Instantiation (1/3)

- Creating types

```
Component boot = Fractal.getBootstrapComponent();
TypeFactory tf = (TypeFactory)boot.getFcInterface("type-factory");

ComponentType rType = tf.createFcType(new InterfaceType[] {
   tf.createFcItfType("m", "M", false, false, false)
});

ComponentType cType = tf.createFcType(new InterfaceType[] {
   tf.createFcItfType("m", "M", false, false, false),
   tf.createFcItfType("s", "S", true, false, false)
});

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

Dynamically Instantiation (2/3)

- Creating template

```
Component boot = Fractal.getBootstrapComponent();
GenericFactory gf =
   (GenericFactory)boot.getFcInterface("generic-factory");

Component rTmpl = gf.newFcInstance(rType,
   "compositeTemplate", new Object[] {"composite", null});

Component cTmpl = gf.newFcInstance(
   cType, "template", new Object[] {"primitive", "CImpl"});

Component sTmpl = gf.newFcInstance(
   sType, "template", new Object[] {"primitive", "SImpl"});
```

Dynamically Instantiation (3/3)

- Filling template

```
ContentController cc =
   (ContentController)rTmpl.getFcInterface("content-
   controller");
cc.addFcSubComponent(cTmpl);
cc.addFcSubComponent(sTmpl);

((BindingController)rTmpl.getFcInterface("binding-controller"))
   .bindFc("m", cTmpl.getFcInterface("m"));

((BindingController)cTmpl.getFcInterface("binding-controller"))
   .bindFc("s", sTmpl.getFcInterface("s"));
```

- Instantiating a component

```
Component r =
   ((Factory)rTmpl.getFcInterface("factory")).newFcInstance();
```

Fractal ADL

- Primitive component

```
<definition name="ClientImpl">
  <interface name="r" role="server"
             signature="java.lang.Runnable"/>
  <interface name="s" role="client" signature="Service"/>
  <content class="ClientImpl"/>
</definition>
```

Fractal ADL

- Composite component

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