Safe Dynamic Reconfigurations of Fractal Architectures with FScript

5th Fractal Workshop

Pierre-Charles David Thomas Ledoux

France Télécom R&D

OBASCO Group (LINA / INRIA)

2006-07-03

Motivation

- Fractal is dynamic and reflexive
  - supports introspection
  - and unanticipated architecture reconfigurations
- However :
  - Fractal APIs are minimalist and orthogonal
    - designed for tools builders, not direct usage (ex : ADL)
  - Language integration (Java) is weak
    - two related, but distinct notions of interfaces
    - lots of casts, `try/catch` needed everywhere
  - Direct usage in a GPL can be dangerous
    - no guarantees (termination, calling dangerous methods, using implementation-specific code...)

Design goals

Provide a Domain-Specific Language to describe and execute safe dynamic reconfigurations on Fractal architectures.

- Domain-Specific Language ?
  - custom notation : closer to the domain
  - custom semantics : can offer guarantees, analysis...
- Safe ?
  - reconfigurations seen as transactions
  - guarantees inspired by ACID properties

FScript main features

Two parts :

1. FPath
   - syntax for navigating in the architecture and selecting elements
   - embedded sub-language of FScript
   - purely without side-effects
   - usable by itself
2. FScript
   - define architectural reconfigurations
   - uses/extends FPath (expressions)
   - imperative, scripting-like language (statements)
   - reconfigurations have transaction-like semantics

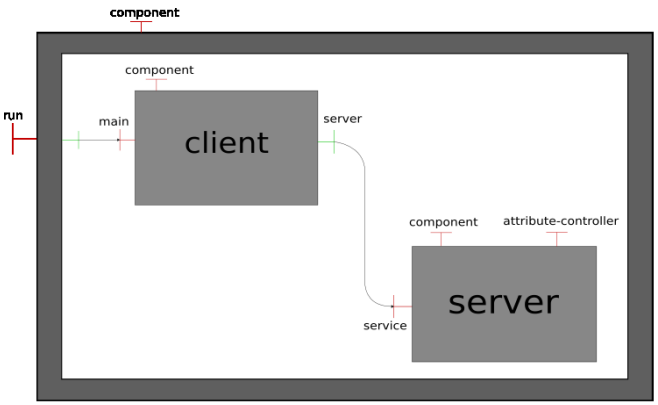
FPath : navigating inside Fractal architectures

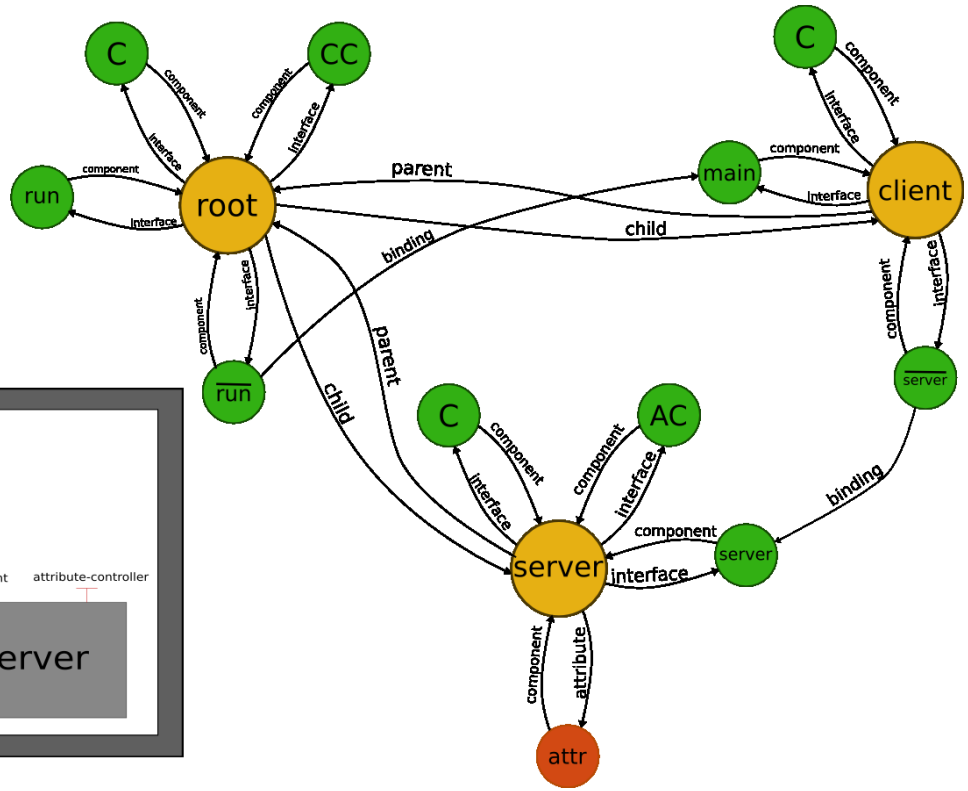
- Notation to navigate in a Fractal architecture and select elements of it
  1. Find the subcomponents of C which provide interface I.
  2. Which components in my application offer configuration attributes, and what are these attributes ?
  3. Which components are shared ?
- Syntax and operational model inspired by XPath [W3C]
  - but implementation does *not* use XML

FPath : data model

Fractal components seen as a (virtual) directed graph

- nodes : components, interfaces, attributes and methods
- arcs with labels : indicate the relations between nodes





FPath : path expressions

- A (relative) path is made of steps : `step1/step2/.../stepN`
- Each step is made of : `axis::test[predicate1][predicate2]`
  1. an axis name, matched against arcs labels
  2. a test (name or *), matched against node names
     - (for components : `NameController.getFcName()`)
  3. an optional list of predicates used for filtering
     - (almost) arbitrary FPath expressions

Example

```
sibling::*/interface::printer[provided(.)][not(bound(.))]
```

FPath also includes "normal" expressions : numbers, strings, arithmetic, comparisons, variables and functions calls

FPath : evaluation

- Evaluating a step
  1. for each of the initial node-set, follow the outgoing arcs whose label matches the step's axis
  2. remove those whose name does not match the test
  3. retain only the node which match all the predicates
- Evaluating a path
  1. starting from an initial node-set,
  2. evaluate the first step as above
  3. feed the resulting node-set to the next step
  4. return the result of the last step

FPath : examples

Find the subcomponents of *C* which provide interface *aService*

```
$c/child::*/interface::aService
$c/child::*/interface::aService/component::*
```

Which components in my application offer configuration attributes, and what are these attributes ?

```
$root/descendant-or-self::*[attribute::*]
$root/descendant-or-self::*/attribute::*
```

Which components are shared ?

```
$root/descendant-or-self::*[count(parent::*) > 1]
```

FPath vs Java

FPath expression

```
$root/child::client/interface::s/binding::*/attribute::header
```

Equivalent Java

```
try {
  Object[] children = Fractal.getContentController(root).getFcSubComponents();
  for (int i = 0; i < children.length; i++) {
    Component kid = (Component) children[i];
    String name = "";
    try { name = Fractal.getNameController(kid).getFcName(); }
    catch (NoSuchInterfaceException nsie) { }
    if (name.equals("client")) {
      try {
        Interface itf = (Interface) Fractal.getBindingController(kid).lookupFc("s");
        if (itf != null) {
          Component server = itf.getFcItfOwner();
          AttributeController ac = Fractal.getAttributeController(server);
          Class klass = ac.getClass();
          try {
            Method meth = null;
            try { meth = klass.getMethod("getHeader", null); }
            catch (NoSuchMethodException nime) { }
            if (meth != null) {
              try { return meth.invoke(ac, null); }
              catch (Exception e) { /* ignore */ }
          } catch (Exception e) { /* ignore */ }
        }
      } catch (NoSuchInterfaceException nsie) { /* ignore */ }
    }
```

FPath axes

- `component` : the component owning a node
- `(internal-)interface` : all the external/internal interfaces of a component
- `attribute` : configuration attributes
- `binding` : from a client interface to the server interface it is bound to
- `child(-or-self)` : direct sub-components
- `parent(-or-self)` : direct super-components
- `descendant(-or-self)` : all sub-components, including indirect
- `ancestor(-or-self)` : all super-components, including indirect
- `sibling(-or-self)` : all components with one direct parent in common
- `method` : the methods of an interface

FScript reconfigurations

- FScript is used to define reconfiguration actions
- Primitive actions : Fractal API
  - `add()`, `remove()`, `bind()`, `unbind()`
  - `new()`, `start()`, `stop()`, `set-value()`...
  - easily extensible (like Fractal)
- Voluntarily limited control structures
  - sequence, `if/then/else`, `foreach`, `return`
  - recursive definitions are forbidden
- Design and implementation guarantee the consistency of the reconfigurations

Exemple FScript reconfiguration

Automatic connection of the interface required by a component

```
action auto-bind(comp) = {
  // Select the interfaces to connect
  clients := $comp/interface : :*[client(.)][mandatory(.)][not(bound(.))];
  foreach itf in $clients do {
    // Search for compatible server interfaces
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)];
    if (not(empty?($candidates))) then
      // Connect one of these candidates
      bind($itf, one-of($candidates));
    }
  }
}
```

Exemple FScript reconfiguration

Automatic connection of the interface required by a component

```
action auto-bind(comp) = {
  // Select the interfaces to connect
  clients := $comp/interface : :*[client(.)][mandatory(.)][not(bound(.))];
  foreach itf in $clients do {
    // Search for compatible server interfaces
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)];
    if (not(empty?($candidates))) then
      // Connect one of these candidates
      bind($itf, one-of($candidates));
    }
  }
}
```

Exemple FScript reconfiguration

Automatic connection of the interface required by a component

```
action auto-bind(comp) = {
  // Select the interfaces to connect
  clients := $comp/interface : :*[client(.)][mandatory(.)][not(bound(.))];
  foreach itf in $clients do {
    // Search for compatible server interfaces
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)];
    if (not(empty?($candidates))) then
      // Connect one of these candidates
      bind($itf, one-of($candidates));
    }
  }
}
```

Exemple FScript reconfiguration

Automatic connection of the interface required by a component

```
action auto-bind(comp) = {
  // Select the interfaces to connect
  clients := $comp/interface : :*[client(.)][mandatory(.)][not(bound(.))];
  foreach itf in $clients do {
    // Search for compatible server interfaces
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)];
    if (not(empty?($candidates))) then
      // Connect one of these candidates
      bind($itf, one-of($candidates));
    }
  }
}
```

Exemple FScript reconfiguration

Automatic connection of the interface required by a component

```
action auto-bind(comp) = {
  // Select the interfaces to connect
  clients := $comp/interface : :*[client(.)][mandatory(.)][not(bound(.))];
  foreach itf in $clients do {
    // Search for compatible server interfaces
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)];
    if (not(empty?($candidates))) then
      // Connect one of these candidates
      bind($itf, one-of($candidates));
    }
  }
}
```

Guarantees offered by FScript

- Transactional approach
  - reconfigurations should not "break" the application

1. *Termination* : guaranteed by language design
   - however, no time bound
2. *Atomicity* : guaranteed by the implementation
   - currently : optimistic approach ("try/repair")
3. *Consistancy* : guaranteed by the implementation
   - test at the end of a reconfiguration $\rightarrow$ rollback in case of problem
   - Julia does most of the checks during the reconfiguration
4. *Isolation* : guaranteed by the implementation
   - two reconfigurations can't interfere
   - currently : global lock
   - ongoing work on fine-grained locking (M. Léger)

FScript usage : API

```
FScriptInterpreter fscript = new FScriptInterpreter();

// Load custom actions from file
fscript.load("myaction.fscript");

// Create a node to represent an existing component
NodeFactory fact = fscript.getNodeFactory();
FractalNode node = fact.newComponentNode(aComponent);

// Call "myaction($node)" programmatically
Object result = fscript.execute("myaction",
                                node,
                                null /* env */);
```

FScript usage : Interactive console

```
% java org.objectweb.fractal.fscript.Console

FScript> !load cache.fscript
Loaded action 'enable-cache'.
Loaded action 'disable-cache'.
FScript> c := new("comanche.Comanche");
FScript> start($c);
FScript> !run $c r
FScript> enable-cache($c);
FScript> !quit
```

Conclusion

- FPath
  - custom notation for navigation and selection in Fractal architectures
  - loosely modeled after XPath
  - side-effects free
  - usable by itself as a query language
  - syntax is generic (+) but verbose (-)
- FScript
  - extends FPath with reconfiguration actions definitions
  - structural reconfigurations + attributes control
  - primitives map standard Fractal operations
    - easily extensible, like Fractal
  - imperative language, minimal control structures
  - reconfigurations are safe
    - transaction-like semantics