Reliable reconfigurations in Fractal

Motivations

In the context of system (self-)administration, dynamic reconfigurations are crucial

- corrective and preventive maintenance, fault-handling, optimizing, context awareness, etc.

But how to ensure consistency of reconfigurations?

- violation of model or architectural invariants, hardware crashes

Propositions

A consistency model for Fractal architectures via integrity constraints

$F = (F_E, F_R, F_P)$

F_E = *Component U Interface U Attribute*

F_R = *hasInterface U hasAttribute U hasChild U boundTo*

F_P = *binary predicates (e.g. noCycle(E,R))*

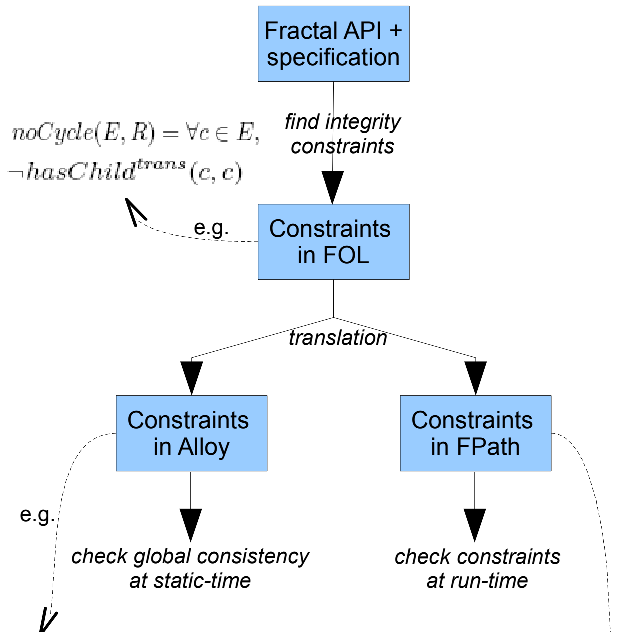
A run-time support for fault-tolerance

- A transactional approach

A Fractal framework with dedicated controllers, ADL extensions

```
<definition name="ClientServer">
  <interface name="main" role="server" signature="java.lang.Runnable" />
  <component name="client"> ... </component>
  <component name="server"> ... </component>
  <binding client="this.main" server="client.main" />
  <binding client="client.service" server="server.service" />
  <constraints>
    <!-- Contrainte : un sous-composant doit fournir l'interface "service" -->
    <constraint value="size(./child::*[./interface::service])==1" />
    <!-- Contrainte : le composant ne doit pas être partagé -->
    <constraint value="no_sharing(.)" />
  </constraints>
</definition>
```

Checking Fractal configurations with invariants




```
fact noCycle {
  all a:Configuration | no c: Component | c->c in a.^child
}
```

e.g.

```
function no_cycle(c) {
  for component : $c::descendant/* {
    if ($c == $component) {
      return false();
  }
  return true();
}
```

Checking Fractal reconfigurations with pre/post-conditions

Addition of a subcomponent : $add(Component\ p, Component\ c)$

Preconditions

$p, c \in E$

$\exists i \in E,\ hasInterface(p,i) \wedge contentCtrlItf(i)$

$\neg(p = c)$

$\neg hasChild^{trans}(c,p)$

Postconditions

$E' = E$

$R' = R \cup \{hasChild(p,c)\}$

translation in FScript

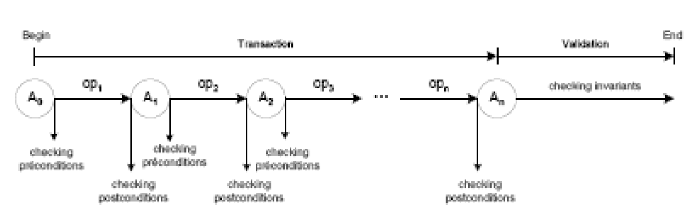
```
action addSafe(p, c) {
  //Preconditions
  assert($p/interface::content-controller)); -- p must be a composite component
  assert(not($p == $c)); // p must not equals c
  assert(size(intersection($p, $c/descendant::*)) == 0); -- p must not be a
                                                          -- descendant of c

  //Operation execution
  add($p, $c);

  //Postconditions
  assert(size(intersection($c, $p/child::*)) == 1); -- c must be a child of p
}
```

A transactional approach (a.k.a FractalTXR)

- Motivation
  - allows fault recovery, architecture consistency, concurrency of reconfigurations, self-repair
- Using FractalTXR
  - automatic demarcation when using FScript



A flat transaction model adapted for dynamic reconfiguration

- **Atomicity:** reconfiguration operations either all occur, or nothing occurs (atomicity with undo operations)
- **Consistency:** ensure that the architecture remains in a consistent state (verify integrity constraints)
- **Isolation:** independent reconfigurations cannot access or see the data in an intermediate state during a transaction (pessimistic approach with locking)
- **Durability:** the result of a reconfiguration is persistent (serialization of ADL definitions)

Marc Léger, Thomas Ledoux, Thierry Coupaye