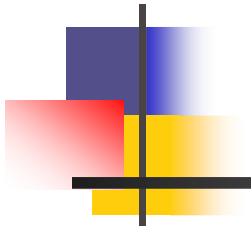
An Aspect-Oriented Approach for Developing Self-Adaptive Fractal Components

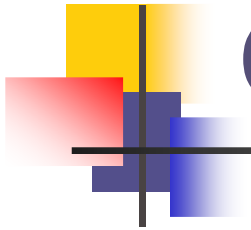


SAFRAN: **S**elf-**A**daptive **FRA**ctal compo**N**ents

Pierre-Charles David* Thomas Ledoux**

* France Télécom, R&D
** OBASCO Group, EMN / INRIA, Lina

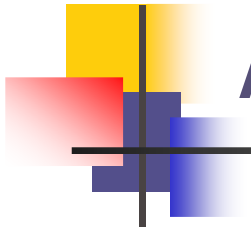
Software Composition 2006
2006-03-25, Vienna



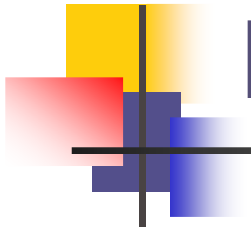
Outline

- Motivation & Problem Statement
- Software Adaptation as an Aspect
- SAFRAN
- Example Scenario
- Conclusion

Towards Self-Adaptive Applications



- Nowadays, how to develop an software application?
- Execution contexts are more and more variable
  - statically: from embedded systems to grids
  - dynamically: resources availability
- Solution: build *self-adaptive applications*
  - *context-aware*
    - ables to adapt to the context's evolutions
    - ables to leverage new possibilities which can appear dynamically
  - *autonomous*
    - applications are both the agent and the target of the adaptation

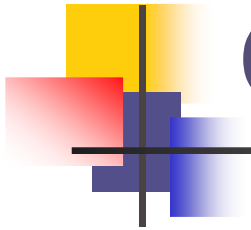


Problem Statement

Need for self-adaptive applications …

… but no easy way to build them!

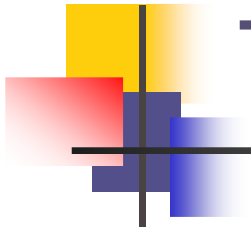
- Problems
  - execution conditions can not be predicted at build time
  - even if they are partially known, conditions change dynamically
  - adaptation decisions are hardwired in applications
  - business code is « polluted » by the adaptation code



Our objective

- Ease the development of self-adaptive applications
  - develop the adaptation code separately…
    - business code becomes simpler and more reusable
  - … and integrate it dynamically into the business code
    - adaptation policies are developped « Just In Time »

Adaptation code is well modularized both spatially and temporally



Towards an Adaptation Aspect

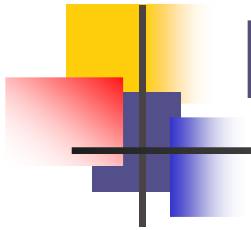
- Software adaptation appears as a *cross-cutting concern*
  - AOP gives us adequate abstractions to solve this issue
- How to « aspectize » the adaptation concern?
  - Make use of similarities between the event-based nature of the adaptation process and the Event-based AOP (EAOP) approach
    - adaptation process: Observation –> Decision –> Action
    - EAOP : join-points are runtime events which trigger advice actions

Structure of the Adaptation Aspect

- Adaptation Aspect = Adaptation policies with reactive rules
- Join-points/Point-cuts
  - Two categories of events:
    - *internal*: classical join-points (cf. Event-based AOP)
    - *external*: evolutions of the execution context

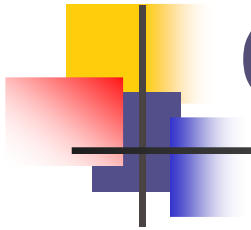
  → *extension of the « traditional » AOP language like AspectJ*
- Advice Language
  - adjust the target application (tuning, parameterization, architectural reconfiguration)
  - Not a GPL but a DSL (Domain-Specific Language) allowing « safe » adaptation

  → *restriction of the « traditional » AOP language like AspectJ*



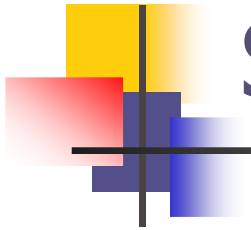
Results

- SAFRAN, an extension of the Fractal model to build self-adaptive applications
  - adaptation aspect is developed separately
  - a Domain-Specific Language (DSL) to express it
  - dynamic integration/weaving with business code



Outline

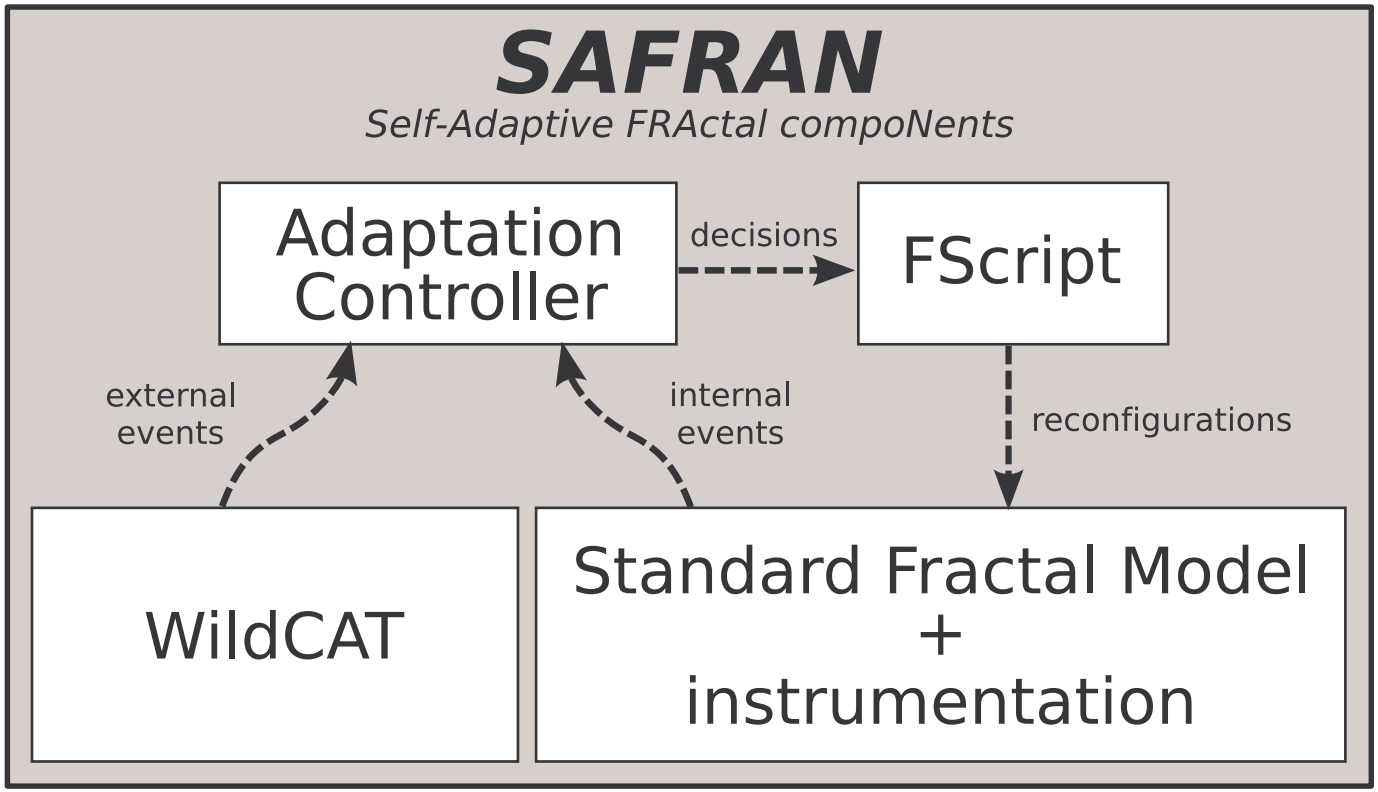
- Motivation & Problem Statement
- Software Adaptation as an Aspect
- SAFRAN
  - Overview
  - Fractal Components: the Base Program
  - Reconfigurations with FScript: the Advice Language
  - Internal & External Events: the Join-Points
  - Adaptation Policies: the Aspect
- Example Scenario
- Conclusion



SAFRAN Overview

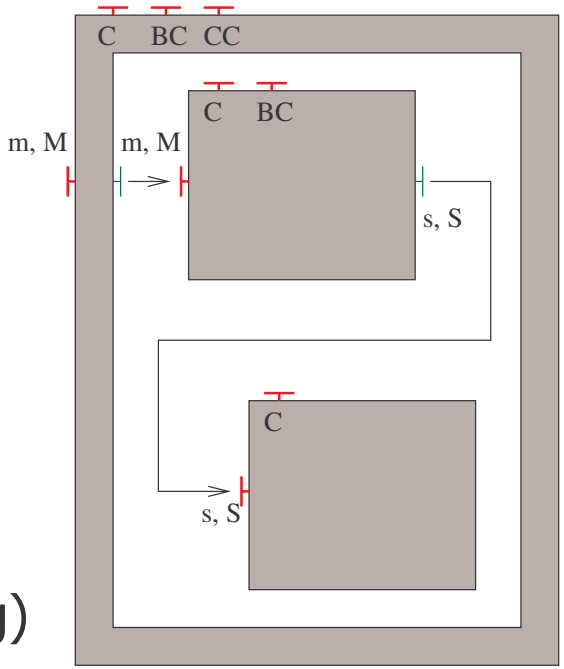
- Base program: Fractal components architecture
- Point-cuts: notification of events
  - *internal*: via code instrumentation
  - *external*: via the WildCAT context-awareness framework
- Advices: safe architectural reconfigurations with the FScript language
- Aspects: adaptation policies dynamically weaved by the AdaptationController
  - synchronises the execution of the adaptation code with the base program and context evolutions

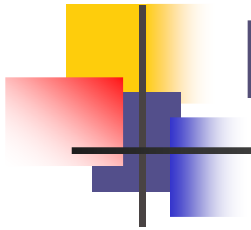
SAFRAN Architecture



The Fractal Component Model

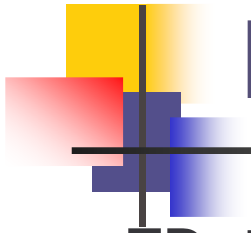
- Component model developed by FT R&D and INRIA
  - `http://fractal.objectweb.org`
  - language independant (*Java*, C, C#, Smalltalk)
- Why Fractal?
  - minimal but extensible kernel
  - supports dynamic reconfigurations
  - reflective → unanticipated reconfigurations
- Fractal components
  - *Controller* manages interactions with the outside
  - Control and services interfaces (ports)
  - Primitive or composites components (with sharing)
  - Connections between type-compatible interfaces
- Enables the construction of adaptable (reconfigurable) applications
  - first step towards *self-adaptive* applications





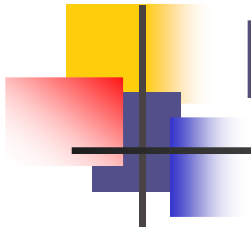
FScript: the Advice Language

- A Domain-Specific Language (DSL) to program Fractal components reconfigurations
- Features
  - « scripting » language, imperative, interpreted (dynamic)
  - syntax, expressivity and execution model adapted
  - access to all the Fractal operations
  - guarantees reconfiguration consistency



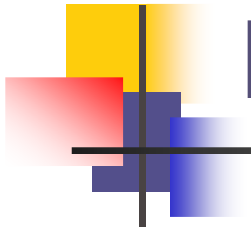
FPath Notation

- FPath (a subset of FScript): notation for *navigation* and *selection* of elements in a Fractal architecture
  - select C's subcomponent which provides interface I.
  - which are the configurable components in my application?
- Syntax and execution model inspired by XPath
  - but does not use XML
- Fractal components seen as a directed graph
  - nodes: components, interfaces and configuration attributes
  - arcs and labels: indicates relationship between nodes
- Syntax
  - *axe1::test1 [pred1]/ axe2::test2[pred2]/…*
- Example
  - `sibling::*/interface::*[provided(.)][not(bound(.))]`



FScript Reconfigurations

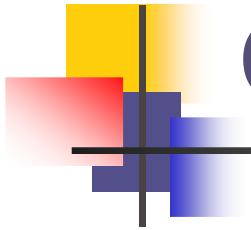
- Allows the definition of reconfigurations actions
- Primitive actions: Fractal APIs
  - `add(), remove(), bind(), unbind()`
  - `new(), start(), stop(), set-value()`
  - easily extensible (like Fractal)
- Control structures voluntarily limited
  - `sequence, if/then/else, foreach, return`
  - recursive definitions forbidden
- Variables manipulation
  - `$varname`



FScript Reconfiguration Example

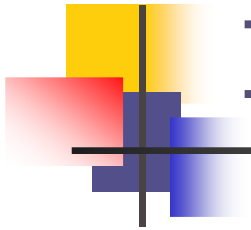
- Automatic connection of required interfaces

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



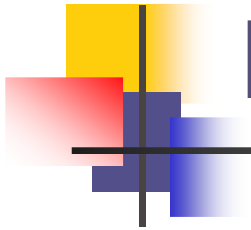
Guarantees Offered by FScript

- Transactional approach
  - an adaptation can not break the application!
- *Termination (*guaranteed by the language design)
- *Atomicity (*guaranteed by the implementation)
  - currently : try-repair approach (undo)
- *Consistency (*guaranteed by the implementation)
  - test at the end of the reconfiguration ! rollback in case of problem
  - Julia, the Fractal implementation used, also offers some guarantees
- *Isolation (*guaranteed by the implementation)
  - currently : global lock (mutex)
  - work in progress: finer-grained synchronisation



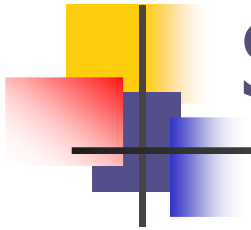
Internal Events

- Dynamic instrumentation of the application components
  - in the component membrane, not application code
- Messages-related events:
  - `message-{received,returned,failed}`
  - ex: `message-received($target/interface::my-service)`
  - execution flow of the application
- Recongurations-related events:
  - `subcomponent-added,binding-destroyed...`
  - ex: `component-started($target/sibling::*)`
  - any change in the architecture



External Events with WildCAT

- The context is usually implicit (by definition)
- WildCAT reifies and gives access to it
  - simple hierarchical data model
  - dynamic
  - pull and push APIs
- Produces external events for SAFRAN
  - `changed(sys://storage/memory#free)`
  - `realized(geo://location/logical#room = "E202")`
  - `appears(sys://devices/input/*)`
  - `disappears(net://host/smtp.example.com/services/smtp)`
- WildCAT is reusable as a general-purpose context-awareness framework

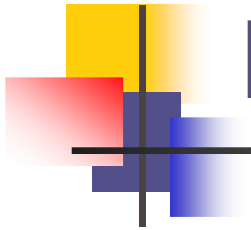


SAFRAN Adaptation Policies

- Integration of the above contributions
- Allows to express reactive adaptation rules:
  - Event → Condition → Action
- Policy = list of rules

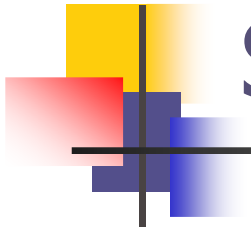
```
policy example() = {
   rule { when <event1> if <cond1> do <action1> };
   rule { when <event2> if <cond2> do <action2> };
   rule { when <event3> if <cond3> do <action3> };
   ...
}
```

- Attached to a `$target` component
- Via the `adaptation-controller`
  - specialized *weaver* for the adaptation aspect



Policy Execution

- Global lifecycle
  - definition → attachment → execution → detachment
- Attachment
  - registration to WildCAT and instrumentation of the component
- Execution phase: waits for events
  - select of the rules to apply
  - compute the global reaction
  - execute it as a reconfiguration transaction
- Fixed composition rules
  - between rules in a policy (executed in sequence, atomically)
  - between policies on a component (reactions executed sequentially, each in its transaction)
  - between adaptive components in the application (partial order defined by the components' composition relation)



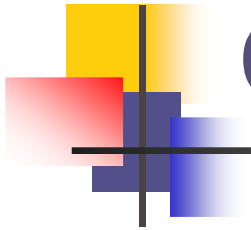
Self-Adaptive Cache Example

- Add a cache to a simple web server
- Problem: how much memory to allocate?
  - not enough! useless
  - too much! thrashing
- Ideal size depends on the amount of free memory
  - changes dynamically, can not be predicted
- Adaptation policy: binds the cache size to the amount of free memory

Self-Adaptive Cache: Adaptation Policy

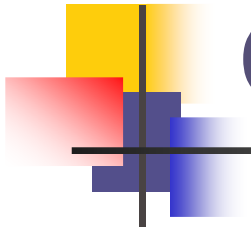
```
policy adaptive-cache = {
 rule {
  when mem : changed(sys://storage/memory#free)
  if (sys://storage/memory#free >= 10*1024)
  do {
   cache := enable-cache($target);
   size := 0.8 * ($mem.new-value + $cache/@currentSize);
   max := sys://storage/memory#used - $cache/@currentSize + $size;
   if ($max < sys://storage/memory#total - 10*1024)
     set-value($cache/@maximumSize, $size);
  }
 }
}
```

- Cache size adjusted automatically
- Adaptation policy does not pollute the cache code
- Policy can be changed dynamically



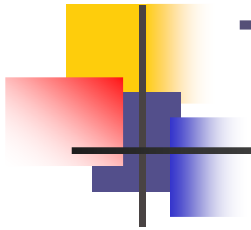
Conclusion: contributions

- An aspect-oriented approach for the development of self-adaptive applications
  - spatial and temporal decoupling
- SAFRAN: an implementation of this approach in the Fractal component model
  - WildCAT: a framework for context-aware applications
  - FPath & FScript: a DSL to program consistent reconfigurations



Conclusion: perspectives

- Extend SAFRAN to allow the adaptation of distributed applications
  - new FScript primitives for distribution-aware reconfigurations
  - new WildCAT context domains
  - execution model to support coordinated adaptation policies of remote components
- Validation of the approach on a bigger application!



Thank you for your Attention!

- Questions?