WildCAT: A Generic Framework For Context-Awareness

Pierre-Charles David* Thomas Ledoux**

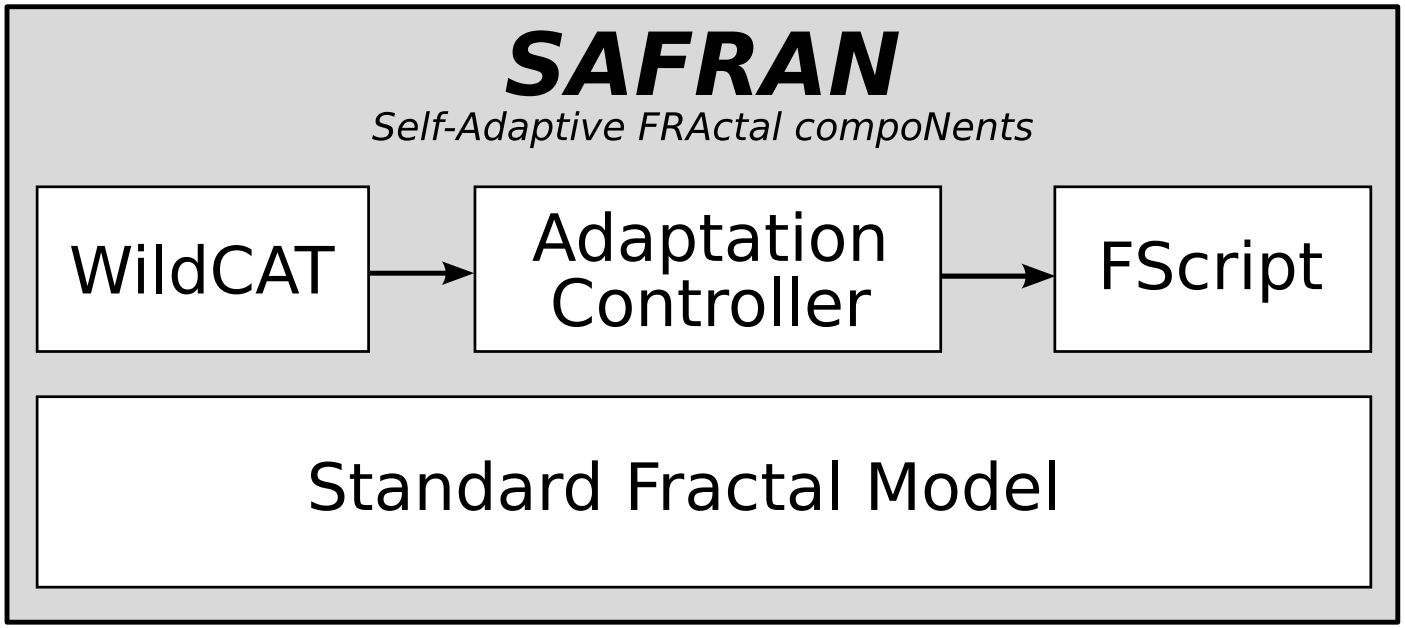
* France Télécom, Recherche & Développement
** OBASCO Group (EMN/INRIA, LINA)

MPAC 2005, Grenoble
2005-11-28

Outline

What is WildCAT ?

- Context-Awareness Toolkit
  - Java library
  - extensible framework
- Part of a bigger system : SAFRAN
  - extended component model for self-adaptive applications
  - uses reactive rules (E-C-A) to program adaptation policies
  - WildCAT is used to trigger context-related events
  - reusable outside of SAFRAN



Requirements

- Need to know the execution context and its evolutions to adapt to it
- By definition, this context is usually implicit

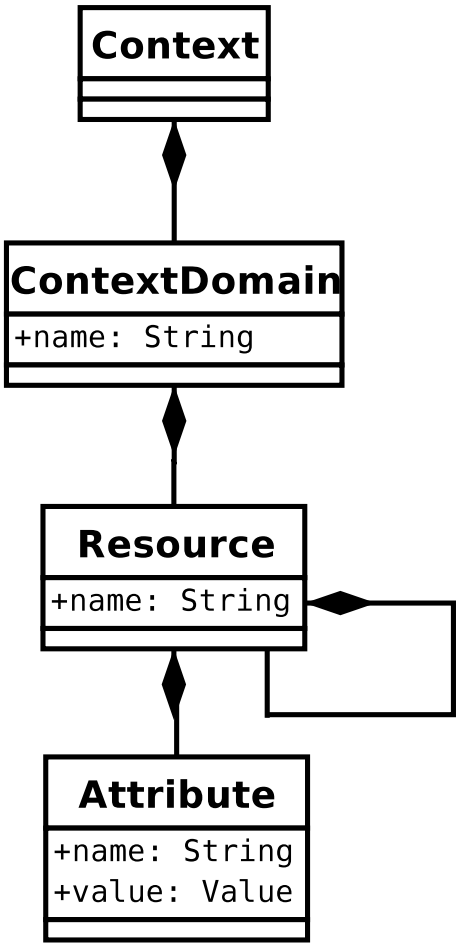
  An application's execution context regroup all the external entities and situations which influence its QoS/performance (quantitative and qualitative) as perceived by users. [my definition]

- WildCAT's roles :
  - reify the context (data acquisition and modeling)
  - give applications access to it (API)
- Objective : ease the creation of context-aware applications
  - pragmatic approach

Design principles (and consequences)

1. Simple to use by application programmers
   - simple data model and APIs
   - usable as a "black-box"
2. Generic (not tied to a particular application domain)
   - pervasive apps *vs* servers, local *vs* distributed...
   - need a configuration interface
3. Allow for efficient implementation
   - minimize framework overhead
   - enable custom implementations for specific domains
4. Extensible
   - built as a framework
   - support different extension scenarios

WildCAT's data model



Context main entry point
singleton + façade
push and pull interfaces

Domain represents an aspect of the context
allows custom implementations
ex. : `sys`, `net`, `geo`

Resource an element in the context or a category
ex. : `mouse`, `memory`, `storage`

Attribute describes a resource
ex. : `#buttons`, `#free`, `#removable`

- Logical data model
  - does not constrain implementation

WildCAT's data model (cont.)

- Simple, but :
  - internally, resources can be shared (DAG)
  - attributes $\longrightarrow$ resources to go beyond simple values
- Tree structures familiar to programmers
- Addressing : `domain://path/to/resource#attribute`
- Model is entirely dynamic
  - every change (potentially) generates an event
  - structural changes, value changes

Application Programmer Interface

- Every API accessed through `Context`
  - clients independent of domains implementations
- Very simple API
  - 2 classes, 2 interfaces
  - easy to learn and use
- Two complementary access modes

Usage

```
Context ctx = new Context();
ctx.addStandardDomain("/path/to/config.xml");
ctx.addDomain(new MyCustomDomain(args));
// Access context (see next slides)
```

Pull API : synchronous requests

- Structure discovery
  - navigate in the tree of resources and attributes
  - denotes elements using `Paths`, not direct pointers

```
Path[] drives = context.getChildren("sys://storage/disk");
Path[] hd = context.getAttributes(drives[0]);
context.exists(hd.getParent());
```

- Request current value of attribute

```
context.resolve("sys://storage/memory#free") ⇒ 54321
```

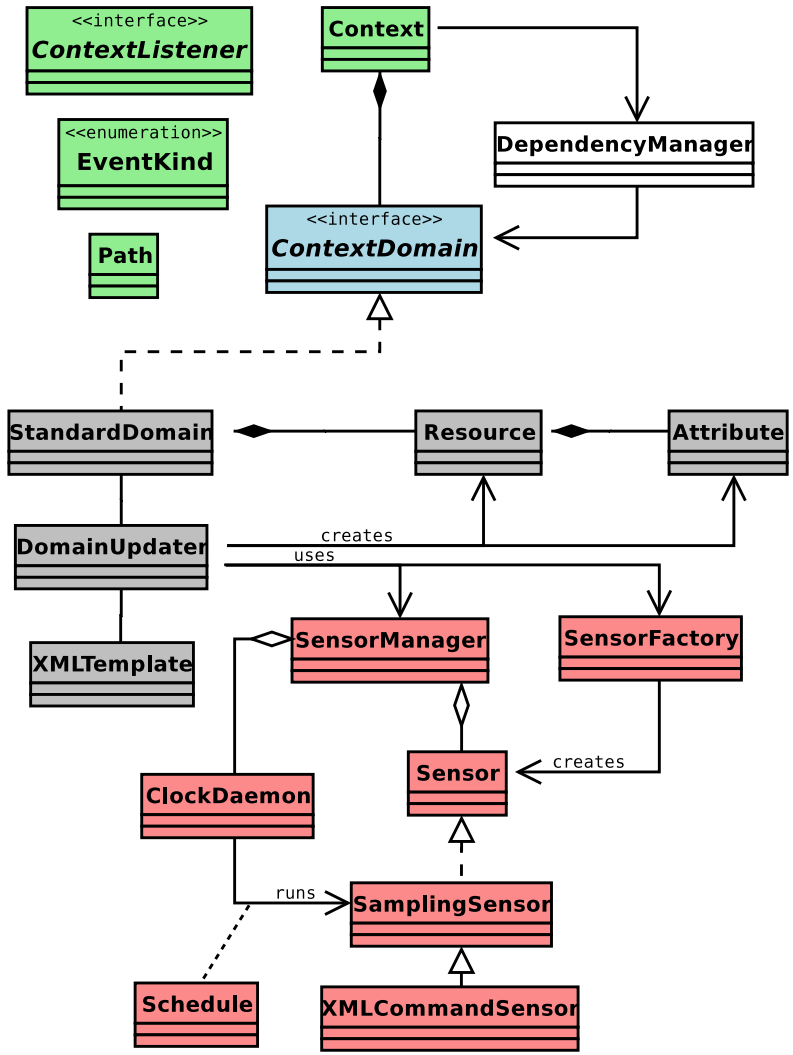
Push API : asynchronous notifications

- Classical publish/subscribe interface
  - `ContextListeners` notified asynchronously
- Basic events
  - `RESOURCE_ADDED/REMOVED`, `ATTRIBUTE_ADDED/REMOVED`
  - `ATTRIBUTE_CHANGED`
- `EXPRESSION_CHANGED`
  - simple expression language
  - expressions can reference paths from any domain
- `CONDITION_OCCURRED`
  - special case of `EXPRESSION_CHANGED`

```
expr = "sys://storage/memory#free < 5000";
context.registerListener(CONDITION_OCCURED, expr, listener);
```

Internal Architecture

- Software framework
- Transparent to application programmers
  - public API in green
- Supports different levels of extension



Data Acquisition Framework

- Raw data acquisition by sensors
  - low-level code
  - no abstraction / interpretation
- Active sensors
  - autonomous
  - send new samples asynchronously
- Passive sensors
  - simpler to program
  - can be made active using a scheduler
- Configuration protocol
  - sensors configured using arbitrary well-formed XML
- Extension scenario :
  - build new sensors & sensor libraries

Default Context Domain Implementation

- Direct implementation of the logical model
  - `Resource` and `Attribute` classes
- Configured using an XML templating language
  - defines initial structure (resources and attributes)
  - declares, configures and binds sensors to resources
    - sensor data becomes primitive attributes
  - (dynamic constructs : `if`, `foreach`)
  - Can define synthetic attributes (*abstraction*)
    - uses a simple expression langage
    - can reference elements from other domains
    - automaticaly recomputed on dependency changes
- Extension scenario :
  - reuse the data model with custom updating logic
  - create new templating/configuration language

The `ContextDomain` interface

- Similar to top-level `Context` interface
  - but limited to a specific domain
- Relatively difficult to implement
- But gives complete freedom on implem. trade-offs
- The default implementation is not a special case
- Extension scenario :
  - write new domains from scratch, with efficient implementation (e.g. `file ://`)
  - wrap "legacy" systems (LDAP, SNMP...)

Conclusion & Future Works

- Characteristics
  1. simple to use
  2. generic
  3. efficient (potentially)
  4. extensible
- Different roles
  - application programmer : needs a simple model and API
  - "configurator" : models the context of an application
  - implementor : extends the framework
- Current version limited
  1. it's just a kernel
  2. more interesting things should go in extensions
  3. ex : distribution, extended data model, support for history...