Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation

(Soutenance de thèse)

Pierre-Charles David

Équipe OBASCO - EMN / INRIA

Université de Nantes

2005-07-01

Plan de la présentation

Constat : variabilité des contextes d'exécution

Les applications s'exécutent dans des contextes de plus en plus variables.
Plusieurs types de variabilité :

Dans l'espace diversité des plate-formes (PDA et systèmes embarqués → grilles et clusters)

Dans le temps disponibilité des ressources logicielles et matérielles évolue en cours d'exécution (périphériques (dé-)branchés à chaud, nomadisme...)

Dans les utilisations applications utilisées dans des situations et par des utilisateurs différents (téléphone portable au bureau, en voiture, dans un lieu public...)

Vers des applications adaptatives

- Solution : construire des applications adaptatives
  - conscientes de leur contexte d'exécution (*context-aware*)
  - capables de s'adapter aux évolutions de ce dernier
  - autonomes
- Problème : prendre en compte tous ces éléments complexifie le développement
  - difficile de connaître au moment du développement les conditions d'exécution précises
  - même si on les connaît en partie, elles sont dynamiques
  - « pollution » de code métier par le code d'adaptation
- *Besoin d'applications adaptatives, mais pas de méthode pour les développer simplement*

Problématique de la thèse

Objectif Faciliter le développement d'applications adaptatives

Approche Deux principes :

1. Considérer l'adaptation comme un aspect afin de le modulariser
2. Proposer un langage spécifique pour exprimer cet aspect

Résultat SAFRAN, une extension du modèle de composants Fractal pour le développement de composants adaptatifs

- développement séparé de l'aspect d'adaptation
- langage dédié à cet aspect
- intégration dynamique avec les composants

Middlewares réflexifs et adaptables

- Intergiciels : médiateurs entre les applications et leur contexte
- Idée : adapter la couche intermédiare ⇒ adaptation de l'application

1. Introduction de la réflexion et de la reconfiguration dynamique
   - OpenCORBA, dynamicTAO, OpenORB...
   - Problème : adaptables, mais souvent pas adaptatifs
2. Approches QoS et « théorie du contrôle »
   - MCF, QualProbes, QuO, QuA...
   - Problème : ne fonctionne bien que pour certaines classes d'applications (flux multimédia par exemple)

- Problème de l'approche middleware :
  - adaptation liée à la sémantique de l'application ⇒ impossible à traiter entièrement dans les couches basses

Modèles de composants adaptables / adaptatifs

- Nouvelle approche (à la « *exokernel* ») :
  - modèles de composants minimalistes mais extensibles
  - $\Rightarrow$ middlewares à la demande
  - adaptation contrôlée par l'application

1. Modèles ad hoc, conçus spécifiquement pour bien supporter l'adaptation
   - Adaptive Components [Boinot], ACEEL [Cherfour & André]
   - Problème : imposent une structure peu naturelle
2. Modèles plus généraux
   - K-Components [Dowling] : adaptations définies statiquement
   - PLASMA [Layaïda & Hagimont] : spécifique au multimédia

Critères : modèle de composants **adaptatifs**, **simple** à utiliser, **dynamique**, **généraliste**

1 Motivation et problématique

2 Étude de l'existant

3 SAFRAN : Self-Adaptive FRActal compoNents
- Sensibilité au contexte avec WildCAT
- Reconfigurations dynamiques avec FScript
- Politiques d'adaptation SAFRAN
- Scénario exemple

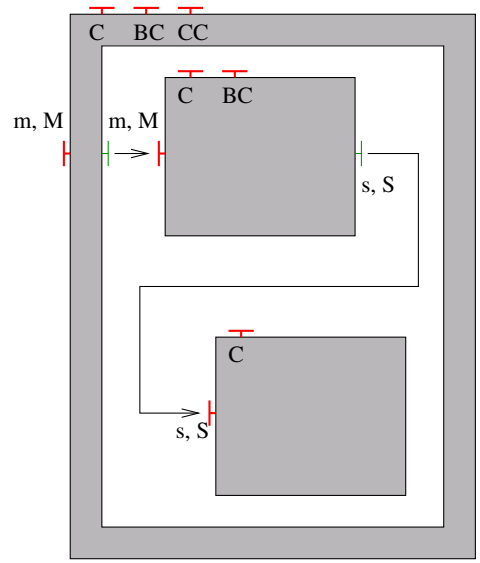
4 Conclusion et perspectives

SAFRAN : Self-Adaptive FRActal compoNents

- Extension du modèle de composants Fractal
- Développement de politiques d'adaptation séparées
- Association dynamique des politiques et des composants
- Politiques réactives programmées dans un langage dédié

Le modèle de composants Fractal

- Modèle de composants développé par FT R&D
  - indépendant du langage, mais surtout Java
- Pourquoi Fractal ?
  - noyau minimal mais extensible
  - supporte les reconfigurations dynamiques
  - réflexif ⇒ reconfigurations non-anticipées
- Composants Fractal :
  - *Contrôleur* régit les interactions avec l'extérieur
  - *Interfaces* de contrôle et de service
  - Composants *primitifs* ou *composites*
  - *Connexions* entre interfaces compatibles
- Permet la construction d'applications adaptables (par reconfiguration)
  - première étape pour obtenir des applications adaptatives



Séparation et composition de l'aspect d'adaptation

Aspects généralistes

- programme de base
- aspects séparés
- tisseur (*weaver*)

Aspect d'adaptation

- composants Fractal
- politiques d'adaptation
- contrôleur d'adaptation

- Contrôleur d'adaptation :
  - association (tissage) dynamique entre composants et politiques
  - synchronise l'exécution des adaptations avec le programme de base et les évolutions du contexte
- Avantage : découplage spatial et temporel code métier / adaptation
  - code métier simplifié et plus réutilisable
  - politiques d'adaptation développées « Just In Time »

Structure des politiques d'adaptation

Aspects généralistes

- points de jonction
- actions (*advices*)

Politiques d'adaptation

- événements
- reconfigurations de composants

- Politiques d'adaptation : règles réactives (ECA)
- Conforme à la nature du processus d'adaptation :
  - Observation/Détection $\longrightarrow$ Décision $\longrightarrow$ Action [ARCAD]
- Deux catégories d'événements :
  - endogènes : points de jonction classiques (EAOP)
  - exogènes : évolutions du contexte d'exécution
- Actions spécifiques : reconfiguration des composants

Interface de SAFRAN

- Composants Fractal étendus : `adaptation-controller`
  - tisseur spécialisé pour l'aspect d'adaptation

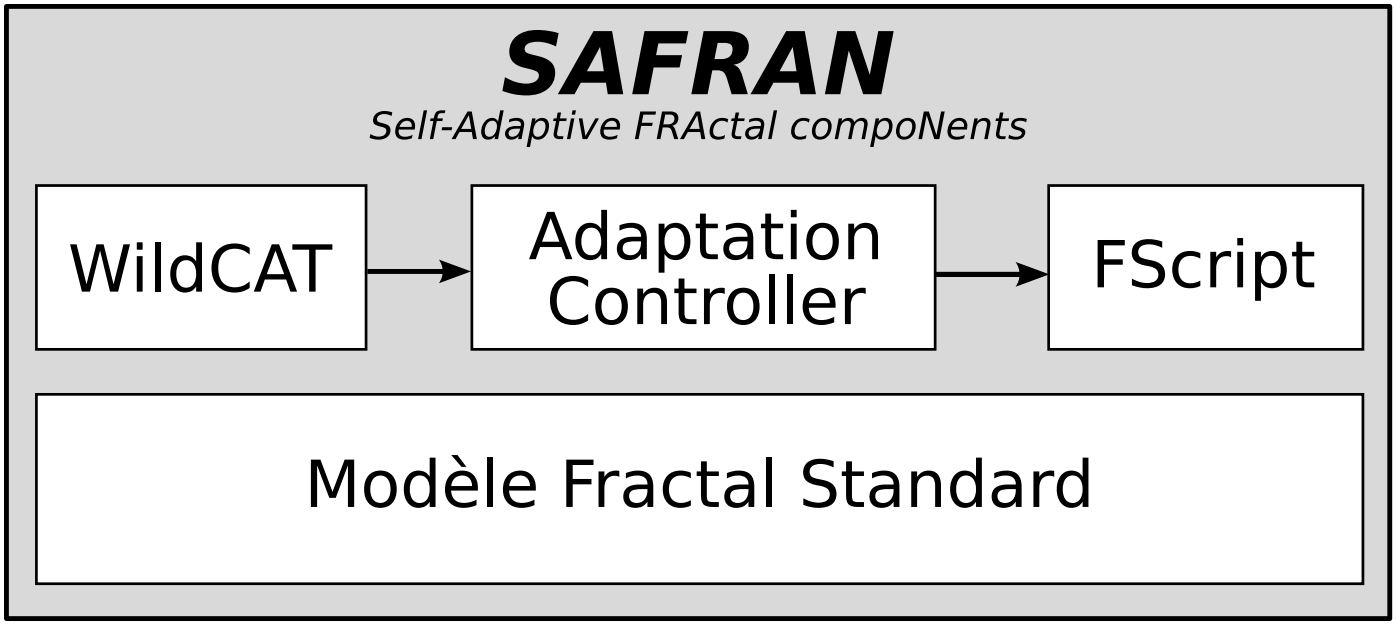
```
interface AdaptationController {
  void attachFcPolicy(AdaptationPolicy policy);
  void detachFcPolicy(AdaptationPolicy policy);
  any[] getFcPolicies();
}
```

- Langage dédié pour programmer les politiques
  - règles réactives type Événement–Condition–Action

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

Architecture de SAFRAN

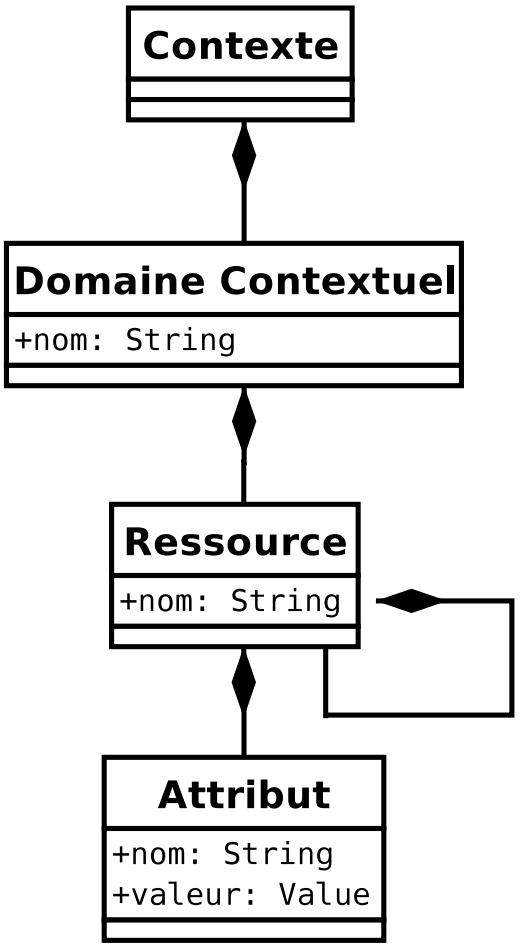
- Conception et implémentation modulaire
  - Observation et détection : WildCAT
  - Décision : règles réactives
  - Action : FScript
- WildCAT et FScript réutilisables en dehors de SAFRAN
- SAFRAN lui-même est l'intégration de ces modules
  - programmer des politiques d'adaptation
  - les intégrer dans des composants Fractal



WildCAT

- On doit connaître le contexte d'exécution et ses évolutions pour pouvoir s'y adapter
- Ce contexte habituellement implicite (par définition)
- WildCAT permet de le réifier et d'accéder au contexte
- Facilite la création d'applications sensibles au contexte
  - utilisable comme une « boîte noire »
  - framework, extensible
- Répond aux étapes d'observation et de détection du processus d'adaptation
  - produit les événements exogènes pour SAFRAN

Le modèle de données logique de WildCAT



Contexte point d'entrée principal
singleton + façade
interfaces push et pull

Domaine représente un aspect du contexte
implémentations spécifiques
ex. : `sys`, `net`, `geo`

Ressource un élément du contexte, ou une catégorie
ex. : `mouse`, `memory`, `storage`

Attribut décrit une ressource
ex. : `#buttons`, `#free`, `#removable`

- Addressage : `domaine://chemin/vers/ressource#attribut`
- Modèle entièrement dynamique

Deux modes d'accès au contexte

- Requêtes synchrones (*pull*)
  - accès à l'état instantané du contexte

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

  - découverte de la structure du contexte

```
Path[] drives = contexte.getChildren("sys://storage/disk")
```

- Abonnement et notifications asynchrones d'événements (*push*) :
  - apparition et disparition de ressources et attributs
  - modification de valeur des attributs
  - modification de valeur d'expressions et occurrence de conditions

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

FScript

- SAFRAN : adaptation par reconfiguration des composants
- Comment spécifier ces reconfigurations ?
  - APIs Fractal conçues pour être minimales ⇒ code verbeux
  - concepts Fractal pas intégrés dans le langage (Java)
  - Java pas assez dynamique, trop puissant
- Solution : FScript
  - langage « de script », impératif, interprété (dynamique)
  - syntaxe, puissance d'expression et mode d'exécution adaptés
  - accès à toutes les opérations supportées
  - garantit la consistance des reconfigurations

Notation FPath

- FPath : notation pour la navigation et la sélection d'éléments d'une architecture Fractal
  - Sélectionner le sous-composant de C qui fournit l'interface I.
  - Quels sont les composants configurables de mon application ?
- Syntaxe et modèle opérationnel inspirés de XPath [W3C]
  - mais n'utilise pas XML
- Composants Fractal vus sous forme de graphes orientés
  - nœuds : composants, interfaces et attributs de configuration
  - arcs avec labels : indiquent la relation entre les nœuds

Exemple

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

Reconfigurations FScript

- Permet la définition d'actions de reconfigurations
- Actions primitives : APIs Fractal
  - `add()`, `remove()`, `bind()`, `unbind()`
  - `new()`, `start()`, `stop()`, `set-value()`...
  - facilement extensible (comme Fractal)
- Structures de contrôle limitées volontairement
  - séquence, `if/then/else`, `foreach`, `return`
  - définitions récursives interdites
- Conception et implémentation garantissent la consistance des reconfigurations

Exemple de reconfiguration FScript

Connexion automatique des interfaces requises par un composant

```
action auto-bind(comp) = {
  // Selectionne les interfaces a connecter
  clients := $comp/interface : :*[required(.)][not(bound(.))] ;
  foreach itf in $clients do {
    // Recherche des interfaces complementaires candidates
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)] ;
    if (not(empty?($candidates))) then
      // Connecte l'une quelconque des candidates
      bind($itf, one-of($candidates)) ;
  }
}
```

Exemple de reconfiguration FScript

Connexion automatique des interfaces requises par un composant

```
action auto-bind(comp) = {
  // Selectionne les interfaces a connecter
  clients := $comp/interface : :*[required(.)][not(bound(.))] ;
  foreach itf in $clients do {
    // Recherche des interfaces complementaires candidates
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)] ;
    if (not(empty?($candidates))) then
      // Connecte l'une quelconque des candidates
      bind($itf, one-of($candidates)) ;
  }
}
```

Exemple de reconfiguration FScript

Connexion automatique des interfaces requises par un composant

```
action auto-bind(comp) = {
  // Selectionne les interfaces a connecter
  clients := $comp/interface : :*[required(.)][not(bound(.))] ;
  foreach itf in $clients do {
    // Recherche des interfaces complementaires candidates
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)] ;
    if (not(empty?($candidates))) then
      // Connecte l'une quelconque des candidates
      bind($itf, one-of($candidates)) ;
  }
}
```

Exemple de reconfiguration FScript

Connexion automatique des interfaces requises par un composant

```
action auto-bind(comp) = {
  // Selectionne les interfaces a connecter
  clients := $comp/interface : :*[required(.)][not(bound(.))] ;
  foreach itf in $clients do {
    // Recherche des interfaces complementaires candidates
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)] ;
    if (not(empty?($candidates))) then
      // Connecte l'une quelconque des candidates
      bind($itf, one-of($candidates)) ;
  }
}
```

Exemple de reconfiguration FScript

Connexion automatique des interfaces requises par un composant

```
action auto-bind(comp) = {
  // Selectionne les interfaces a connecter
  clients := $comp/interface : :*[required(.)][not(bound(.))] ;
  foreach itf in $clients do {
    // Recherche des interfaces complementaires candidates
    candidates := $comp/sibling : :*/interface : :*[compatible?($itf, .)] ;
    if (not(empty?($candidates))) then
      // Connecte l'une quelconque des candidates
      bind($itf, one-of($candidates)) ;
  }
}
```

Garanties offertes par FScript

- Approche transactionelle
  - empêcher une adaptation de « casser » l'application

1. *Terminaison* : garantie par la conception du langage
2. *Atomicité* : garantie par l'implémentation
   - actuellement : approche « try-repair » (undo)
3. *Consistance* : garantie par l'implémentation
   - test à la fin d'une reconfiguration → rollback en cas de problème
   - Julia, l'implémentation de Fractal, offre aussi certaines garanties
4. *Isolation* : garantie par l'implémentation
   - empêche deux reconfigurations de se chevaucher
   - actuellement : verrou global (mutex)
   - futur : synchronisation plus fine (nécessite de limiter la portée des actions FScript)

Politiques d'adaptation SAFRAN

- Intégration des contributions précédentes
- Permet d'exprimer des règles d'adaptation réactives :
  - Événement → Condition → Action
- Politique = liste de règles

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

- Associées à un composant cible ($target)

Descripteurs d'événements

- Forme générale : `type-evenement(arguments)`
- Événements endogènes :
  - exécution : `message-{received,returned,failed}`
  - reconfigurations : `subcomponent-added`, `binding-destroyed`...
  - exemple : `component-started($target/sibling::*)`
- Événéments exogènes (WildCAT) :
  - `changed(expression)`, `realized(condition)`
  - `appears(expression)`, `disappears(expression)`
  - exemple : `appears(sys://devices/input/*)`
- Capture des occurrences

```
when room:changed(geo://location/logical#room)
if ($room.new-value = "G. Besse")
do { set-value($target/attribute::delay, 0); }
```

Exécution des politiques

- Cycle de vie global :
  - définition → attachement → exécution → détachement
- Attachement
  - abonnement auprès de WildCAT et des composants
- Phase d'exécution : attente de réception d'un événement
  1. sélection des règles concernées
  2. calcul de la réaction globale
  3. exécution de la transaction de reconfiguration
- Règles de composition :
  - entre règles d'une même politique
  - entre les politiques sur un composant donné
  - entre les composants adaptatifs de l'application

Applications réalisées

- Lecteur de courrier électronique
  - application graphique, interactive
  - Scénario 1 : modes de notification adapté aux circonstances
  - Scénario 2 : gestion du mode déconnecté
- Serveur web (Comanche)
  - application non-interactive
  - **Scénario 1 : composant cache adaptatif**
  - Scénario 2 : adaptation du nombre de threads

Cache adaptatif

- Ajout d'un cache pour améliorer les performances d'un serveur web
- Problème : quelle taille lui allouer ?
  - trop petite $\longrightarrow$ inutile
  - trop grande $\longrightarrow$ inefficace (trashing)
- Taille idéale dépend de la mémoire libre
  - variable dynamiquement, imprévisible
- Politique d'adaptation : assujettir la taille du cache à la quantité de mémoire disponible

Cache adaptatif : politique d'adaptation

```
policy adaptive-cache = {
    rule emergency;
    rule adjust-cache-size;
}
```

- Règles définies en dehors de la politique
- `emergency` :
  - **lorsqu'**il reste moins de 10Mo de mémoire libre
  - **alors** réduire la mémoire allouée au cache pour tenter de préserver 10Mo, et le désactiver complètement si ce n'est pas possible
- `adjust-cache-size` (régime normal)
  - **lorsque** la quantité de mémoire libre varie
  - **si** au moins 10Mo sont disponibles
  - **alors** allouer jusqu'à 80% de la mémoire disponible au cache, mais toujours garder 10Mo de libre

Détail de la règle adjust-cache-size

```
rule adjust-cache-size = {
  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) ;
    }
  }
}
```

- Action enable-cache définie séparément
  - création et intégration du cache dans l'architecture
- Taille du cache ajustée automatiquement
- Politique d'adaptation ne pollue pas le code du cache
- Politique modifiable dynamiquement

Conclusion

Contributions :

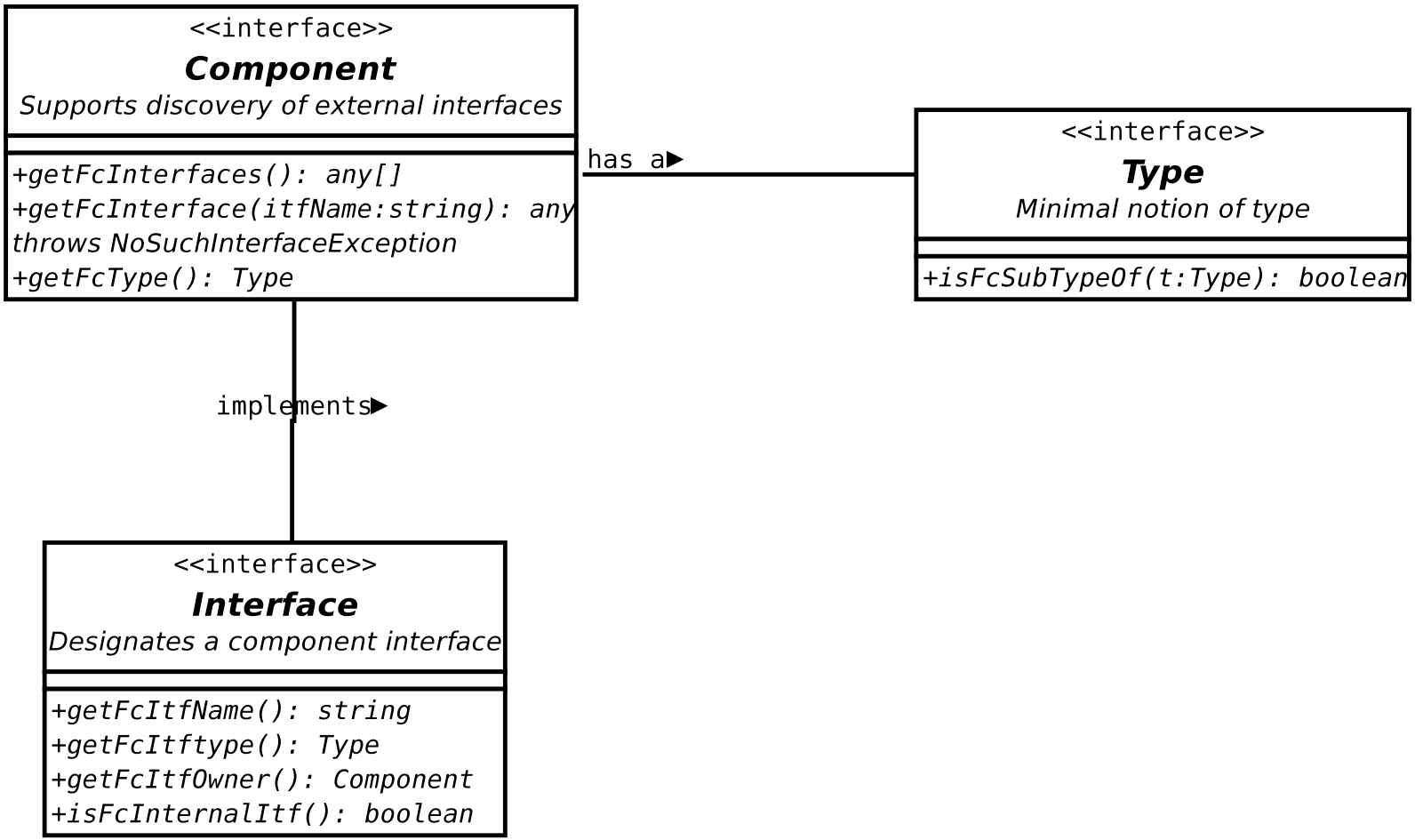
- une approche par aspect pour le développement d'applications adaptatives
  - découplage spatial et temporel
- SAFRAN : une mise en œuvre de cette approche dans le cadre de Fractal
  - langage dédié pour les politiques d'adaptation
- WildCAT : un système pour le développement d'applications sensibles au contexte (*context-aware*)
  - réifie le contexte d'exécution
  - deux modes d'accès complémentaires et simples d'utilisation
- FPath & FScript : un langage pour programmer des reconfigurations consistantes d'applications Fractal
  - syntaxe et sémantique adaptées au problème
  - garanties sur les reconfigurations

Perspectives

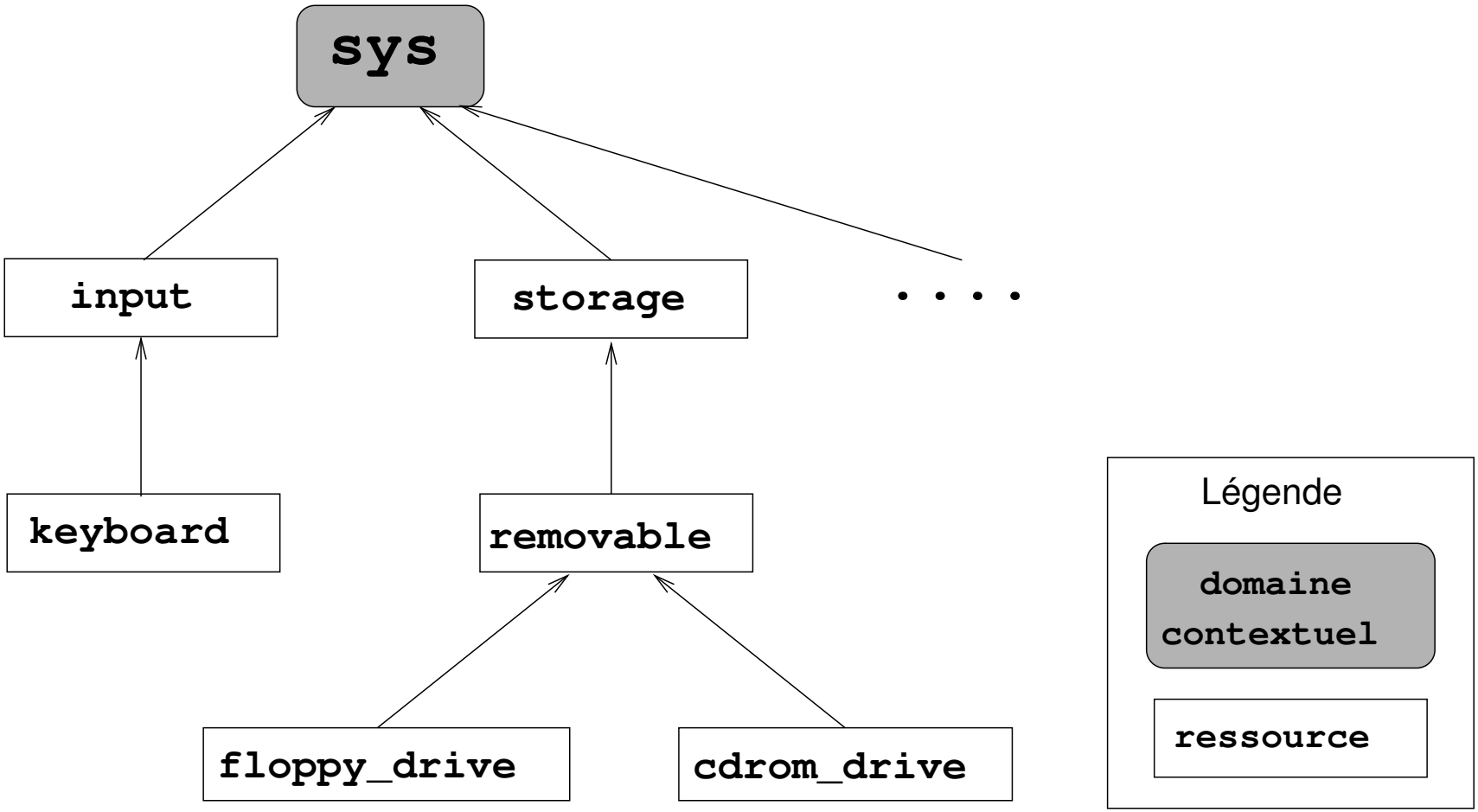
Pas de remise en cause de l'architecture de SAFRAN, mais amélioration des différents composants.

- Analyses statiques
  - système de type pour FScript
  - compatibilité entre politiques et composants
- Autres opérations de reconfiguration (Fractal)
  - spécialisation de composants [Bobeff et Noyé]
  - sérialisation $\Rightarrow$ passivation, mise-à-jour (transfert d'état), migration...
- Distribution : plusieurs étapes
  1. distribution des informations contextuelles (WildCAT)
  2. reconfigurations distribuées : migration de composants, gestion de connexions distantes (RMI...)
  3. distribution des politiques elles-mêmes : adaptation coordonnée, raisonnement distribué et coopératif...

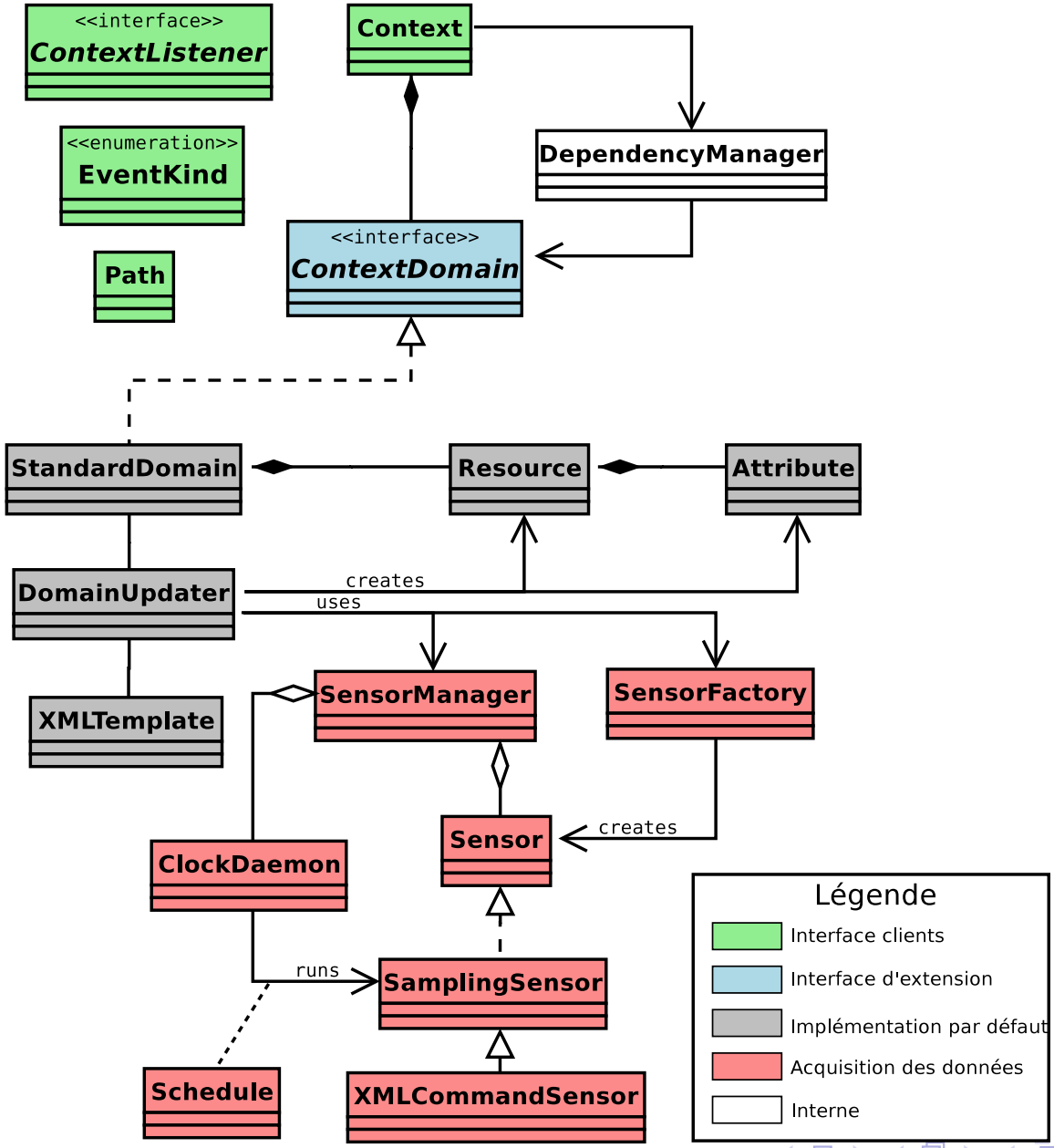
Noyau Fractal



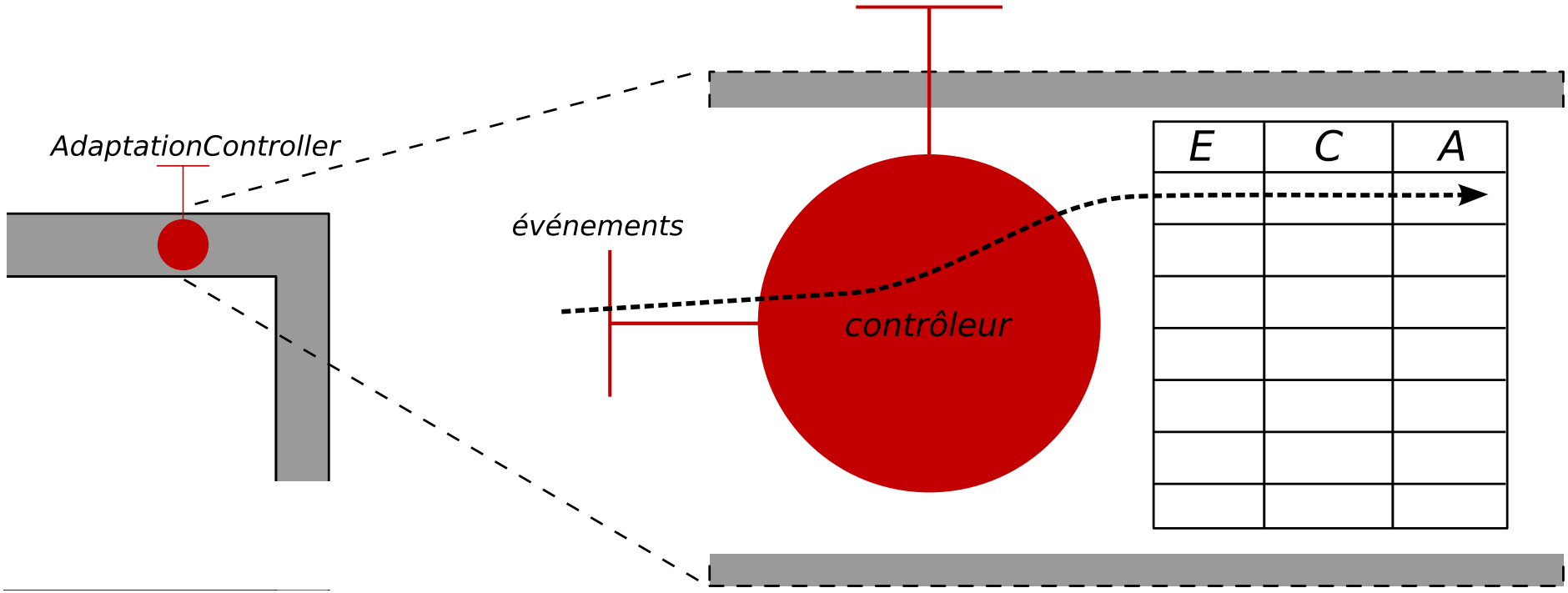
Exemple de domaine contextuel WildCAT



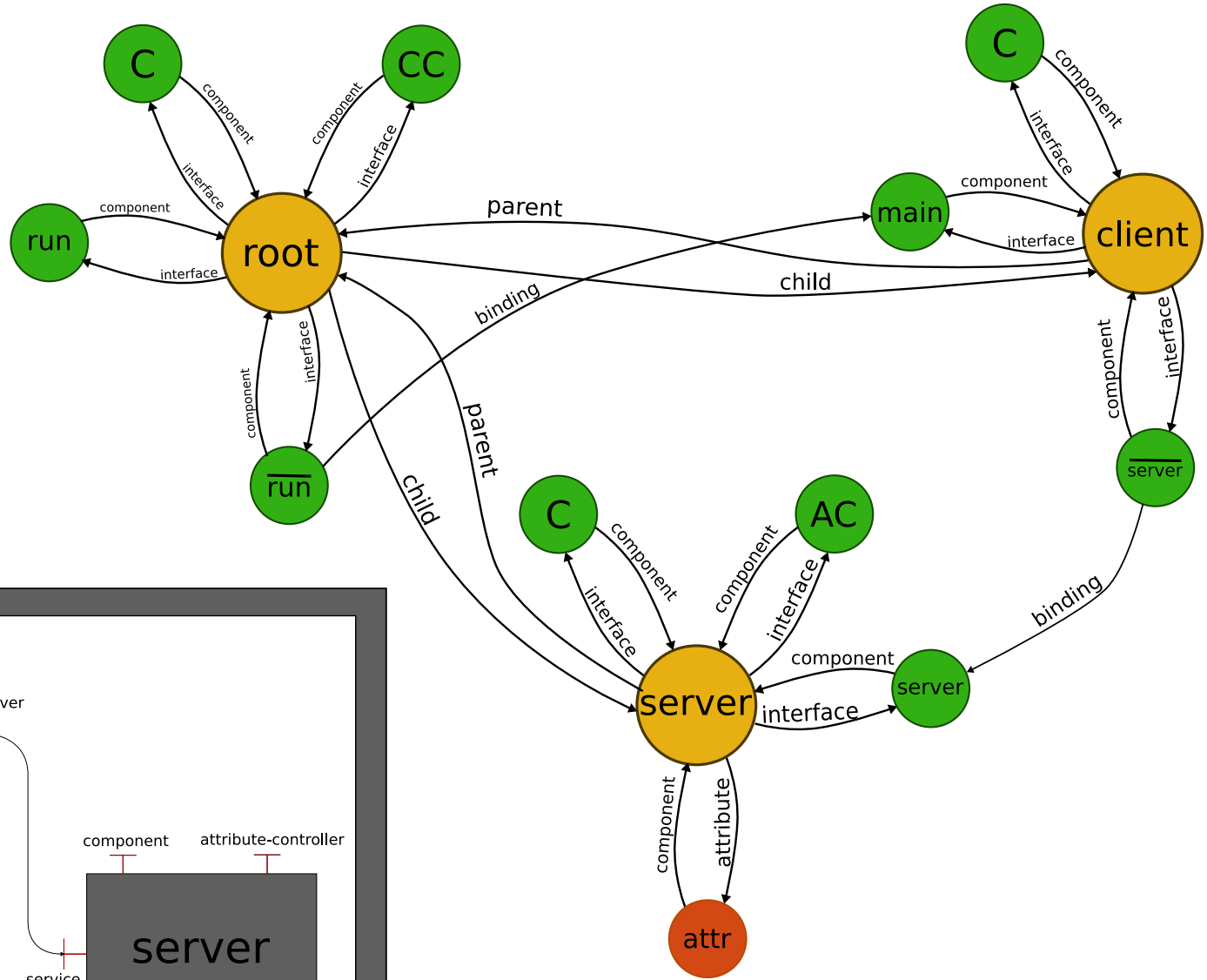
Architecture de WildCAT

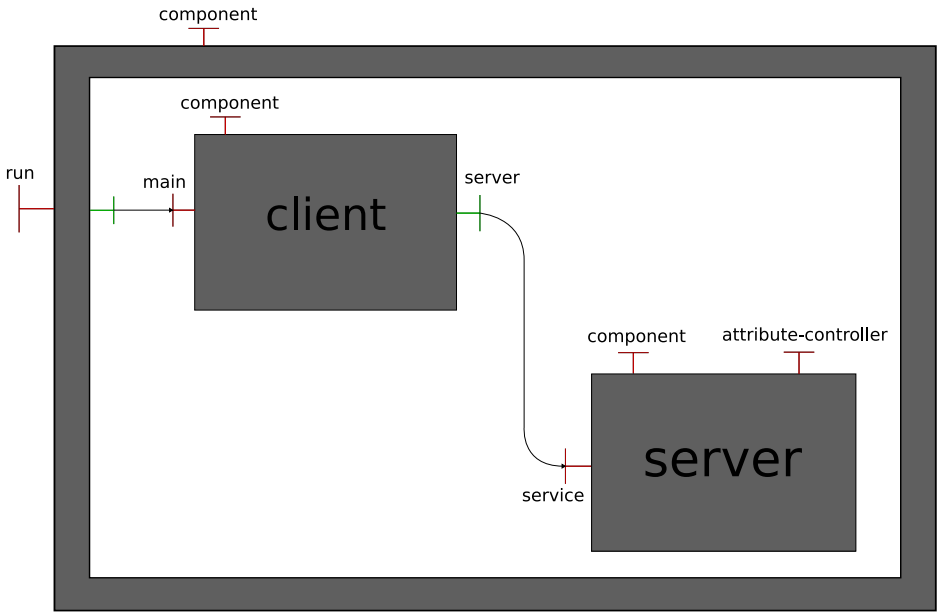


Implémentation du contrôleur d'adaptation

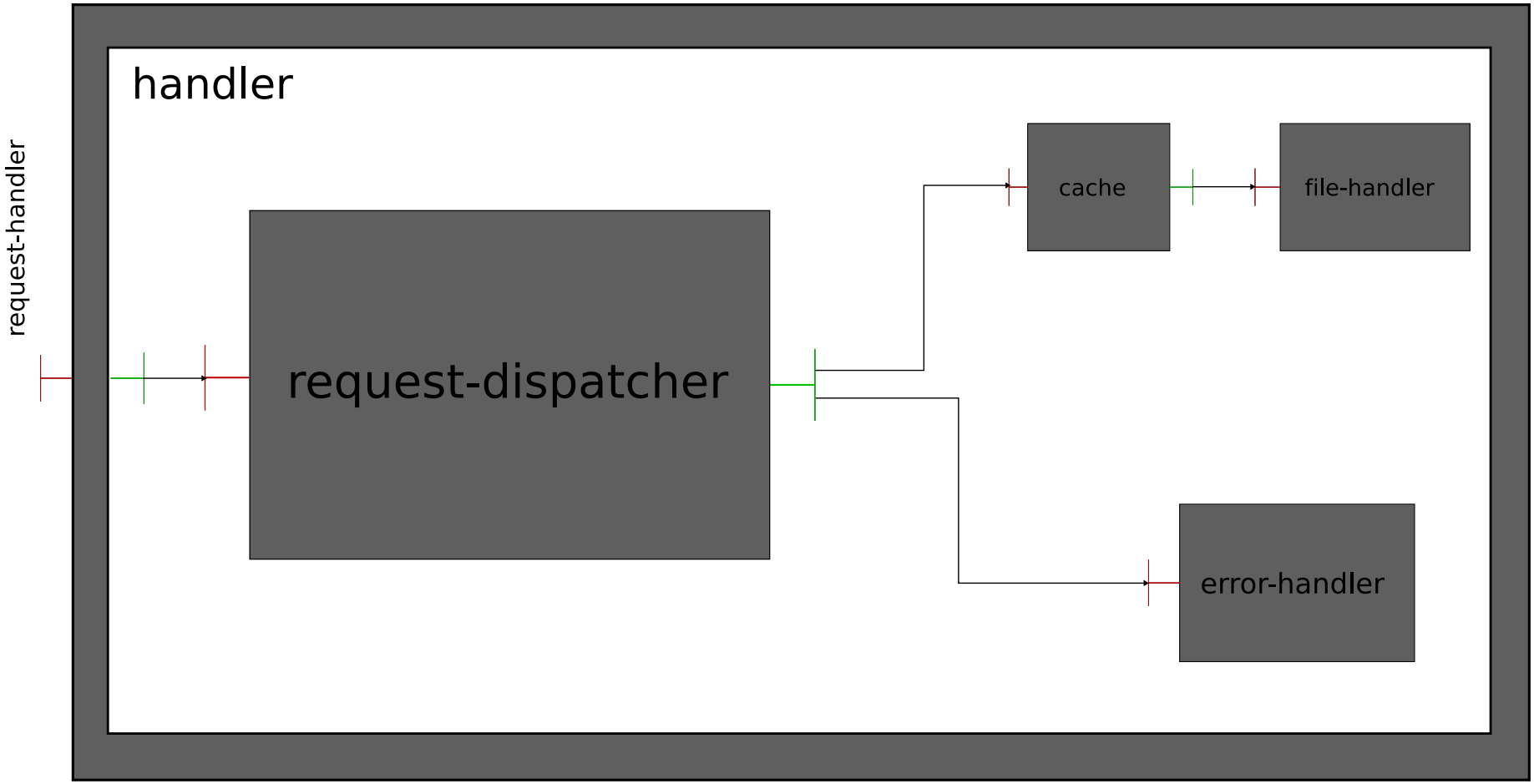


Modélisation de composants Fractal sous forme de graphe





Introduction d'un cache dans Comanche



Code source de l'action enable-cache

```
action enable-cache(handler) = {
  dispatcher := $handler/child::request-dispatcher;
  if (name($dispatcher/#handler/component::*) != 'cache') then {
    unbind($dispatcher/interface::handler);
    file-handler := $handler/child::file-handler;
    cache := $handler/child::cache;
    bind($dispatcher/interface::handler, $cache/interface::request-handler)
    bind($cache/interface::handler, $file-handler/interface::request-handle
  }
}
```

Processus de développement des applications avec SAFRAN