Une approche par aspects pour le développement de composants Fractal adaptatifs

Pierre-Charles David [1] Thomas Ledoux

France Télécom, Recherche & Développement
Groupe OBASCO - EMN / INRIA, LINA

JFDLPA, 2005-09-15

[1] Ce travail a été effectué dans le cadre de la thèse de l'auteur au sein du groupe OBASCO.

Plan de la présentation

Constat : variabilité des contextes d'exécution

- Les applications s'exécutent dans des contextes de plus en plus variables :
  - diversité des plate-formes
  - dynamicité des ressources logicielles et matérielles
- Objectif : construire des applications adaptatives
  - conscientes de leur contexte d'exécution et de ses évolutions
  - capables de s'y adapter de façon autonome
- 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

Problématique

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 (ASL)
- intégration dynamique avec les composants

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*)

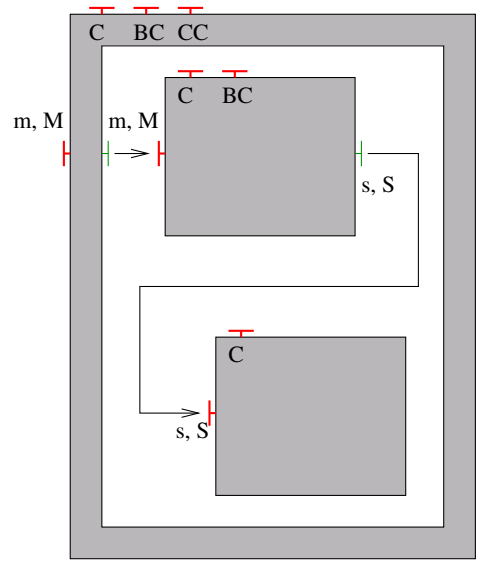
Politiques d'adaptation

- 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

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 $\Rightarrow$ 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



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 aspects/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> };
  ...
}
```

Descripteurs d'événements (E–C–A)

Correspondent aux points de jonction de l'aspect d'adaptation...

- avec l'application ⇒ événements endogènes :
  - exécution : `message-{received,returned,failed}`
  - reconfigurations : `subcomponent-added`, `binding-destroyed`...
  - exemple : `component-started($target/sibling::*)`
- avec le contexte ⇒ événéments exogènes (WildCAT) :
  - `changed(geo://location/logical#room)`
  - `realized(user://activity#working)`
  - `appears(sys://devices/input/*)`
  - `disappears(net://remote-services/example.com#http)`

Actions/advices : FScript (E–C–A)

- SAFRAN : adaptation par reconfiguration des composants
  - correspondent aux advices d'AspectJ
- FScript : un DSL pour programmer des reconfigurations Fractal
  - langage impératif, interprété (dynamique)
  - syntaxe, puissance d'expression et mode d'exécution adaptés
  - garanties : terminaison, atomicité, consistance, isolation
- Actions primitives : APIs Fractal
  - `add()`, `remove()`, `bind()`, `unbind()`
  - `new()`, `start()`, `stop()`, `set-value()`... (extensible)
- Navigation et sélection avec FPath
  - notation inspirée d'XPath, mais adaptée à Fractal
  - ex : `$comp/interface::*[required(.)][not(bound(.))]`
- Structures de contrôle limitées volontairement
  - séquence, `if/then/else`, `foreach`, `return`
  - définitions récursives interdites

Politiques d'adaptation SAFRAN

- Intégration des contributions précédentes
- Règles réactives de type Événement → Condition → Action :
  - Événement ≡ Coupe (*pointcut*)
  - Condition ≡ test sur l'événement ou l'état du système
  - Action ≡ *Advice* (corps de l'aspect)

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

- Associées (tissées) à un composant cible (`$target`)
- 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

Exemple : un 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 {
    when mem:changed(sys://storage/memory#free)
    if (sys://storage/memory#free >= 10*1024)
    do {
      cache := enable-cache($target);
      size := compute-optimal-size($cache, $mem);
      set-value($cache/@maximumSize, $size);
    }
  }
}
```

- Politique simplifiée, une seule règle :
  - **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
- Aspect/politique d'adaptation ne pollue pas le code du cache
- Aspect/politique tissable dynamiquement

Conclusion

Une approche par aspect pour le développement de composants et d'applications adaptatives.

- Découplage spatial et temporel de l'aspect d'adaptation
  - évite de polluer le code métier
  - permet de faire évoluer les politiques d'adaptation à l'exécution
- SAFRAN structuré comme un système à aspects
  - Programme de base : composants Fractal
  - Aspects : politiques d'adaptation
    - Coupe : événements (endogènes et exogènes)
    - Action : reconfiguration FScript
  - Tisseur : contrôleur d'adaptation
- Approche dédiée à un domaine / une classe d'aspects
  - compromis « langage généraliste » *vs* DSL