Une infrastructure pour middleware adaptable

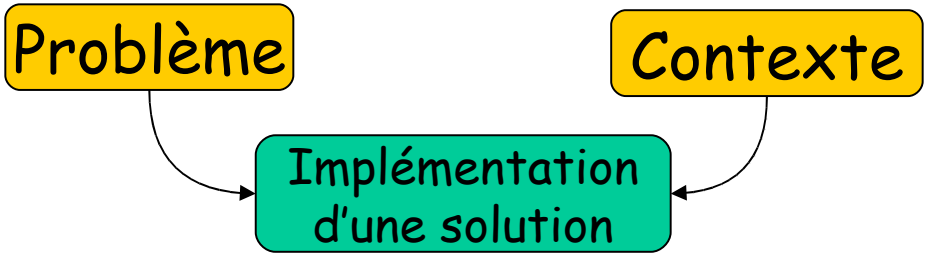
Pierre-Charles David (**pcdavid@emn.fr**)
DEA Informatique, École des Mines de Nantes
Encadreur: Thomas Ledoux (**ledoux@emn.fr**)

Plan

- Problématique
- Analyse de l'adaptabilité
- Solution proposée
- Conclusion
- Questions ouvertes & travaux futurs

Problématique

- Système informatique:



- Définition du problème: tend à se stabiliser
- Contexte d'exécution: évolution constante
  - nouvelles machines et technologies (PDAs…)
  - ressources disponibles variables (bande passante…)
- La solution doit rester valide malgré les évolutions de l'environnement: elle doit **s'adapter**

Solution partielle

- Concept de *middleware* (intergiciel):
  - couche logicielle intermédiaire
  - isole le programme de l'environnement

- Solution *partielle*: on a seulement déplacé le problème!
- *Middleware adaptable* bénéficierait à toutes les applications à moindre coût

Objectifs

- Définir une infrastructure générique
  - au niveau du middleware,
  - permettant d'adapter les applications,
  - indépendante d'une application particulière,
  - et donc configurable.

$\Rightarrow$ *une infrastructure pour middleware adaptable*

Analyse de l'adaptabilité

- Eléments à définir:
  - Sujet de l'adaptation (Quoi?)
  - Acteur de l'adaptation (Qui?)
  - Moments de l'adaptation (Quand?)
  - Mécanismes de l'adaptation (Comment?)

Sujet de l'adaptation

- Qu'est-ce qui est adapté?
- Le *code fonctionnel* (objets métier)
  - choix d'une implémentation parmi plusieurs
  - ex: Adaptive Components, MolèNE
- Le *code non fonctionnel* (services techniques)
  - association des services aux objets métier
  - ex: JavaPod, dynamicTAO
- Code non fonctionnel générique (réutilisable)
  - solution plus générale

Acteur de l'adaptation

- Qui réalise l'adaptation?
- Le *programmeur de l'application*
  - au mieux, le middleware se contente de le notifier des évolutions de l'environnement
  - *ex*: R-RIO, LEAD++
- Le *middleware* lui-même
  - solution plus automatisée, mais pas totalement
  - besoin d'informations de configuration (politiques)
  - *ex*: Olan (version dynamique)
- Systèmes *adaptables* vs *adaptatifs*

Moments de l'adaptation

- Conception et codage
  - conception modulaire
  - séparation code *fonctionnel* / *non-fonctionnel*
- Déploiement
  - *configuration* par rapport à un environnement donné (type de machine par exemple)
- Exécution
  - *observation* des conditions d'exécution
  - *reconfiguration dynamique* du système

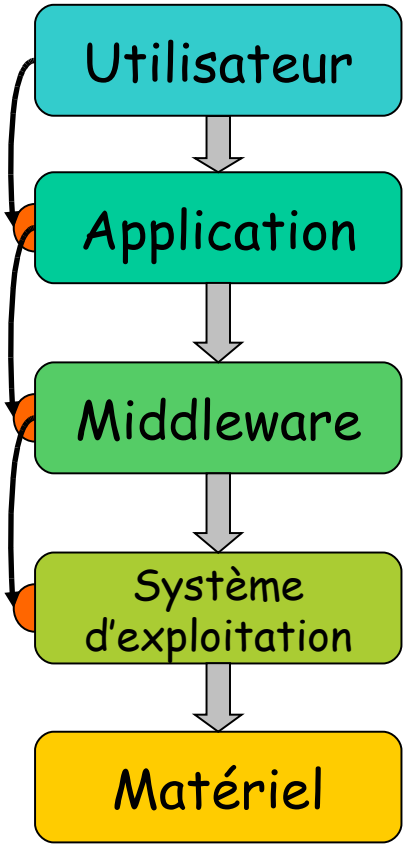
Mécanismes d'adaptation
Pour une approche ouverte

- Les couches hautes doivent pouvoir spécifier leurs "besoins" aux couches basses

$\forall \Rightarrow$ *Open Implementation*

  - approche "boîte grise"

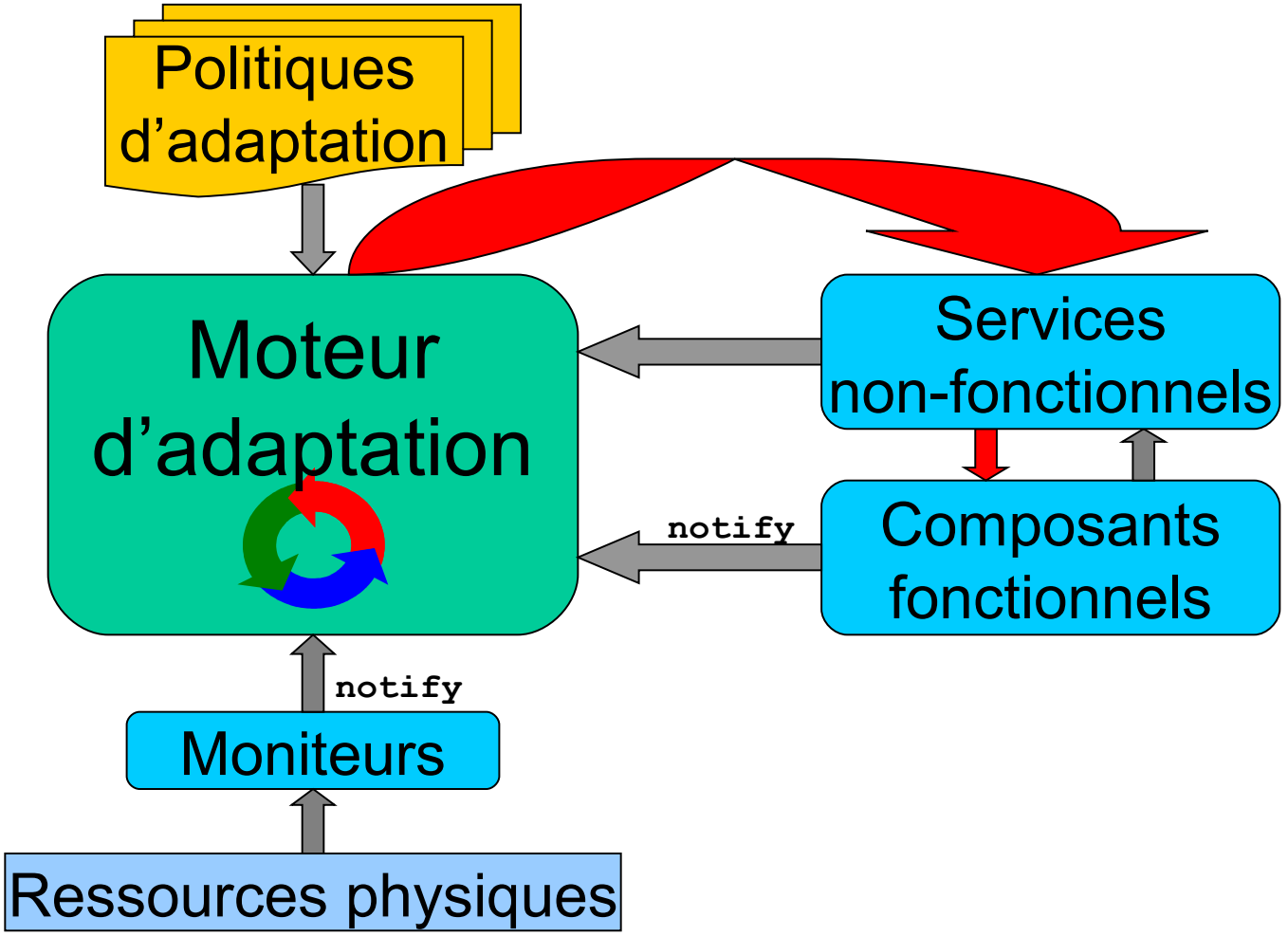
- Ajout d'une interface de configuration générique



Fonctionnalités requises pour une infrastructure adaptable

- Modularité et flexibilité du code applicatif
  - approches objets, composants, AOP
- Observation et contrôle du système
  - réflexion (introspection + intercession)
- Observation de l'environnement
  - sondes logicielles, moniteurs
- Adaptation guidée par la sémantique de l'application et par l'utilisateur
  - politiques d'adaptation

Architecture générale



Rappels sur la réflexion

- Système réflexif: auto-référentiel
- Deux aspects duals (connexion causale):
  - introspection (auto-représentation)
  - intercession (auto-modification)
- Deux niveaux d'abstraction: *base* et *méta*
- Langages à objets:
  - *objets de base* et *métaobjets*
  - *Meta Object Protocols (MOPs)*

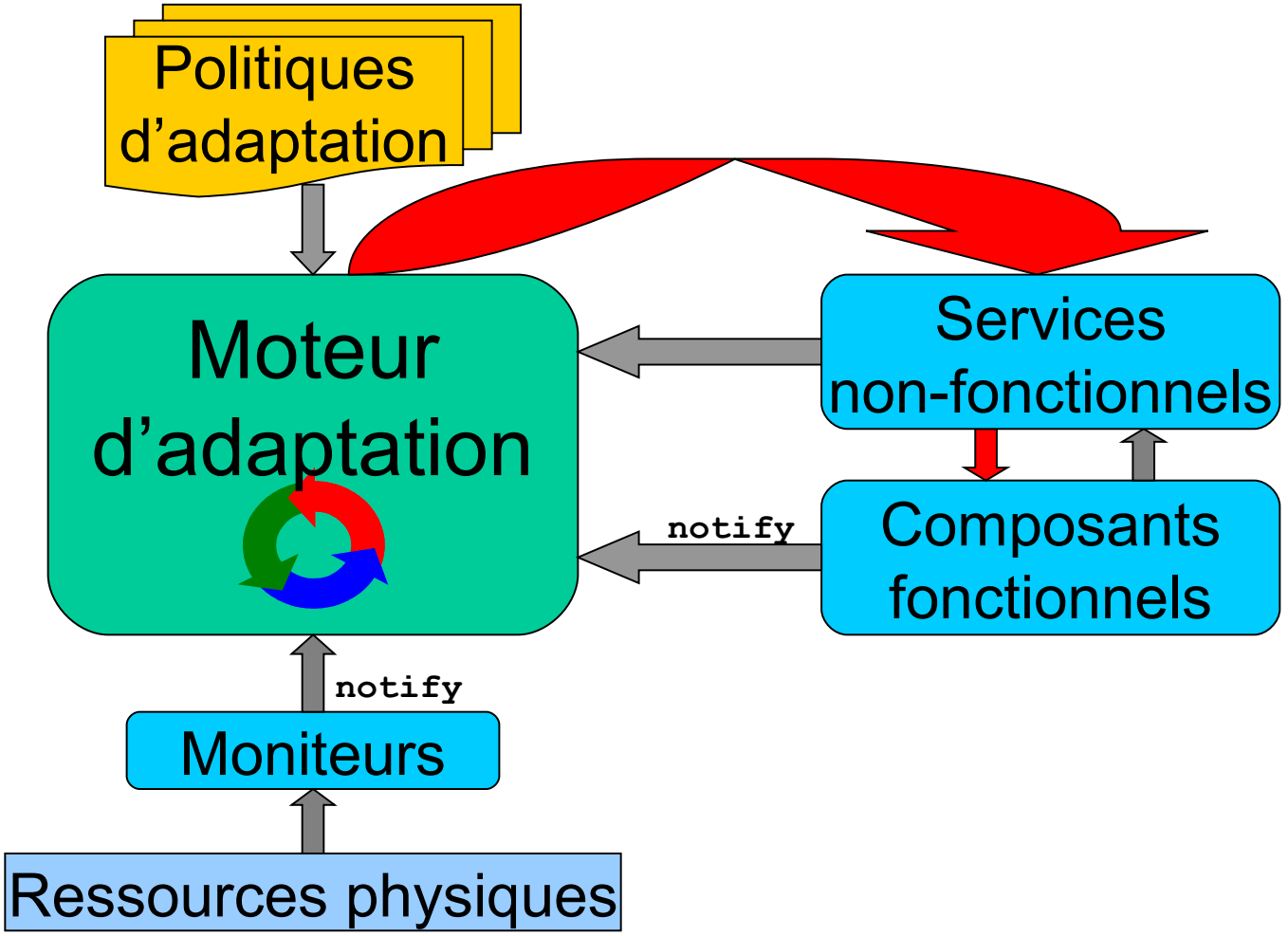
Composants fonctionnels

- Objets Java réflexifs (MOP RAM)
- Un métaobjet principal: **`Container`**
  - représente le composant du point de vue du reste du système
- Attachement d'*attributs* au **`Container`**
  - 2 interfaces: requêtes et notification
  - peuvent refléter certains champs de l'objet
- Ajout/retrait dynamique d'autres métaobjets

Services non-fonctionnels

- Fournissent les aspects non-fonctionnels
  - ex: **`misc.trace`**, **`communication.rpc...`**
- Une ou plusieurs implémentations concrètes
  - ex:**`com.sun.RMIProvider`**, **`org.omg.IIOPProvider`**
- Indépendants de l'application (réutilisables)
  - génériques ⇒programmés au niveau méta
- Adaptation: reconfiguration dynamique des associations entre services et composants
  - attachement, détachement, reconfiguration

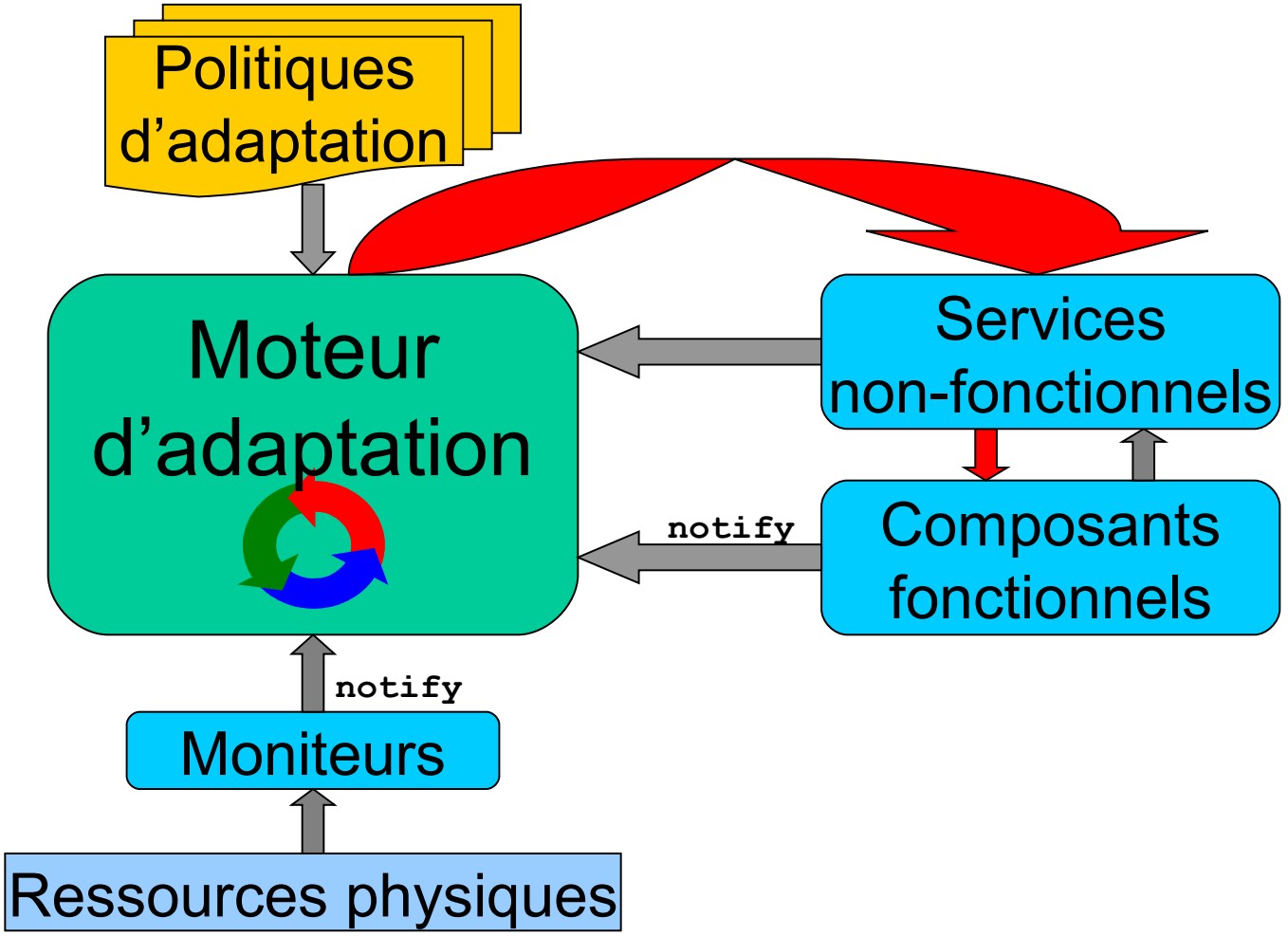
Architecture générale



Observation des ressources

- *But*: rendre disponible au reste du système l'état des ressources physiques
- Organisation hiérarchique de `MonitoredResource`
- Chaque `MonitoredResource` exporte des attributs dynamiques (générés par des sondes)
- Système de notification:
  - changement de valeur d'un attribut
  - apparition / disparition d'une branche (ressource)
- Possibilité d'avoir des attributs synthétiques
  - ex: charge moyenne durant les 5 dernières minutes

Architecture générale



Politiques d'adaptation

- Guident le moteur d'adaptation
- Lui indiquent comment réagir…
  - attacher/détacher des services à certains composants
- … dans quelles circonstances
  - relatives aux conditions d'exécution
  - relatives aux composants eux-même
- 2 types de politiques (2 niveaux d'abstraction):
  - *système* et *applicatives*

Politiques système

- Définies indépendamment d'une application
  - fournies avec le système, réutilisables
  - enrichies par de nouveaux services
- Ensemble de règles: *condition* ⇨ *actions*
- Condition:
  - relative à l'environnement d'exécution
  - ex: 'la bande passante passe en dessous de 5kb/s'
- Action:
  - activation d'un ou de plusieurs service(s), avec des paramètres

Exemple de politique système

```
<system-policy name='distribution'>
  <rule>
    <when><true/></when>
    <ensure>
      <attached service='communication.rpc' role='server'/>
    </ensure>
  </rule>
  <rule>
    <when>
      <less-than>
        <attribute-value name='/system/network.free-bandwidth'/>
        <number value='500'/>
      </less-than>
    </when>
    <ensure>
      <attached service='cache'/>
    </ensure>
  </rule>
</system-policy>
```

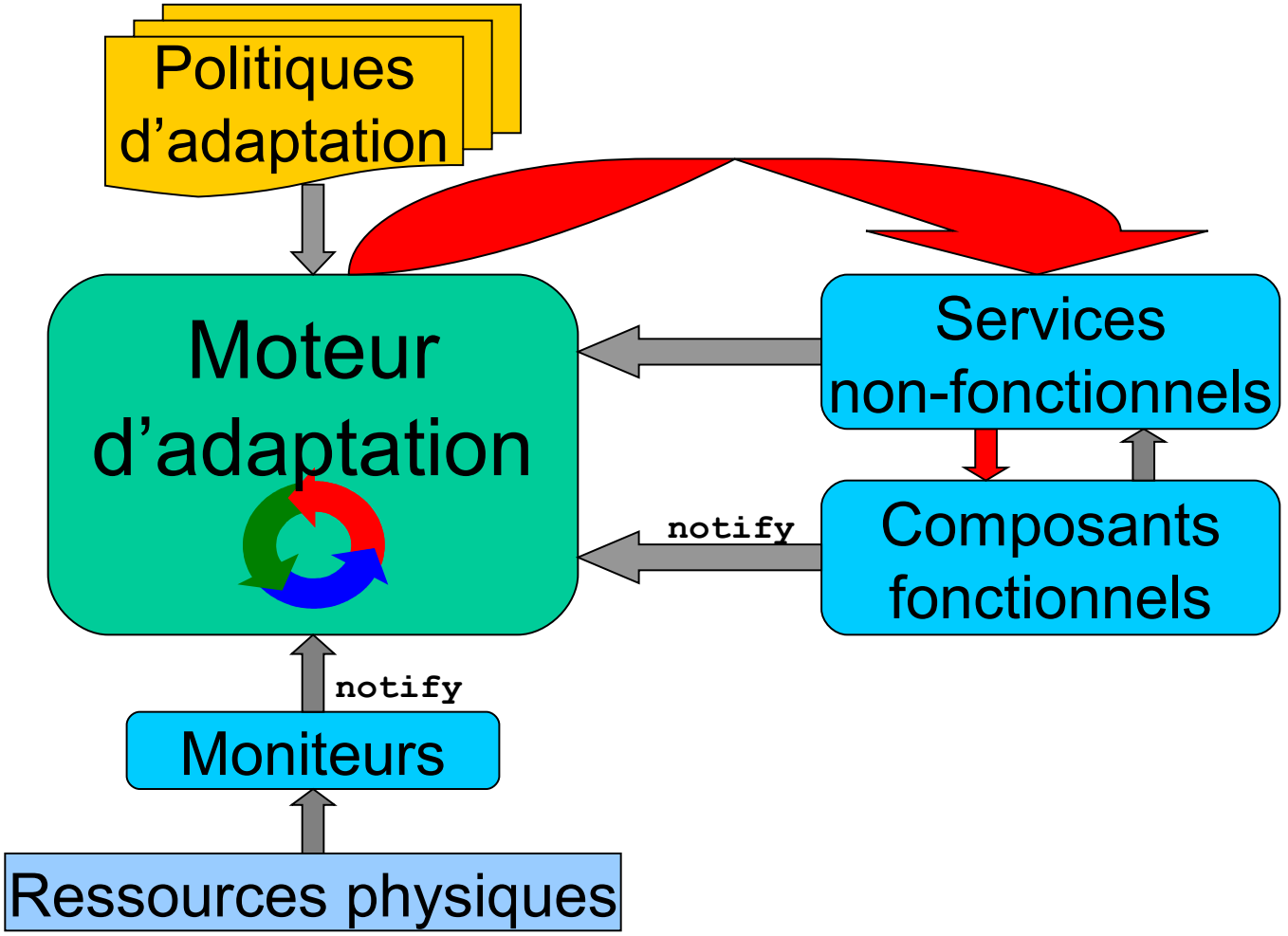
Politiques applicatives

- Ecrites par le programmeur d'application
- Définissent des groupes de composants
  - composants liés sémantiquement (attributs)
  - doivent être traités de la même façon
  - contenu mis à jour dynamiquement
- Associent ces groupes à:
  - des politiques système
  - ou directement des services

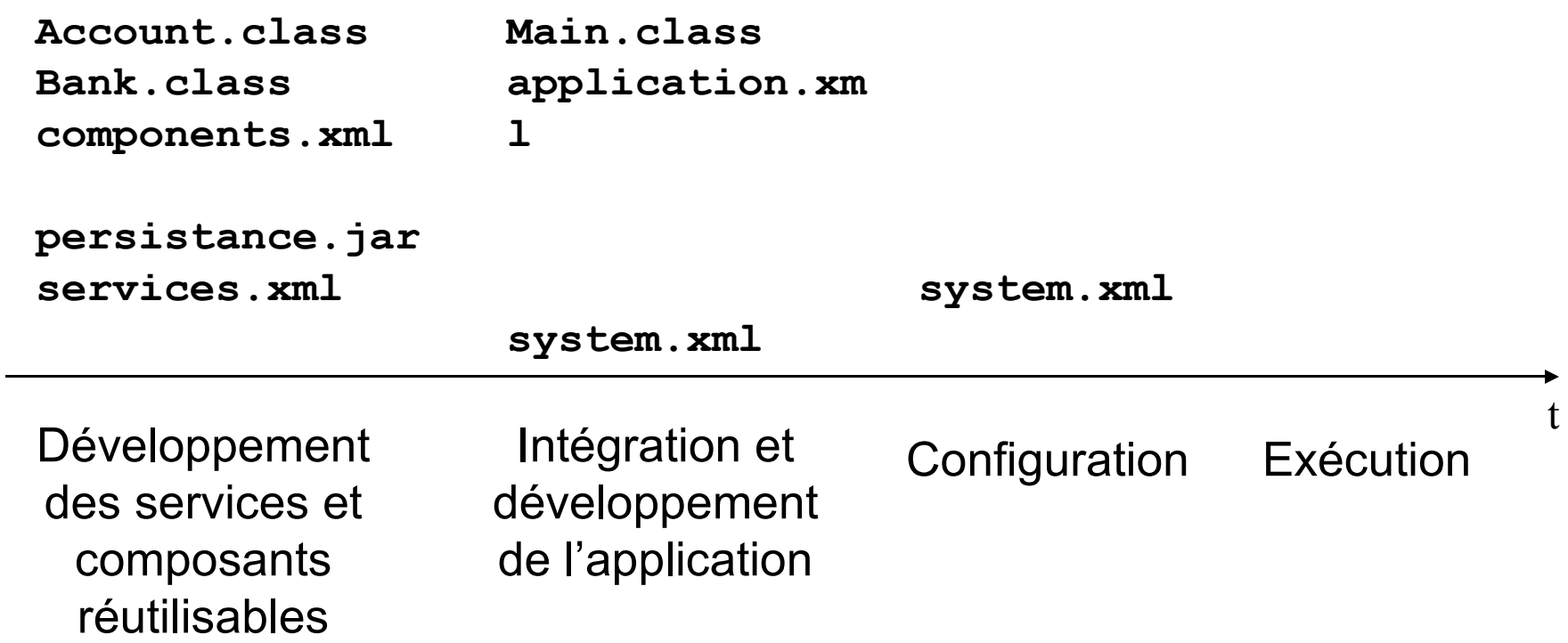
Exemple de politique applicative

```
<application-policy>
  <group name='comptes-surveillés'>
    <select from="all">
      <and>
        <equals>
          <attribute-value name='className'/>
          <string value='demo.Account'/>
        </equals>
        <less-than>
          <attribute-value name='balance'/>
          <number value='500'/>
        </less-than>
      </and>
    </select>
    <bind system-policy='tracer'>
      <parameter name='prefix' value='COMPTE'/>
    </bind>
  </group>
</application-policy>
```

Architecture générale



Processus de développement



Conclusion

- Infrastructure adaptable intégrant:
  - *Observation*
    - des ressources (moniteurs)
    - des programmes (introspection)
  - *Décision:* **politiques d'adaptation**
    - définies statiquement par *les* programmeurs
    - interprétées à l'exécution par le moteur
  - Action
    - modification des associations services/composants
    - reconfiguration *dynamique*

Questions ouvertes & travaux futurs

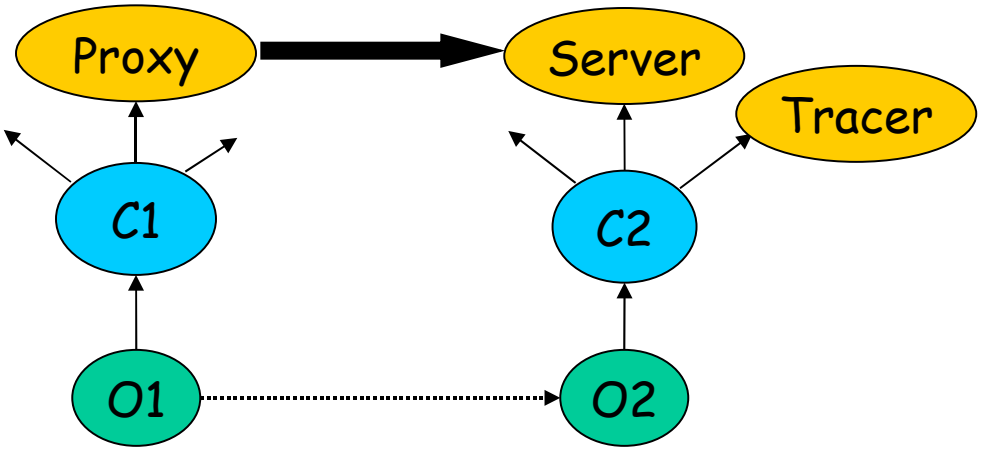
- Fonctionnement dans un cadre distribué
  - coopération entre les hôtes
- Composition de services
- Formalisation (synchronisation)
- Politiques plus déclaratives
- Réutilisation de l'infrastructure en remplaçant RAM par ProActive ou JOnAS

Bibliographie

- *An approach for constructing dynamically adaptable component-based software systems using LEAD++,* Amano & Watanabe, *OORASE99*
- *A declarative approach for designing and developping adaptive components*, Boinot et al., *ASE2000*
- *A generic approach to satisfy adaptability needs in mobile environments*, André & Segarra, *HICSS33*
- *JavaPod: une plate-forme à composants adaptables et extensibles*, Bruneton & Riveill, *Rapport INRIA 3850*
- *Monitoring, security and dynamic configuration with the dynamicTAO reflective ORB*, Kon et al., *Middleware 2000*
- *On the integration of configuration and meta-level programming approaches*, Loques et al, *OORASE2000*
- *Aspects dynamiques des langages de description d'architectures logicielles*, Riveill & Senart, *L'Objet X/2001*

Services, rôles et métaobjets

- Un service peut impliquer plusieurs composants
  - notion de rôles (ex: `proxy`, `server`)
- Plusieurs rôles coopèrent pour implémenter un service
- Métaobjets implémentent les rôles



Observation des ressources

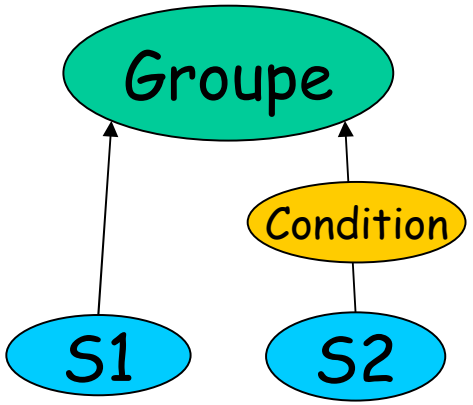
- Exemple:

```
/system
    uptime_s: 3754
    hostname: 'www.foo.com'
    /cpu
        vendor: 'intel'
        type: 'pentium3'
        speed_mhz: 600
    /disk
        /0
            capacity_kb: 15 000 000
            free_kb: 234 521
```

- Désignation: `'/system/cpu.vendor'` => `'intel'`

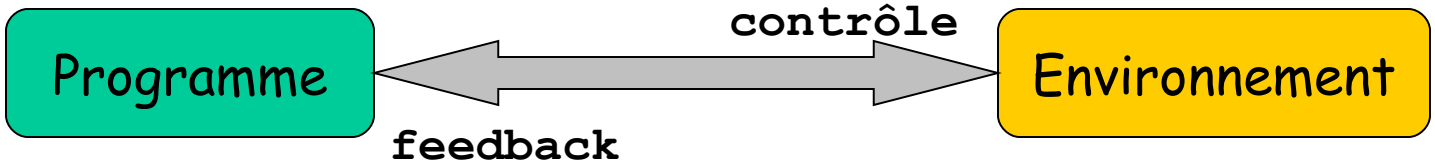
Politiques applicatives

- Ecrites par le programmeur d'application
- Définit des groupes de composants
  - composants liés sémantiquement
  - doivent être traités de la même façon
- Associe ces groupes à:
  - des services
  - des politiques système



Motivation

- Système informatique:



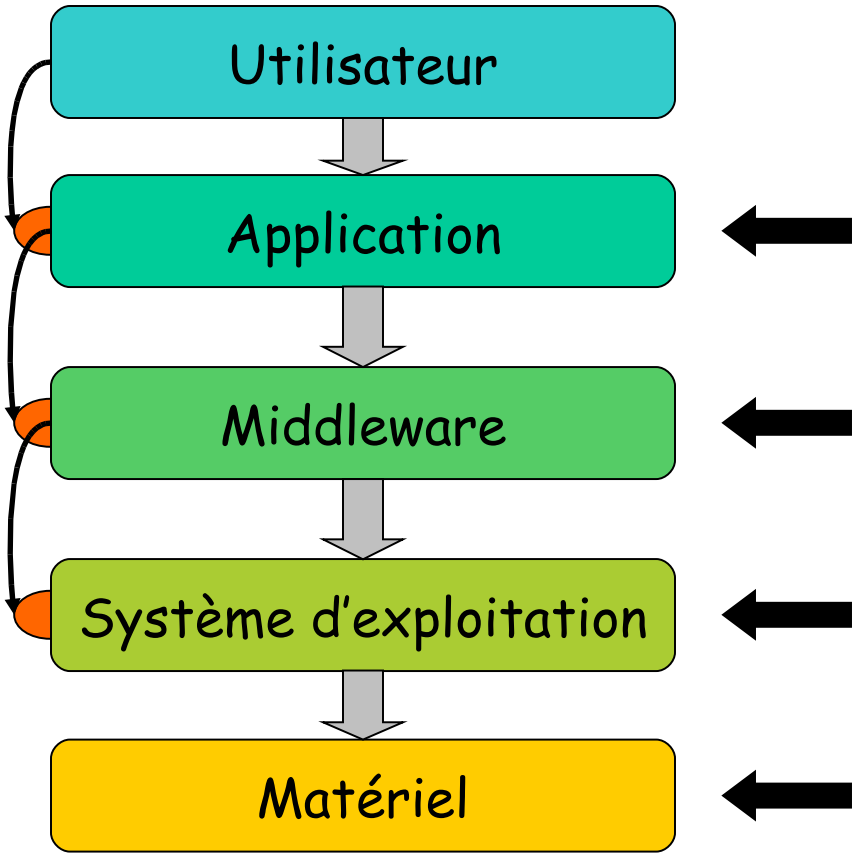
- Fort couplage entre les deux éléments
- Environnement très variable et complexe:
  - statiquement (spectre matériel: du PDA au cluster)
  - dynamiquement (conditions d'exécution changeantes)
- Le programme doit être **adaptable**
- *Problème*: dans ce schéma, c'est au programme de gérer les fluctuations de son environnement

Solution partielle

- Découplage par introduction d'un niveau d'indirection: le *middleware* (intergiciel)

- Agit comme un médiateur, rend le programme moins dépendant des variations de l'environnement.
- Deux rôles:
  - fournit des abstractions standards
  - prend en charge les aspects non-fonctionnels
- *Solution partielle*: on a seulement déplacé le problème !

Couches logicielles



Composants fonctionnels

Services non-fonctionnels (persistance, transaction, crypto…)

Services de bas niveau (filesystem, threads, VM…)

Ressources physiques (CPU, mémoire, connectivité…)