Master II Informatique
Parcours : Génie logiciel

SILOC : Spécification et Implémentation des Langages à Objets et à Composants - Spécification et transformation de contraintes sur les architectures à base de composants -

Chouki.TIBERMACINE@lirmm.fr
Maître de conférences en informatique

http://www.lirmm.fr/~tibermacin/ens/siloc/

Plan du cours

- **Partie 1 :** Généralités sur les composants et architectures logicielles à base de composants
- **Partie 2 :** Spécification des contraintes architecturales
- **Partie 3 :** Un modèle de composants pour les contraintes architecturales
- **Partie 4 :** Transformation des contraintes architecturales
- **Partie 5 :** Génération de code à objets à partir de contraintes architecturales
- **Partie 6 :** Génération de composants à partir des contraintes

Contenu de cette partie

- Problématique abordée
- Solution basée sur le langage ACL
- Transformation de contraintes ACL
- Outil support
- Travaux en cours

Contenu de cette partie

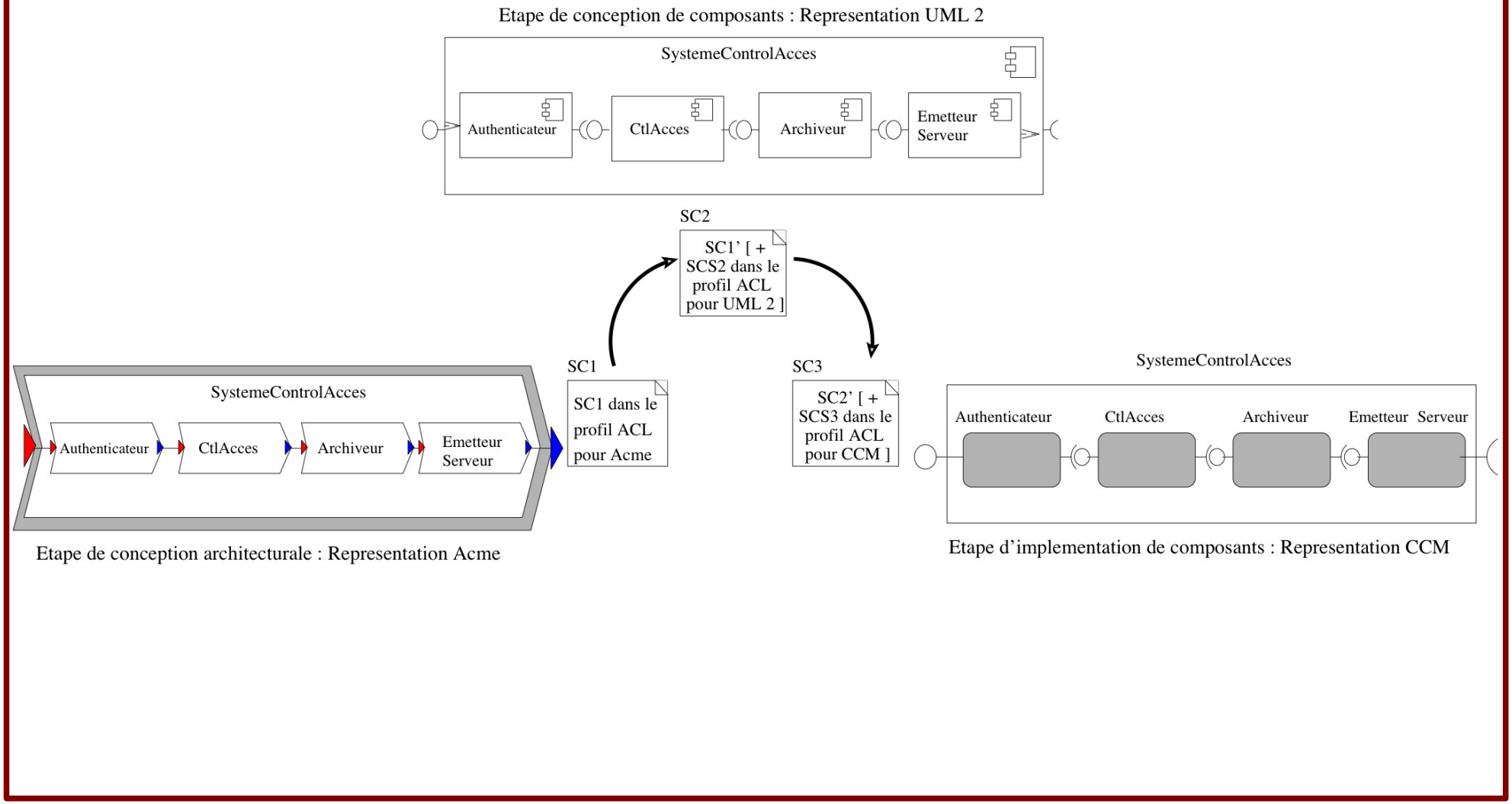
- Problématique abordée
- Solution basée sur le langage ACL
- Transformation de contraintes ACL
- Outil support
- Travaux en cours

Problématique lors du développement

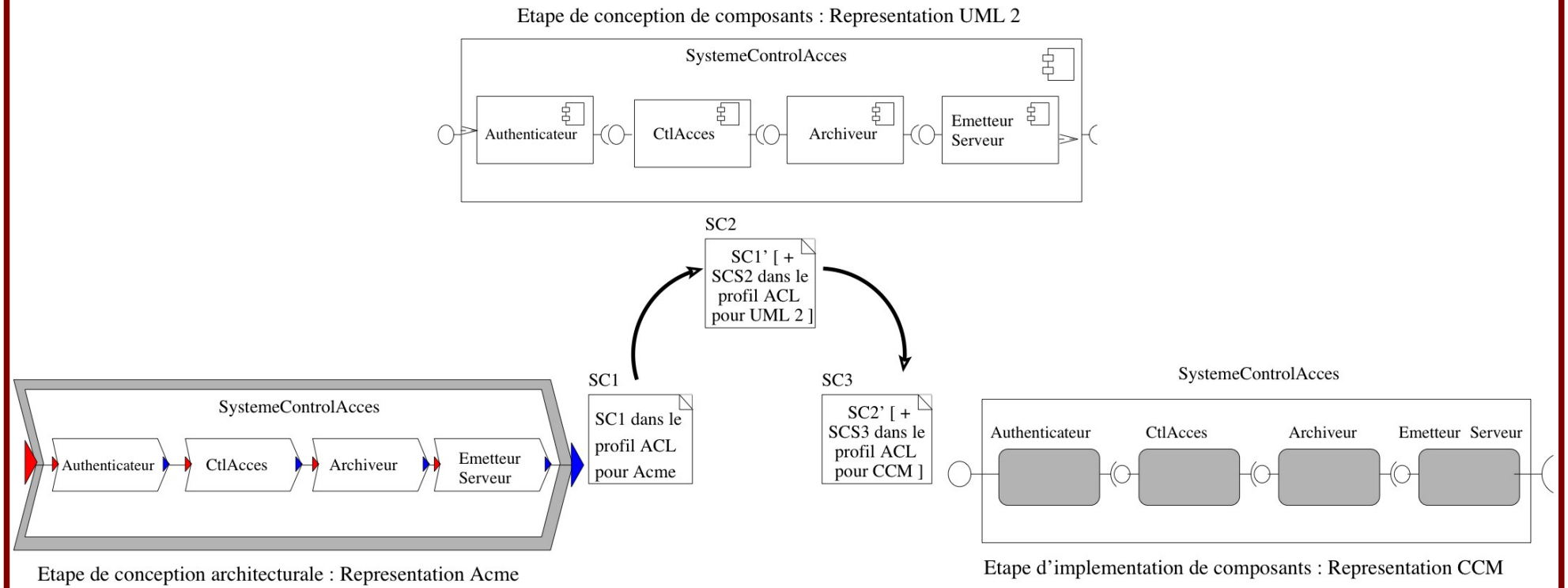
- En transitant d'une étape à une autre dans le processus de développement (dans un contexte d'IDM) :
  - une décision architecturale peut être affectée
  - par conséquent, les attributs qualité correspondants peuvent être altérés

- Par exemple :
  - **Implémenter** un connecteur entre deux composants non-adjacents dans un système **conçu** selon le style d'architecture en couches
    ⇒ le choix du style architectural en couche est affecté
      ⇒ le système est moins maintenable

Besoins de préservation des décisions



Besoins de préservation des décisions -suite-



- On doit être capable de décrire AD1, ADS2 et ADS3
- On doit être capable d'effectuer une évaluation facile de AD1 dans le contexte de la phase 2 (AD1'), et AD2 dans le contexte de la phase 3 (AD2')

Contenu de cette partie

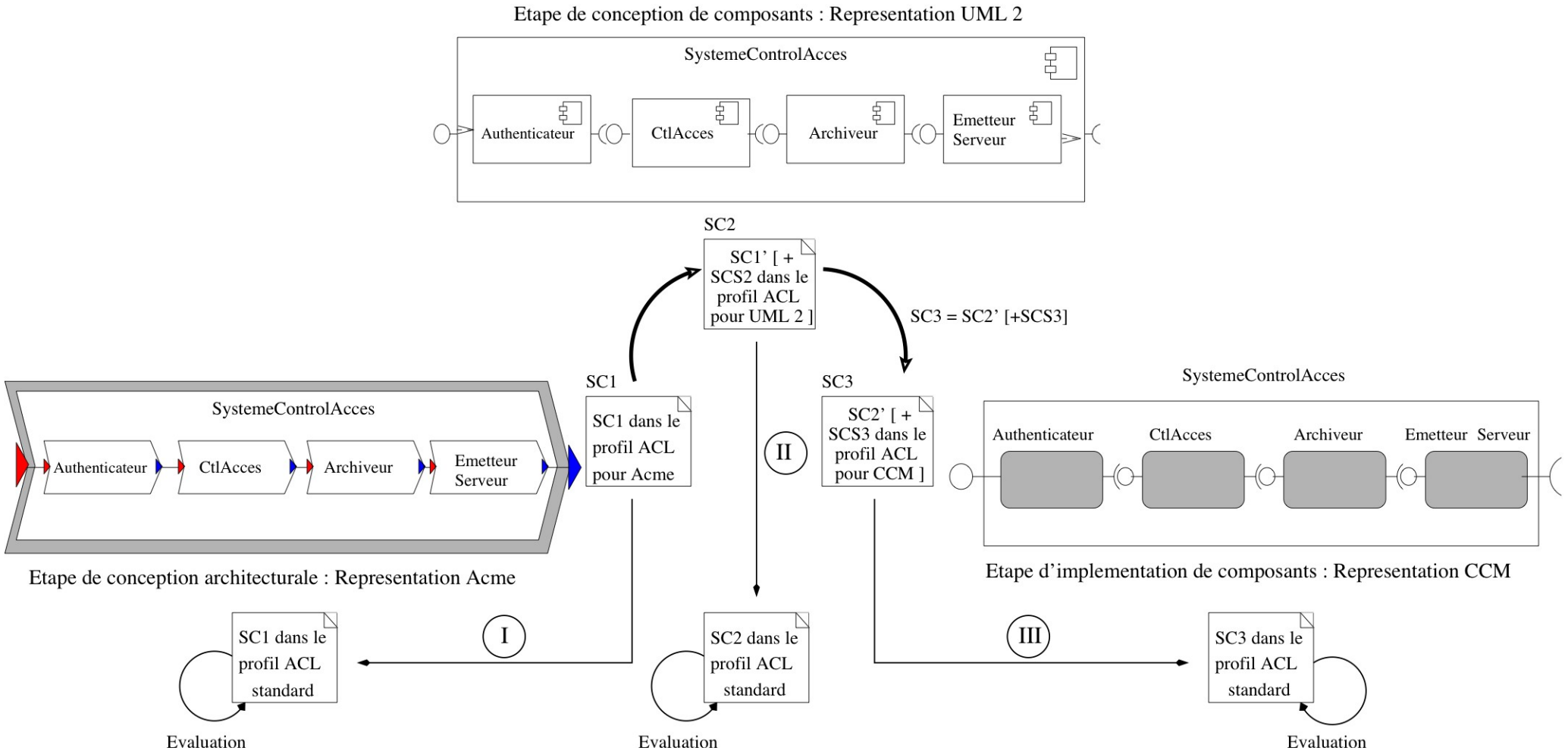
- Problématique abordée
- Solution basée sur le langage ACL
- Transformation de contraintes ACL
- Outil support
- Travaux en cours

ACL pour répondre à ces besoins

- ACL présente des propriétés intéressantes pour la préservation des décisions architecturales :
  - utilisable tout au long du processus de développement (description de AD1, ADS2, ADS3)
  - une même méthode d'évaluation quelque soit la phase (évaluation de AD1’ et AD2’)

- A une phase donnée, il est possible d'évaluer toutes les contraintes définies dans les phases en amont

ACL pour la traçabilité



Passage par un modèle intermédiaire

- Face à une diversité des ADL et des technologies de composants
- Face à une diversité des profils ACL pour ces derniers
- Proposition d'un profil unique évaluable
- Les contraintes écrites dans les différents profils sont d'abord transformées vers ce profil pour être évaluées

Passage par un modèle intermédiaire -suite-

Contraintes architecturales - Profil ACL pour xAcme

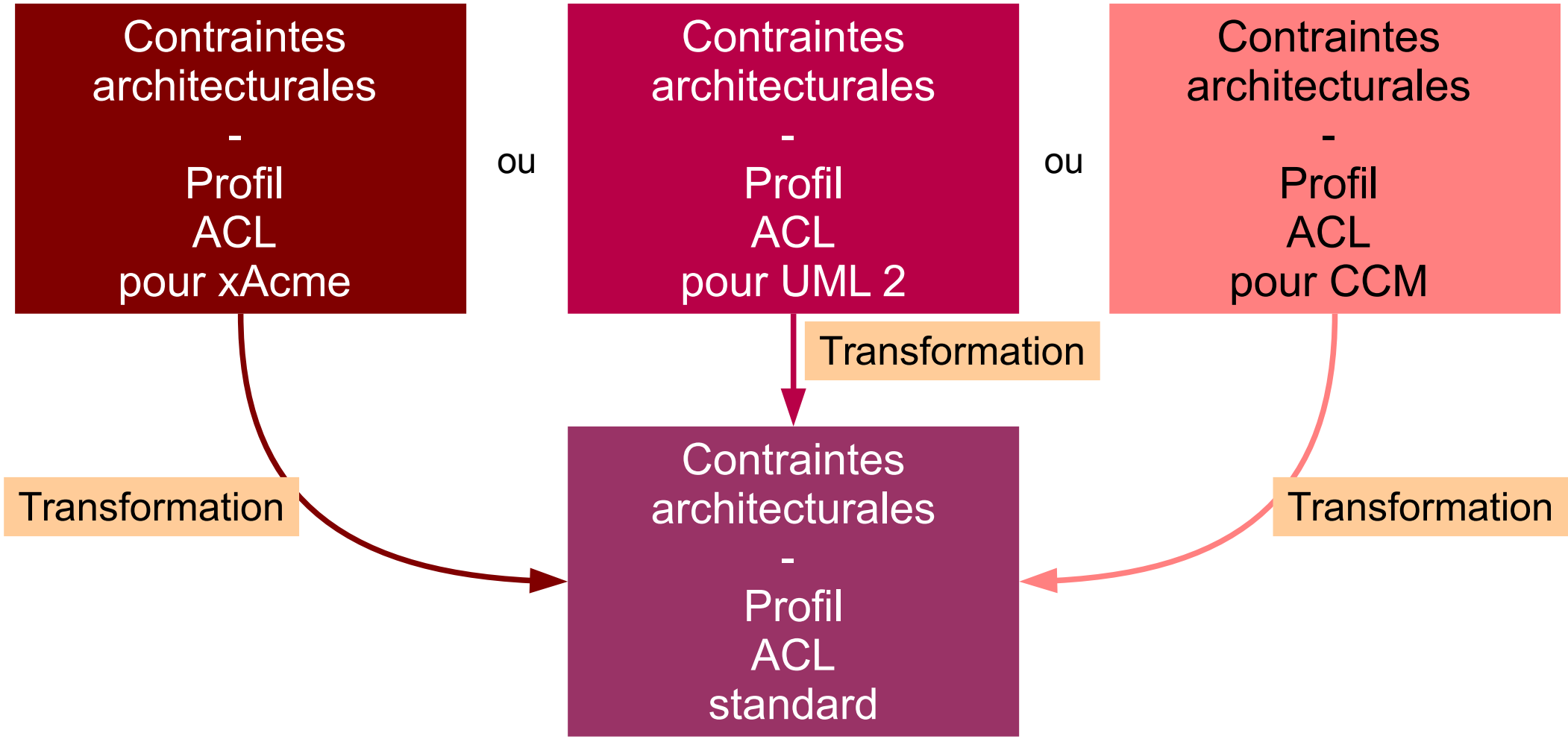
ou

Contraintes architecturales - Profil ACL pour UML 2

ou

Contraintes architecturales - Profil ACL pour CCM

Passage par un modèle intermédiaire -suite-



Passage par un modèle intermédiaire -suite-

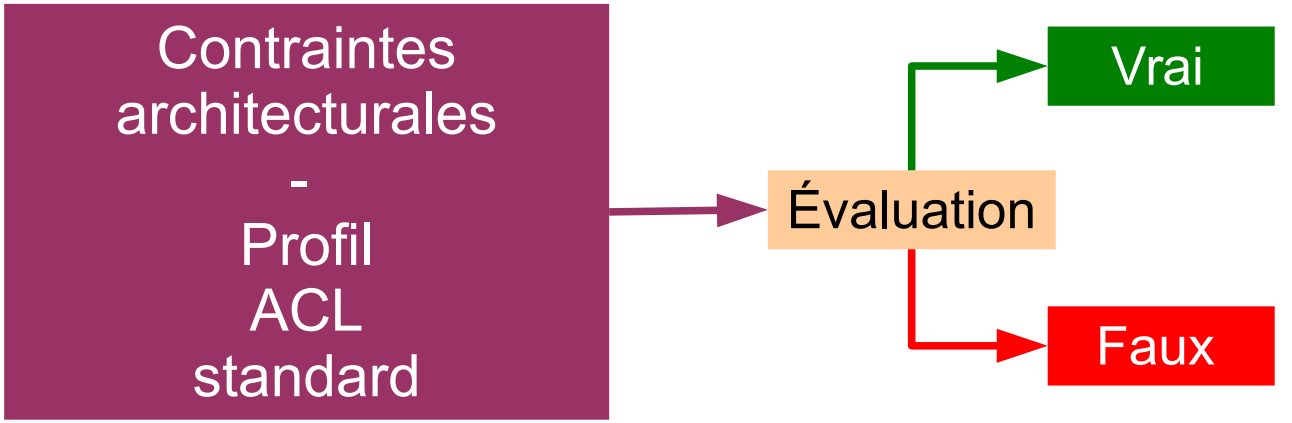
Contraintes architecturales - Profil ACL pour xAcme

ou

Contraintes architecturales - Profil ACL pour UML 2

ou

Contraintes architecturales - Profil ACL pour CCM

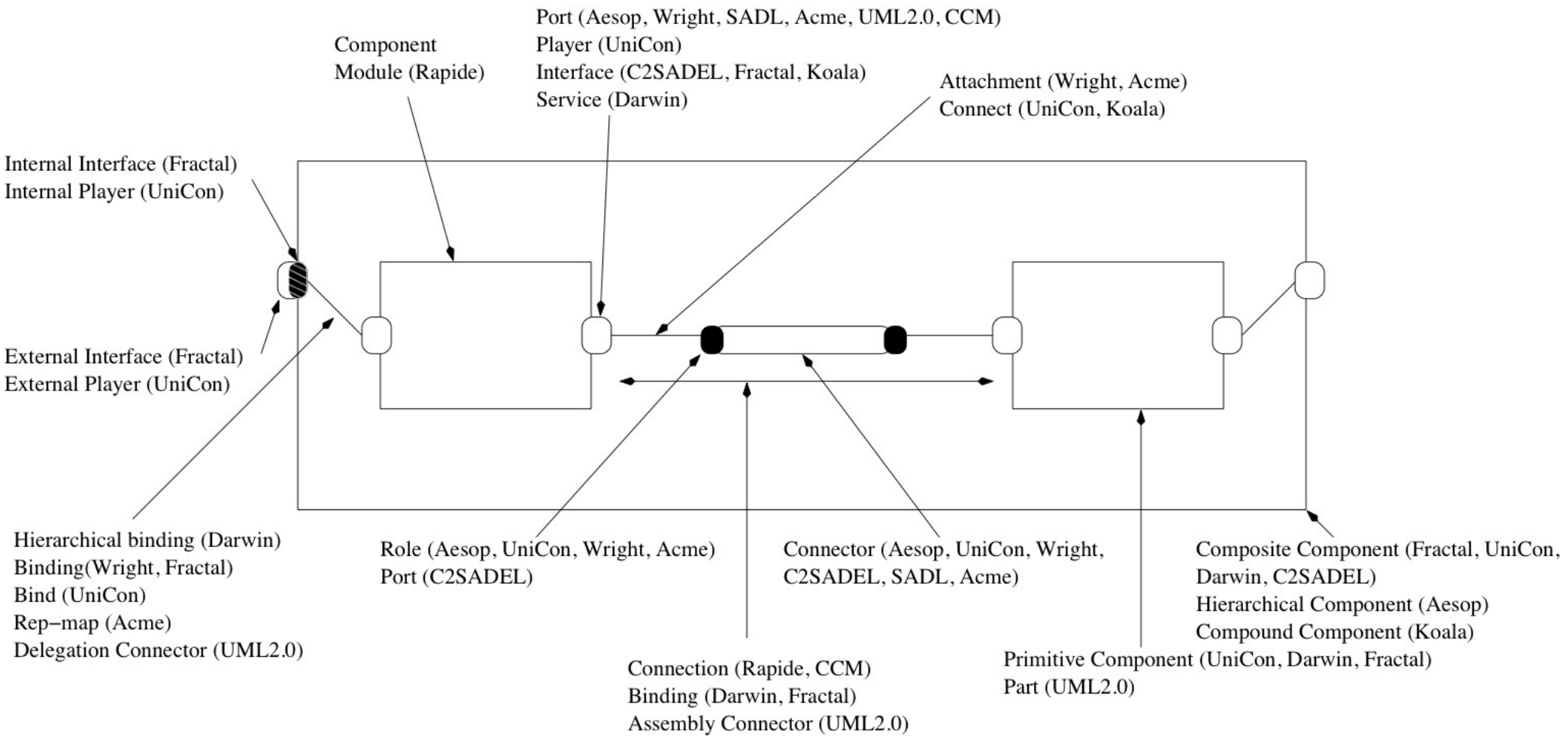


Profil ACL standard

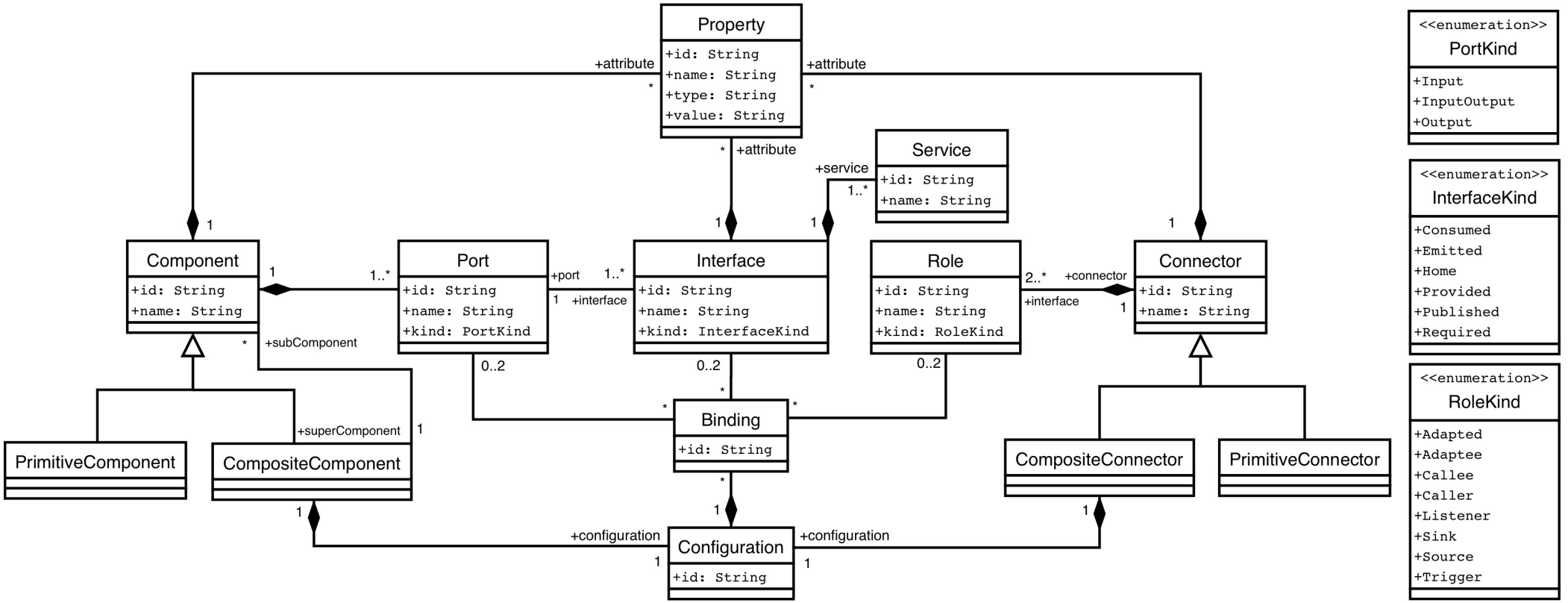
- Profil ACL standard = CCL + ArchMM

- ArchMM : méta-modèle générique englobant les abstractions architecturales retrouvées dans :
  - les langages de description d'architecture proposés dans la littérature et l'industrie (xAcme, C2SADEL, Darwin, Koala, ...)
  - les diagrammes de composants dans UML 2
  - les technologies de composants (EJB, CCM et Fractal)

Abstractions architecturales dans les composants



Le méta-modèle ArchMM



Comparaison des contraintes

- Contrainte dans le profil ACL pour xAcme :

```
context MACS :ComponentInstance inv :
MACS.subArchitecture.archInstance.componentInstance
->forAll(c | c.interfaceInstance
->select(i : InterfaceInstance | i.direction = #in)
->size() < 5)
```

- Contrainte dans le profil ACL standard :

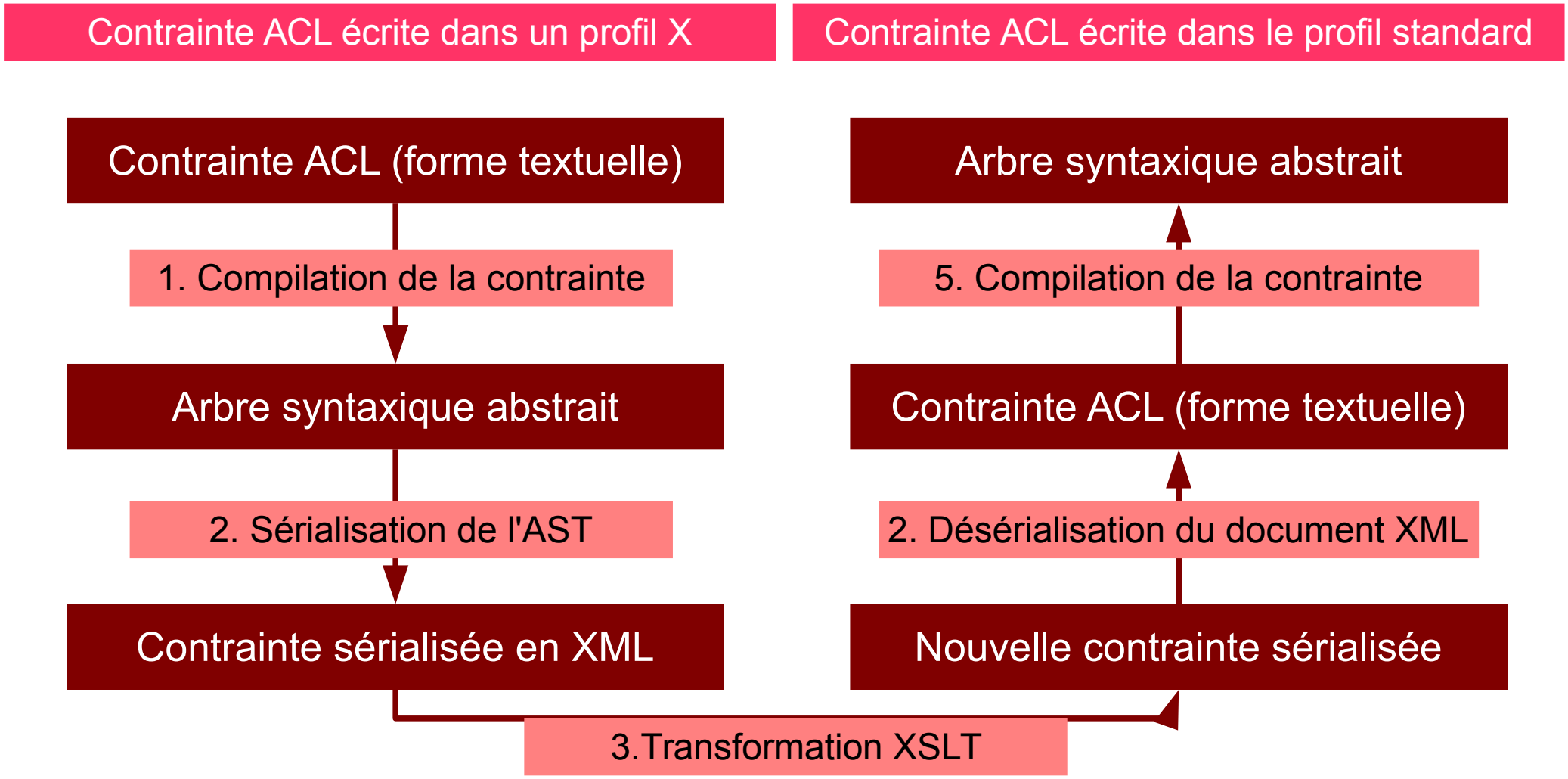
```
context MACS :CompositeComponent inv :
MACS.subComponent
->forAll(c | c.port.interface
->select(i : Interface | i.kind = #Provided)
->size() < 5)
```

- Seuls les concepts architecturaux sont à projeter

Contenu de cette partie

- Problématique abordée
- Solution basée sur le langage ACL
- Transformation de contraintes ACL
- Outil support
- Travaux en cours

Transformation des contraintes vers le profil standard



Éléments concernés par la transformation

- Qu'est-ce qui est transformé ?
  - **les abstractions architecturales :** Composants, interfaces, ports, connecteurs, ...
  - **les relations directes entre ces abstractions :** les interfaces décrivant un port, les rôles d'un connecteur, ...
  - **des patrons de navigation :** toutes les interfaces requises des instance de composants dans l'architecture interne, ...
- Chacun des items précédents est projeté vers le profil standard ACL à travers une ou plusieurs règles de transformation
- L'ordre de transformation :
  1. les patrons de navigation
  2. les relations directes entre abstractions
  3. les abstractions restantes

Exemples de projections

- Projections entre xAcme et ArchMM :
  - Abstractions architecturales :

```
ComponentInstance -> CompositeComponent
SubArchitecture -> Configuration
```

  - Relations directes entre abstractions :

```
connectorInstance.interfaceInstance
->
connector.role
```

  - Patrons de navigation :

```
componentInstance.subArchitecture.archInstance
.componentInstance
->
component.subComponent
```

Par rapport aux techniques existantes

- Classification de Czarnecki et Helsen :
  - les règles de transformation sont purement déclaratives
  - la portée de l'application des règles : sur l'ensemble des modèles source et destination
  - les relations modèle source - modèle cible : modèles cibles sont de nouveaux modèles créés
  - la stratégie de l'application des règles : déterministe
  - l'ordonnancement des règles : implicite dans les règles
  - l'organisation des règles : templates XSLT réutilisables
  - la traçabilité entre modèles source et cibles : non préservée
  - la direction : unidirectionnelle

Évaluation des contraintes

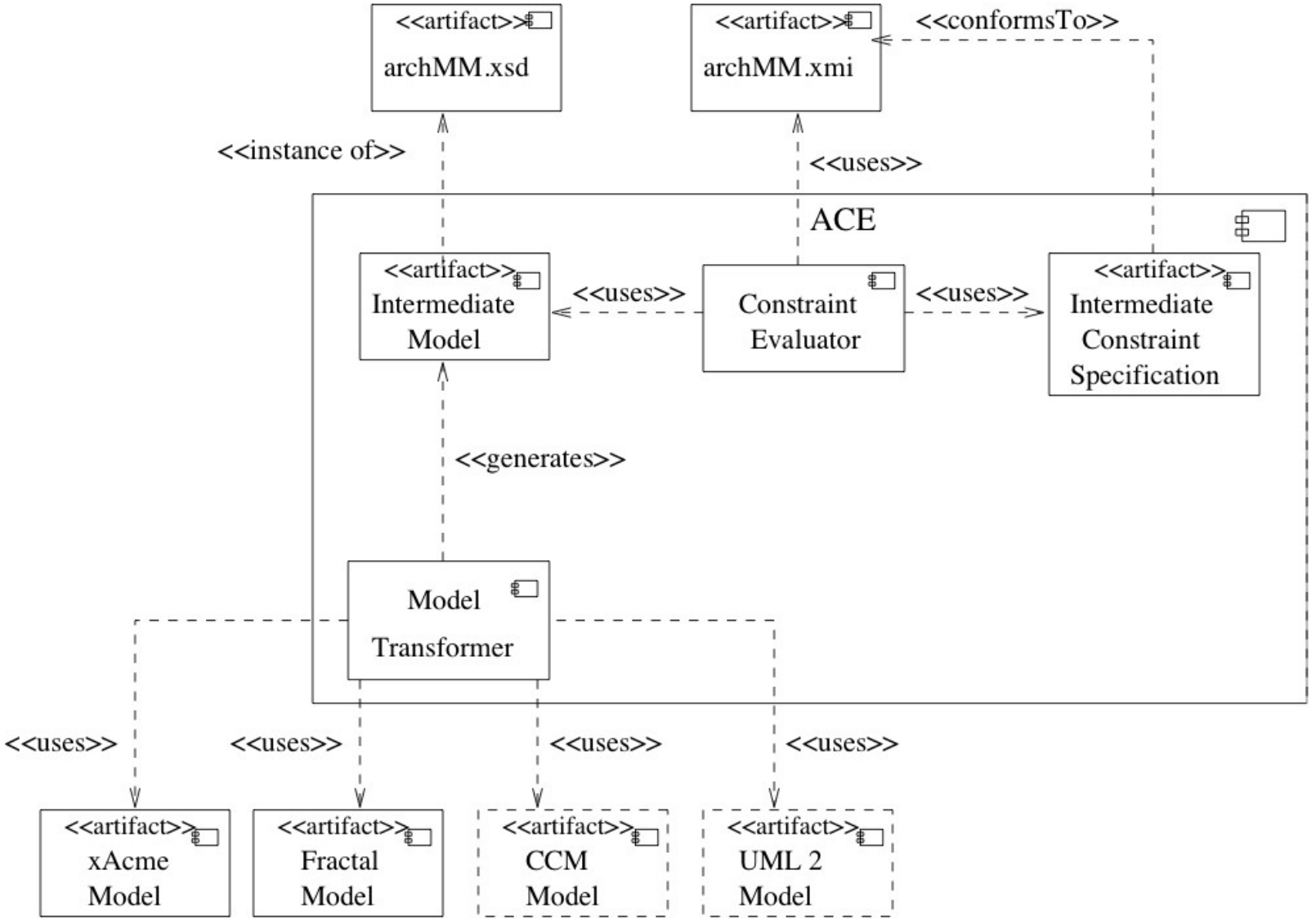
- Les contraintes sont compilées avec une version modifiée d'OCL Compiler[1] (version développée avec JAVA)
- L'évaluation est ensuite lancée à partir de la racine des arbres syntaxiques produits lors de la phase précédente :
  - toutes les classes des nœuds ont été enrichies avec des méthodes pour l'évaluation qui propagent l'évaluation et retournent les résultats
- Les erreurs sont notifiées au développeur (syntaxiques, de type, ou évaluation à faux) pour validation des contraintes
- Le risque d'erreurs syntaxiques est minimisée grâce à une aide à l'écriture des contraintes (en s'appuyant sur le méta-modèle)

[1]Site du compilateur OCL sur source forge : http://dresden-ocl.sourceforge.net/

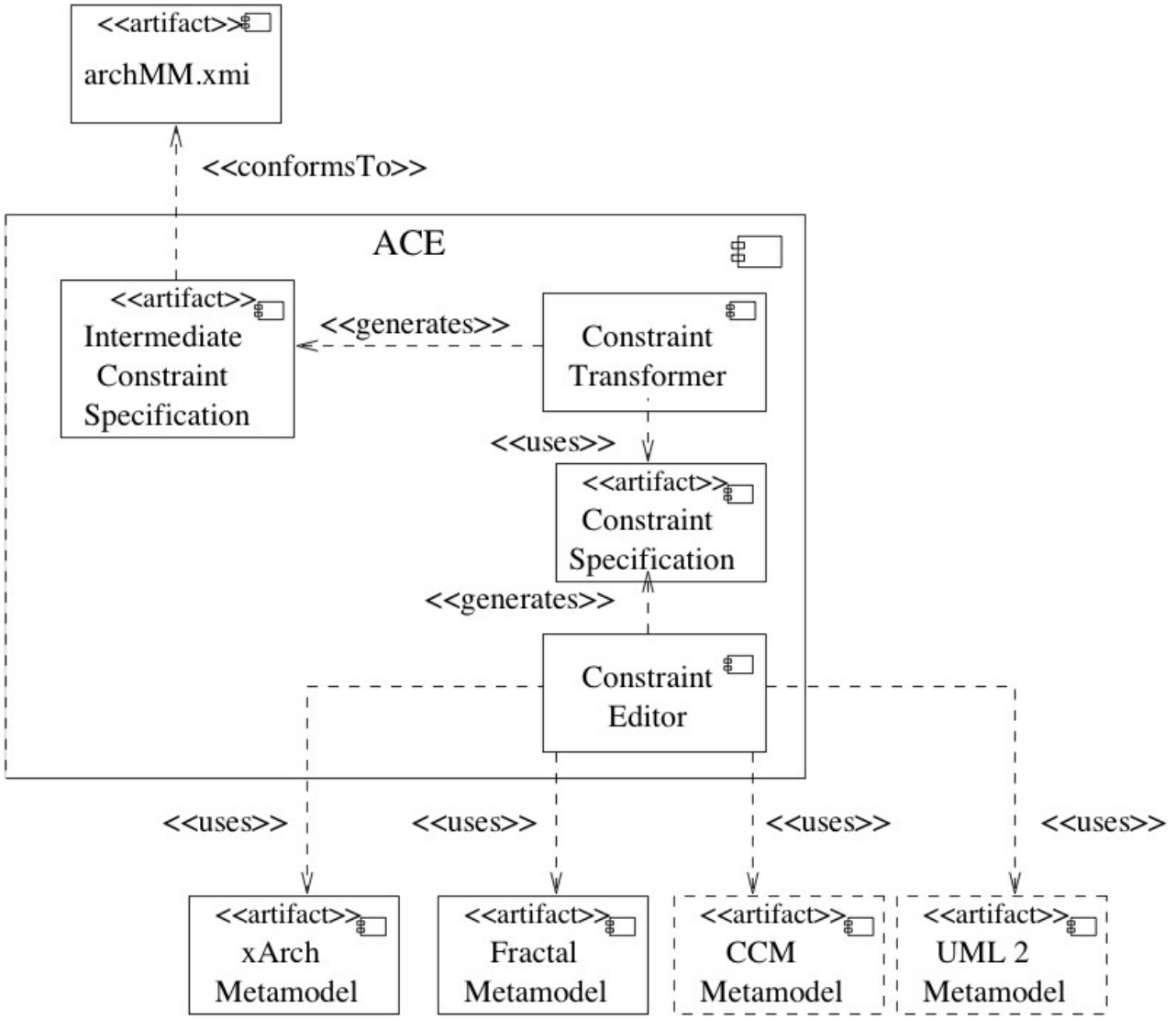
Contenu de cette partie

- Problématique abordée
- Solution basée sur le langage ACL
- Transformation de contraintes ACL
- Outil support
- Travaux en cours

ACE : évaluateur de contraintes



ACE : évaluateur de contraintes -suite-



Contenu de cette partie

- Problématique abordée
- Solution basée sur le langage ACL
- Transformation de contraintes ACL
- Outil support
- Travaux en cours

Thèse terminée et travail en cours ...

- Thèse commencée en septembre 2007 et terminée en 2010 à l'université Bretagne-Sud
- Le travail continue ...
- Projet en cours : Transformation de contraintes basée sur les langages de transformation de modèles
- L'idée :
  - Exploiter les règles de transformation déjà définies pour la transformation des modèles pour transformer les contraintes architecturales spécifiées pour ces modèles
- Autre projet en cours : Transformation de contraintes OCL en CSP

Questions