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

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

Constats et problématique

- Les contraintes architecturales sont souvent réutilisées
- Quand elles sont réutilisées, elles sont sujettes à une personnalisation avant leur association à une description d'architecture
- Les contraintes architecturales sont sujettes à une composition (concevoir des contraintes architecturales complexes à partir de contraintes plus simples)

Problématique par l'exemple 1

- Contrainte vérifiant le style d'architecture en couches :

```
context ACS:CompositeComponentDescriptor inv:
-- Each layer should be connected
ACS.internalArchitecture.instance->forAll(c: ComponentInstance |
c.port.inConnector->union(c.port.outConnector)->notEmpty())
and
-- Each layer should be connected to at most two other layers
ACS.internalArchitecture.instance->forAll(c: ComponentInstance |
c.port.inConnector.toPort.instance->union(c.port.outConnector
.fromPort.instance)->asSet()->size() <= 2)
and
-- The number of layers that are connected to only one other layer
-- is equal to two (top and bottom layers)
ACS.internalArchitecture.instance->select(c:ComponentInstance |
c.port.inConnector.toPort.instance->union(c.port.outConnector
.fromPort.instance)->asSet()->size() = 1)->size() = 2
```

Problématique par l'exemple 1 -suite-

- Contrainte vérifiant le style d'architecture « Pipe & Filter » :

-- The number of components having input and output ports
-- is equal to the total number of components – 2
... and
-- There is only one component having only input ports
... and
-- There is only one component having only output ports
... and
-- Each filter should be connected
self.internalArchitecture.instance->forAll(c:ComponentInstance | c.port.inConnector->union(c.port.outConnector)->notEmpty())
-- Connectors between each pair of components should go
-- in the same direction
... and
-- Connectors between all components should go in the same direction
...

Remarques sur ces exemples

- La contrainte vérifiant que chaque couche ou chaque filtre est connecté dans une architecture interne est réutilisée :

ACS.internalArchitecture.instance->forAll(c: ComponentInstance | c.port.inConnector->union(c.port.outConnector)->notEmpty())

- Cette contrainte peut être utilisée dans d'autres contextes. Elle stipule plus généralement :
« Chaque instance de composant dans l'architecture interne d'un composant composite doit être connectée à au moins une autre instance de composant »

- Contraintes architecturales : entités réutilisables

Remarques sur ces exemples -suite-

- Les différentes parties de ces contraintes sont combinées pour réaliser une contrainte architecturales plus complexes

- Architecture en couche =
  - Chaque couche doit être connectée
    and
  - Chaque couche doit être connectée à au plus 2 autres couches
    and
  - ...

- Contraintes architecturales : entités composables

Problématique par l'exemple 2

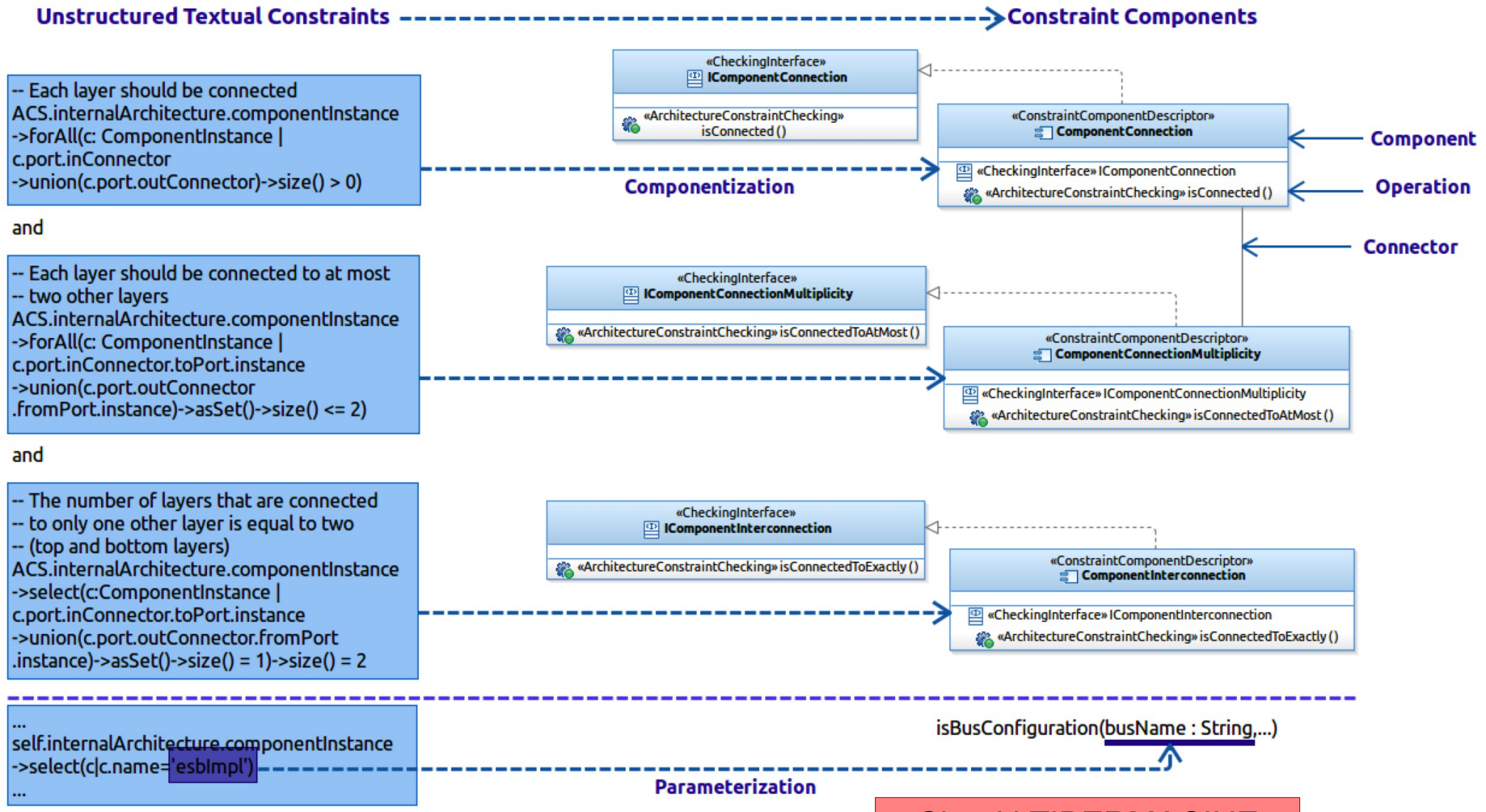
- Contrainte vérifiant le patron d'architecture en bus :

```
let bus:ComponentInstance= self.internalArchitecture.instance
->select(c | c.name='esbImpl')->first() ,
customers:Set(ComponentInstance)=self.internalArchitecture
.instance->select(c | (c.name='cust1')or(c.name='cust2')
or (c.name='cust2')) ,
producers:Set(ComponentInstance)=self.internalArchitecture
.instance->select(c | (c.name='prod1')or(c.name='prod2')
or (c.name='prod2'))
in
-- Le bus doit avoir au moins un port Input et un port Output
bus.port->exists(p1,p2 | (p1.kind = PortKind::Input)
and (p2.kind = PortKind::Output))
and ...
```

Remarques sur cet exemple

- Dans les expressions en gras, on utilise des références d'éléments d'architecture particuliers visés par la contrainte (noms d'instances de composants)
- Il serait plus judicieux de paramétrer la contrainte avec ces références pour la rendre plus réutilisable
- Contrainte d'architecture : entités personnalisables (paramétrables)

Objectifs de ce travail



Idée générale : composants-contraintes

- Construire les contraintes d'architecture sous la forme de composants, dits « **composants-contraintes** »
- Ces composants exportent des ports (Input) décrits par des interfaces contenant des opérations pour vérifier des contraintes (contraintes que ces composants savent vérifier)
- Ces composants peuvent définir des ports (Output) décrits par des interfaces contenant des opérations requises (contraintes dont ces composants ont besoin et qui sont vérifiées par d'autres composants)
- Ces composants peuvent être assemblés avec des connecteurs

Ports de vérification de contraintes et ports d'introspection

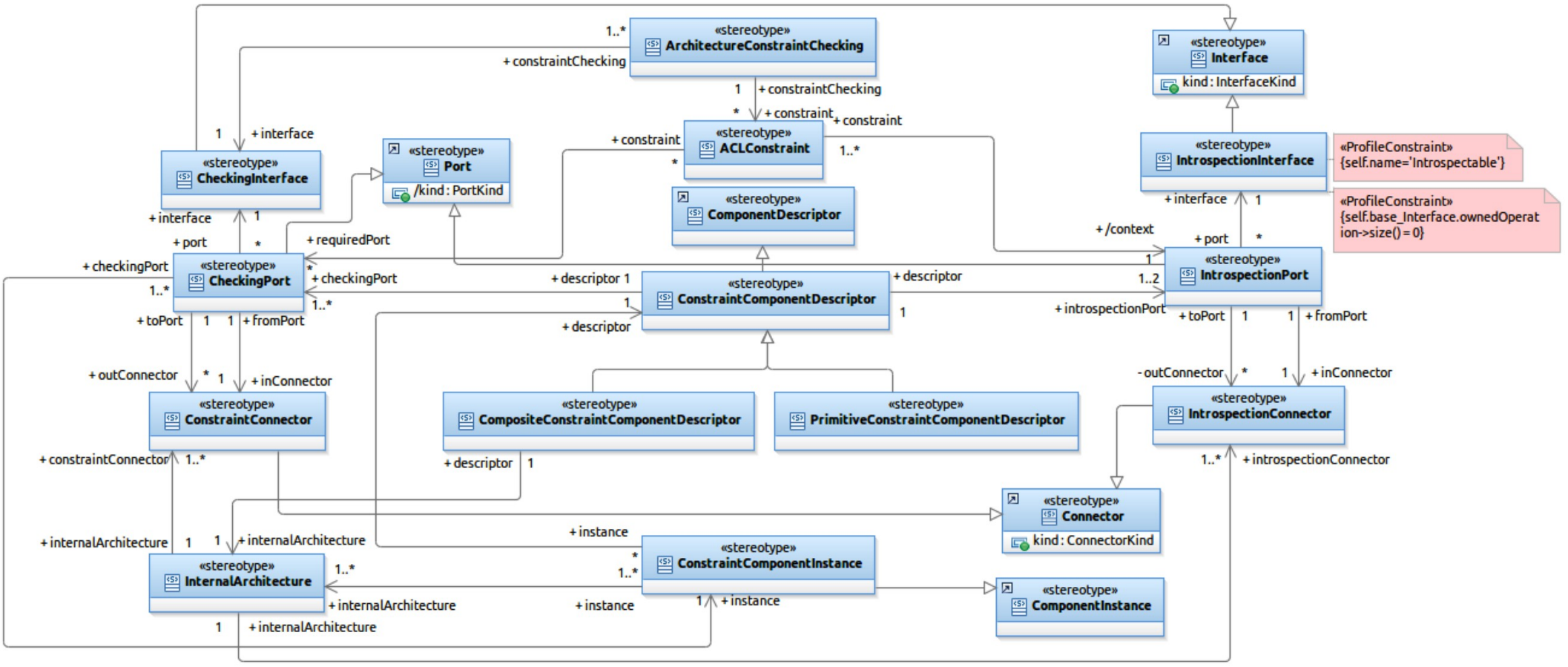
- Un composant-contrainte possède au moins un port de vérification de contraintes : port à travers lequel les contraintes peuvent être vérifiées (ce port doit être de type Input. Le composant peut avoir un ou plusieurs ports de type Output)

- Un composant-contrainte possède aussi systématiquement un port Output (décrit par une interface requise de marquage : Introspectable) qui s'appelle « port d'introspection »

- Ce port a pour nom : « context ». Il sert à connecter le composant-contrainte au composant « métier » qui doit être « *checké* »

- Les contraintes ACL utilisent ce port pour référencer le contexte

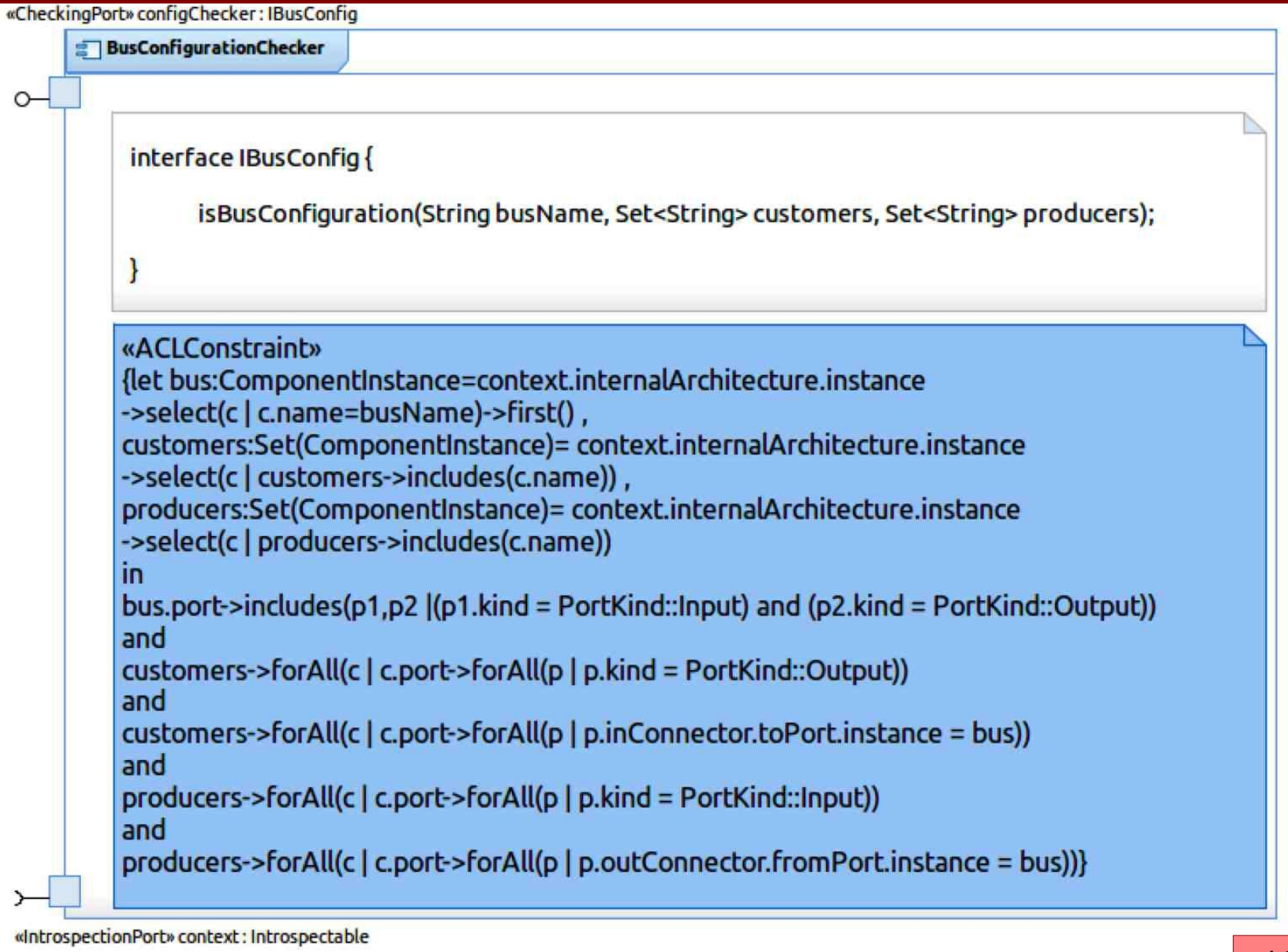
Méta-modèle des composants-contraintes



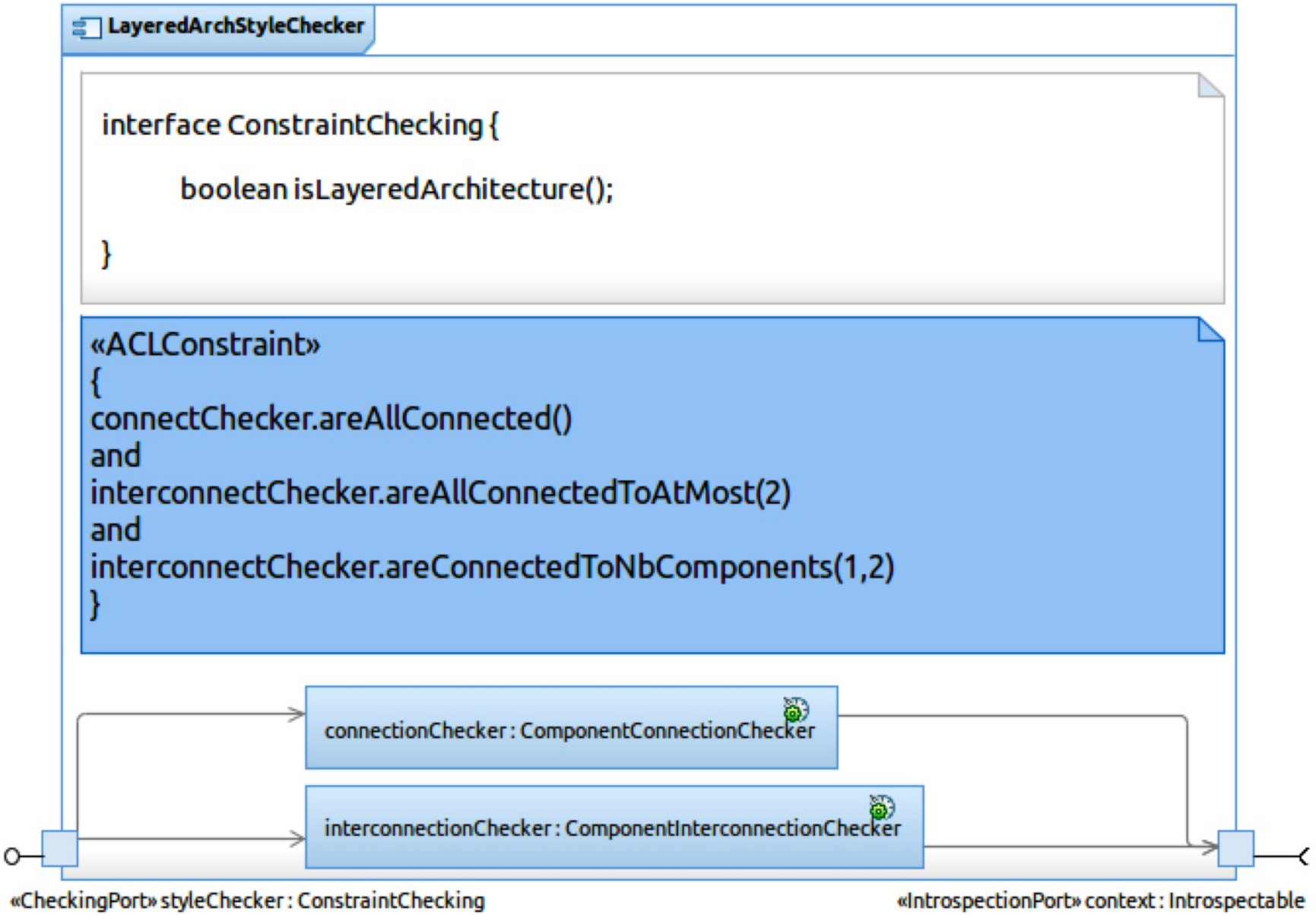
Spécifier des composants-contraintes

- Rôle : développeur de composants-contraintes
  - On décrit en premier les contraintes ACL
  - On leur donne des signatures de la forme :
    public boolean nomOperation(<liste-paramètres>)
  - On groupe les signatures dans des interfaces dites « *checking interfaces* »
  - On définit un descripteur de composant-contrainte
  - Si le composant est primitif, déclarer ses ports « Input » décrits par les « *checking interfaces* »
  - Si dans les contraintes ACL on utilise des ports « Output » décrits eux aussi par des « *checking interfaces* », les déclarer
  - Si le composant est composite, lui définir une architecture interne avec des instances de composants-contraintes
  - Le port d'introspection et les connecteurs de délégation vers celui-ci sont générés automatiquement

Exemple de composant-contrainte primitif



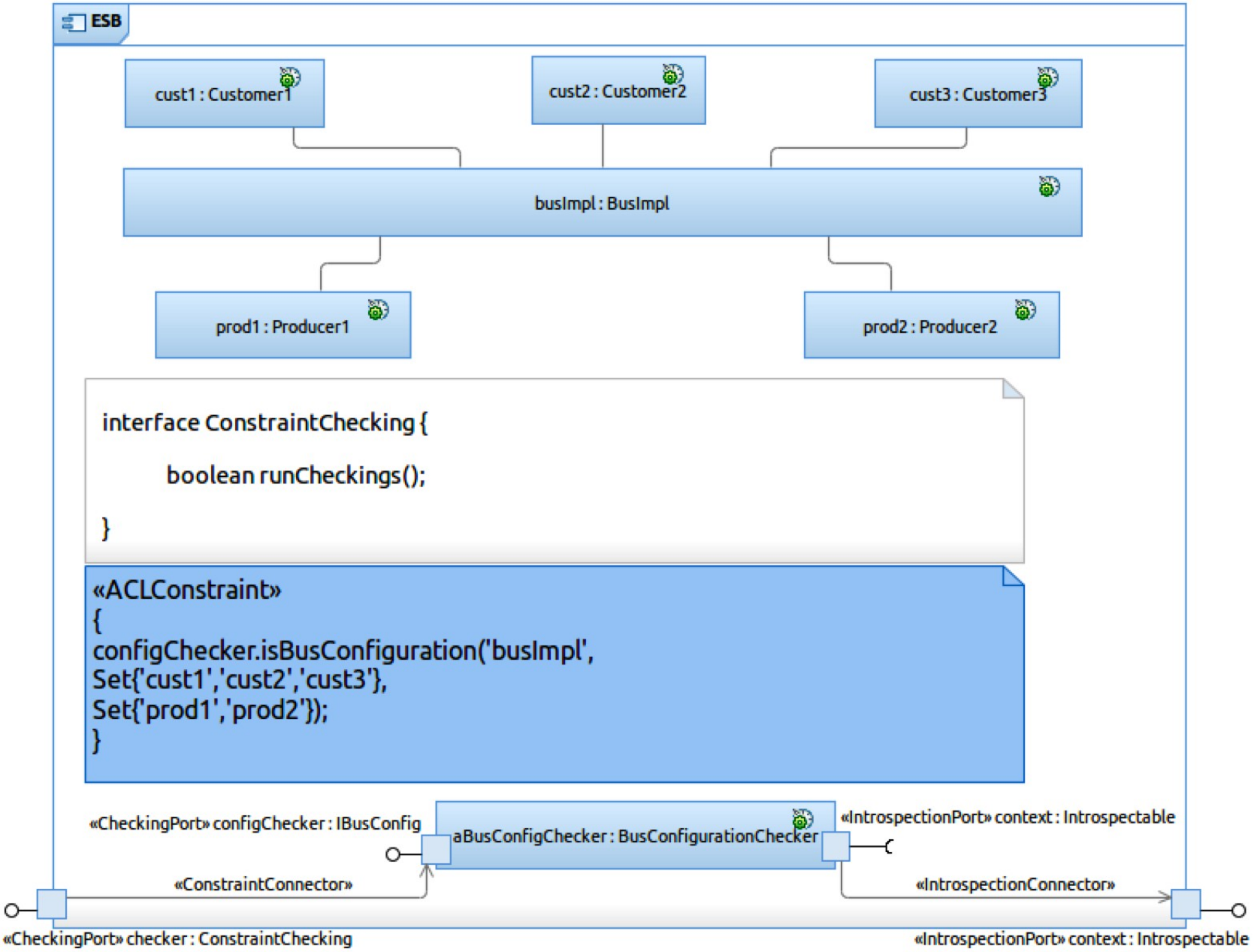
Exemple de composant-contrainte composite



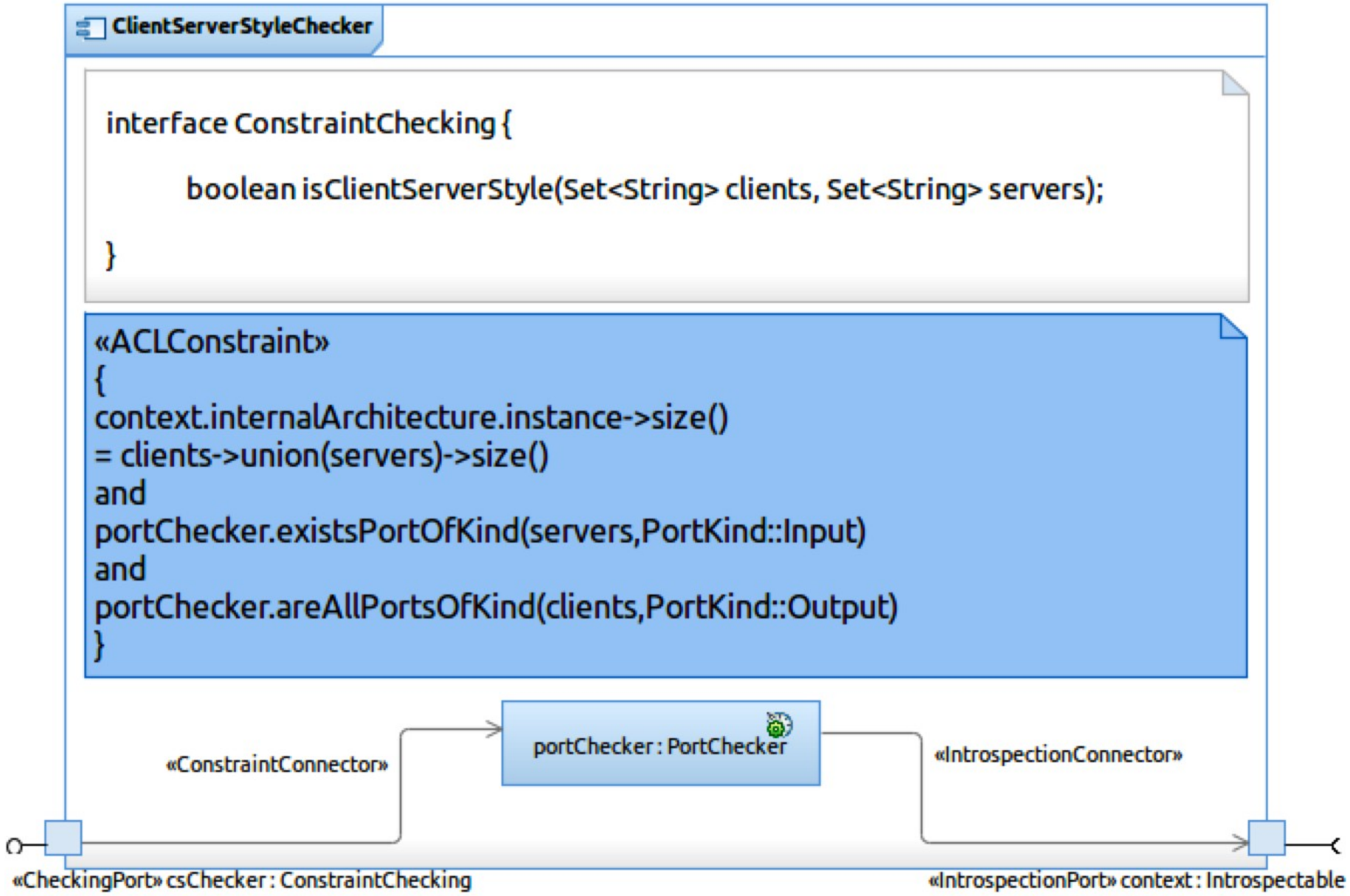
Connecter des composants-contraintes à des composants « métier »

- Rôle : utilisateur de composants-contraintes
  - Déclarer un port de vérification de contraintes de type « Input »
  - Déclarer une instance d'un composant-contrainte dans l'architecture interne du composant métier
  - Connecter, par le biais d'un connecteur de délégation, le port du composant métier à un des ports du composant-contrainte
  - La création d'un port d'introspection et la connexion de celui-ci au port du composant-contrainte se fait automatiquement

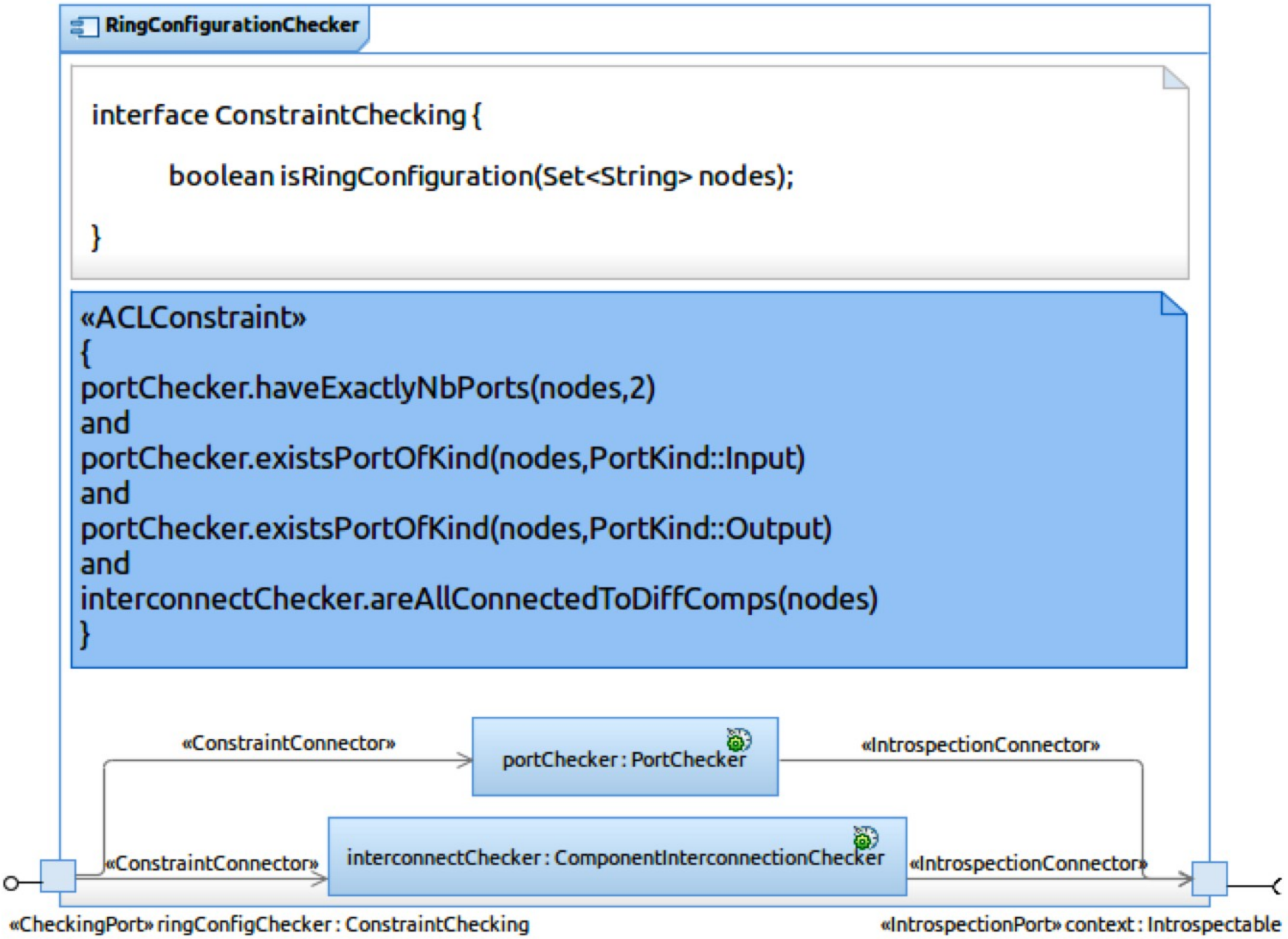
Exemple de composant « métier » connecté à un composant-contrainte



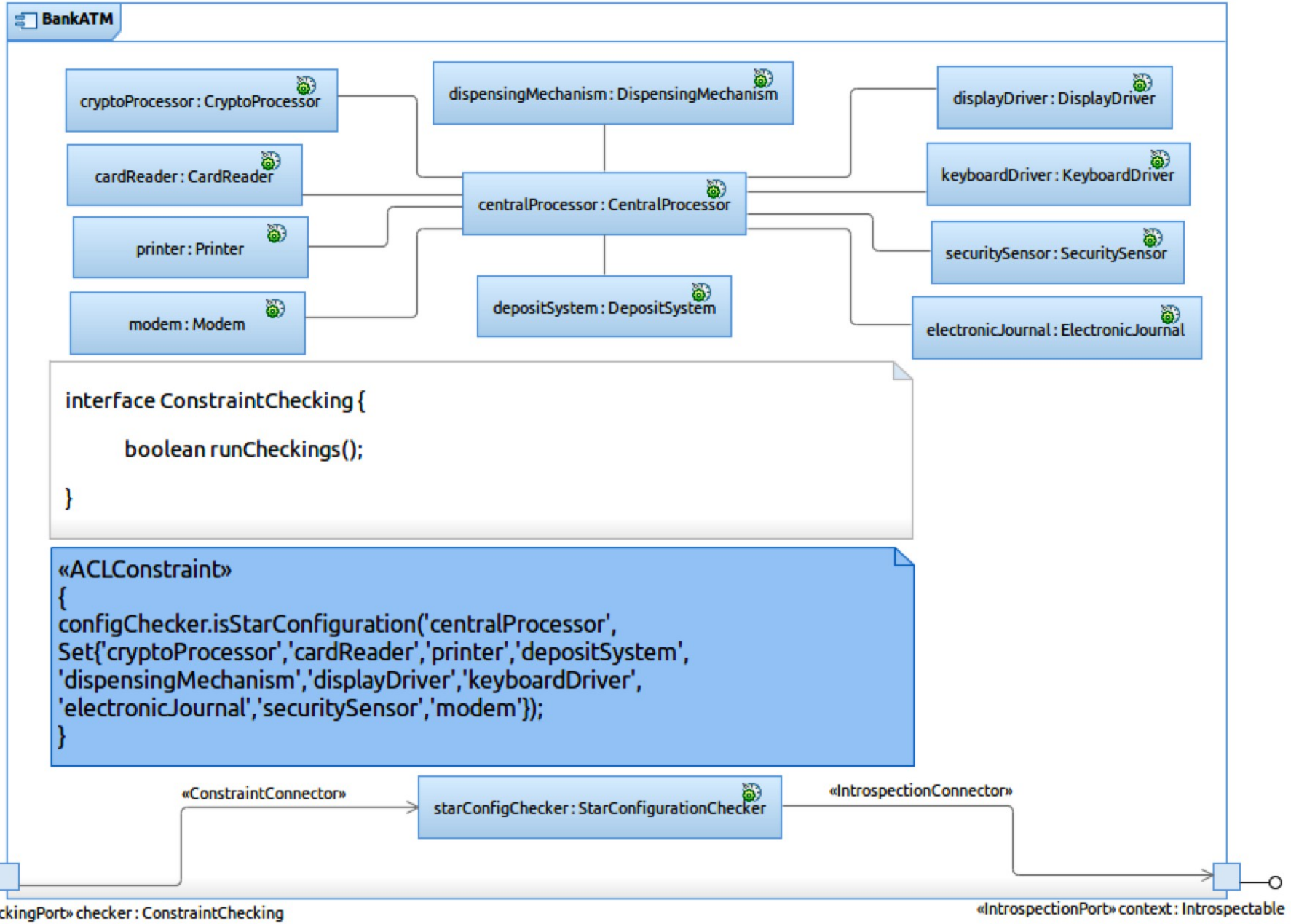
Exemple 1 : style d'architecture client/serveur



Exemple 2 : patron d'architecture en anneau



Exemple 3 : patron d'architecture en étoile

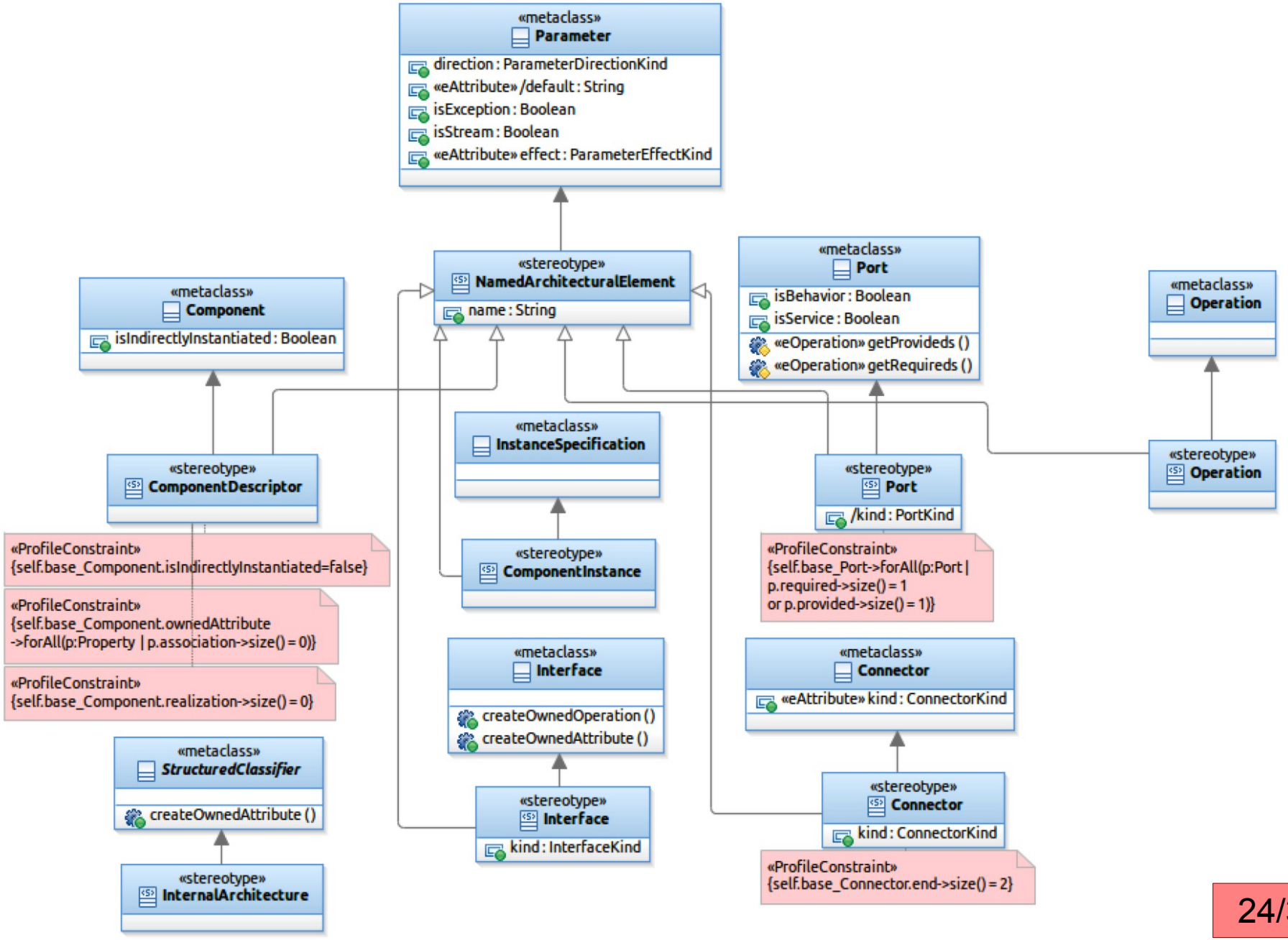


Implémentation des composants-contraintes

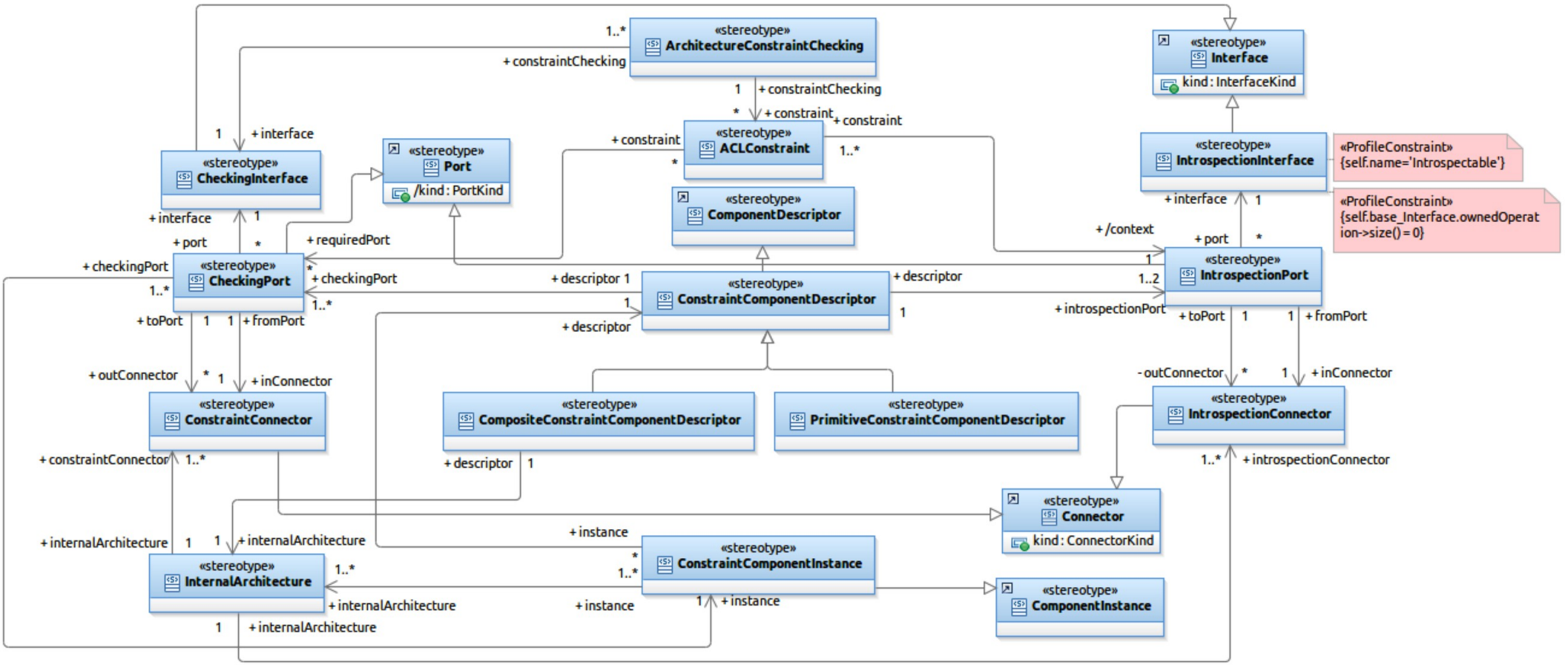
- Implémentations du méta-modèle des composants-contraintes :
  - Nouveau méta-modèle EMF/GEF sous Eclipse[1]
  - Profil UML 2.4 sous RSA (Rational Software Architect)
- Les contraintes sont évaluées sur la base du plugin OCL d'Eclipse
- Évaluation statique, à la conception, des contraintes architecturales

[1]C. Tibermacine et al. Component-based Specification of Software Architecture Constraints. Dans les actes du 14th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE'11), Boulder, Colorado, USA, June 2011. ACM Press.

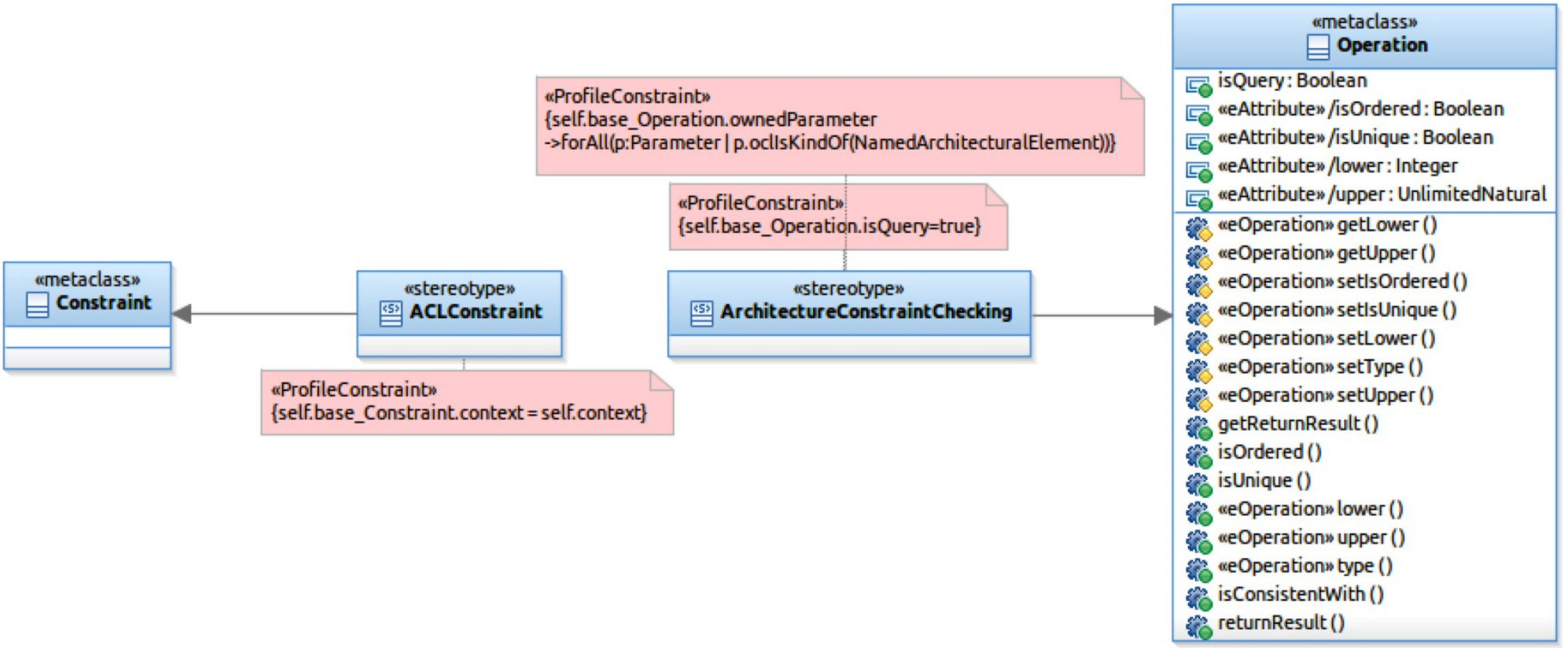
Profil UML pour les composants « métier »



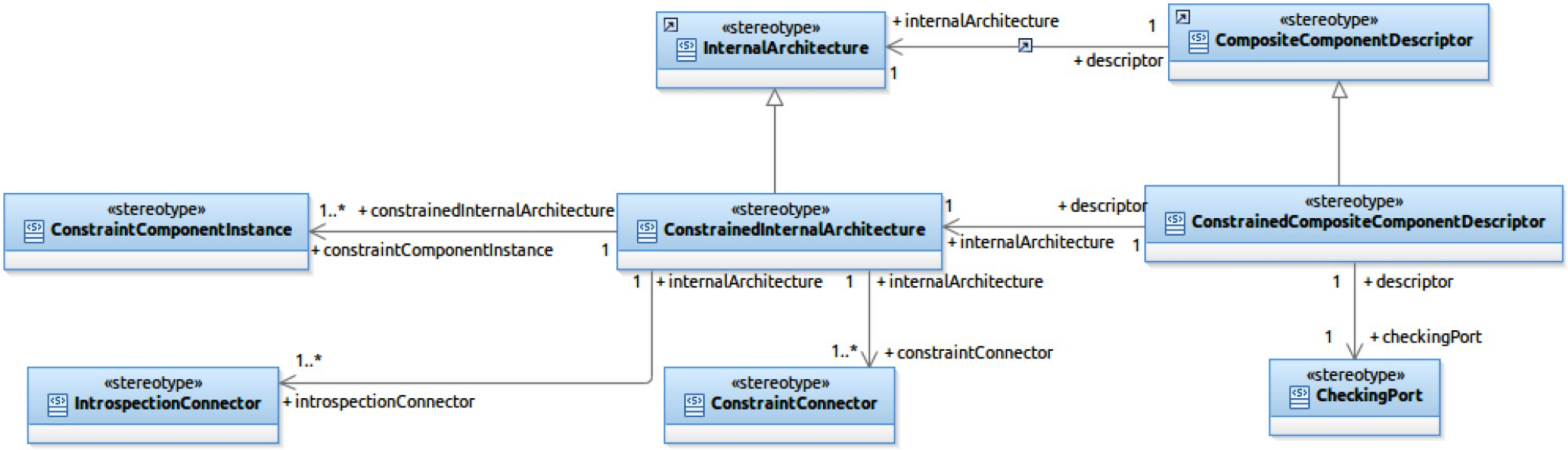
Profil UML pour les composants-contraintes



Profil UML pour les composants-contraintes -suite-



Profil UML pour les composants-contraintes -suite-



Conclusion

- Un modèle de composants pour la spécification des contraintes architecturales :
  - Composants = entités réutilisables
  - Composants = entités composables
  - Composants = entités personnalisables
- Implémentation de ce modèle de composants sous Eclipse et RSA

Travaux futurs

- Langages de programmation :
  - Compo : proto sous Smalltalk (thèse de Petr Spacek, en cours)
  - CoPoLa : framework Java (projet en cours)
- Ajout d'une couche réflexive à ces langages permettant de faire de l'introspection
- Implémentation du modèle de composants-contraintes dans ces langages
- Donner à ces composants-contraintes la capacité directe d'introspection/intercession

Questions