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

Plan de cette partie

- Notion de contrainte d'architecture
- Le langage ACL
- Contraintes sur les architectures à services Web

Plan de cette partie

- Notion de contrainte d'architecture
- Le langage ACL
- Contraintes sur les architectures à services Web

Contrainte d'architecture

- Une contrainte est une règle que doit respecter une description d'architecture

- Elle restreint :
  - le nombre d'éléments dans une architecture (composants, ports, connecteurs, ...)
  - la configuration de ces éléments (quel composant est connecté à quel autre composant, ...)
  - ...

- Nous nous limitons ici aux contraintes qui portent sur la structure (certaines descriptions d'architecture incluent une spécification comportementale : machines d'état, par exemple. Nous ne les traitons pas ici)

Contrainte d'architecture vs CSP

- Ne pas confondre contrainte d'architecture et la programmation par contraintes (avec CSP, par exemple)
- Ce ne sont pas des conditions qui doivent être respectées dans un problème d'optimisation : trouver la solution optimale dans un espace de recherche assez large
- Contraintes d'architecture : conditions évaluées pour vérifier si une « seule » solution (notre description d'architecture) satisfait les contraintes
- Si une contrainte d'architecture n'est pas vérifiée, ce n'est pas une nouvelle solution qui est recherchée (la description d'architecture doit être modifiée)

Langages existants

- La plupart des ADL suggèrent de documenter ces contraintes sous forme textuelle. Aucun moyen de les vérifier si la description d'architecture évolue

- Certains ADL proposent des langages pour les spécifier de façon formelle

- Parmi ces ADL, nous retrouvons : Acme (Armani) et Wright

Langage Armani[1]

- C'est le langage de contrainte proposé avec l'ADL Acme
- Il permet d'exprimer des prédicats dans la logique du premier ordre
- Deux sortes de contraintes :
  - Invariants (contraintes à respecter) :
    ```
    invariant Forall c1,c2: component in sys.Components |
      exists conn : connector in sys.Connectors |
      Attached(c1,conn) and Attached(c2,conn);
    ```
  - Heuristiques (contraintes qui peuvent être violées) :
    ```
    heuristic Size(Ports) <= 5;
    ```

[1] R. T. Monroe, "Capturing software architecture design expertise with Armani," Rap. tech School of Computer Science, Carnegie Mellon University, USA, 2001.

Langage Wright

- ADL permettant de spécifier de façon formelle la structure et le comportement d'une architecture logicielle, et particulièrement les connecteurs

- Son langage de contrainte permet d'exprimer des prédicats dans la logique du premier ordre

- Exemple :
  $\forall$ c : Components • Type(c) = Filter $\Rightarrow$ $\forall$ p : Ports(c);
  • Type(p) = Input $\vee$ Type(p) = Output

Langage FScript

- FScript est un langage de script pour Fractal ADL
- Il est basé sur un langage de navigation : FPath (~ xPath)
- Exemple :

```
-- Tests whether the client interface $itf is bound to
-- ( an interface of ) component $comp .
function bound-to (itf , comp ) {
  servers = $itf/binding::*/component::* ;
  return size($servers & $comp) > 0 ;
}
```

[1]Tutorial FScript : svn ://forge.objectweb.org/svnroot/fractal/tags/fscript-2.0

Limites de ces langages

- Manque d'expressivité :
  - Langages ad-hoc qui ne comportent que quelques structures syntaxiques pour exprimer des contraintes assez simples
  - Très difficile, voire impossible, d'écrire des contraintes complexes impliquant beaucoup de règles
- Dépendance du langage de contraintes avec le modèle de composants sur lequel s'appliquent les contraintes
  - Syntaxe du langage doit être adaptée si le modèle de composants évolue

Plan de cette partie

- Notion de contrainte d'architecture
- Le langage ACL
- Contraintes sur les architectures à services Web

Notre Langage : ACL

- ACL = Architecture Constraint Language[1]

- Langage basé sur un standard de l'OMG, le langage OCL : Object Constraint Language

- Langage ayant deux niveaux d'expressivité :
  - Prédicats dans la logique du premier ordre (CCL : Core Constraint Language)
  - Analyse de l'architecture effectuée à travers la navigation dans un méta-modèle de composants

[1]C. Tibermacine et al. A Family of Languages for Architecture Constraint Specification. In Journal of Systems and Software (JSS), Elsevier, 2010

Langage ACL

- ACL est en réalité une famille de langages :
  - Différents méta-modèles de composants : ADL, UML, frameworks de programmation par composants
  - Chaque membre de cette famille est appelé profil ACL :
    - Profil ACL pour xAcme : CCL + méta-modèle xAcme
    - Profil ACL pour UML 2 : CCL + méta-modèle de composants UML 2
    - Profil ACL pour CCM (composants CORBA) : CCL + méta-modèle CCM

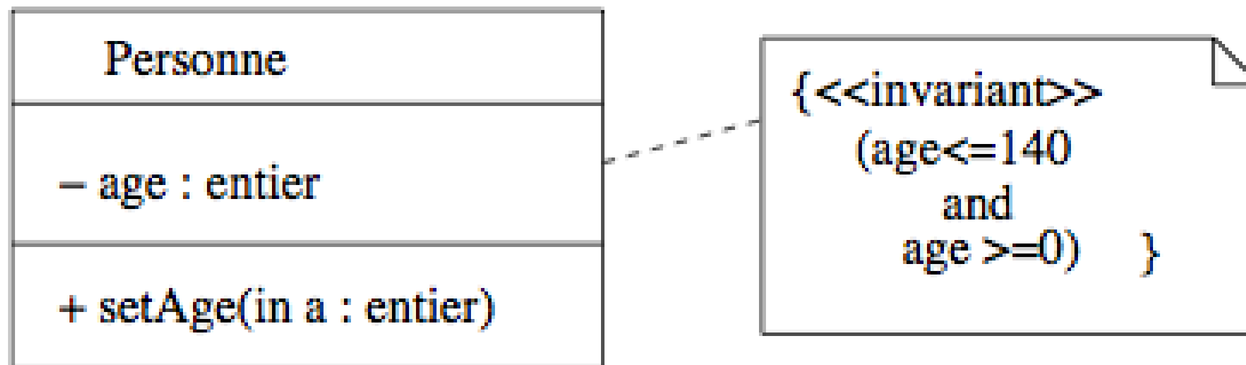
- Chaque profil peut être utilisé avec son modèle de composants

CCL = variante d'OCL : Object Constraint Language

- OCL est un standard de l'OMG, qui a été intégré à UML 1.1 en 1999, mais depuis, il possède une spécification qui lui est propre « dissociée de celle d'UML »
- Il permet d'exprimer ce que UML ne permet pas : enrichir la sémantique des diagrammes UML
- Il permet d'exprimer des règles de façon non ambigüe, qui vont restreindre les instances d'un élément de modélisation UML (expressions ayant une valeur booléenne)
- Cet élément de modélisation est dit le **contexte** de la contrainte
- Contraintes OCL possibles : invariants, pre- et post-conditions, body, init et derive (voir cours de M. Huchard)

Exemple de contrainte OCL

- Une classe Personne et un invariant :



- La contrainte sous forme textuelle :

-- l’âge est compris entre 0 et 140 ans (commentaire)
context Personne inv :
(age <= 140) and (age >=0)

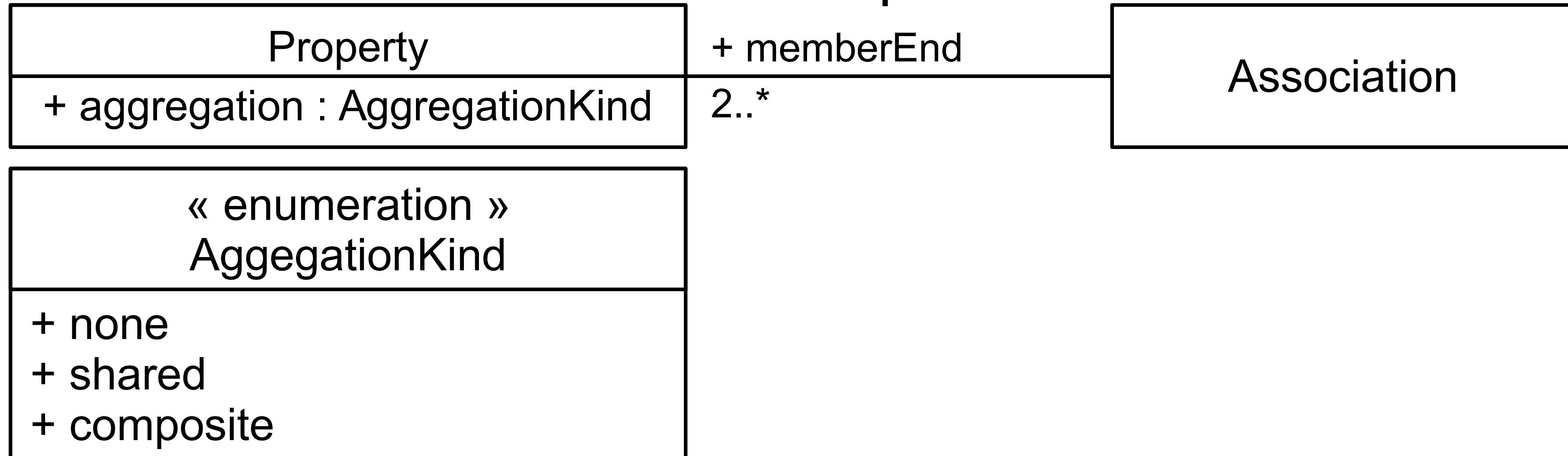
Contraintes dans le langage OCL

- La contrainte précédente s'applique sur toutes les instances de la classe Personne : tout objet Personne doit avoir pour son attribut age une valeur comprise entre 0 et 140

- Nous pouvons écrire des contraintes OCL, non pas au niveau modèle, mais au niveau méta-modèle

Contraintes OCL sur un méta-modèle

- Méta-classe Association dans la spécification UML 2.4 : Extrait



- Exemple : Contrainte extraite de la spécification UML 2.4

-- Only binary associations can be aggregations
context **Association** inv:
self.memberEnd->exists(aggregation <> Aggregation::none)
implies self.memberEnd->size() = 2

Introspection

Contraintes dans le langage ACL

- La contrainte précédente est définie au niveau du méta-modèle UML
- Elle va donc s'appliquer à tous les diagrammes de classes UML, comportant des associations
- Dans le cas des contraintes architecturales, on a besoin de ce mécanisme d'introspection (naviguer dans un méta-modèle), mais on voudrait appliquer les contraintes à un seul modèle (une seule description d'architecture)
- On a donc légèrement modifié le langage OCL

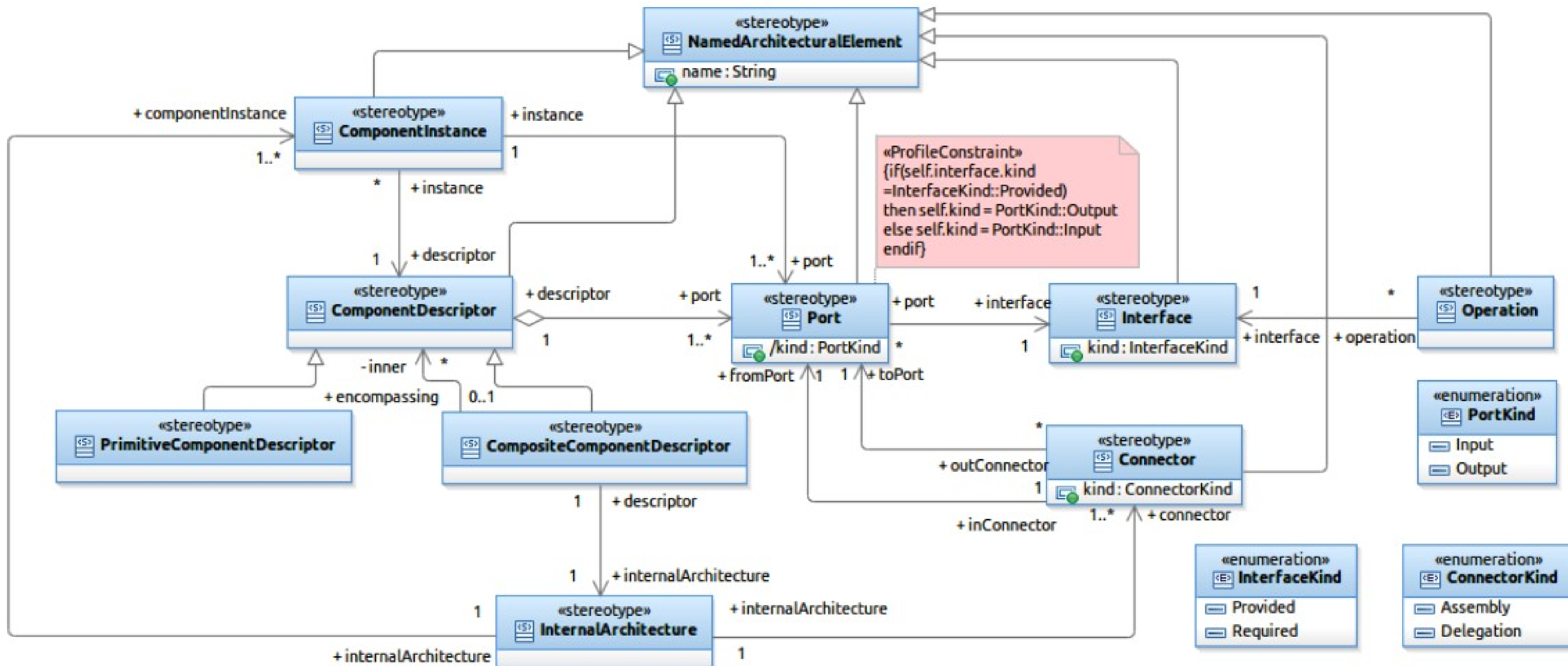
Langage CCL

- Seuls les invariants sont exprimables
- Dans la partie contexte, on identifie quel élément de l'architecture est concerné par la contrainte (sur lequel doit s'appliquer la contrainte)
- Exemple :

```
-- Pas plus de 5 ports pour le composant nommé «ACS»
context ACS:ComponentDescriptor inv:
ACS.port->size() < 5
```

- Pour faire de l'introspection (ACS.port->size()), la contrainte navigue dans un méta-modèle de composants

Méta-modèle de composants de notre ADL (CLACS)



Structures langagières nécessaires pour les contraintes

- Objectif voulu à travers les contraintes d'architecture :
  - Faire de l'introspection sur un descripteur d'architecture à base de composants et imposer des conditions booléennes

- Pour cela, il nous faut :
  - Utiliser la navigation entre méta-classes (et méta-attributs). Grâce à ça, on obtient des collections d'instances (ensemble de ports dans un descripteur, par exemple)
  - Utiliser des quantificateurs : universel (pour tout : ∀) et existentiel (il existe au moins un : ∃)
  - Appeler des opérations sur les collections (select, size, ...)
  - Exprimer des expressions de comparaison : < > <= >= = <>
  - Utiliser des opérateurs logiques : not and or xor implies et parfois des opérateurs arithmétiques : + - * /

Navigation dans le méta-modèle

- Naviguer à travers une association entre méta-classes ou vers les méta-attributs d'une méta-classe
  - Utiliser la notation du point (.)
  - Exemples :
    ACS.port.kind (l'ensemble des genres de ports du comp. ACS) ou ACS.internalArchitecture.instance (l'ensemble des instances de composants déclarées dans son architecture interne)
- Descendre à travers un lien d'héritage
  - Utiliser une conversion de type forcée (cast) : oclAsType(UneMetaClasse)
  - Exemple :
    unDescComp.oclAsType(CompositeComponentDescriptor) .internalArchitecture

Quantificateurs

- Les deux quantificateurs :
  - Pour tout (∀) :
    - Syntaxe : ->forAll(...)
    - Exemples : Tous les ports de ACS sont de type Output
      ACS.port->forAll(p : Port | p.kind = PortKind::Output)
      ou ACS.port->forAll(p | p.kind = PortKind::Output)
      ou ACS.port->forAll(kind = PortKind::Output)
  - Il existe au moins un (∃) :
    - Syntaxe : ->exists(...)
    - Exemples : Il existe au moins un composant dont le nom est « authenticator »
      ACS.internalArchitecture.instance
      ->exists(c : ComponentInstance | c.name='authenticator')

Types et types collection

- Les types qu'on utilise dans les contraintes d'architecture sont les suivants : Real, Integer, String et Boolean
- En plus de ces types, nous avons les méta-classes de notre méta-modèle : Port, Interface, ...
- Nous avons quatre types de collections :
  - Set : Ensemble au sens mathématique (pas de doublons)
  - OrderedSet : Ensemble ordonné
  - Bag : Famille au sens mathématique (existence de doublons)
  - Sequence : Famille ordonnée

Quelques opérations sur les collections

- L'appel de ces opérations est précédé d'une flèche : ->
- Calcul du nombre d'éléments :
  - size(), isEmpty(), notEmpty()
  - count(unObjet)

- Opérations avec itérateurs :
  - includes(unObjet), excludes(unObjet) : retournent un booléen
  - includesAll(uneCollection), excludesAll(uneCollection) : booléen
  - including(unObjet), excluding(unObjet) : une collection avec/sans unObjet
  - select(uneCondition), reject(uneCondition), collect(uneCondition) : une collection
  - one(uneCondition) : booléen

Quelques opérations sur les collections - suite -

- Autres opérations (certaines sont propres à certains types) :
  - union(uneCollection), intersection(uneCollection) : une collection
  - Conversion de type : asSet(), asSequence(), ...
  - Obtention d'une instance à une certaine position : at(unEntier), first(), last(), ...
  - ... (voir la spécification OCL)

Expression « let »

- Parfois une sous-expression se répète dans une contrainte
- On peut assigner sa valeur à une variable grâce à une expression « let »
- Syntaxe :

let uneVar : SonType = saValeur (la sous-expression qui se répète)
in
laContrainte

- Exemple :

let layers : Set(ComponentInstance) = ACS.internalArchitecture
.instance->select(name='authenticator' or name='accessControler'
or name='dataLogging' or name='remoteArchival')
in ...

Exemples de contraintes architecturales

- Styles d'architecture :
  - Style d'architecture en couches
  - Style d'architecture « Pipe & Filter »
  - Style d'architecture « Pipeline »
  - Style d'architecture « client/serveur »
  - Style d'architecture « 3-tiers »
- Patrons d'architecture :
  - Patron Façade
  - Patron des composants répliqués
  - Patron d'architecture en bus
  - Patron d'architecture en étoile
  - Patron d'architecture en anneau
  - Patron d'architecture MVC (Model-View-Controller)
- Plusieurs variantes de chaque cas
  - Exemple : une couche = un composant ou un {composants}

Style d'architecture en couches

- Contraintes textuelles :
  - Chaque couche doit être connectée
  - Chaque couche doit être connectée à au plus deux autres couches
  - Le nombre de couches qui sont connectées à une seule autre couche est égal à 2 (les couches supérieure et inférieure)

Style d'architecture en couches -suite-

- Contraintes ACL :

```
-- Chaque couche doit être connectée
self.internalArchitecture.instance->forAll(c: ComponentInstance |
  c.port.inConnector->union(c.port.outConnector)->notEmpty()
) and
-- Chaque couche doit être connectée à au plus deux autres
-- couches
self.internalArchitecture.instance->forAll(c: ComponentInstance |
  c.port.inConnector.toPort.instance->union(c.port.outConnector
  .fromPort.instance)->asSet()->size() <= 2
) and
-- Le nombre de couches qui sont connectées à une seule autre
-- couche est égal à 2 (les couches supérieure et inférieure)
...
```

Patron d'architecture en bus

- Contraintes textuelles :
  - Le bus doit avoir au moins un port Input et un port Output
  - Les composants clients ne doivent avoir que des ports Output
  - Les composants clients ne doivent être connectés qu'au bus
  - Les composants fournisseurs ne doivent avoir que des ports Input
  - Les composants fournisseurs ne doivent être connectés qu'au bus

Patron d'architecture en bus -suite-

- Contraintes ACL :

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 ...

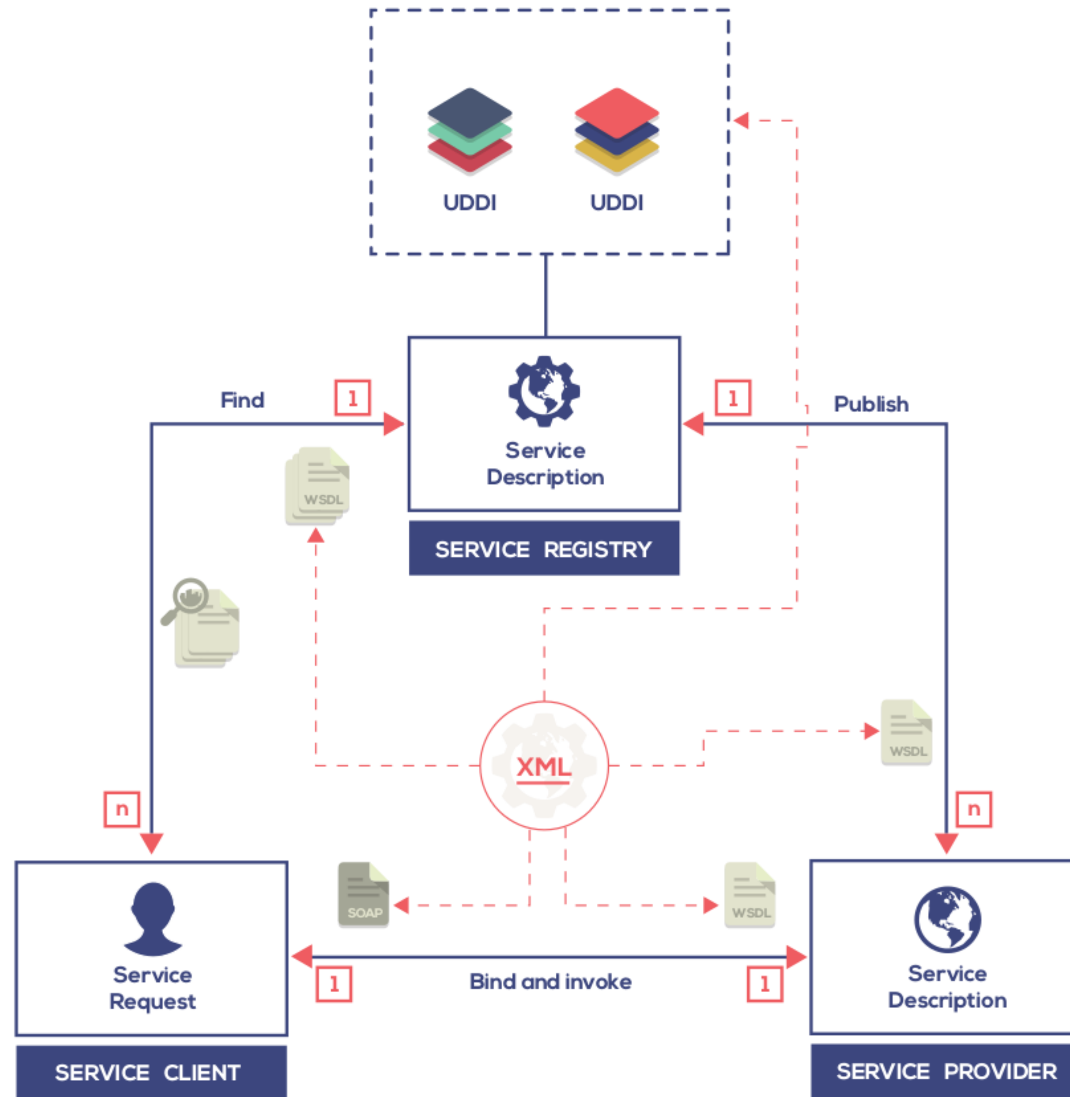
Plan de cette partie

- Notion de contrainte d'architecture
- Le langage ACL
- Contraintes sur les architectures à services Web

Qu'est-ce qu'un service Web ?

- Un composant logiciel identifié par un URI (Universal Resource Identifier) sur Internet
- Son interface (ensemble d'opérations) et ses bindings sont décrits avec des langages/protocoles standards : XML, WSDL, SOAP, HTTP, ...
- Sa description d'interface peut être « découvert » par d'autres systèmes en utilisant des moteurs de recherche et des annuaires, entre autres
- Les systèmes intéragissent avec le service Web en invoquant à distance ses opérations et en lui passant des messages SOAP/XML (ou de simples messages HTTP, pour les services REST) sur Internet

Modèle de référence



Compositions de services Web

- Une composition de services Web est construite en réutilisant des services Web existants
- Il existe deux sortes de compositions de services :
  - Chorégraphies de services : une collaboration entre des services Web pairs (ex de langages : BPMN)
  - Orchestrations de services : un workflow qui invoque les opérations de services Web (ex de langages : BPEL)

Langage BPEL

- Prononcé Bipeul
- *Business Process Execution Language*
- Standard du consortium OASIS (IBM, Microsoft, SAP, Intel, ...) depuis 2007
- Une orchestration = un processus avec des activités dans lesquelles nous pouvons invoquer des opérations de services Web
- Une orchestration = un programme exécutable grâce aux moteurs BPEL (des plugins Netbeans et Eclipse existent)

Contraintes sur les orchestrations de services

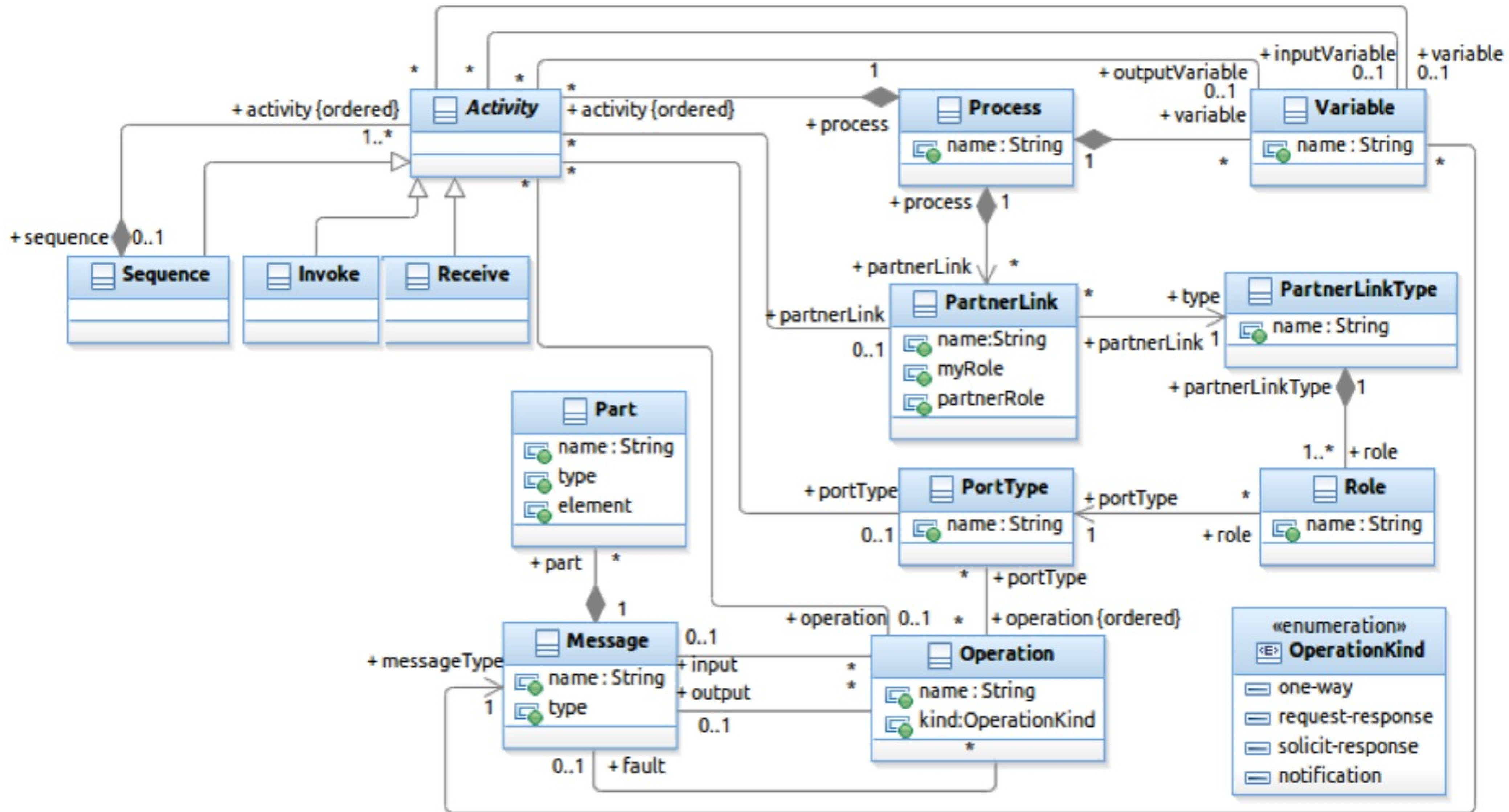
- Contraintes d'architecture vérifiables sur des processus BPEL

- On les spécifie avec un profil ACL pour BPEL
  - CCL
  - Méta-modèle de BPEL

- On les utilise[1] pour vérifier lors de l'instanciation de patrons SOA[2] dans une orchestration, ou lors de son évolution si les patrons précédents sont toujours respectés

[1]T. Zernadji, C. Tibermacine, F. Cherif and A. Zouioueche. Integrating Quality Requirements in Engineering Web Service Orchestrations. In JSS Nov. 2015
[2]Thomas Erl, SOA Design Patterns, Prentice Hall, 2009.

Méta-modèle du langage BPEL



Exemple de contrainte

- Patron SOA : Passive Replication
- Contraintes :
  - Le service à répliquer doit être encapsulé dans une activité BPEL Scope pour permettre de capturer les éventuelles erreurs dans une activité BPEL FaultHandlers
  - Dans les catchs attachés au Scope, un seul Reply doit exister (réponse en cas de défaillance du service)
  - Il existe plusieurs Invoke dans le scope, qui représentent les réplicats du service à fiabiliser
  - ...

Exemple de contrainte : formalisation

```
context TRS: Process inv:
let scp :Set(Activity)= self->closure(oclAsType(Activity))->select(a:Activity|
 a.oclAsType(Scope).name='aScope') in
--The service to be replicated should be wrapped in a 'Scope' activity
 scp.oclAsType(Scope).activity->exists(b:Activity|
 b.oclAsType(Invoke).name='serviceTobeReplicated')
and
let cth :OrderedSet(Activity)=
 scp->closure(oclAsType(Activity))->select(c:Activity|
c.oclIsKindOf(Catch))->asOrderedSet() in
 let rep: Set(Activity)=
 cth->closure(oclAsType(Activity))->select(c:Activity|c.oclIsKindOf(Reply)) in
-- In all the 'Catch' elements attached to the 'Scope' it should exist only one 'Reply'.
-- This latter represents the fault response case (if any) of all the replicated services
-- and should be in the last 'Catch' element
 rep.oclAsType(Reply)->size()=1 and cth->last()->exists(c:Activity|c.oclIsKindOf(Reply))
and
let ink: Set(Activity)=
 cth->closure(oclAsType(Activity))->select(c:Activity|c.oclIsKindOf(Invoke)) in
```

Exemple de contrainte : formalisation -suite-

```
--The number of 'Invoke' activities equals the one of 'Catch' activities minus one.
-- The last 'Catch' intercepts the failure case of the last replicated service if any.
 ink.oclAsType(Invoke)->size()>=1 and ink.oclAsType(Invoke)->size()=
cth.oclAsType(Catch)->size()-1
and
 let fhandlers :OrderedSet(Activity)=
 scp->closure(oclAsType(Activity))->select(c:Activity| c.oclIsKindOf(FaultHandler))
->asOrderedSet() in
--The service invocations are organized in a hierarchical way
 if fhandlers->size() > 1 then
 fhandlers->excluding(fhandlers->last())->forAll(aa,bb:Activity|
aa.oclIsKindOf(FaultHandler) and
 aa->exists(bb.oclIsKindOf(FaultHandler))) else false endif
```

Questions