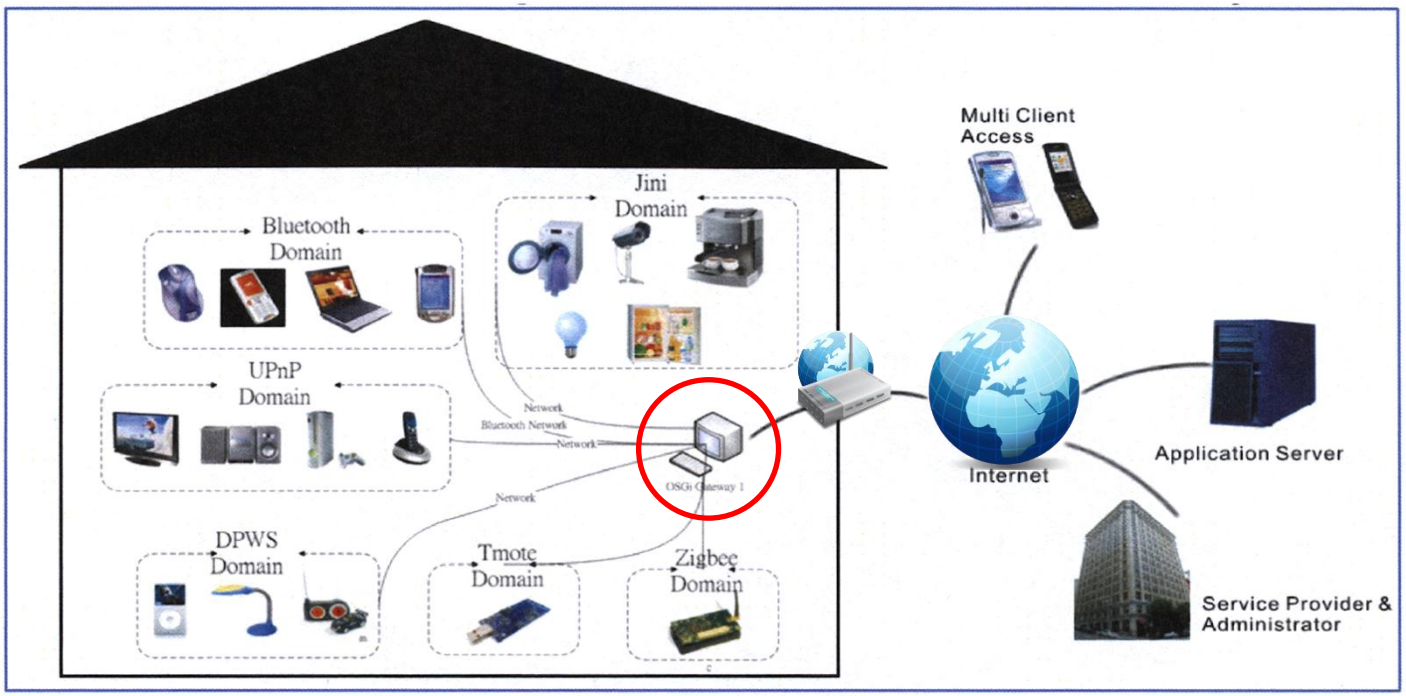
Rose et APAM
Concepts de Base

Germàn Vega, Jacky Estublier

UNIVERSITÉ DE GRENOBLE

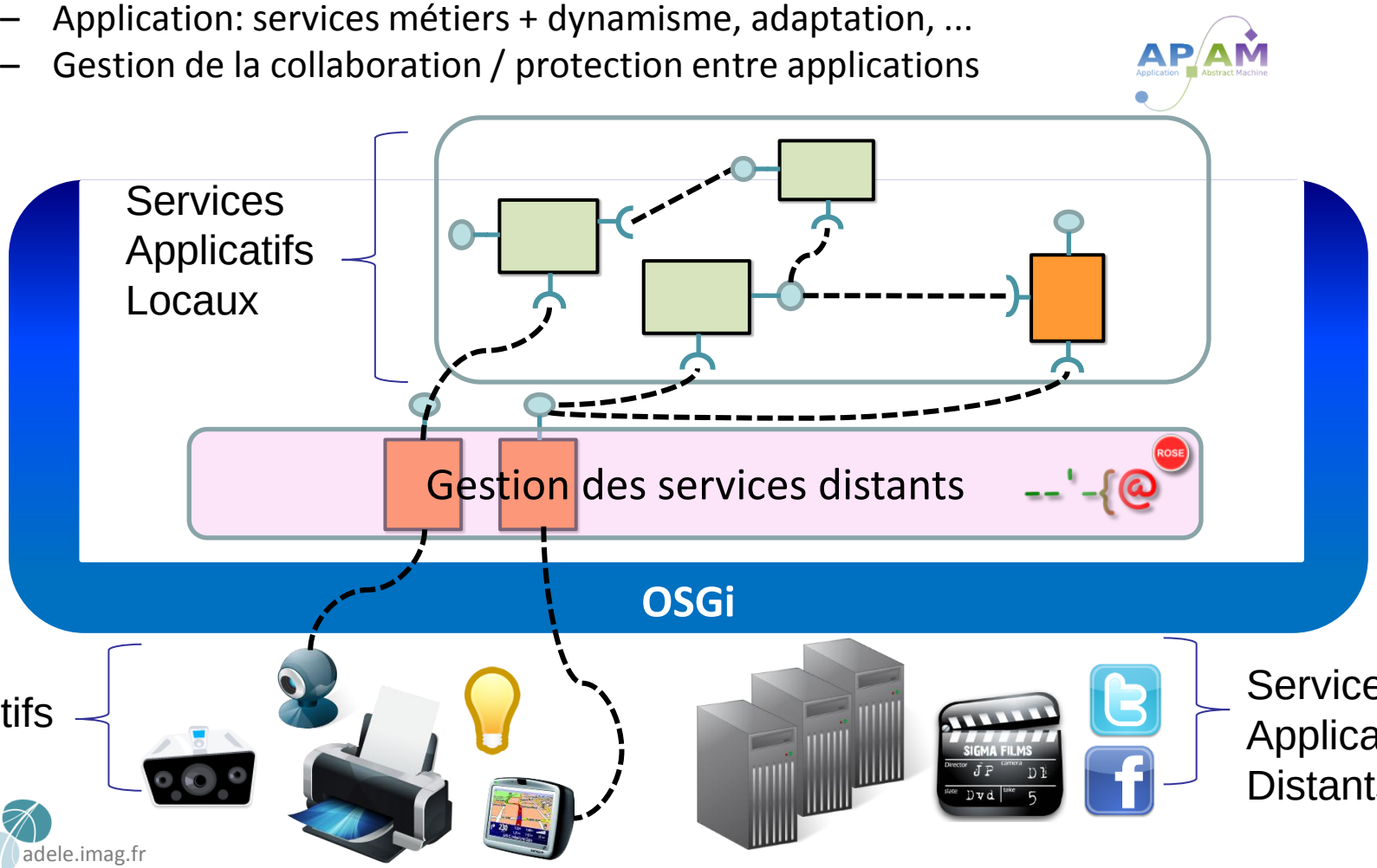
La vision d'origine

- La passerelle au centre des échanges entre le réseau de dispositifs et les services applicatifs.
- Le point d'entrée pour la gestion des dispositifs.
- Hébergement de services locaux.



Approche globale

- Faciliter la programmation
- Séparation des diverses préoccupations.
  - Dispositifs, services distants: hétérogénéité, distribution, découverte, …
  - Application: services métiers + dynamisme, adaptation, ...
  - Gestion de la collaboration / protection entre applications

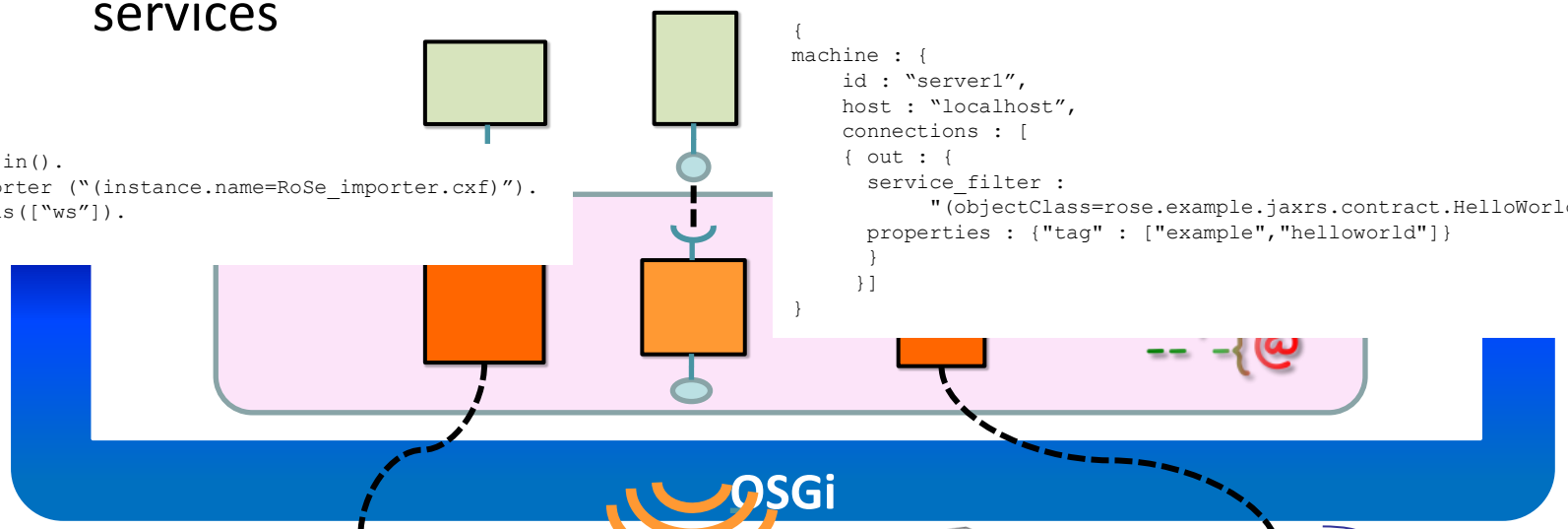


RoSe : faciliter la programmation

- Transparence:
  - Création dynamique de stub/skeleton
  - Vue unifié des dispositifs
- Découverte/Publication dynamique
  - Refléter la disponibilité des dispositifs dans la plate-forme à services

```
machine.in().
withImporter ("(instance.name=RoSe_importer.cxf)").
protocols(["ws"]).
add();
```

```
{
machine : {
    id : "server1",
    host : "localhost",
    connections : [
    { out : {
      service_filter :
           "(objectClass=rose.example.jaxrs.contract.HelloWorld)",
      properties : {"tag" : ["example","helloworld"]}
      }
     }]
}
```



Dispositifs

Services Applicatifs Distants

RoSe : protocoles actuellement supportés

Protocole/Standard	Style	Librairie	Import	Export
JSON-RPC	RPC	Jabsorb.org	✓	✓
JAX-WS	SOAP	Apache CXF	✓	✓
JAX-RS	REST	Jersey		✓
XML-RPC	RPC	Apache XML-RPC	✓	✓

CE N'EST PAS un bridge entre protocoles

		rte
UPnP	✓	✓
DPWS	✓	✓
Dns-sd	✓	✓
Modbus		✓
Comet-d	✓	✓
Zookeeper	✓	✓
Pubsubhubbub	✓	✓

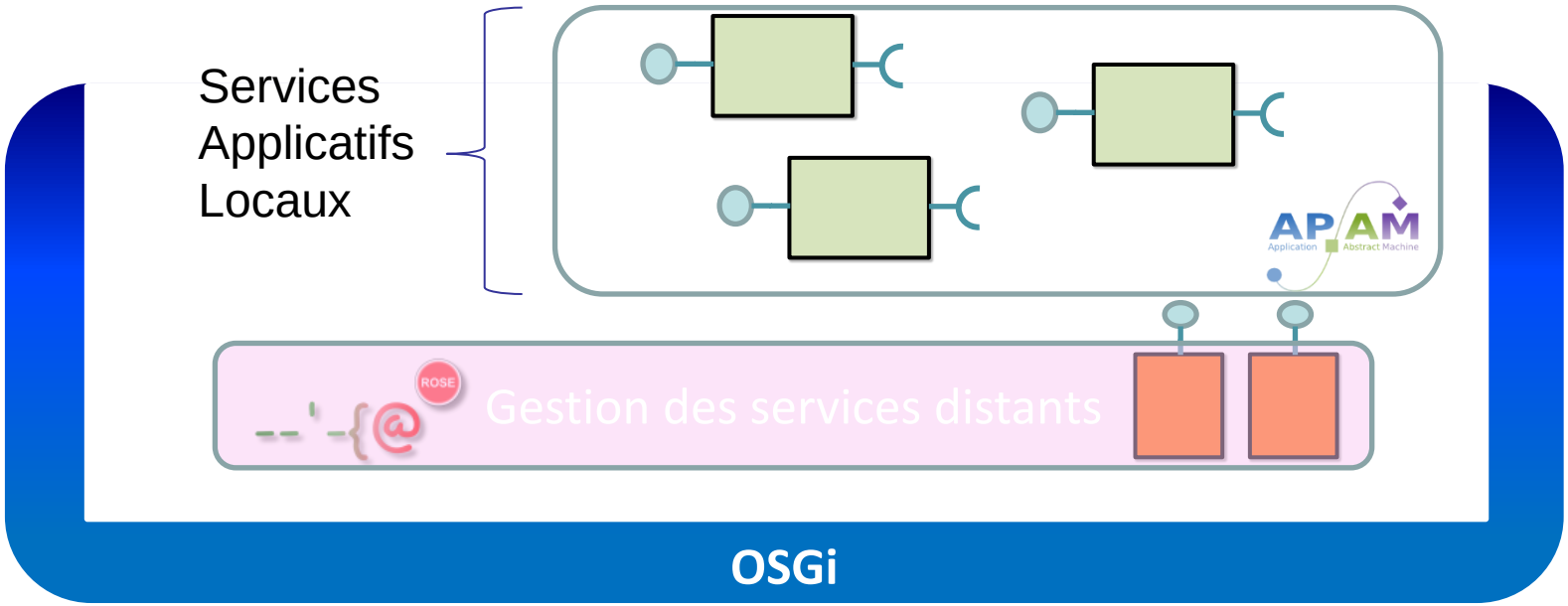
RoSe

- ✓ Focalisé sur la distribution
- ✓ Vision unifiée en termes de services
  - Découverte/disparition dispositifs reflétée en termes de disponibilité de services
- ✓ Multi-protocole
- ✓ Extensible

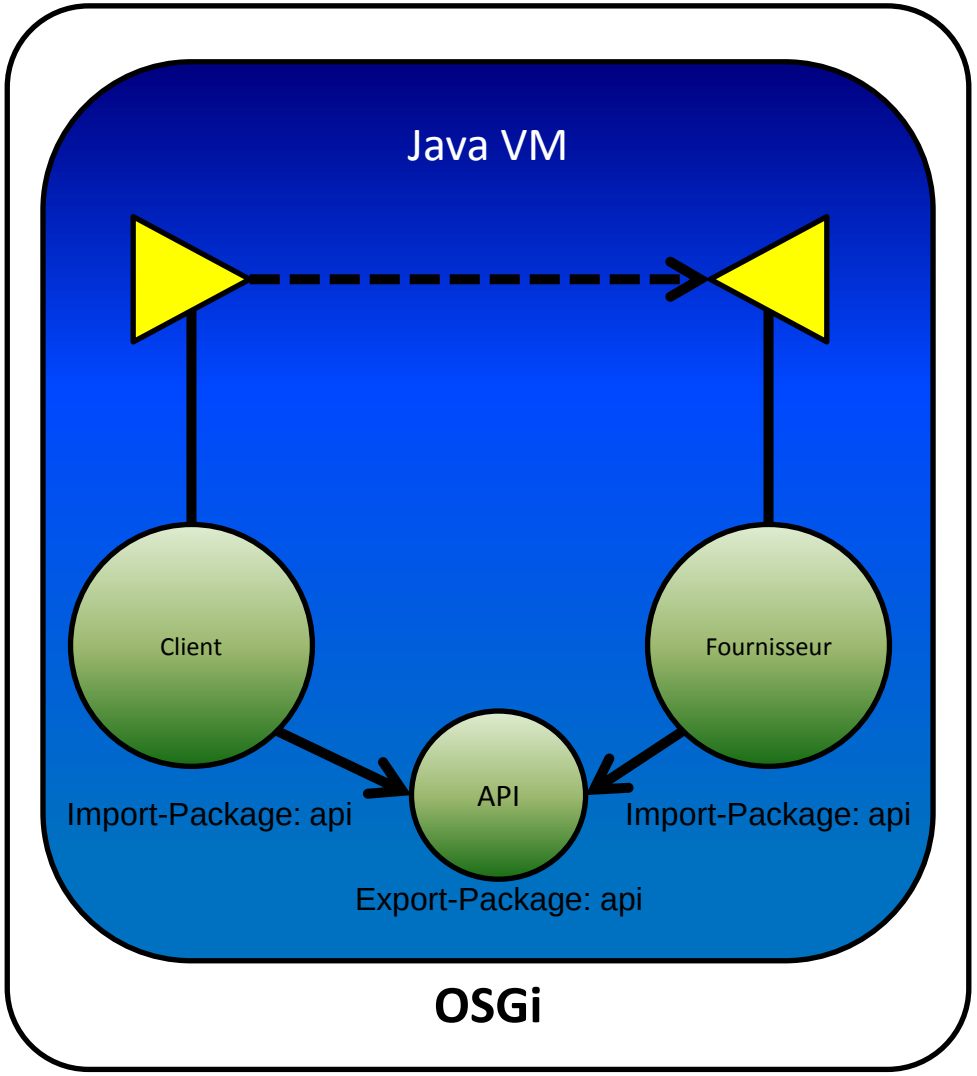
- ✗ Bas de niveau d'abstraction
  - comment structurer l'application?
  - comment réagir aux évènements dynamiques?
- ✗ Java-Only

Gestion du dynamisme

- Dynamisme des dispositifs et services distants
- Dynamisme lié aux mises à jour
- Impact sur l'application?



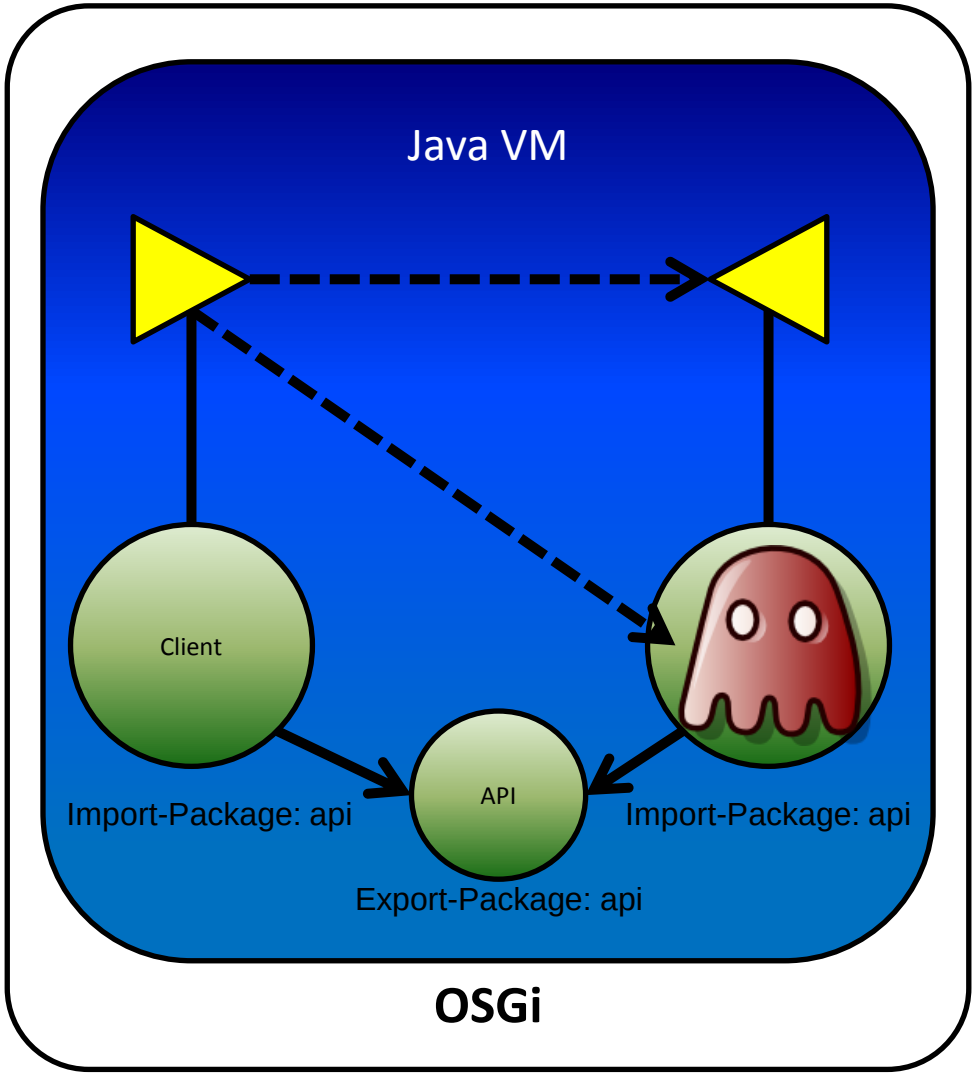
Gestion du dynamisme : SOA



Utiliser les patrons SOA pour gérer l'interaction entre modules

- ✓ Séparer le contrat de service de son implémentation
- ✓ Le fournisseur publie le service dans l'annuaire
- ✓ Le client cherche le service dans l'annuaire (requête avec filtre)
- ✓ Invocation dynamique

SOA dynamique: programmation complexe

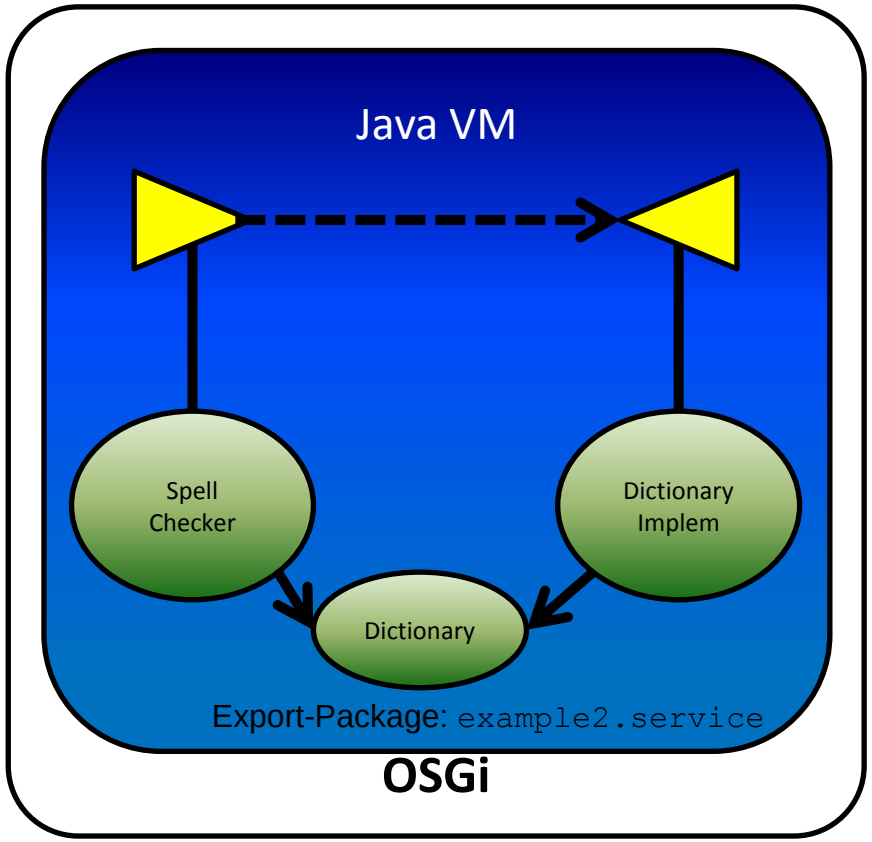


- ✘ Le client est responsable de faire la demande de service (requête) et d'attendre une notification
- ✘ Le fournisseur est responsable de la publication du service
- ✓ La plate-forme notifie les clients intéressés
- ✓ La plate-forme efface l'enregistrement et notifie les clients
- ✘ Responsabilité du programmeur de libérer toutes les références à l'ancienne version

SOA dynamique: Les pièges

- Déchargement des classes
- Dépendances cachées
- Il ne suffit pas d'avoir les primitives « bind » et « unbind »
- Programmation par évènements et multi-thread, même si l'application n'a aucun parallelisme
- Mélange de préoccupations (code métier vs gestion du dynamisme) que difficulté la compréhension

iPOJO : simplifier la programmation



- Définir le contrat de service

```
package example2.service;

public interface DictionaryService {
  public boolean checkWord(String word);
}
```

- Faire la requête pour attendre les notifications de service

```
package example2;
import example2.service;

public class SpellCheckerImpl  {
  private DictionaryService[] dictionaries;

  public String[] check(String passage) {
  for(DictionaryService dictionary:dictionaries) {
    if (! dictionary.checkWord(word))
      System.out.println("error in "+ passage);
  }}
}
```

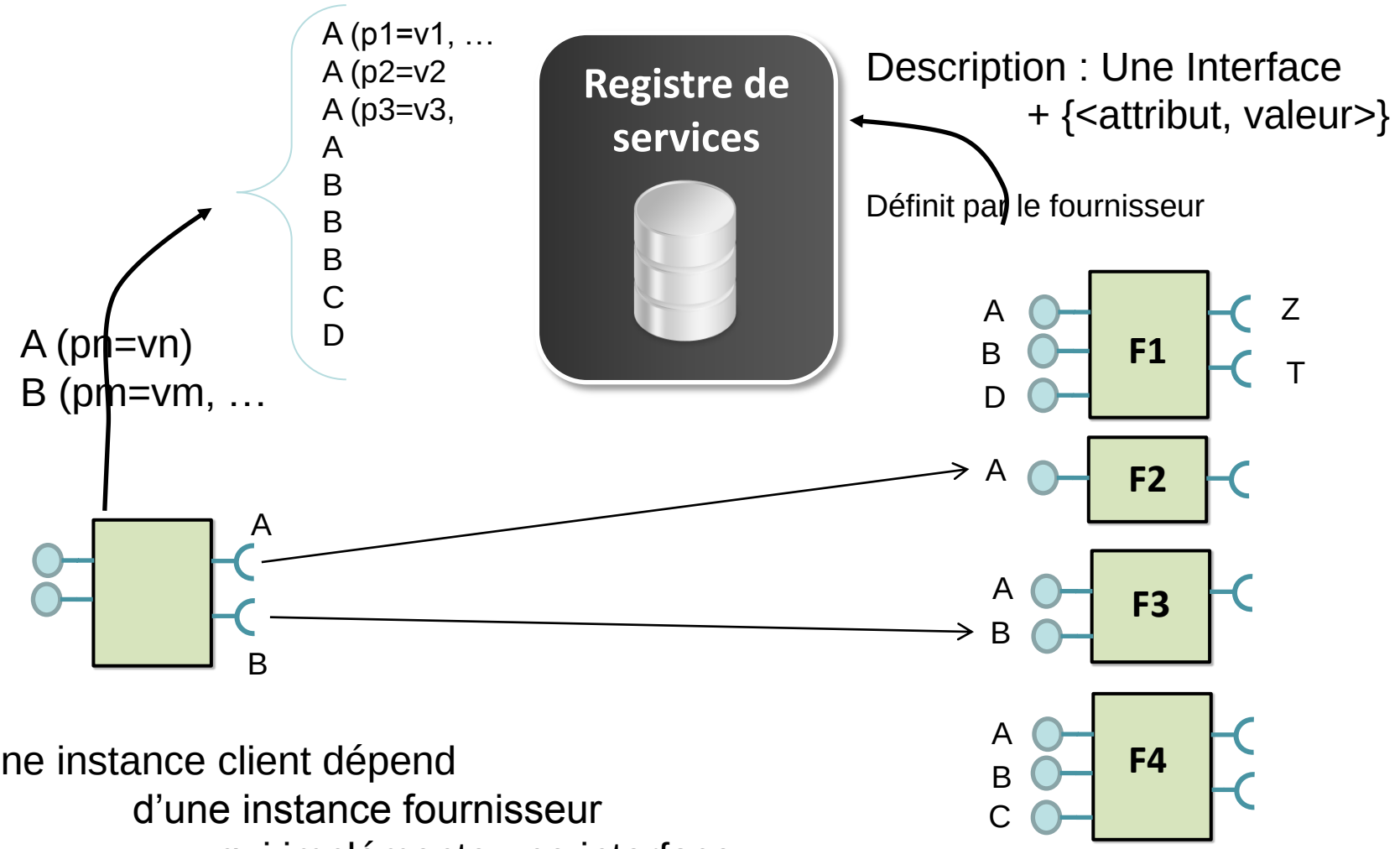
ipojo.xml

```
<ipojo>
<component name="example2.SpellCheckerImpl">
     <require field="m_dictionaries"
              filter="(Language=*)"/>
 </component>
</ipojo>
```

apam.xml

```
<apam>
<implementation name="example2.SpellCheckerImpl">
     <dependency field="m_dictionaries"  >
          <constraints>
              <instance filter="(Language=*)"/>
          </constraints>
      </dependency>
 </implementation>
</apam>
```

Gestion des Dépendances Dynamiques: Les plates-formes à Service



Une instance client dépend
d'une instance fournisseur
qui implémente une interface

Gestion des Dépendances Dynamiques: Challenges

- Automatiser et Simplifier

  => Injection de code

- Fiabiliser la sélection

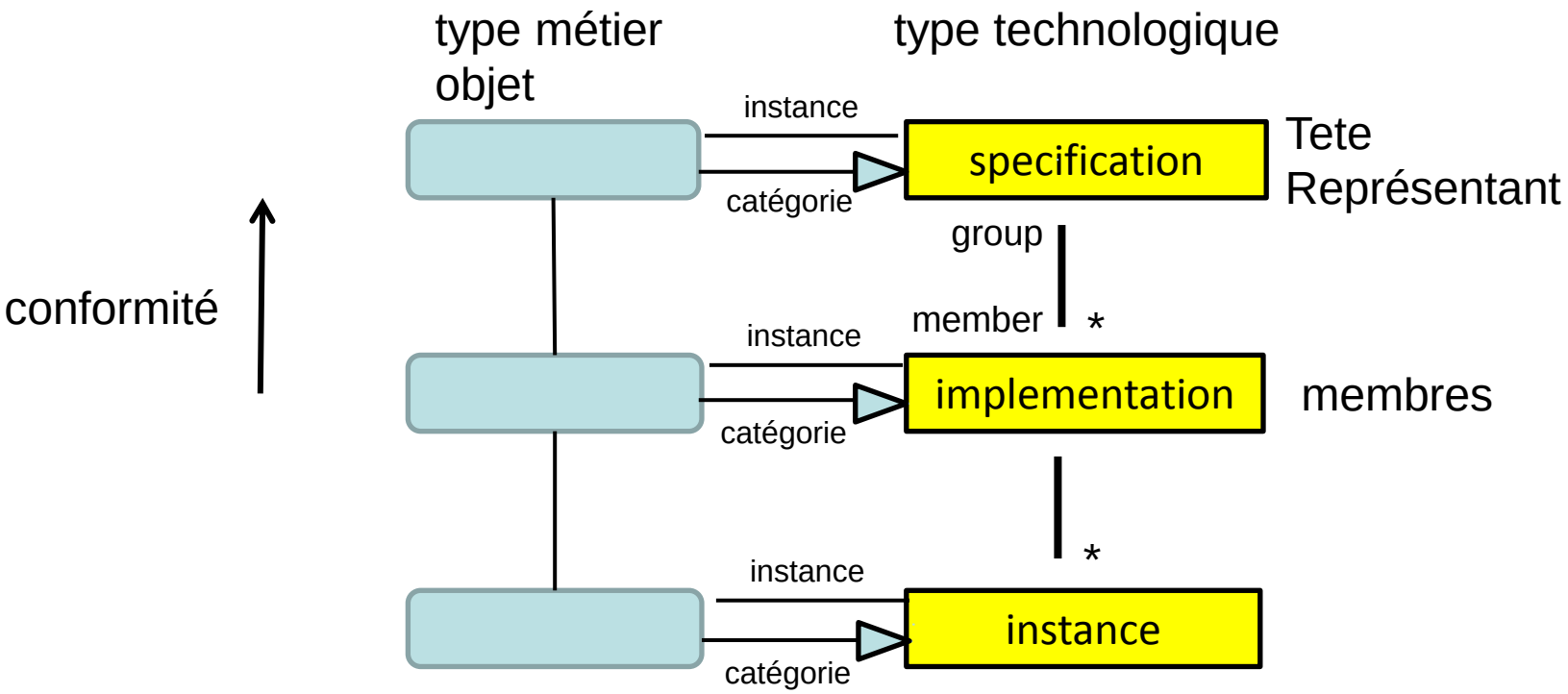
  => Ensembles d'équivalence et typage fort

- Donner le plus grand espace de choix possible.

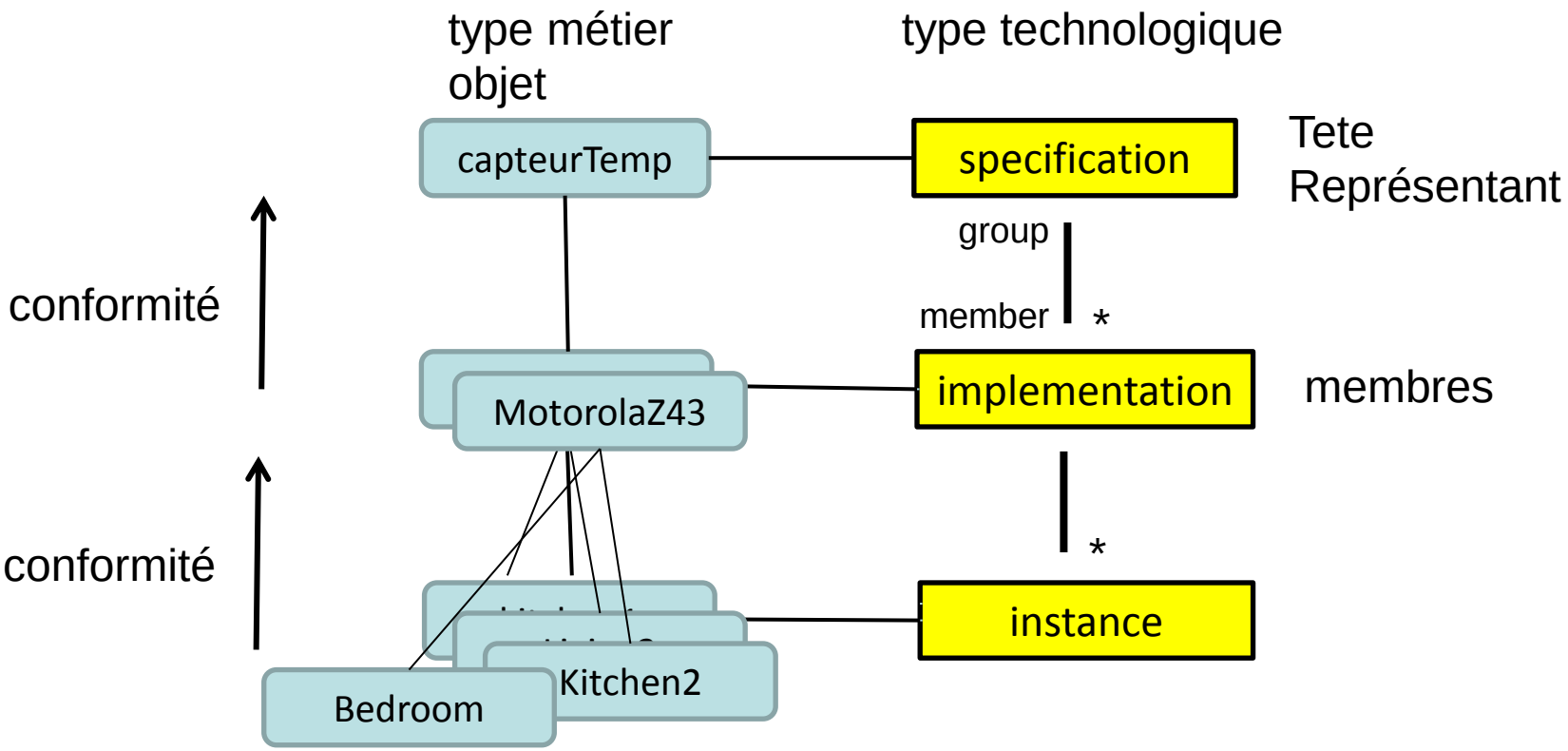
  => Espaces de sélection multiples (local, distant …)

- Donner à l'administrateur / architecte les moyens de définir les stratégies de gestion du dynamisme

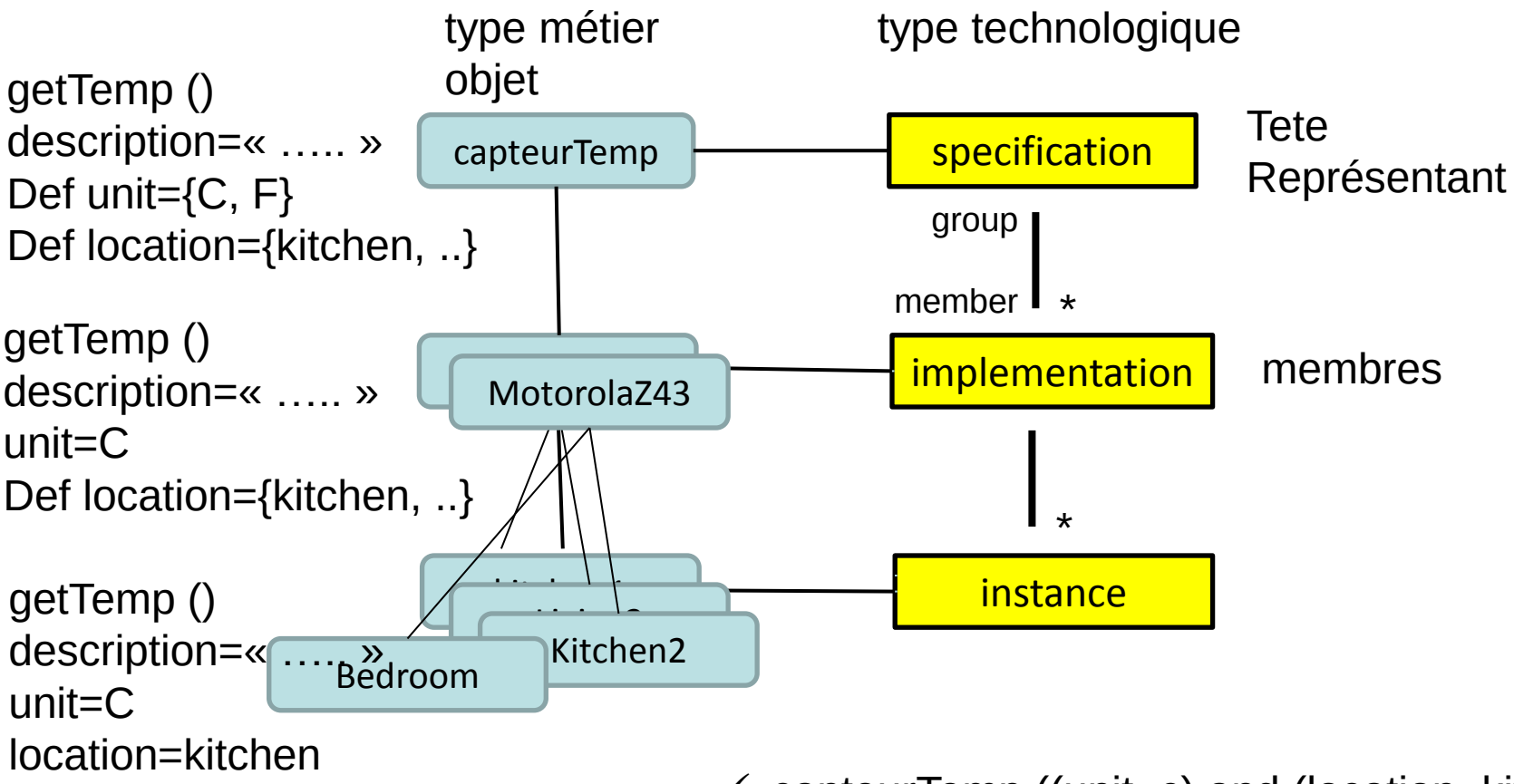
Groupes : généralisation de la matérialisation et power types



Groupes

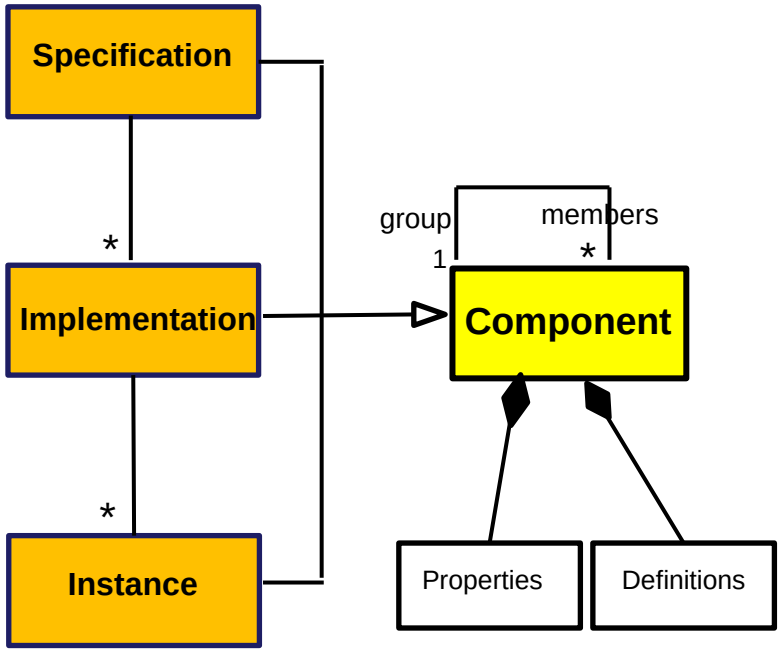

type métier
objet
type technologique
capteurTemp
specification
Tete
Représentant
group
member
*
conformité
MotorolaZ43
implementation
membres
conformité
*
instance
Kitchen2
Bedroom

Groupes: La Conformité



✓ capteurTemp ((unit=c) and (location=kitchen))

✘ capteurTemp ((unit=celsius) and (piece=kitchen))


Specification
Implementation
Instance
Component
group
members
1
*
*
*
Properties
Definitions

```
<specification name="CapteurTemp"
         interfaces="{apam.demo.CapteurTemp}" >
         <property name="description" value="Un capteur de température …"
                  type="string"/>
         <definition name="unit" type="{C, F}" value="F" />
         <definition name="location" type="{living, kitchen, bedroom}" />
         <definition name="OS" type="{Linux, Windows, Android, IOS}" />
</specification>

<implementation name="MotorolaZ43" specification="CapteurTemp"
         classname="apam.demo.MotorolaZ43" >
         <property name="unit" value="C" />
         <property name="OS" value="Linux, Android" />
         <definition name="rate" type="{high, low, medium}" />
</implementation>

<instance name="Kitchen1" implementation="MotorolaZ43" >
         <property name="location" value= "kitchen" />
         <property name="rate" value="high" />
</instance>
```

Vérification des contraintes dans les dépendances

```
<specification name="test" >
    <dependency specification="CapteurTemp" >
        <constraints>
            <implementation filter="(&amp;(unit=celsius)(piece=kitchen))" />
        </constraints>
    </dependency>
</specification>
```

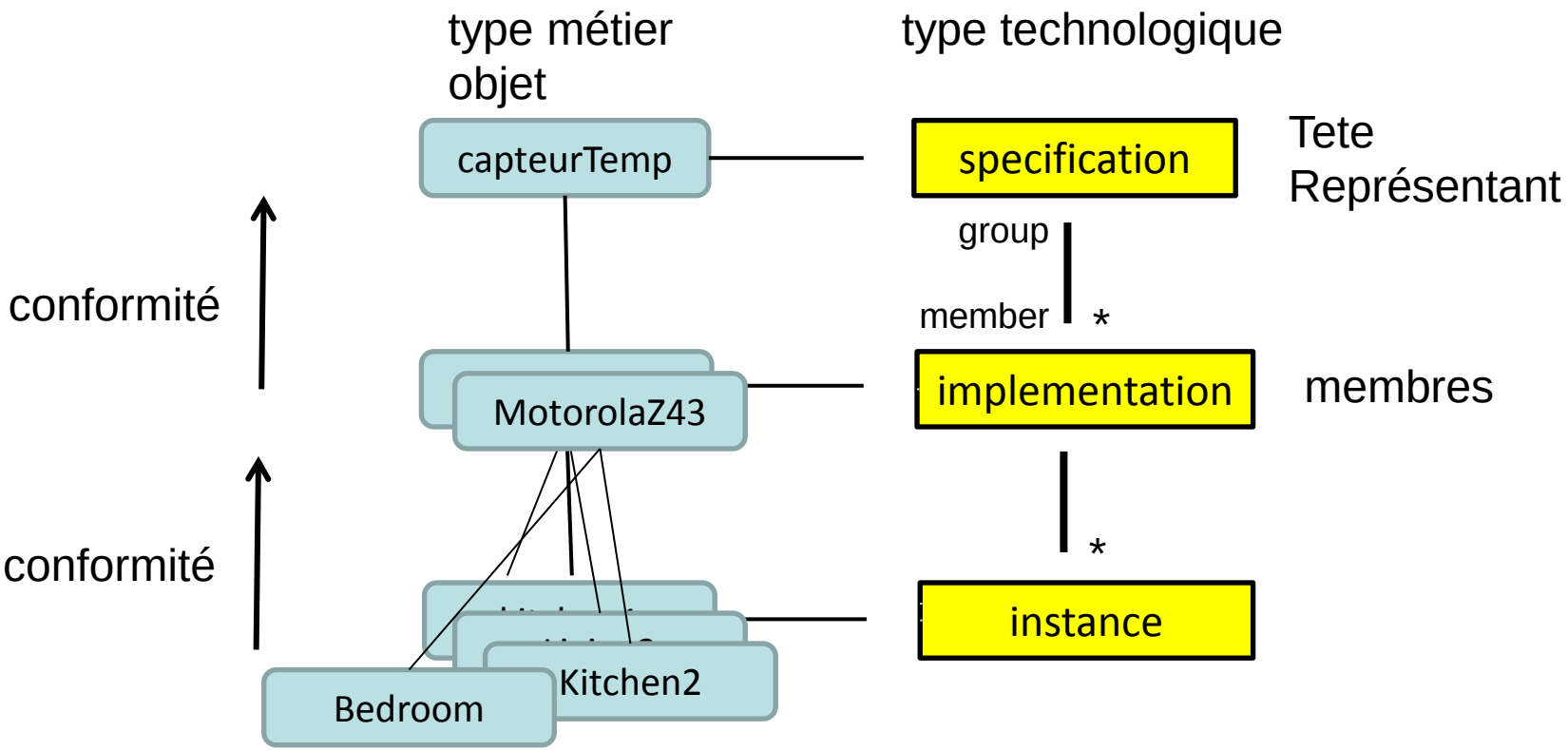
```
Checking specification test ...
ERROR - Invalid attribute value(s) "celsius" for attribute "unit".  Expected
subset of: {C, F}
ERROR - Members of component CapteurTemp cannot have property piece. Invalid
constraint (&(unit=celsius)(piece=kitchen))
```

launch MotorolaZ43 root

```
spec CapteurTemp
----- [ ASMSpec : CapteurTemp ] -----
   Interfaces: apam.demo.CapteurTemp
   Effective Required specs:
   Required by:
   Implementations:
         MotorolaZ43
   Properties :
      description = Un capteur de …

      name = CapteurTemp
      spec-name = CapteurTemp
      shared = true
      instantiable = true
      singleton = false
```

```
implem MotorolaZ43
----- [ ASMImpl : MotorolaZ43 ] ----
  specification : CapteurTemp
  In composite types:
        rootCompositeType
  Uses:
  Used by:
  Instances:
        Kitchen1
        MotorolaZ43-1
  Properties :
     description = Un capteur de …
     unit = C
     OS = Linux, Android

     name = MotorolaZ43
     impl-name = MotorolaZ43
     spec-name = CapteurTemp
     shared = true
     singleton = false
     instantiable = true
```

Groupes

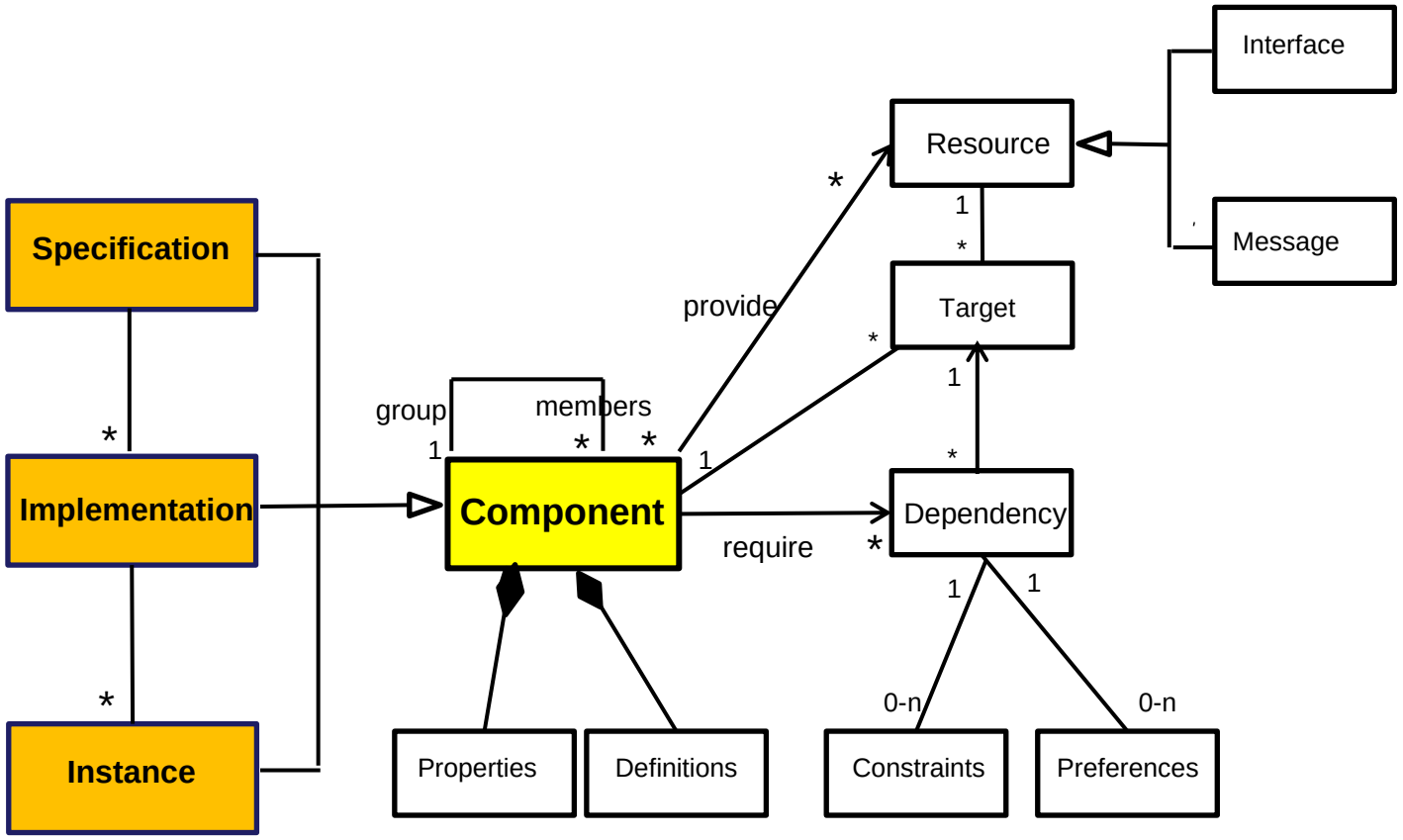

type métier
objet
type technologique
capteurTemp
specification
Tete
Représentant
group
member
*
conformité
MotorolaZ43
implementation
membres
conformité
*
instance
Kitchen2
Bedroom

Spécialisation de « Implémentation »

```
<implementation name="S2ImplCompile2" specification= "S2"
    classname="fr.imag.adele.apam.test.compile.S2Impl"
    interfaces="{fr.imag.adele.apam.test.compile.S2, A.B}"

    messages="{fr.imag.adele.apam.test.M1}"
    message-fields="{p2, p1}"

    singleton="false" instantiable="false" shared="false" >

    <!- Specialisation de "S2" -->
    <properties …
    <definitions …
    <dependency …
```

```
inst Kitchen1
----- [ ASMInst : Kitchen1 ] -----
   Dependencies:
   Called by:
   specification  : CapteurTemp
   implementation : MotorolaZ43
   in composite   : rootComposite
   in application : null
   Properties :
      description = Un capteur de température …
      unit = C
      rate = high
      location = kitchen
      OS = Linux, Android

      shared = true
      name = Kitchen1
      impl-name = MotorolaZ43
      spec-name = CapteurTemp
      instantiable = true
      singleton = false
      inst-name = Kitchen1
```

```
spec CapteurTemp
----- [ ASMSpec : CapteurTemp ] -----
   Interfaces:
         interface fr.imag.adele.apam.test.dependency.CapteurTemp[]
   Effective Required specs:
   Required by:
   Implementations:
         MotorolaZ43
   Properties :
      description = Un capteur de température …
      name = CapteurTemp
      shared = true
      spec-name = CapteurTemp
      instantiable = true
      singleton = false

   Declaration of CapteurTemp
      Provided resources:
         interface fr.imag.adele.apam.test.dependency.CapteurTemp
      Properties:
         description = Un capteur de température …
      Attribute definitions:
         name: unit. Type: {C, F}. default value: F
         name: location. Type: {living, kitchen, bedroom}. default value: null
         name: OS. Type: {Linux, Windows, Android, IOS}. default value: null
```


Interface
Resource
Message
Specification
provide
Target
group
members
Implementation
Component
Dependency
require
Instance
Properties
Definitions
Constraints
Preferences
*
1
0-n

Résolution des Dépendances

Exemple de résolution

```
public class Dependency implements S2, ApamComponent, Runnable {

    // Apam injected
    S3_1      s3;
    S3_2      s3bis;
    Set<S3_1> s3_1set;
    S3_2[]    s3_2array;


@Override
    public void run() {
        System.out.println("S3bis = " + s3bis.getName());
        for (S3_1 s3 : s3_1set)
                System.out.println("s3_1set : " + s3.getName());
        for (int i = 0; i < s3_2array.length; i++)
                System.out.println("s3_2array : " + s3_2array[i].getName());
}

@Override
    public void apamInit(Instance inst) {
        new Thread(this, "test dependency").start();
    }
```

```
<specification name="SDep" interfaces="apam.demo.S2" messages="{apam.demo.M1}" />

<specification name="S3" interfaces="{apam.demo.S3_1,apam.demo.S3_2}" messages="{apam.demo.M2}">
    <definition name="location" type="{living, kitchen, bedroom}"    />
    <definition name="OS" type="{Linux, Windows, Android, IOS}" />
    <definition name="MyBool"    type="boolean" />
    <dependency specification="S4" />
</specification>

<implementation name="Dependency"  classname="apam.demo.Dependency"
          specification="S2" message-fields="p1" >

    <dependency field="s3" /> <!--  interface="apam.demo.S3_1" multiple="false" -->

    <dependency field="s3bis" id="s3bisDep" >
          <constraints>
              <implementation filter="(OS*&gt;Android)" />
              <instance       filter="(&amp;(location=living)(MyBool=true))" />
          </constraints>
          <preferences>
              <implementation filter="(OS*&gt;Linux, IOS, Windows)" />
              <implementation filter="(OS*&gt;Linux, IOS)" />
              <implementation filter="(OS*&gt;IOS)" />
          </preferences>
    </dependency>

    <dependency specification="S3" id="S3Dep"> <!--  multiple = true -->
          <interface field="s3_1set" />
          <interface field="s3_2array" />
    </dependency>
</implementation>
```

Résolution des Dépendance: algorithme

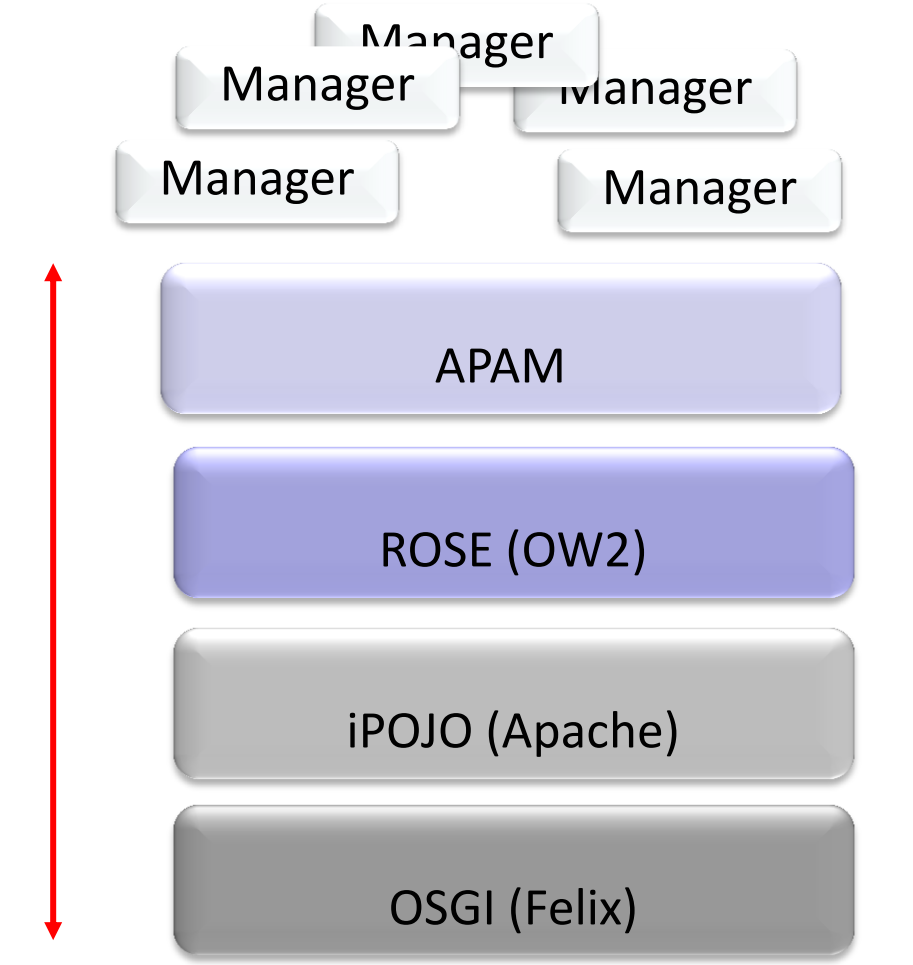
- ✓ Invoqué lors de l'accès à une variable
  - *s3bis.getName()*
- ✓ Le code injecté détecte que la variable n'est pas initialisé
- ✓ Apam est appelé avec:
  - Le client instance : *Dependency-0*
  - la dépendance à résoudre : *s3bisDep*
- ✓ Apam délègue la résolution à un ensemble de managers
  - Dans l'ordre … avec les contraintes …
  - C'est le premier qui trouve une solution (une instance fournisseur)
- ✓ Les managers:
  - Prédéfinis: ApamMan, ObrMan, DynaMan, DistriMan, …
  - domaines spécifiques …

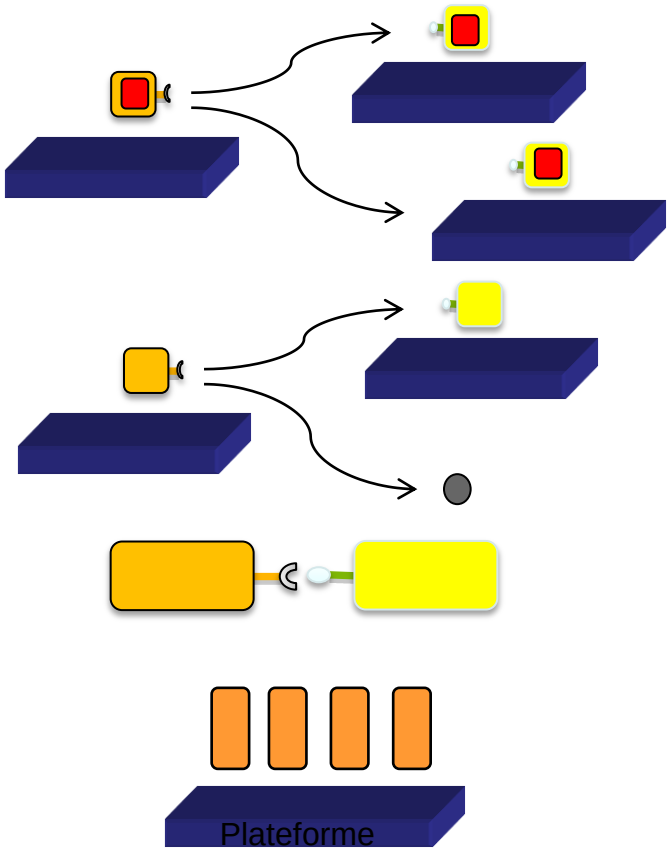
Extensibility: Domain Specific platform

Applications

NF & Business domain

Generic Technical domains





Example de résolution

```
public class Dependency implements S2, ApamComponent, Runnable {

    // Apam injected
    S3_1      s3;
    S3_2      s3bis;
    Set<S3_1> s3_1set;
    S3_2[]    s3_2array;


@Override
    public void run() {
        System.out.println("S3bis = " + s3bis.getName());
        for (S3_1 s3 : s3_1set)
                System.out.println("s3_1set : " + s3.getName());
        for (int i = 0; i < s3_2array.length; i++)
                System.out.println("s3_2array : " + s3_2array[i].getName());
}

@Override
    public void apamInit(Instance inst) {
        new Thread(this, "test dependency").start();
    }
```

```
launch Dependency
Resolving Dependency on the root composite
- Looking for implementation Dependency:
- APAMMAN
- OBRMAN
   OBR: looking for a component matching(name=Dependency)
   -->Component Dependency found in bundle : TestDependency
      From root repositories : [file:/F:/Maven/.m2/repository.xml]
S3Impl Started : S3LunixIOS-Okbis
S3Impl Started : S3LunixIOS-Ok
S3Impl Started : s3Linux-living
S3Impl Started : s3Linux-kitchen
-  : deployed Dependency
g! Dependency test Started : Dependency-0
- Resolving dependency s3bis from instance Dependency-0
- Looking for an implem with dependency id: s3bis. toward interface apam.demo.S3_2
         Implementation Constraints
            (OS*>Android)
         Instance Constraints
            (&(location=living)(MyBool=true))
         Implementation Preferences
            (OS*>Linux, IOS, Windows)
            (OS*>Linux, IOS)
            (OS*>IOS)
- APAMMAN
-  : deployed S3LunixIOS
- Looking for an instance of S3LunixIOS:
- APAMMAN
- Selected S3LunixIOS-Okbis
S3bis = S3LunixIOS-Okbis
```

```
- Resolving dependency S3Dep from instance Dependency-0
- Looking for all implems with dependency id: S3Dep. toward  specification S3
        Implementation Constraints
           (OS*>Android)
        Instance Constraints
           (&(location=living)(MyBool=true))
- APAMMAN
-  : selected S3LunixIOS
-  : deployed S3Android
-  : deployed S3IOS
-  : deployed S3Linux
- Looking for an instance of S3LunixIOS:
- APAMMAN
- Looking for an instance of S3Android:
- APAMMAN
- OBRMAN
- Looking for an instance of S3IOS:
- APAMMAN
- OBRMAN
- Looking for an instance of S3Linux:
- APAMMAN
- Selected set [s3Linux-living, S3LunixIOS-Okbis, S3LunixIOS-Ok]
s3_1set : S3LunixIOS-Ok
s3_1set : s3Linux-living
s3_1set : S3LunixIOS-Okbis
s3_2array : s3Linux-living
s3_2array : S3LunixIOS-Okbis
s3_2array : S3LunixIOS-Ok
```

Résolution des dépendances : Stratégies.

- Stratégie classique : eager.
  - On charge toutes les ressources au démarrage.
- Strategy Apam par défaut: lazy.

  C'est quand une variable est utilisée que Apam essaie de résoudre.

  ✓ On ne charge que les ressources utilisées

  ✓ La ressource peut ne pas être disponible au départ …

  ✓ L'instance peut avoir changé

  ✘ Une résolution peut échouer

Que se passe t-il si une résolution échoue ?

Echec de résolution

- Le programmeur doit savoir si l'échec est possible.
  - Par défaut, les dépendances sont optionnelles:
    - On retourne « null » si la résolution échoue
      - If (s3 == null) { je m'y prends autrement …}
  - Dépendances en « wait »
    - Si la résolution échoue, le thread est mis en attente, jusqu'à ce que la résolution soit satisfaite.
    - Le programmeur considère la dépendance toujours satisfaite.

```
<dependency field="s3" fail="wait" />
```

  - Erreur
    - Le programmeur considère que c'est une exception

```
<dependency field="s3" fail="Exception"
        exception="fr.imag.xxx" />
```

Dépendances Dynamiques

Un service disparait

- Apam retire les « wires » vers ce service
  - La variable redevient non initialisée
- La résolution sera refaite au prochain usage de la variable.
  - Avec les memes stratégies.
  - Si une (autre) solution est trouvée, il y a substitution transparente des fournisseurs.

Un service apparait.

- Si des clients sont en wait, ils sont débloqués.
- Si le nouveau service satisfait des dépendances multiples, il est ajouté à ces dépendances multiples.

Applications dynamiques: Difficulté de conception

- SOA Dynamique => Flexibilité, adaptabilité.
- Définition d'application flexible
  - Peut difficilement être dans le code des composants
    - Vision globale / réutilisation
- Dynamisme : évolution des architectures
  - Quelle architecture ?
  - Comment décrire son évolution (permise) ?
- Eviter les compositions statiques
  - Architecture à composant
- Eviter le non-déterminisme
  - Plates-formes à service

Scoping. Portée.

- Vision « service » traditionnelle
  - tout composant (instance) voie tous les autres composants (instance) présents sur la plateforme.
    - Dynamique, opportuniste, flexible, adaptable
    - Imprévisible dans un monde ouvert
  - Pas d'encapsulation, pas de porté, pas de protection …
- Vision « composant » traditionnel
  - Un composant (implémentation) est lié à d'autres composants (implémentation) dans une application.
    - Rigide, « statique », difficilement adaptable,
    - Déterministe, reproductible, fiable …
  - Structuration forte, encapsulation hiérarchique.
  - Applications indépendantes, contextes indépendants

Services vs Composants

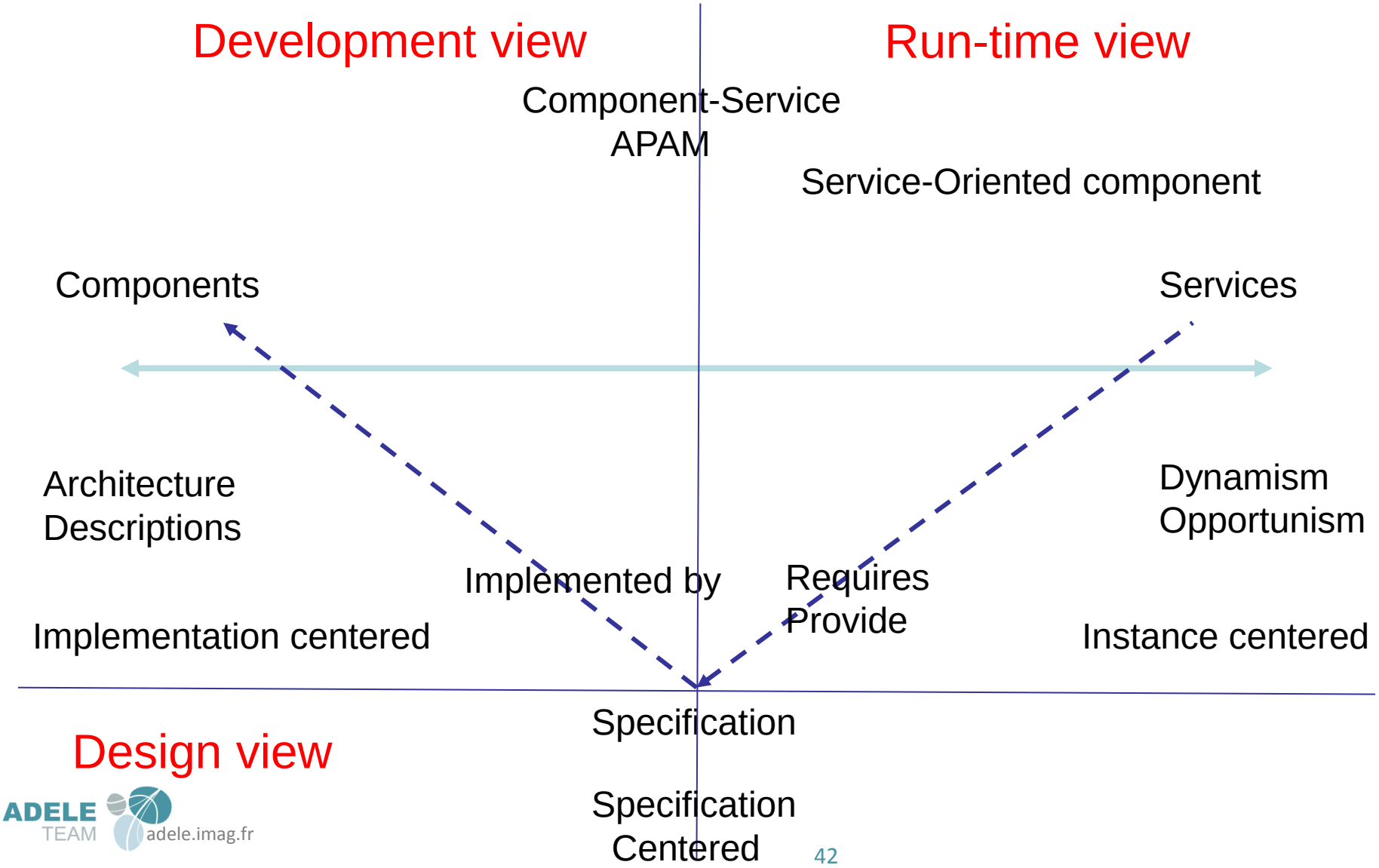
Pourquoi les services n'offrent pas de structuration ?

- Encapsulation: Propriété de composants bien connus
- Service: Partage de fournisseurs anonymes

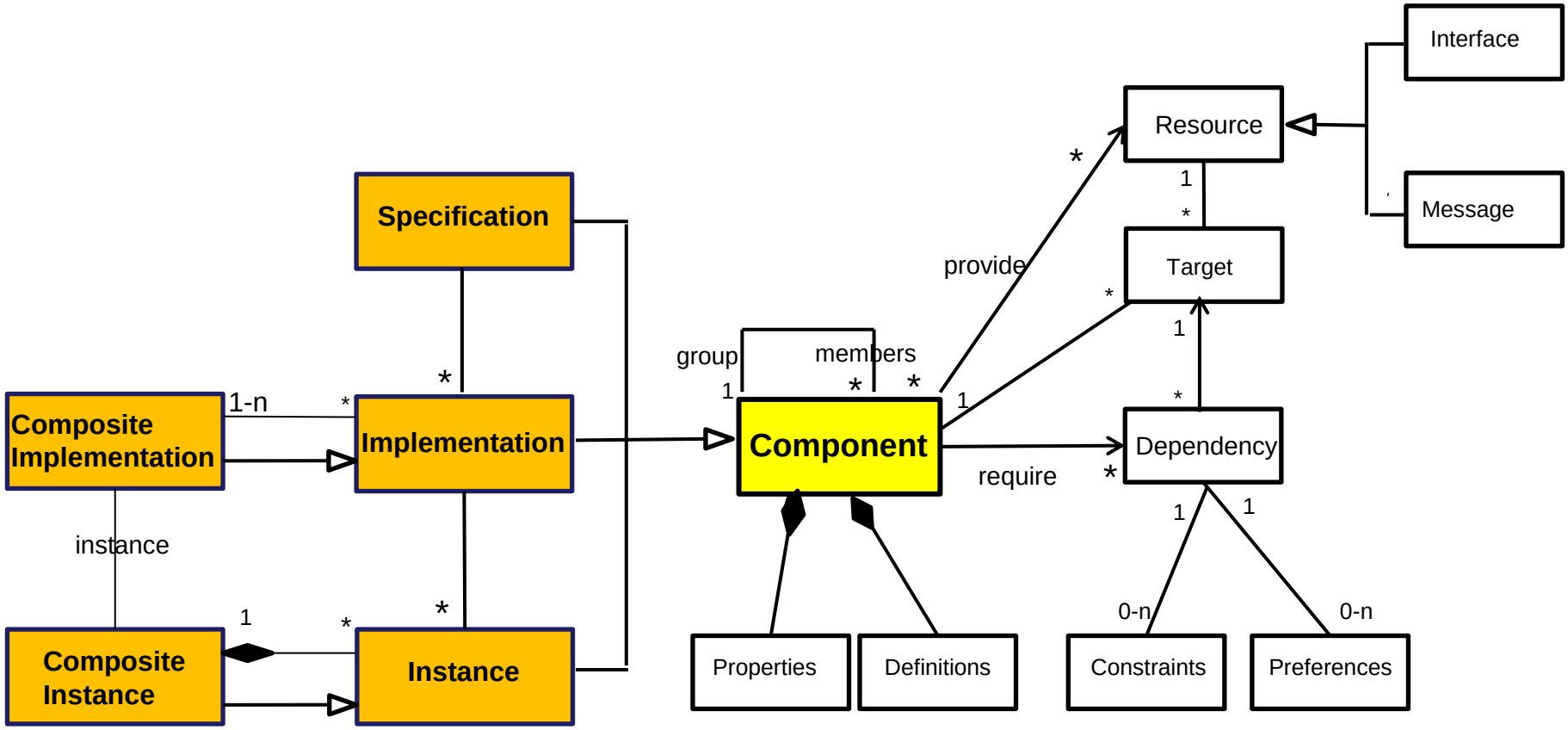
Besoins: encapsulation et contrôle de visibilité

- Respecter le protocole client / fournisseur
  - Le fournisseur peut ne pas être connu statiquement
  - Le fournisseur peut être fourni par d'autres (third party)
- La visibilité des fournisseurs doit être contrôlée
  - usage réservé à certain clients

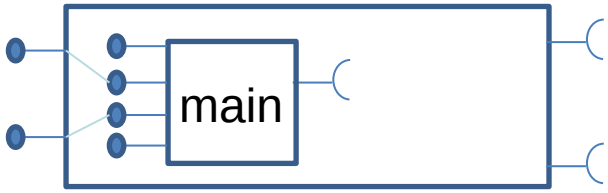
Service-Component model



Composites


Specification
Composite Implementation
1-n
Implementation
instance
Composite Instance
1
Instance
group
members
Component
provide
require
Resource
Interface
Message
Target
Dependency
Properties
Definitions
Constraints
Preferences
0-n
0-n

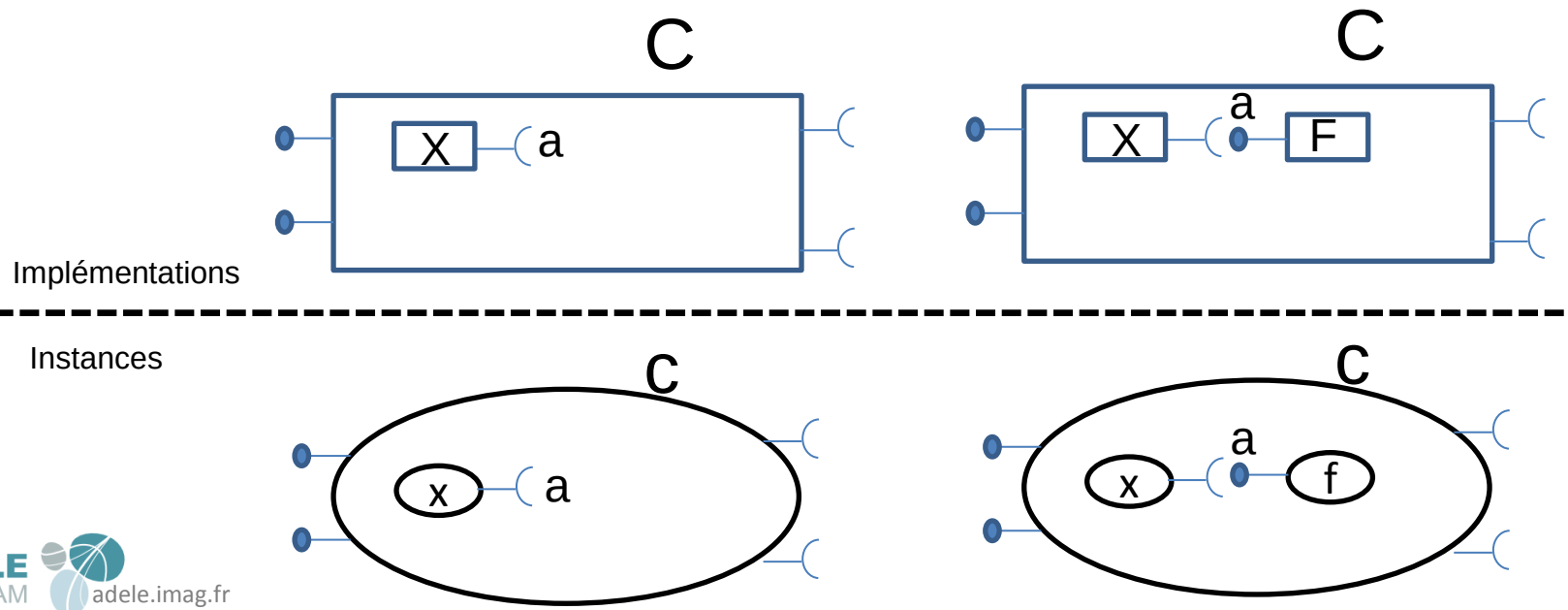
Déclaration de Composite Implementation



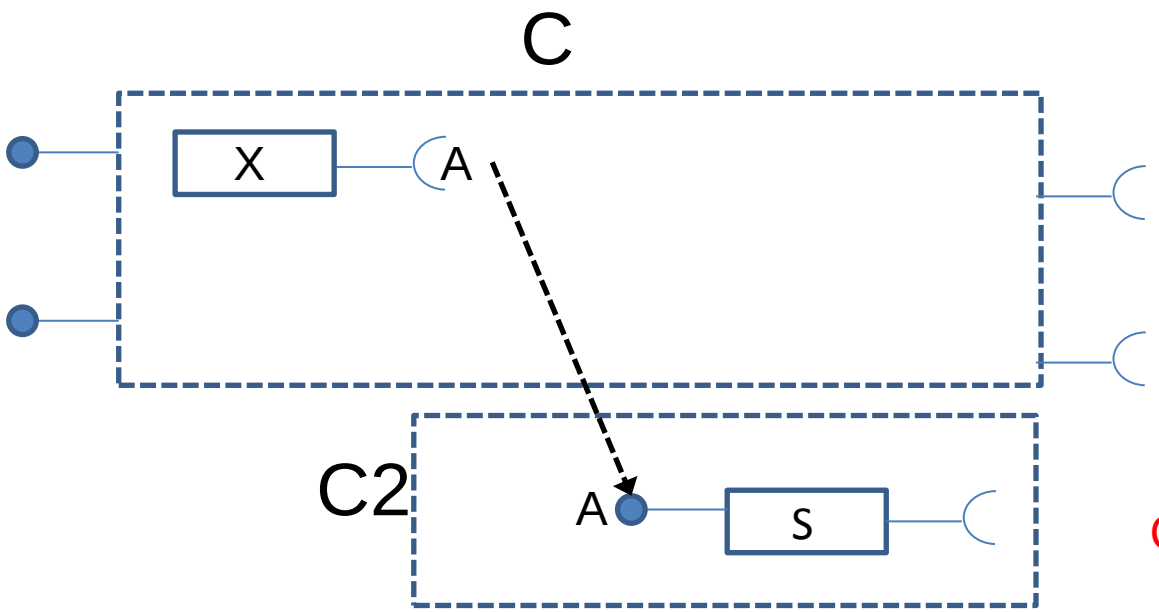

```
<composite name="GoodComposite"  main="S2Impl" />

<composite name="ContentComposite" specification="S2"  main="S2"  >
    <dependency specification="S4" id="compoS4" />
    <dependency implementation="xx" …>
        <constraints> ….</constraints>
        <preferences> … </preferences>

    <contentMngt>
    </contentMngt>
</composite>
```

Contenu des Composites

- Création d'une instance « c » du composite « C »
- On démarre la main instance « x », instance de « X »
- Demande de résolution de «a» pour «x »
  - Si « F » est déployée, « F » appartient à « C ».
  - Si une instance « f » de F est créée, «f» appartient à «c».

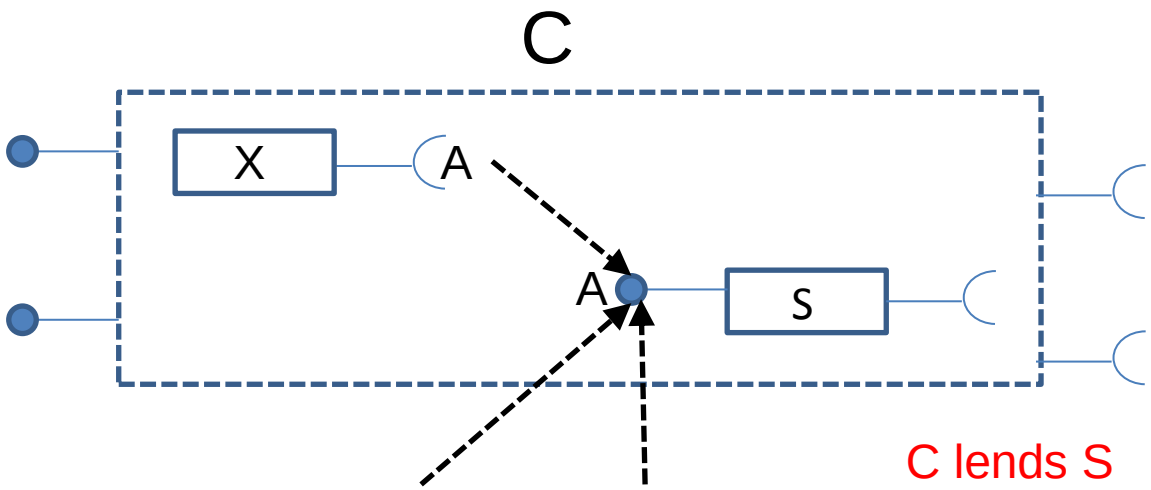


Control de visibilité: borrow

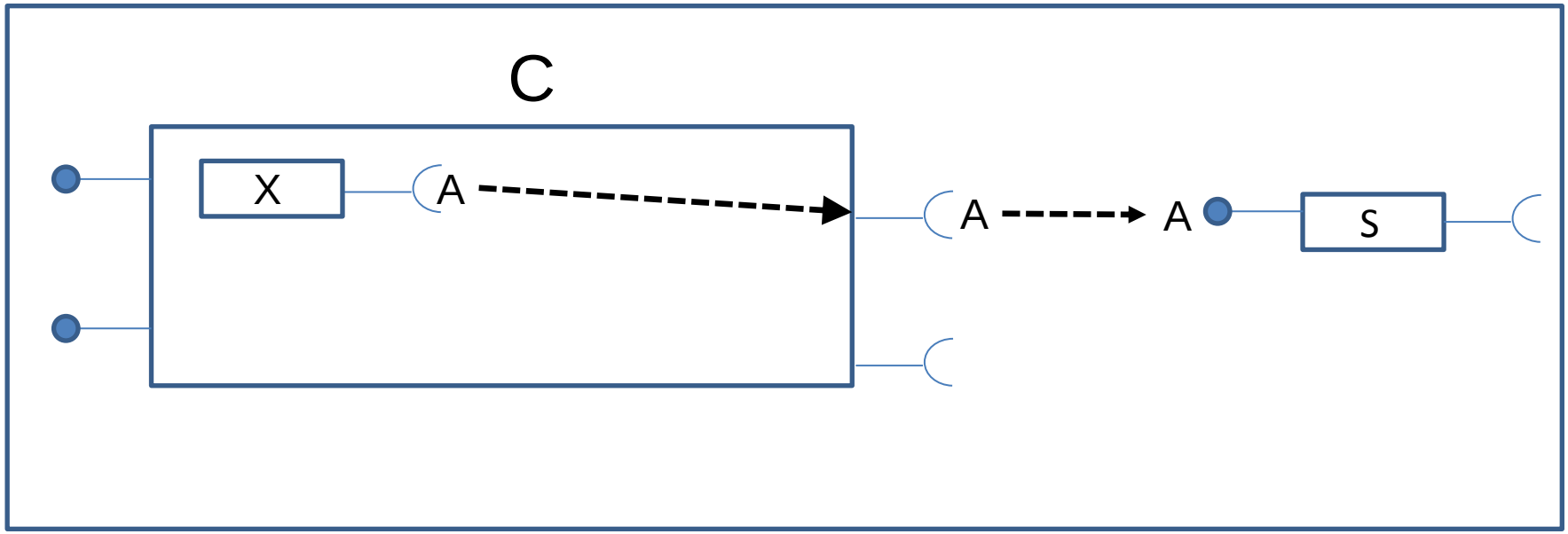


C borrows S from C2

Contrôle de visibilité: lending

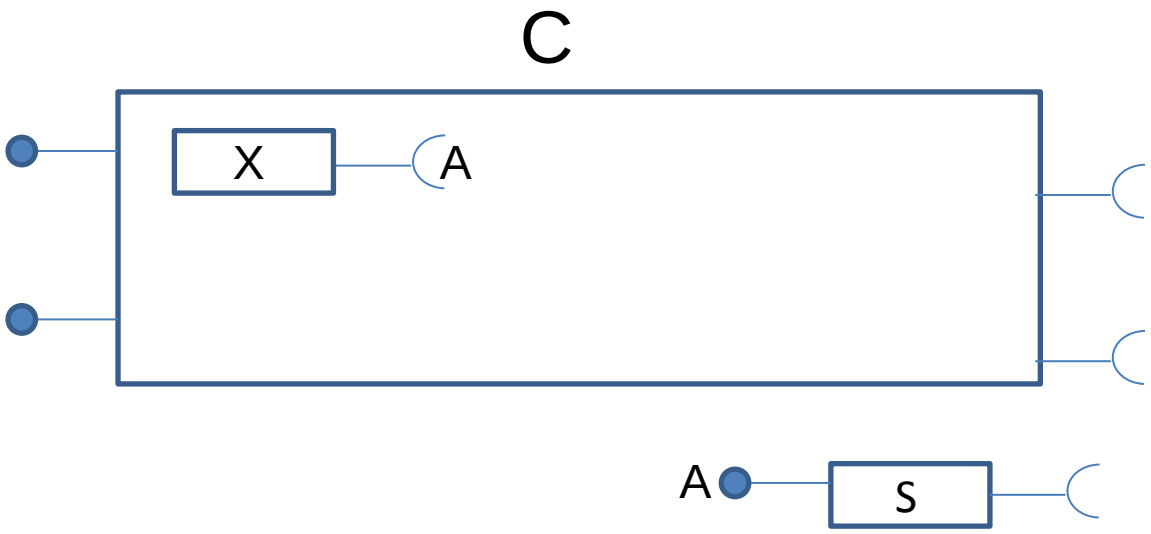


Contrôle de visibilité: Promotion



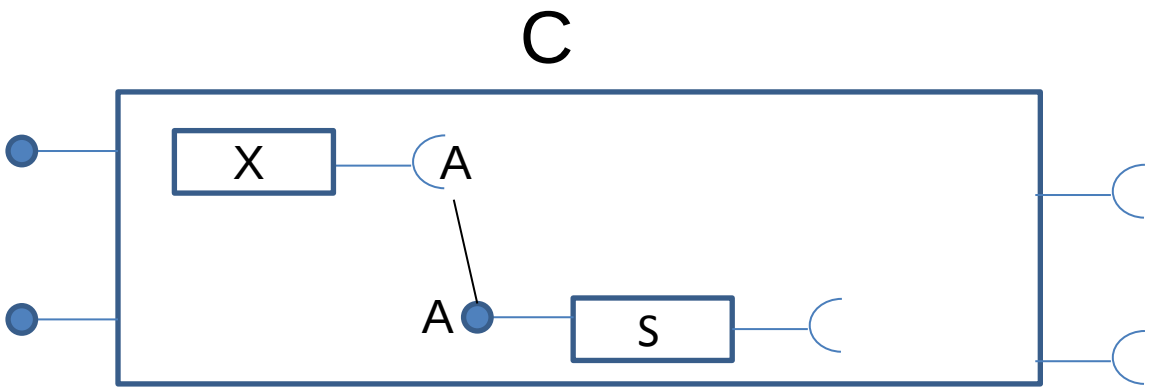
Full encapsulation: the black box

- Service X in composite C needs a service interface A
  - C deploys / instantiate S for itself.
  - S is visible inside C only. No one else can use S

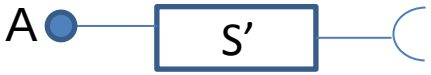


Full encapsulation: the black box

- Service X in composite C needs a service interface A
  - C deploys / instantiate S for itself.
  - S is visible inside C only. No one else can use S



- We say that C
  - **lends** S to no one
  - **borrows** A(S) from no one.

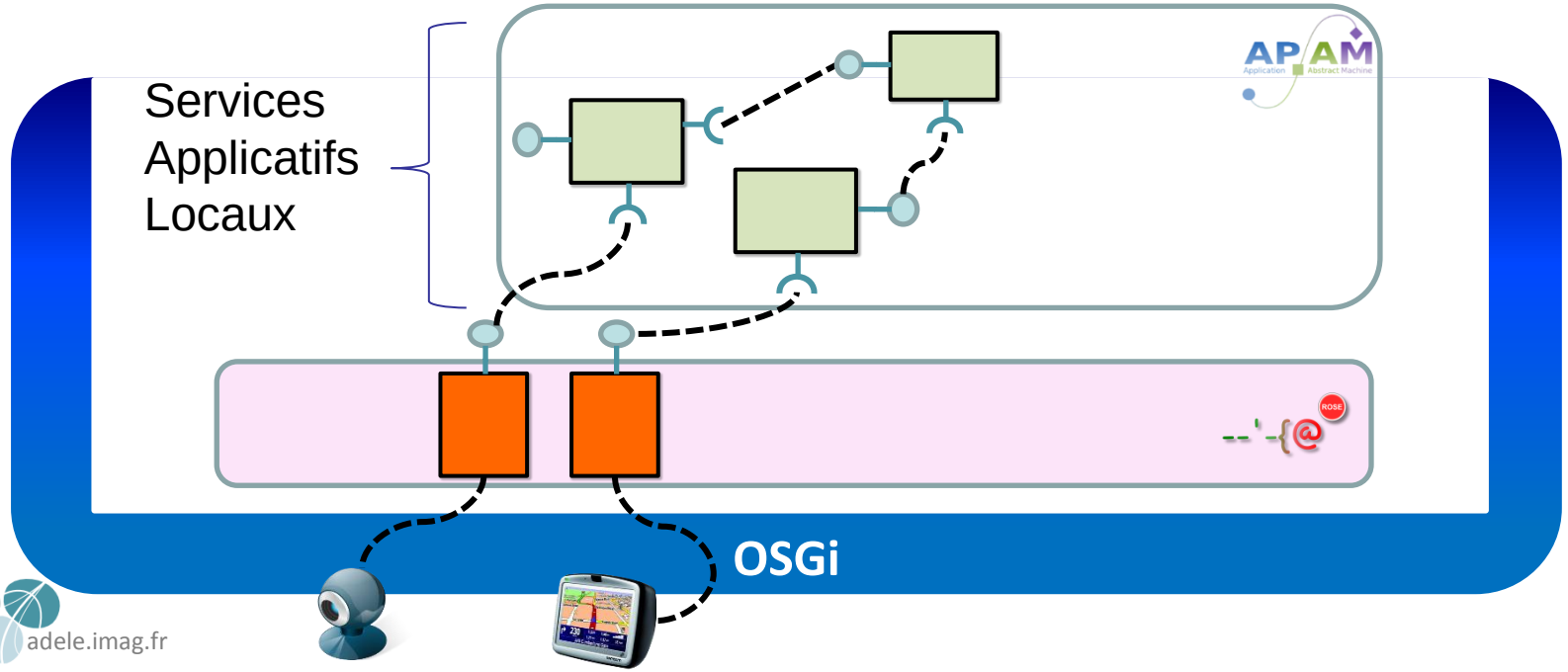


Apam Visibility control

```
<composite name="ContentComposite" specification="S2"  main="S2"  >
    <dependency specification="S4" id="compoS4" />
    <dependency implementation="xx" …>
        <constraints> ….</constraints>
        <preferences> … </preferences>

    <contentMngt>
         <borrow implementation="(OS=Android)" instance="false" />

         <local  implementation="(OS=Linux)" instance="true" />
         <friend implementation="(name=xxx)" instance="(….)" />
         <application />
   </contentMngt>
</composite>
```

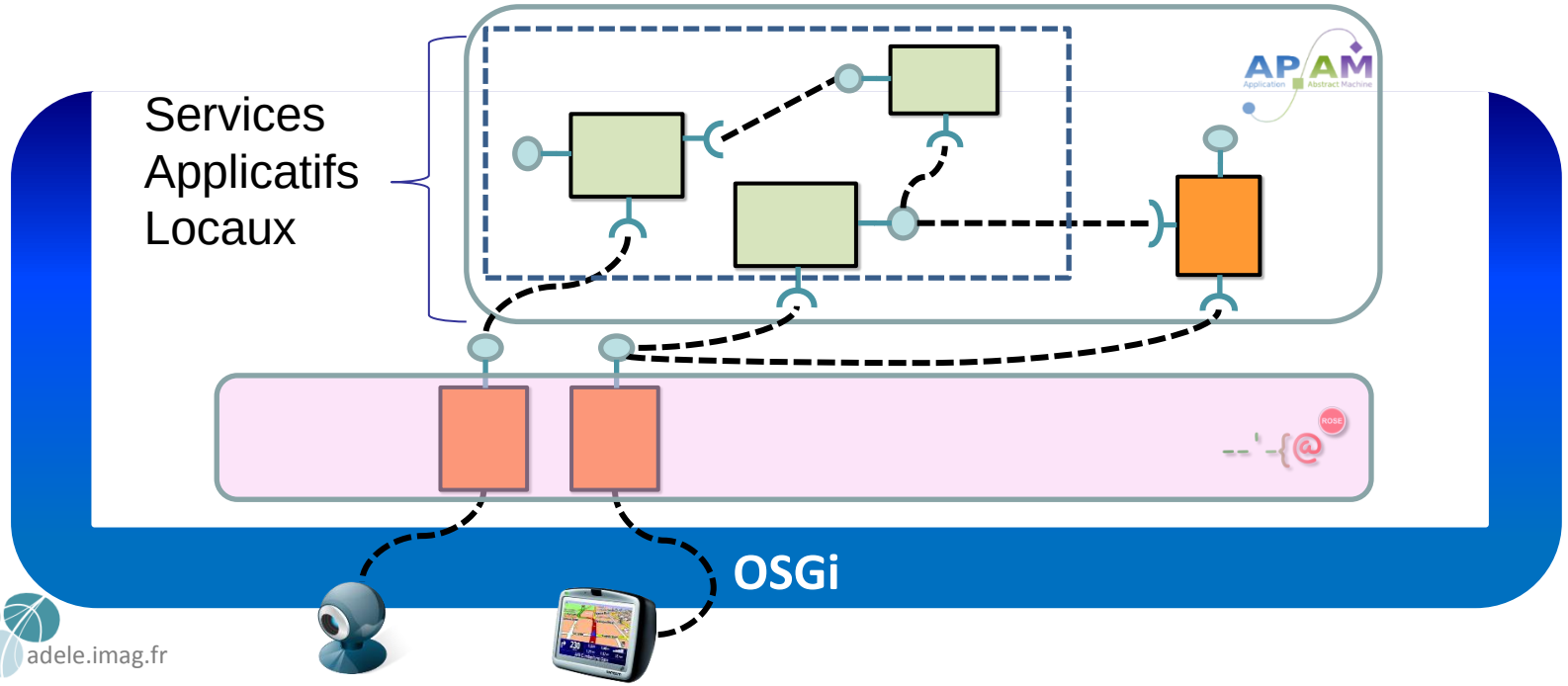
Dynamisme: propriétés globales

- Intrinsèque vs. Contextuel => composites
- Raffinement de la gestion de dépendances dans le composite
- Lazy + backtracking => reconfiguration opportuniste

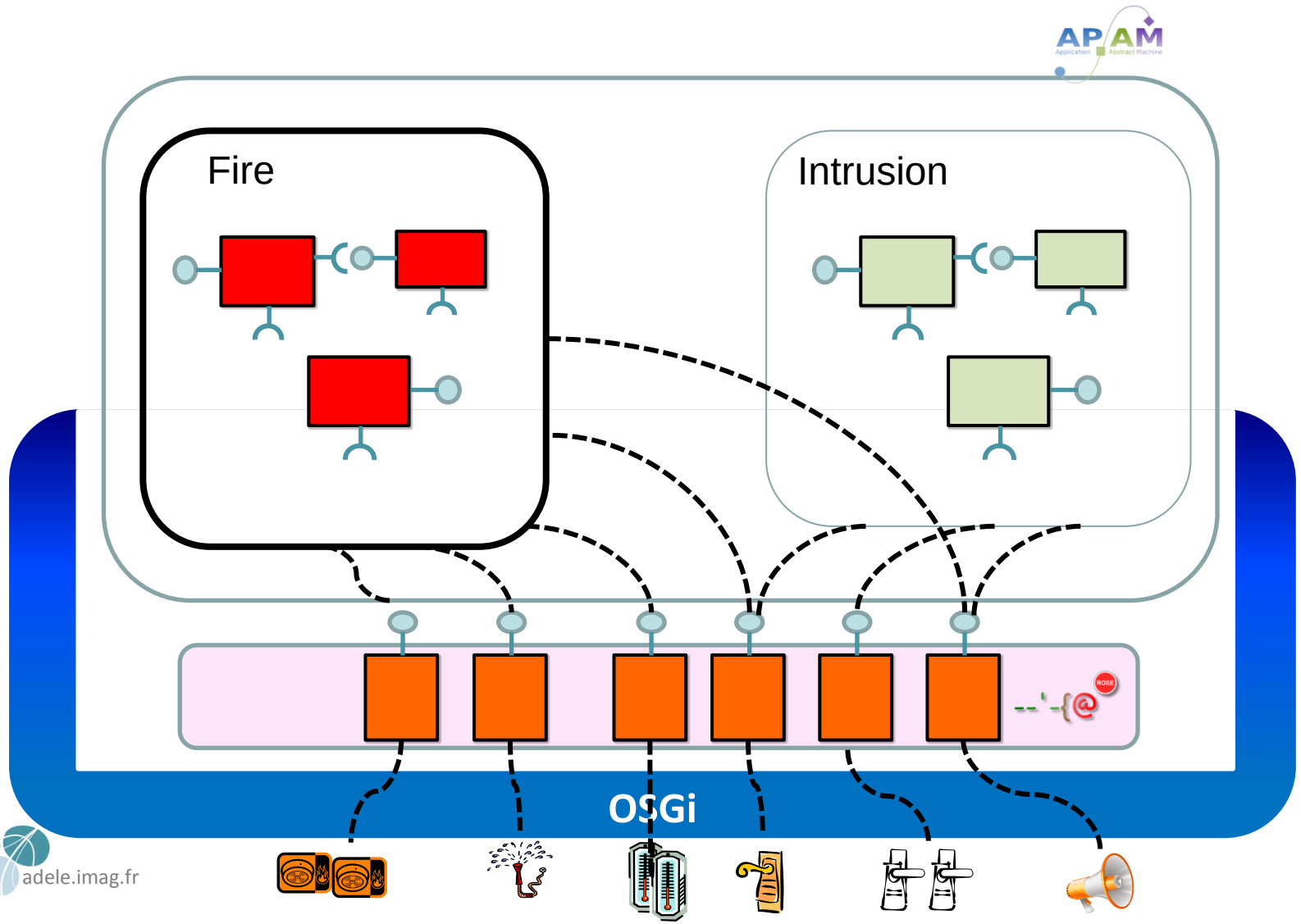


Dynamisme: partage et conflits

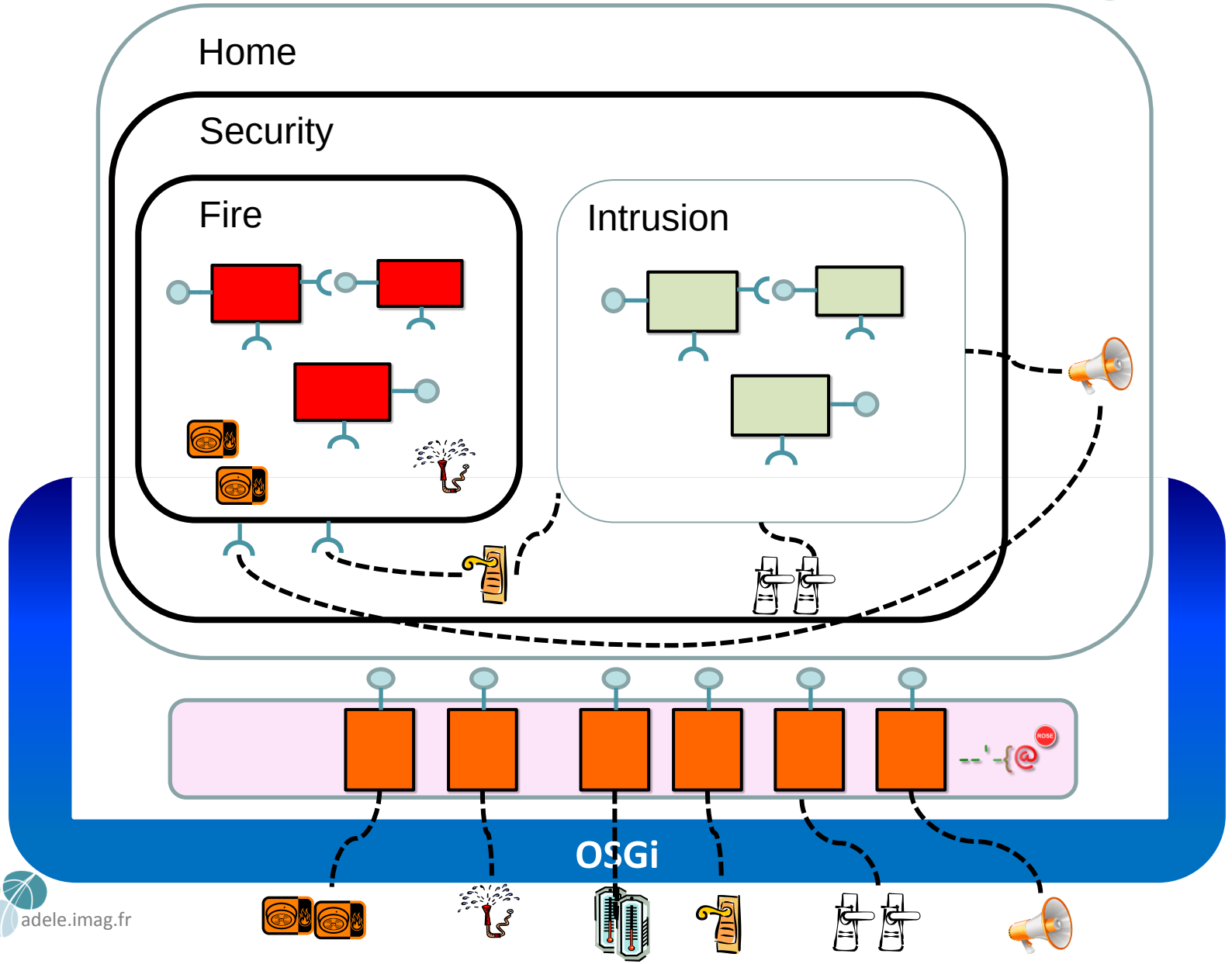
- Monde de la machine versus Monde réel
- Le fournisseur du service ne peux pas anticiper toutes ses possibles usages
- Le développeur d'une application ne peux pas connaître à l'avance toutes les configurations d'exécution
- On a besoin d'une vision globale contextuelle



Dynamisme: partage et conflits



Dynamisme: partage et conflits



Dynamisme: partage et conflits

```
<Composite name=”FireCompo" specification=”Fire”
               mainComponent=”Fire” >

      <dependency specification="Alarm”>
      <dependency specification="Door" id=”doors” >
           <constraints>
                <instance filter=”(location=entrance)”>
            <constraints/>
      </dependency >

       <contentMngt>
          <borrow implementation="false" instance="false" />
          <owns Specification=”{Smoke,Sprinkler}”/>
       </contentMngt>
</composite>
```

Dynamisme: partage et conflits

```
<Composite name=”Security">

    <contentMngt>

        <start component=”FireCompo”/>;
        <start component=”IntrusionCompo” />;
        <owns specification=”Door” />
        <local instance="true" />

        <state values=”Normal, Empty, Intrusion, Emergency” />
        <grant component=”Fire” dependency=”doors”
               when=”Emergency”/>
        <grant component=”Intrusion” dependency =”Door”
               when=”Intrusion”/>

    </contentMngt>
</composite>
```

APAM : a Component-Service platform

- Manages **applications** in the full range from
  - “pure” service based, to
  - “pure” component based
- Manages various applications running concurrently
  - Protection, scope, visibility, resource sharing …
- Extensible
  - Spécialisable for domain specific & life cycle needs.
- Based on the concepts :
  - Reified and reflexive application state (APplication Abstract Machine).
  - Specification
  - Composite