COMPO : Un continuum pour le développement par Components

(Components and Architectures)@runtine

Christophe Dony* - Petr Spacek** - Chouki Tibermacine* - Frédéric Verdier *** - Anthony Ferrand ***

** MAREL - LIRMM - Université Montpellier*
*** CCMI : - Center for Conceptual Modeling and implementation - Czech Technical University*
**** Master Génie Logiciel (AIGLE) - Université Montpellier*

Discussion autour de :

Petr Spacek, Christophe Dony, and Chouki Tibermacine.
A Component-based meta-level architecture and prototypical implementation of a Reflective Component-based Programming and Modeling language. CBSE'14.

Cours M2 AIGLE - HMIN307- Université Montpellier

Table des matières

1 Contexte et Problématique

1.1 Contexte : CBSE

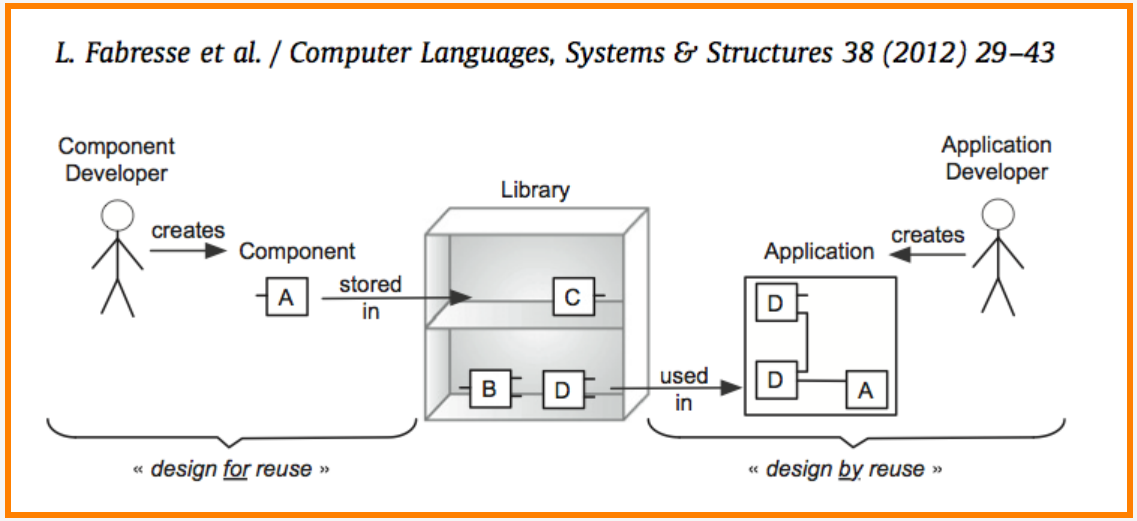


Figure *(1)* – *Connections et Incrémentalité - la métaphore de la réutilisation en électronique - requètes/réponses ou data-flow - mais avec des données structurées.* `Composant, étagère, port, interface, connection, architectures, encapsulation, ...`

- MD McIlroy :
  - **Mass produced software components**, NATO Conference on Software Engineering, Naur P, Randell B. (eds.) 1968
  - ... known for having originally developed **Unix pipelines**, software componentry and several Unix tools, such as spell, diff, sort, ...” [Wikipedia]

1.2 Phase d'émergence, beaucoup d'idées autour d'un terme générique

Etat de l'art, voir thèse de Petr Spacek, chapitre 2.

1. **Architecture Description Languages** (ADLs) (Approche Générative)

   "*software architecture is becoming a valuable abstraction*" : TSE 1995 - *Special issue on Sotware Architecures.*

   Wright, C2, Rapide, Darwin, Fractal ADL, AOKell, DiaSim...

2. **Component frameworks** (Distribution, Conteneurs, déploiement)
   OMG CORBA CCM, EJBs, Fractal, OpenCOM, FraSCAti, ...

3. **Module Interconnection Languages** (MILs)
   OSGI, Jiazzi, MzScheme, Knit, ML, ...

4. **Component-oriented Programming languages** (COPLs)
   ArchJava, CompJava, ACOEL, ComponentJ, SCL, ...

1.3 Constats ... ADLs et Approche générative ...

- Approches **multi-langages** :
  - de description d'architecture (ADL),
  - de spécification d'interfaces (IDL),
  - de contraintes architecturales (OCL),
  - de transformation de modèles (...),
  - d'implantation (Java, ...).
- **Absence de continuum**

  "*most component models use standard programming languages ... for the implementation stage*" [Crnkovic&al, TSE 2011]

  => Debugging, reverse engineering, ... : complexes

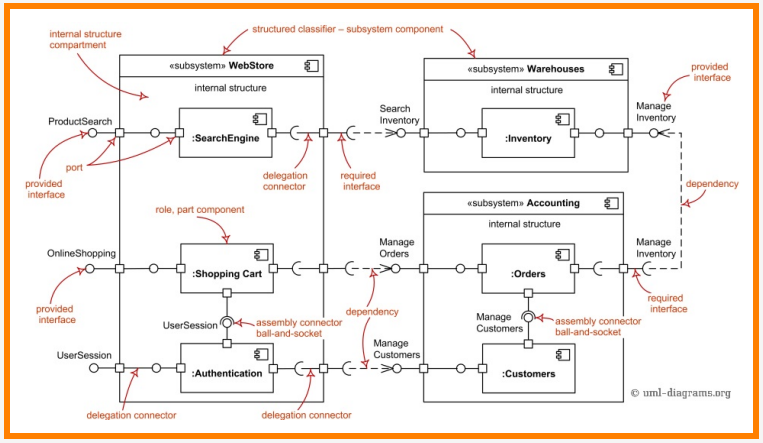


Figure *(2) – Un exemple d'architecture avec UML components [http ://www.uml-diagrams.org/].*

1.4 Continuum : pour la phase d'implantation, Il manque des choses aux objets ?

- **requis** explicite
- **architectures** explicites
- **ports** et **connections**
- Découplage $- - - - - - - - - - \rightarrow$
- Points de vues (différents ports fournis sur un même composant)
- ...

```
1  class X {
2    protected Y y;
3    public somewhere() { ... y = new Z();
         ... }
4    ... }

6  class Z implements Y { ... }
```

Couplage en Java

1.5 Des langages de programmation par composants pour la phase de d'implantation ?

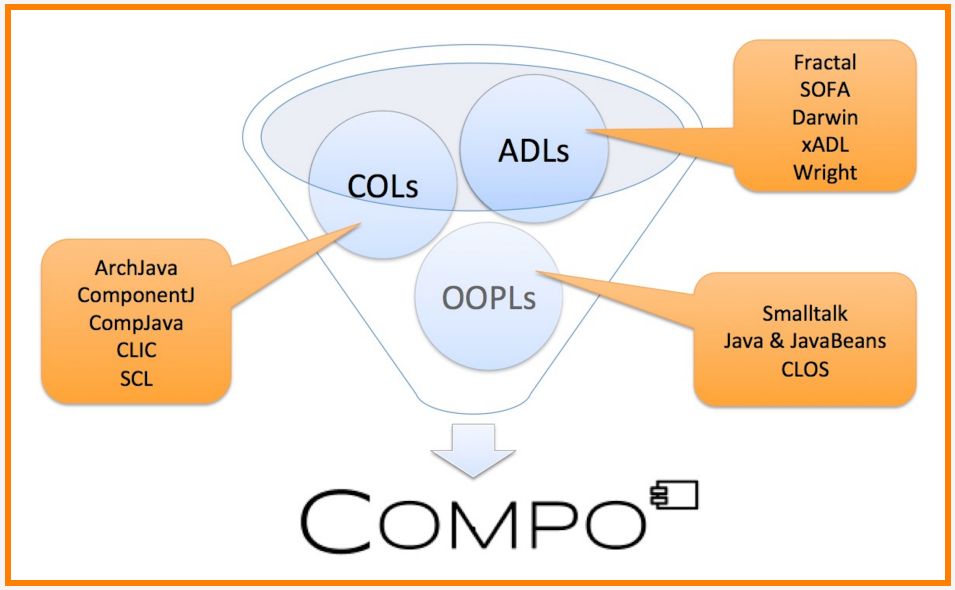
Les premiers langages de programmation par composants (COLs) : ArchJava , CompJava, ACOEL, ComponentJ, ...

... ne le proposent pas.

- **"Archjava : connecting software architecture to implementation"**.
  J. Aldrich, C. Chambers, and D. Notkin. ICSE 2002,
- Introduit l'instruction `connect`.
- *Problème* : mixe objets et composants
  les composants se connectent, les objets se passent en arguments
  qu'est-ce qui est un objet, qu'est-ce qui est un composant ?

Challenge : expérimenter un espace conceptuel unifié

Tout faire avec les même composants ... le même langage.



Compo tries to unify the following ideas:

- architectures are connected to implementation	COLs
- requirements are explicit	ADLs & COLs
- architectures are explicit	ADLs
- everything is a component	Smalltalk
- communication uniquely through ports	SCL
- design reuse via inheritance	OOP
- components are instances of descriptors	OOP

2 Un ADL et un COPL

- Histoire : SCL 2003-2007 (thèse Luc Fabresse) - Compo 2011-2014 (thèse Petr Spacek)
- Modeler et Programmer en intégration continue ...
  - un langage de **description d'architectures** et **programmation**
  - intégrer les schémas de **modularité**
  - intégrer les schémas de **réutilisation** (liaison dynamique, inversion de contrôle, ...)
  - Intégrer le niveau méta (IDM, vérifications, contraintes, transformations, MODELS@RUNT
  - potentiellement compatible avec l'**approche distribuée** (conteneurs / déploiement / annuaires) - travaux avec Hinde Bouziane
  - potentiellement compatible avec l'**approche packaging** (paquets de distributions) - travaux avec Djamel Seriai

2.1 Descripteurs et Composants

- Descripteur (texte)
- Composant, instance d'un descripteur
- Déclaration de Port :
  interne/externe - requis/fourni - ...
- Description de port :
  signature* ou nom de descripteur
- Connection : RF ou de délégation
  RR - FF
- Architecture :
  Ensemble de connections
  Opérateur de désignation `@` :

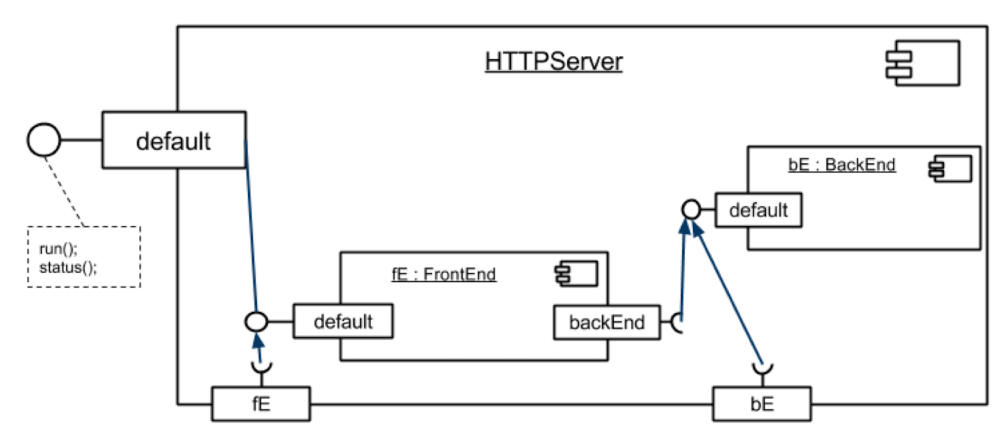


Figure *(3)*

```
Descriptor HTTPServer
{
   provides: {
      default : { run(); status(); }
   }
   internally requires {
      fE : FrontEnd;
      bE : BackEnd
   }
   architecture {
      connect fE@self to default@(FrondEnd.new());
      connect bE@self to default@(BackEnd.new());
      connect backEnd@fE to default@bE;
      delegate default@self to default@fE;
} }
```

Figure *(4)*

- Un composant n'est utilisable que via un port requis connecté à un de ses ports fournis,
- Architectures :
  - Opérateur de désignation `p@pr`, utilisable dans un `connect`, rend le port `p` du composant connecté à `pr`.
  - `new` rend un port requis connecté au port fourni par défaut du nouveau composant,
  - découplage : Impossibilité de connecter un port requis externe

```
Descriptor HTTPServer
{
   provides: {
      default : { run(); status(); }
   }
   internally requires {
      fE : FrontEnd;
      bE : BackEnd
   }
   architecture {
      connect fE@self to default@(FrondEnd.new());
      connect bE@self to default@(BackEnd.new());
      connect backEnd@fE to default@bE;
      delegate default@self to default@fE;
} }
```

2.3 Services et Invocation de service

- Service (opération, méthode)
  exemple `isListening()`
- Invocation de service, exemple : `fE.isListening()`
  - recherche d'un service compatible de nom `isListening` sur le composant dont un port fourni déclarant ce service est connecté à `fE` (liaison dynamique)
  - déréférencement automatique
    réalisé par l'interpréteur, opérateur non accessible au programmeur.
  - sémantique : valeur rendue, passage d'argument (...)

```
1  Descriptor HTTPServer {
2   provides {
3      default : { run(); status() }
4    }
5   internally requires {
6      fE : FrontEnd;
7      bE : BackEnd;
8    }
9   architecture {
10     connect fE to default@(FrontEnd.new());
11     connect bE to default@(BackEnd.new());
12     delegate default@self to default@fE;
13     connect backEnd@fE to default@bE;
14  }

16  service status() {
17     if (fE.isListening())
18         { return 'running' }
19         else { return 'stopped'} }}
```

Listing (1) *– un même langage pour décrire l'architecture et le code des services*

2.4 et les éléments primitifs ? entiers, chaînes,

- Tout élément d'un type primitif ou d'un type non encore décrit en Compo ...
- est interprété comme un composant possédant en tout et pout tout un port fourni "default"
- auquel il est possible de se connecter, pour invoquer ses services

```
requires i Integer;
i connect to 1;
```

IMPLANTATION

- Implantation de Compo : Petr Spacek ... après SCL (Luc Fabresse)
- Implanté en **Pharo** (http://pharo.org/)
- Implantation du module graphique, du passage d'arguments, des composants primitifs : Petr Spacek, Frédéric Verdier, Anthony Ferrand

3 Schémas de réutilisation : un système d'Héritage

Présentation, etat de l'art, travaux connexes :

Petr Spacek, Christophe Dony, Chouki Tibermacine, and Luc Fabresse. **An Inheritance System for Structural and Behavioral Reuse in Component-based Software Programming**. In procs. of 11th Int. Conf. on Generative Programming and Component Engineering (GPCE'12), pages 60-69. ACM Press, September 2012.

3.1 Schémas de Réutilisation (GOF) - Composition et/ou Héritage ?

Réutilisation par Composition : La solution ComponentJ ...

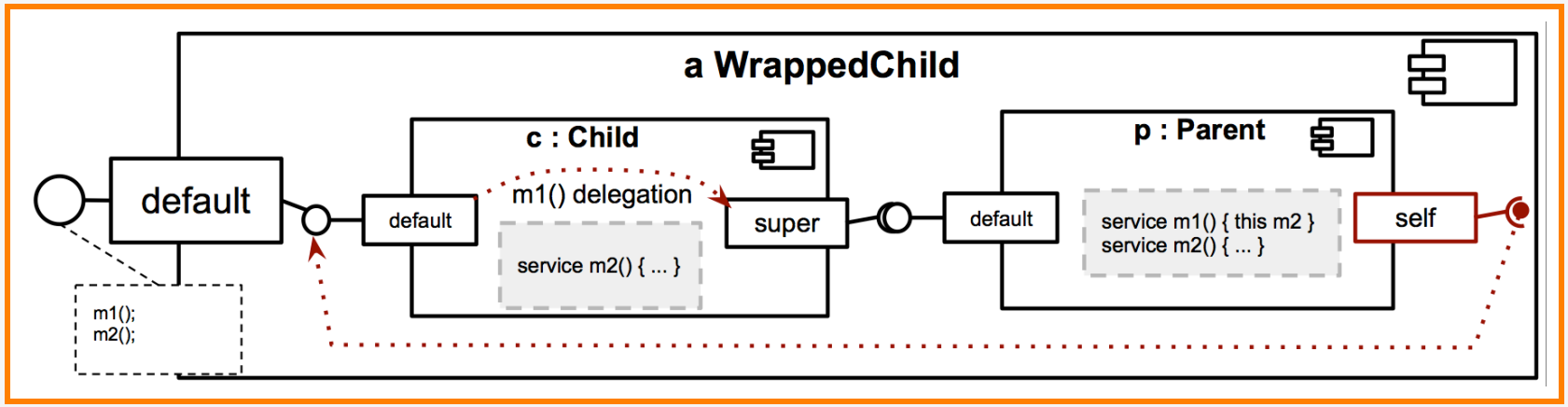


Figure ***(5)*** – *La solution de* ComponentJ *au problème de perte du receveur initial rencontré en simulant l'héritage par la composition.*

... complexe et non incrémentale.

Nécessite de recalculer l'interface de *child* à chaque ajoût de parent.

3.2 L'héritage avec Compo

- Comportements
  - ajoût de services
  - spécialization de services (port "super")
- Architectures
  - ajoût ou spécialisation de ports fournis
  - ajoût ou spécialisation de requis interne
  - specialisation d'architecture
  - ajoût de ports requis externe (autorisé ? covariant ! !)

3.3 Exemple : spécialisation d'architecture

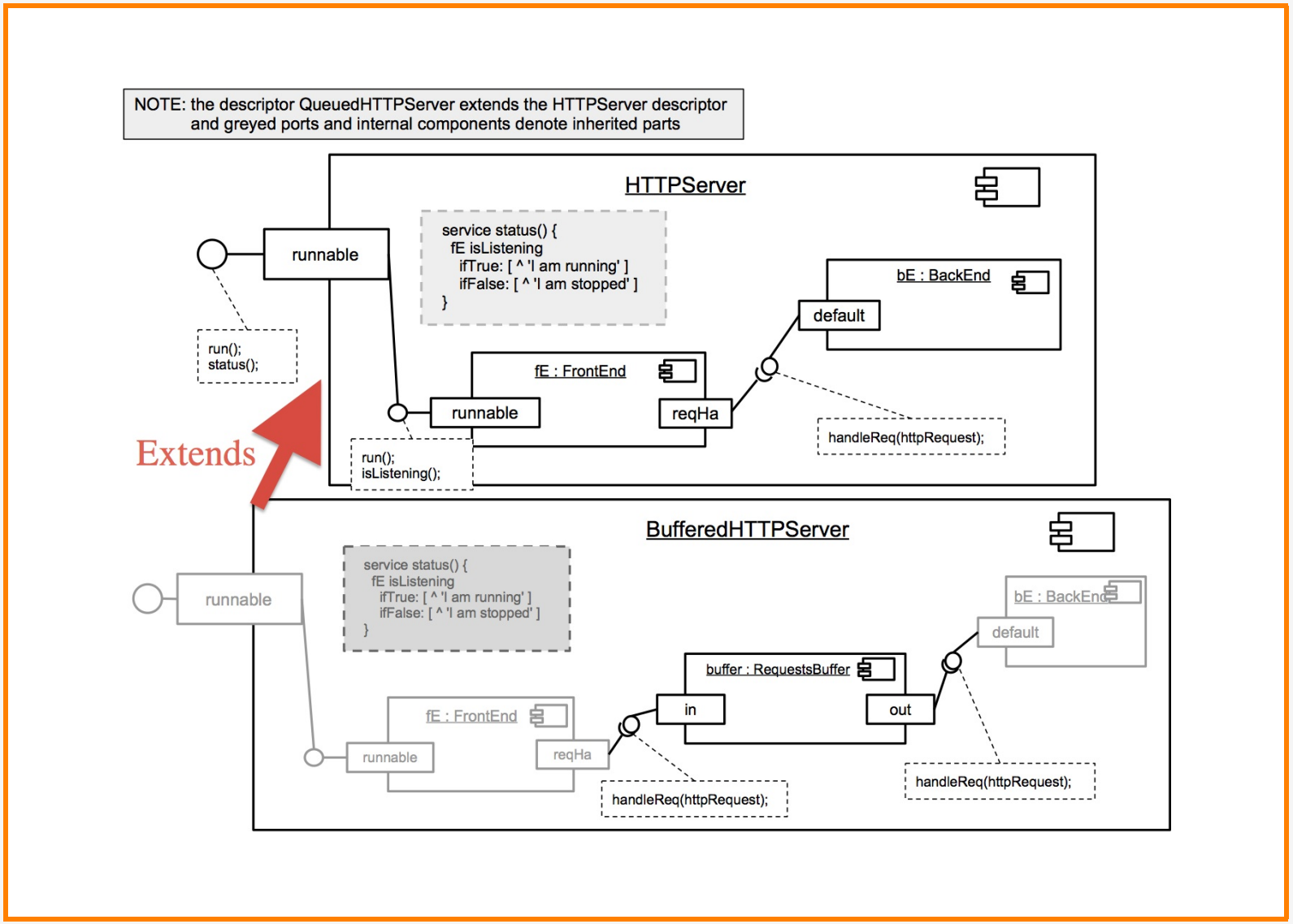


Figure *(6)* – Specialisation d'architecture *: un server HTTP bufférisé possède un composant interne additionnel.*

Exemple : spécialisation d'architecture - suite

```
1  Descriptor BufferedHTTPServer extends HTTPServer
2  {
3    internally requires {
4      buffer : RequestsBuffer
5    }
6    architecture {
7      connect buffer@self to default@(RequestsBuffer.new());
8      disconnect backEnd@fE from default@bE;
9      connect backEnd@fE to in@buffer;
10     connect out@buffer to default@bE;
11   }
12   ...
13 }
```

Listing (2) – *Specialisation d'architecture.*

ex : class A { int i; ...}
class B extends A{ int j; ...}
2 cas possibles

a) new B (z,u) -> ok avec constructeur 2 params
b) remplace un A par un B dans un framework

b1) A a = new A(3) -> ok mais si on fait A a = new B(3) il faut re-définir le constructeur comme dans b2) sinon cela causera une erreur car j non défini
b2) B(int x) {
i = x;
j = x+1;
}-> ok mais peut causer une erreur car j non initialiser mais rarement

3.4 Exemple : ajoût de requis

```
1  component descriptor Emailer
2  {
3   provides {
4    in : { sendMessage(); ... }
5   }
6   requires {
7    syntaxChecker : { spelling(); }
8   }
9   ...
10 }
11
12 component descriptor SmartEmailer
13   extends Emailer
14 {
15  requires {
16   syntaxChecker : { grammarChecking(); }
17   semanticsChecker : {checkAttachements();}
18  }
19  ...
```

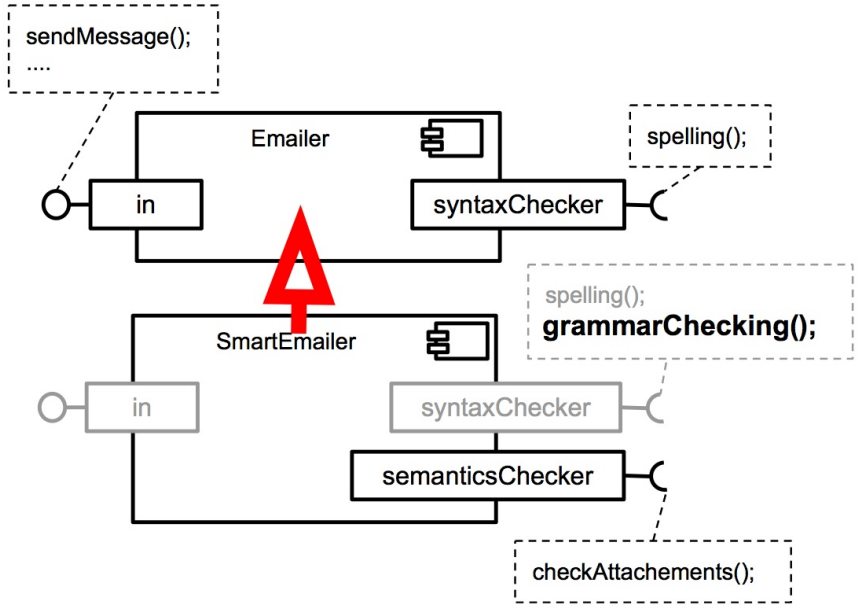


Figure *(7) – Sous-descripteur et spécialisation du requis. Le remplacement dans une architecture d'un* Emailer *par un* SmartEmailer *nécessite des contrôles et/ou une primitive spécialisée* `(newCompatible)`

4 Compo : Un langage réflexif

Un méta-modèle a) accessible b) exécutable

Présentation, etat de l'art, travaux connexes :

Petr Spacek, Christophe Dony, and Chouki Tibermacine. **A Component-based meta-level architecture and prototypical implementation of a Reflective Component-based Programming and Modeling language**. In Procs. of ACM CBSE'14, pages 13-23. July 2014.

4.1 Intérêts

MDE pour et par le développement par composants

Vérifications, Transformations : statiquement et à l'exécution

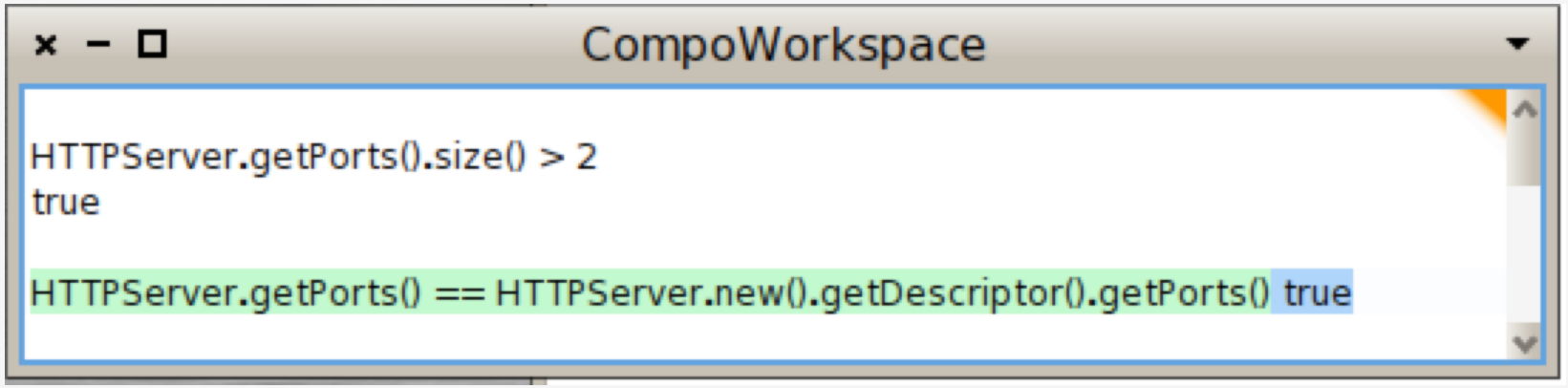


Figure ***(8)*** – *Exemple : Vérification de modèles et de programmes.*

Adaptabilité dynamique des architectures et de leurs implémentations

Models@runtime

- Mettre du code dans les modèles
  [Muller&al Weaving executability into object-oriented meta-languages. MoDELS'05]
- Mettre des modèles dans le code
  [3-Lisp, ... lignée des systèmes tout-ou-partie réflexifs] $\leftarrow - - - - - Compo$

une solution : une version réflexive de Compo

un challenge : expérimenter "tout est composant" au méta niveau

4.2 Le méta-modèle de Compo

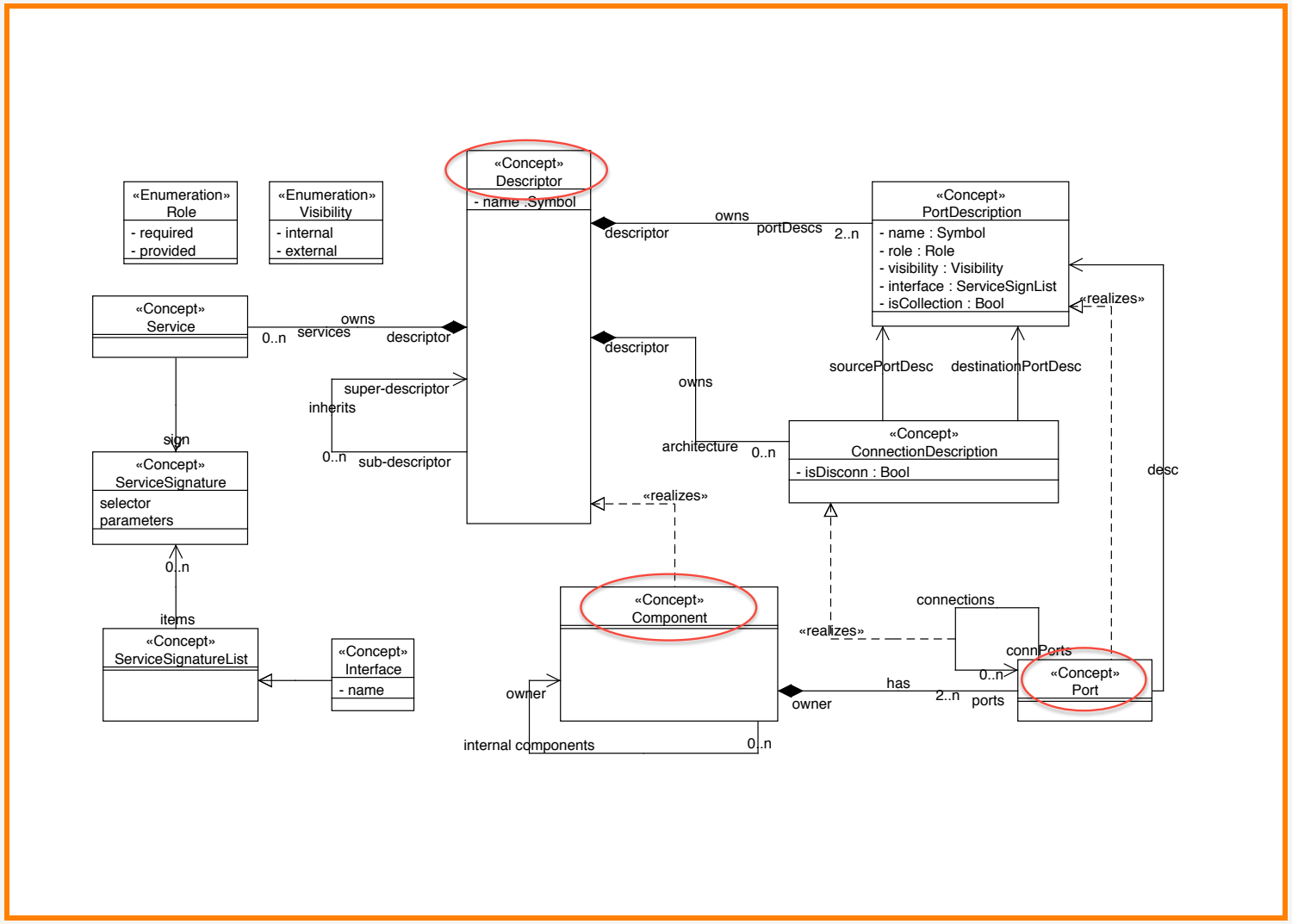


Figure ***(9)*** – *Reifier (componentifier) tous les concepts, ... en particulier ... Descripteurs et Ports*

4.3 Le méta-modèle - intégration de la réflexivité

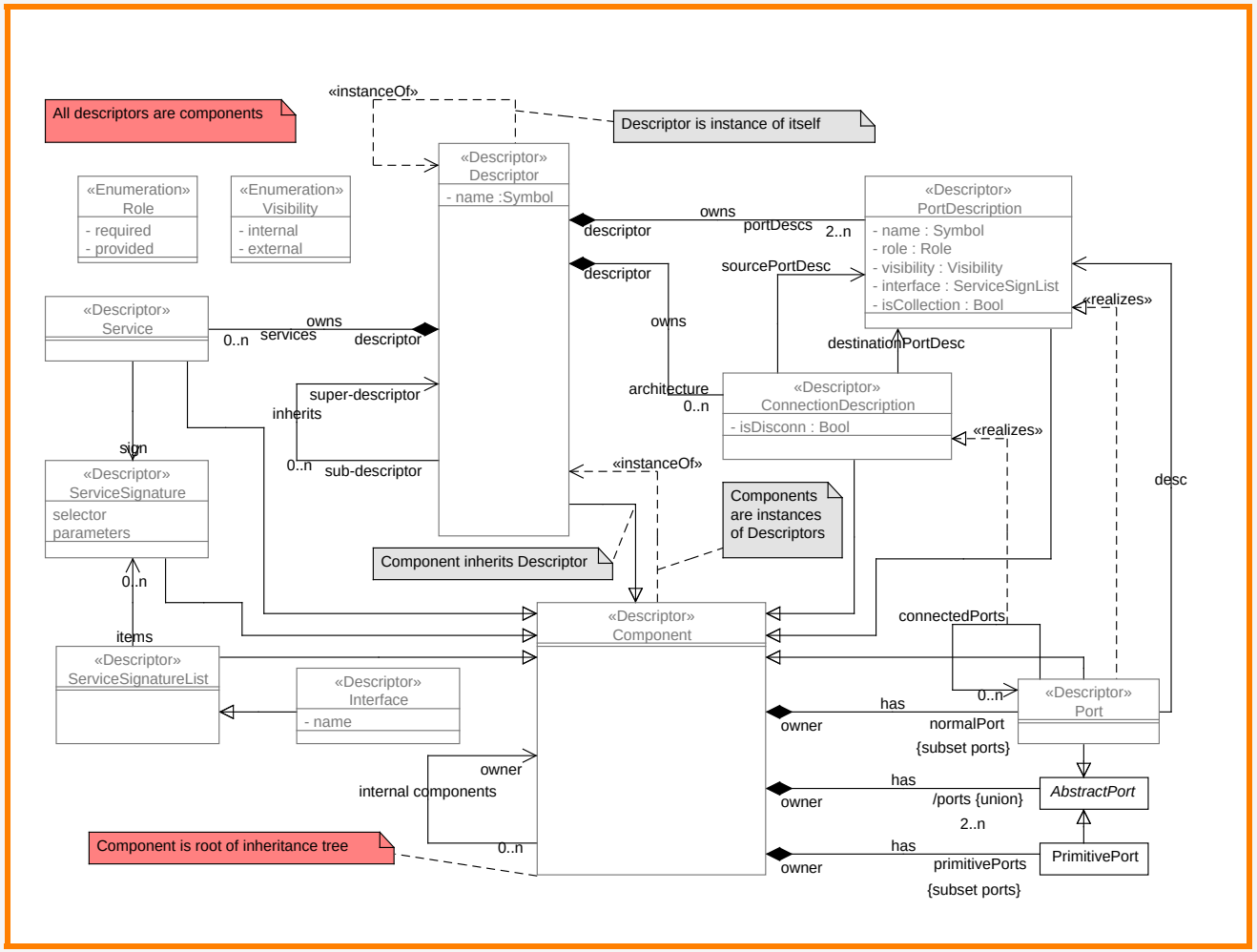


Figure ***(10)*** – *Tout élément de ce méta-modèle qui hérite de* `Component` *est accessible en introspection & intercession (lecture et écriture) donc admissible à verification, transformation (de modèles).*

4.4 Tout est composant : le cas des descripteurs

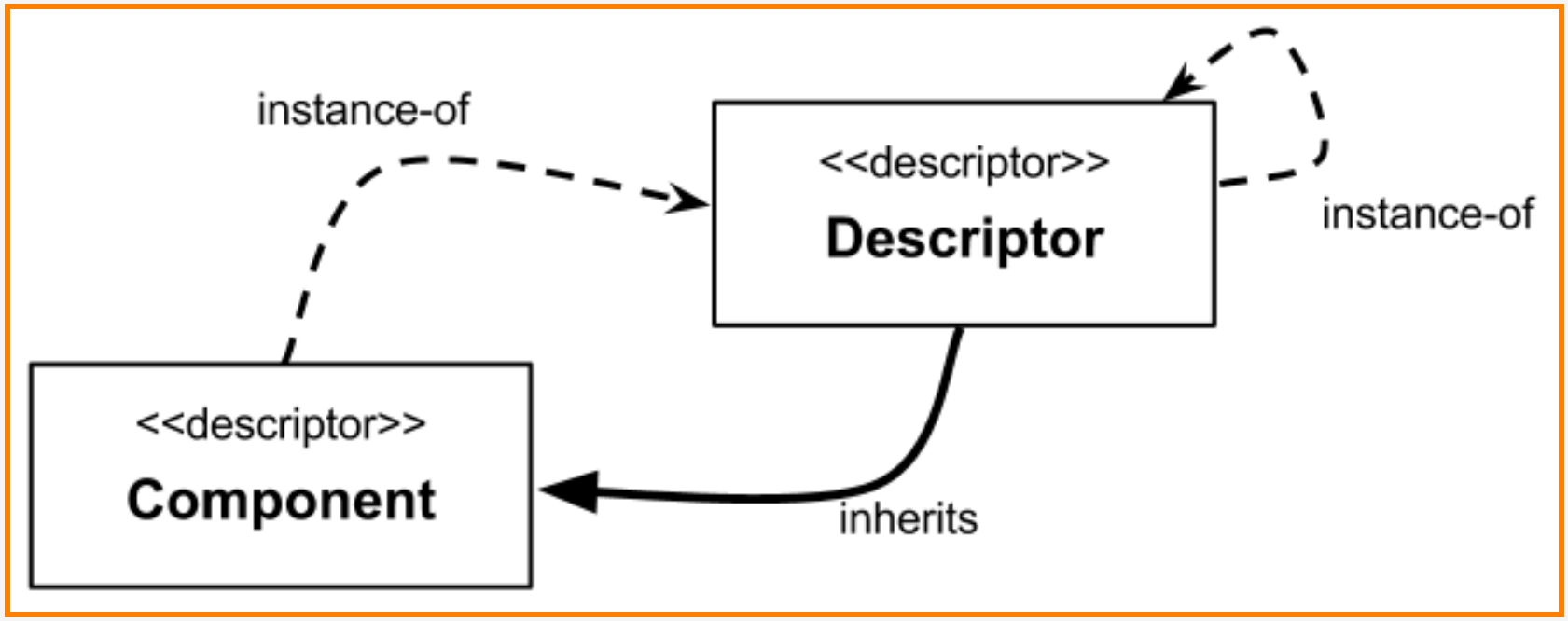


Figure ***(11)*** – *Une application directe de la solution* ObjVlisp *[Cointe - OOPSLA'1987] ou MOF fonctionne ! - Bootstrap.*

4.5 Tout est composant : le cas des ports - #1

Faire le lien avec *First-Class references*.

- **Problème 1** : Une invocation de service s'effectue **via** (déréférencement) un port. Comment invoquer un service **d'** un port ?
- une solution : un opérateur d'accès au port vu comme un composant (operateur &).

```
1  randGen.getRandomNumber();
       #--> un nombre
       aléatoire si le composant
       est connecté

3  &randGen.isConnected();
       //--> false
```

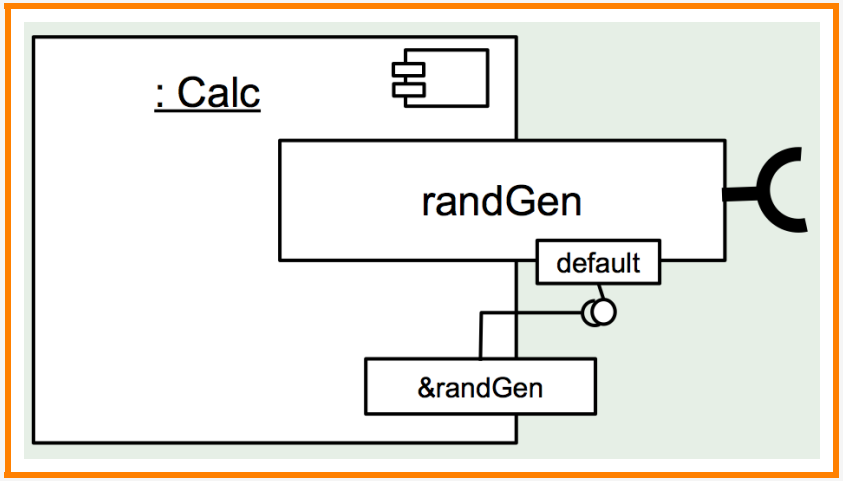


Figure *(12)*

Tout (presque) est composant : le cas des ports - #2

- **Problème 2** : un port réifié (réprésenté comme un composant) possède des ports, etc.
- une solution : les ports des ports sont **primitifs**.
  &randGen est primitif : créé et géré par la VM, non introspectable.

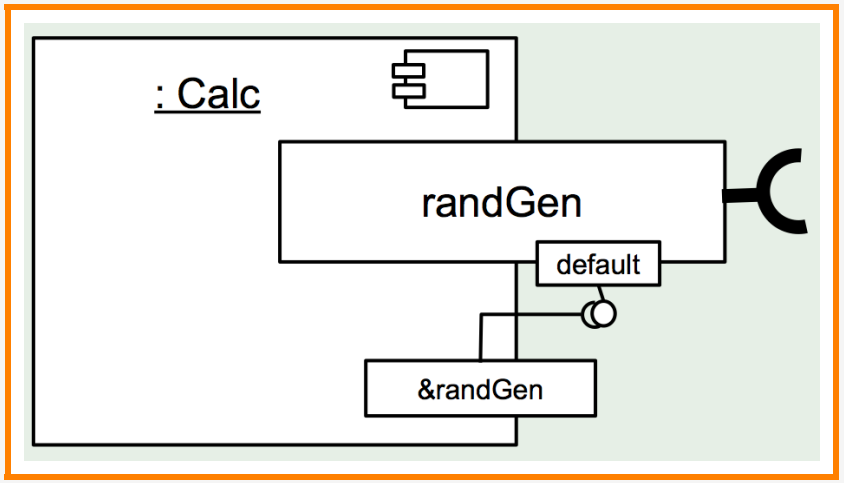


Figure *(13)*

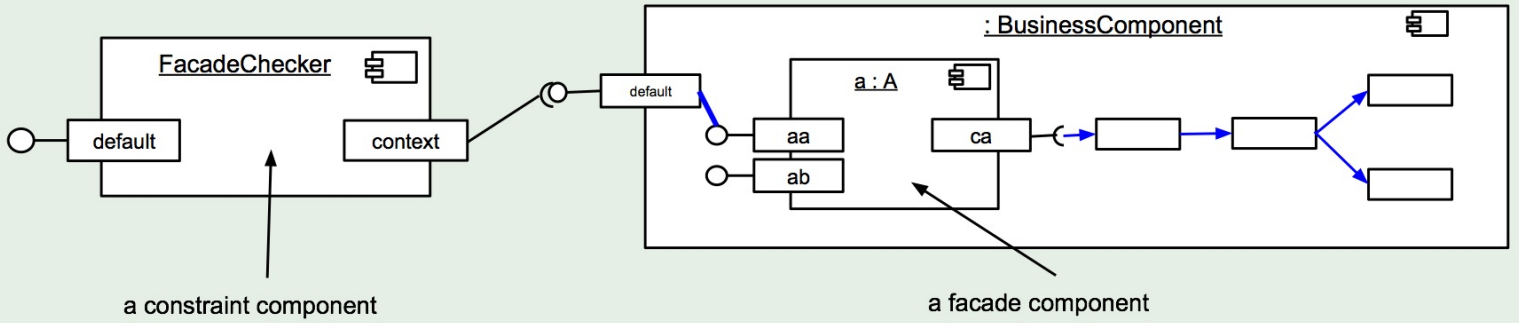
4.6 Exemples d'Application

- Implantation de nouveaux types de ports, *aspectPorts*, *readOnlyPorts*
- implantation de contraintes sur les modèles (type OCL) en COMPO - composants contraintes
- transformations d'architectures ...

4.6.1 Exemple 1 : Vérification de contraintes d'architecture - composants contraintes

- architecture constraints are assertions like:
  - "*is this architecture a bus-like architecture?*"
  - "*are these ports connected?*"
- constraints can be modeled as components[7]

Connecting a constraint component to a business component



[7] Chouki Tibermacine et al. "Component-based specification of software architecture constraints". In: *Proceedings of the 14th CBSE*. Boulder, Colorado,

Figure ***(14)*** – *Un composant contrainte via son port requis* `context` *peut accéder à et tester le méta-niveau (le descripteur, l'architecture) du composant auquel il est connecté.* (`Extrait` *soutenance Petr Spacek)*

4.6.2 Exemple 2 : Transformation d'architecture

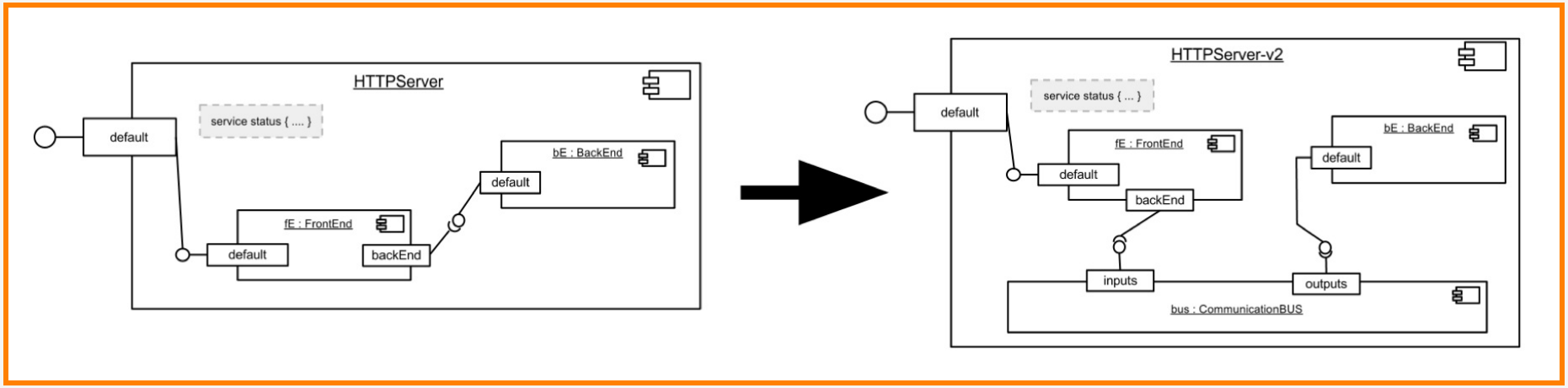


Figure *(15)* – *transformation de l'architecture définie par le descripteur HTTP-Server en une architecture avec BUS.*

Applicable statiquement ou à l'éxécution [a].

```
1  transformer connectTo ToBusTransformer.new();
2  connect target@transformer to default@HTTPServer;
3  transformer.step1-AddBus();
4  ...
```

a. Si la transformation est faite à l'exécution, il faut mettre à jour les instances.

```
1  Descriptor ToBusTransformer {
2   requires { target : IDescriptor }

4    service step1-AddBus() {
5      requires portDesc;
6      requires cd;

8      connect portDesc to PortDescription.new();

10     portDesc.setName('bus');
11     portDesc.setRole('required');
12     portDesc.setVisibility('internal');
13     portDesc.setInterface('IBus');
14     target.addPortDescription(portDesc);

16     connect pc to connectionDescription.new();
17     cd.setSourcePort('bus');
18     cd.setSourceComponent('self');
19     cd.setDestinationPort('default');
20     cd.setDestinationComponent('Bus.new()');

22     target.addConnectionDescription(cd);
23 }
```

4.6.3 Exemple 3 : modification et extensions du méta-niveau

Exemples : ReadOnlyPorts

ou

AdvicePorts (en attendant des aspects)

```
1  Descriptor AdvicePort extends RequiredPort {
2      requires { advicer : {before(); after(); }}
3
4      service invoke(service) {
5          aspectComp.before();
6          super.invoke(service);
7          aspectComp.after();
8      }
9  }
```

5 Implantation

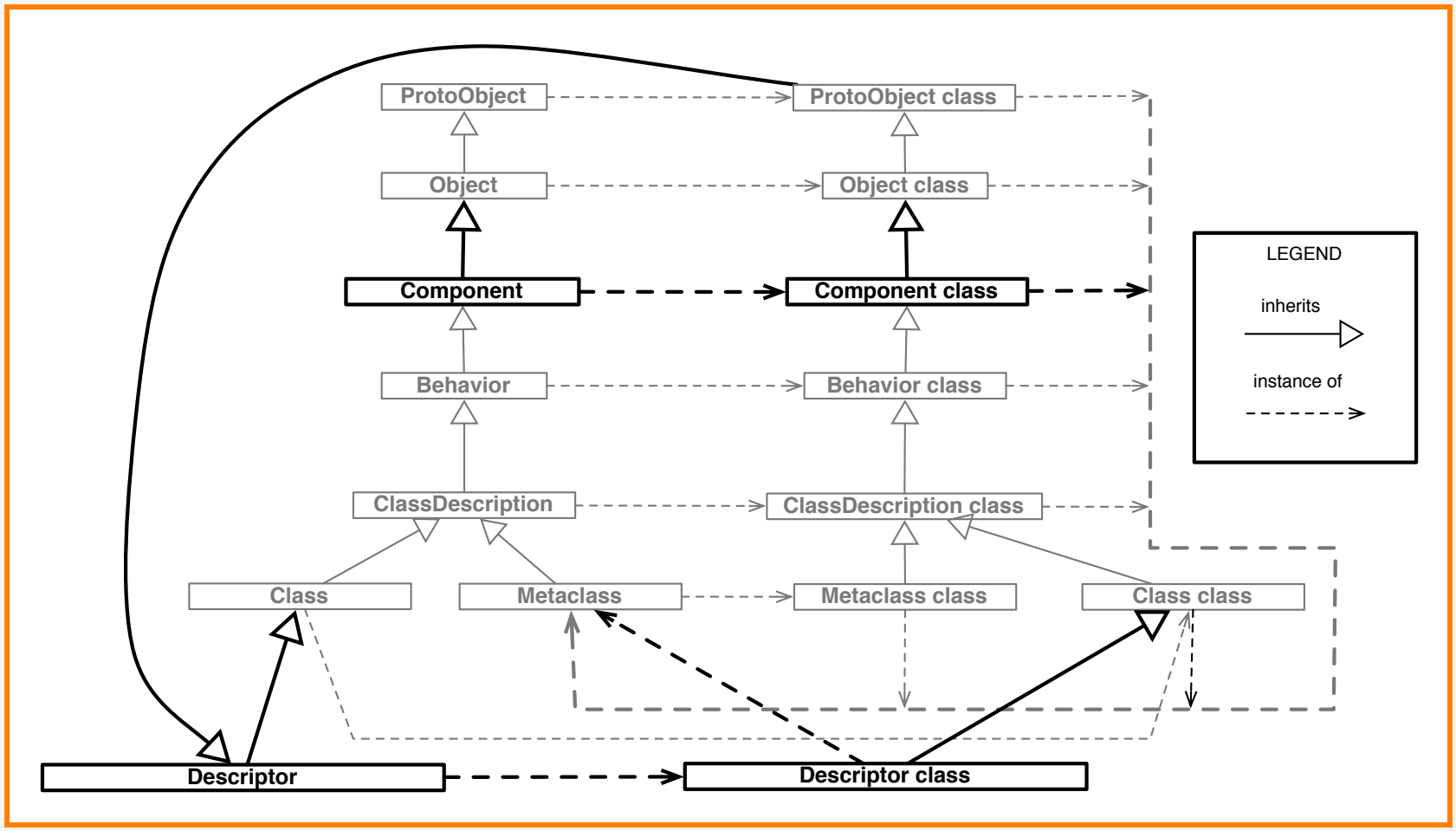


Figure *(16)* – *Clé de l'implantation actuelle : intégration dans le système de méta-classes de Smalltalk-80 (schéma ClassTalk[Briot-Cointe - Oopsla'89]. Prochaine étape : Combonent-oriented Virtual Machine ?*

6 Bilan, Perspectives

Un langage réfexif de programmation et modélisation par composants autorisant la vérification et transformation des modèles et des programmes statiquement ou à l'exécution.

- C'est partiel (intégration de beaucoup de choses a minima) mais c'est donc faisable !
- Un pont entre communautés (ADLs, Langages réflexifs)
- L'explicitation du requis, des architectures, ... est (me semble-t-il) un modèle d'une évolution en cours : voir le @Component de Spring ([http ://howtodoinjava.com/2015/01/23/how-to-use-spring-component])

```
1  @Component
2  public class EmployeeDAOImpl implements EmployeeDAO {
3      ...
4  }
```

- Pose de façon explicites diverses questions, par exemple :
  - ajoût de requis sur un sous-descripteur (traitée dans l'exposé)
  - pourquoi et comment passer un composant en argument ? (travail en cours avec David Delahaye et Chouki Tibermacine) ?
    qu'est-ce qu'un composite ? (En Java, passer en argument un objet référencé privé)
    Comment garantir à l'architecte que sa substitution d'un composant par un autre dans l'architecture sera effective ?
    Que signifie passer "par fourni" ou "par requis" (lien avec les modes de passage classiques) ?
- Beaucoup de choses à faire :

- Traiter la partie dynamique des architectures.
- Ecrire un JIT Compiler et une machine virtuelle réflexive à composants (discussions avec Luc Fabresse)
- Intégrer plus d'abstraction - Architecture logicielle à trois niveaux (discussions avec C. Urtado, M. Huchard),
- Faire le lien conceptuel avec l'approche packaging (un bundle OSGI versus un composant Compo) - Collaboration avec D. Seriai.
- Etablir la sémantique formelle de la version non réflexive (travail en cours avec David Delahaye)
- Machine virtuelle réflexive à Composants (Petr Spacek - Czech Technical University)
- Voir Compo comme un environnement de prototypage - générer du fractal ADL + IDL + OCL + ATL + Java à partir d'un programme Compo
- Passer des composants en argument ...
- passer par requis
- passer par fourni
-