Université Montpellier-II
Faculté des Sciences - Département Informatique
Master Informatique

(Modularité et Réutilisation : les composants logiciels).

Introduction

Notes de cours

Christophe Dony

1 Contenu du cours

- Suite sur les schémas des réutilisation objet et sur l'architecture des lignes de produits (frameworks).
- Limites de l'approche objet. Présentation de l'idée de composant.
- Diverses interprétations possibles de l'idée de "développement par composants".
- Le concept d'aspect, orthogonal à celui d'objet ou de composant. (optionnel)
- Composants assemblales. L'exemple des Java-Beans. Etude des schémas de connexion par évènements associés (*Observer*, *Adapter*, application au *MVC)*. Etude des différents types de propriétés typiques (propriétés liées).
- Composants distribués. L'exemple des composants EJB dans la technologie JEE :
  - *session bean* pour traiter les requètes metier des clients,
  - *message-driven bean* pour gérer l'asynchronisme des communications,
  - Abstraction de la persistance et des transactions
  - Architecture globale d'une application JEE de base avec JSP et Servlet.
- Composants applications WEB :
  - composants et réutilisation avec le framework OSGI/Spring.

- La gestion des dépendances (dont l'injection) avec les composants SPRING et OSGI (Open Service Gateway Initiative),
- ...

2 Rappel sur les schémas de réutilisation

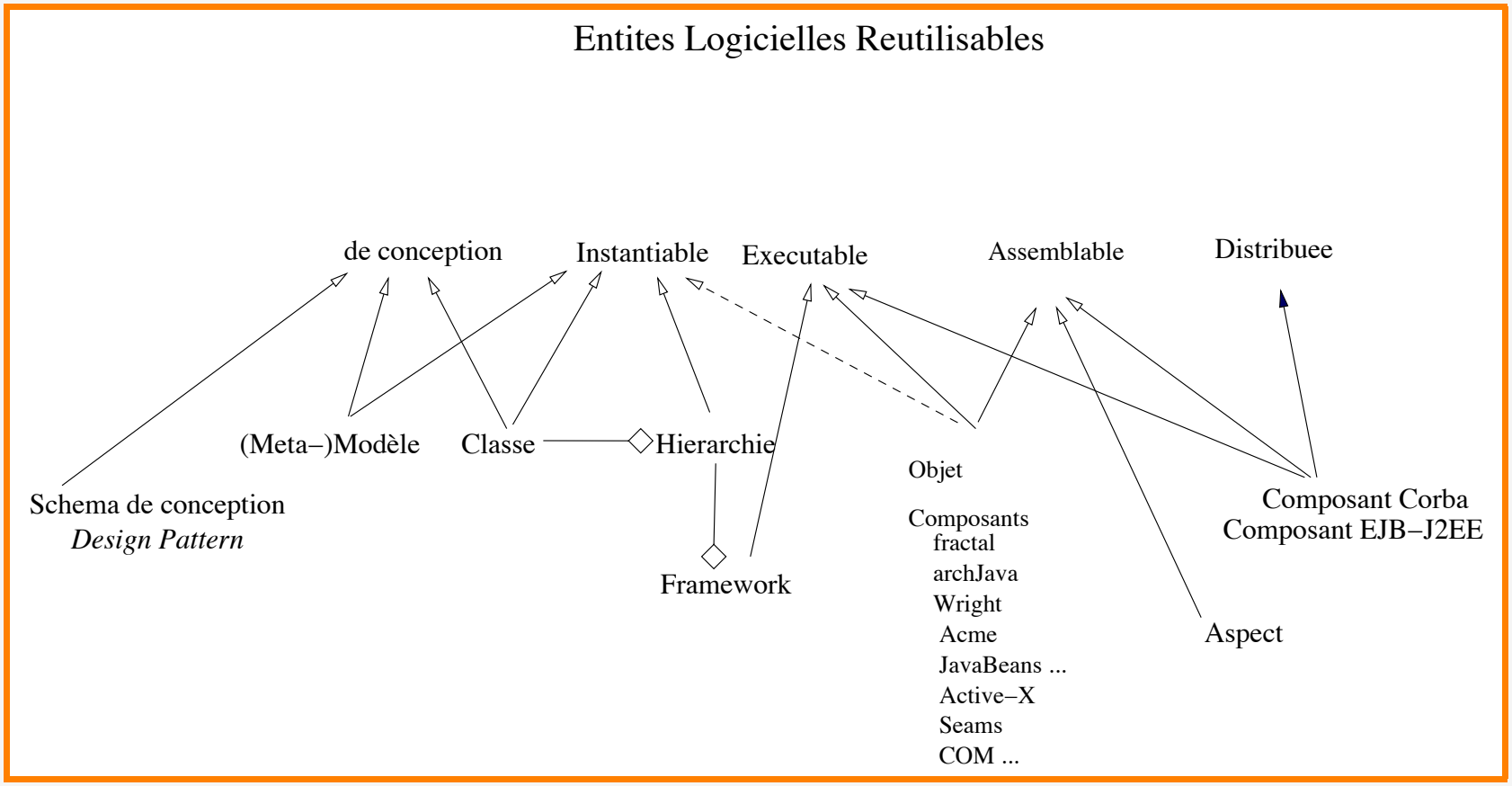

Figure (1) *– Les entités réutilisables du génie logiciel à objets*

La programmation par objets introduit des schémas :

- d'extension (description différencielle, héritage)

- de paramétrage : passage de fonctions (ordre supérieur) en argument via les objets et liaison dynamique.

Objet : encapsulation de données dotées d'un ensemble de fonctions.

Passer un objet en argument, c'est passer un ensemble de fonctions.

2.1 Paramétrage par Spécialisation

```
1  public interface I {
2    int m1(); }

4  public abstract class A implements I {
5      int attribut;
6      public int m1() {
7         if (this.m2(this))
8           return attribut + this.m3();
9           else return attribut + this.m4();
10     }
11     public boolean m2(A a) {return true;}
12     public abstract int m3();
13     public abstract int m4();
14 }
```

Listing (1) –

```
1  abstract class Produit{
2    protected int TVA;

4    int prixTTC() {
5      return this.prixHT() * (1 + this.getTVA())}
6    abstract int prixHT();
7    int getTVA() {return TVA;}}

9  class Voiture extends Produit {
10     int prixHT() {return (prixCaisse()+prixAccessoires()+ ...)} ... }

12 class Livre extends Produit {
13     protected boolean tauxSpecial = true;
14     int prixHT() {...}
15     int getTVA() {if (tauxSpecial) return (0,055) else return (0,196);}
```

Listing (2) – *Exemple de paramétrage par spécialisation*

2.2 Paramétrage par composition

Code générique (méthode *service*) adapté par un plugin défini sur un composite.

```
1  class Compiler{
2    Parser p;
3    CodeGenerator c:
4    Compiler (Parser p, CodeGenerator c){
5      this.p = p;
6      this.c = c;}

8    public void compile (SourceText st)
9      AST a = p.parse(st);
10     generatedText gt = c.generate(a);}
```

2.3 Applications

- **API**, **Bibliothèques de classes**
  Ensemble de classes spécifiques d'un domaine.
  Ex l'API *java.lang.io* ou les bibliothèque de *Collection*.
- **Hiérarchie de classe**
  Une hiérarchie de classe est une bibliothèque thématique et extensible.
  Article historique : *R.E.Johnson, B.Foote : Designing reusable classes. Journal of Object-oriented Programming, Vol 1, No 2, pp 22-35, July 1988.*

- **Frameworks - Lignes de Produits**

  Application pré-packagée spécialisable et Adaptable selon divers schémas (spécialisation, composition, fonctions d'ordre supérieur, descripteurs) pour en faire des applications spécifiques.

  Invention du pattern **Inversion of Control** ou **Hollywood Principle**.

- **Plugin**

  Entité destiné à paramétrer un framework.

  Exemple courant : l'environnement de développement *ECLIPSE* est extensible en divers points bien définis de son code par des *plugins* permettant d'étendre ou d'adapter ses fonctionnalités.

3 Limites de l'approche objet

... software reuse in the large has never been achieved by object-oriented development ... W. Emmerich *Distributed Component Technologies and their Software Engineering Implications - ICSE'2002.*

3.1 Le couplage

Le couplage est la mesure du degré d'interdépendance entre différents modules logiciels, de la possibilité d'utiliser l'un sans l'autre ou d'avoir une alternative.

Un bon logiciel doit avoir un couplage faible.

Problème du Couplage fort (ou explicite)

Le couplage explicite dénote une référence explicite, dans un élément logiciel A, via son nom ou son adresse, à un autre élément logiciel B nécessaire à A.

Exemple avec des classes :

```
1  class A{
2    B b = new B();
3    public ma(){ b.mb(); }

5  class B{
6    public mb(){ ... }}
```

Problèmes :

- Impossible de voir que A est dépendant de B sans regarder le code de A.
- Impossible pour une application de réutiliser A avec autre chose que B, sans modifier le code de A.
- Difficulté ou impossibilité à réaliser l'**inversion de contrôle** (Hollywood Principle - voir Frameworks).

Solution : **couplage faible** (*loose coupling*) (ou a posteriori) :

3.2 Solutions objet pour le couplage faible

3.2.1 Injection de dépendances

Injection de dépendance : Passage à A, au moment de la configuration de son utilisation, des références vers les éléments qui lui sont utiles

Trois solutions (complémentaires) pour l'injection de dépendances :

- Injection par les constructeurs
- Injection par accesseurs en écriture
- Injection par interface

Exemple : injection de dépendance en paramétrage par composition.

```
1  class A{
2    IB b;
3    public void setB(IB b){this.b = b;}
4    public int ma(){ return (1 + b.mb()); }

6  interface IB {public mb();}

8  class B implements IB {public int mb(){ return(2); }}

10 class Application{
11   public main () {
12     A a = new A();
13     Ib b = new B();
14     a.setB(b);
15     a.ma();
```

L'injection de dépendance préfigure la connexion externe des langages à composants dont elle constituera une des solutions.

3.2.2 Le schéma *Fabrique (Factory)*

Le schéma *factory* donne un nom à une solution pour l'injection de dépendance en paramétrage par spécialisation utilisée depuis l'invention du schéma *MVC* (1977).

> Factory methods are common in toolkits and frameworks where framework code needs to create objects of types which may be subclassed by applications using the framework. *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides : Design Patterns : Elements of Reusable Object-Oriented Software Addison Wesley, 1994.*

> Subclasses of *View* in Smalltalk-80 specify `defaultControllerClass` as the factory method that subclasses should override and which returns a class used by the View class, `defaultController`, to create instances.

Exemple en typage dynamique (langage hypothétique, syntaxe à la Java) :

```
1  Class View {
2    ...
3    Constructeur(){
4       myController := self defaultControllerClass new
5       ...}
6  }

8  Class MyApplicationView extends View {
9     méthode defaultControllerCLass(){
10       return MyApplicationController; }
11    ...
12 }

14 Class MyApplicationController { ... }
```

3.3 Cohésion

La cohésion est la mesure du degré de proximité entre les éléments logiciels nécessaires à la réalisation d'une tâche donnée.

Comment regrouper de façon appropriée des objets qui réalisent une fonctionnalité globale.

3.4 Problème de la cohésion faible

Les langages fonctionnels ou à objets n'offrent à la base pas de solutions pour la cohésion.

Les packages où les modules sont des unités de modularité mais pas de cohésion.

Extension aux objets : les paquets (jars, bundle, ...) permettant de sauvegarder et transférer un ensemble d'éléments logiciels disparates

3.5 Synthèse

voir “cohésion et découplage”.

3.6 Problème de l'anticipation

Dans l'exemple de la programmation par évènements, comment décider à l'avance quels évènements doit émettre un objet ? Comment deviner quel évènement va intéresser un client ?

La non anticipation consisterait à laisser un utilisateur d'un objet décider a posteriori et sans modifier le code de l'objet, quels évènements, relatifs à la vie de l'objet, l'intéressent.

La **programmation par aspects** et par extension la programmation par annotations propose une solution pour la non anticipation.

3.7 Problème de l'expression du requis

- En programmation par objets, seul le fourni est (parfois) spécifié "explicitement", pas le requis.

```
1  class Line {
2    Point p1, p2;
3    public Point getP1() { return p1; }
4    public Point getP2() { return p2; }
5    public void setP1(Point p1) { p1 = p1; MoveTracking.setFlag(); }
6    public void setP2(Point p2) { p2 = p2; MoveTracking.setFlag(); } }
```

La découverte du requis nécessite une fouille code. Plus simple en typage statique que dynamique.

3.8 Problème de l'expression des architectures

Les architectures de composition ne sont pas explicites. Leur perception ou modification nécessite d'avoir accès à des documents de modélisations (diagramme de classes UML) et/ou la lecture du texte du programme.

Un diagramme de classe UML peut être vu comme un modèle ou une description de l'architecture "objet" d'un programme.

Mais ce diagramme ne fait pas partie du programme.

Il est imprécis, par exemple il n'indique pas le nombre d'objets crées ni leur distribution.

Difficile de modifier de telles architectures une fois le code écrit.

3.9 Problèmes de séparation des aspects fonctionnels et non fonctionnels

- **Aspect ou Préoccupation** : point particulier de la réalisation d'un logiciel.

- **Aspect fonctionnel** : aspect relatif aux fonctionnalités que doit fournir le logiciel à ses clients.

- **Aspect non fonctionnel (ou technique)** : aspect relatif à un problème annexe dans la réalisation du logiciel.

- Problème logiciel : **séparation des préoccupations**

nombreux aspects non fonctionnels (applications internet ou client-serveur d'envergure) :

- la distribution, la concurrence, la persistance, le déploiement des objets et des fonctions, le transactionnel, etc

Solutions pour la séparations des préoccupations

- La programmation par aspects. Chaque fonctionnalité accessoire peut également s'adapter à diverses fonctionnalités essentielles.
  Exemple : aspect *Synchronized* d'une méthode Java.

```
1  class CompteBanque

3     float solde = 0;

5     public void synchronized retrait(float montant)
6           throws CreditInsuffisant {
7        if (solde >= montant)
8           solde = solde - montant;
9           else throw new CreditInsuffisant();}

11    ...
```

- Les composants (aspects fonctionnels) et leurs conteneurs (aspects non fonctionnels).

4 Composant

- Une définition :
  entité logicielle, prête à l'emploi, qui peut
  - être paramétrée de diverses façon pour fonctionner de façon spécifique dans différents contextes,
  - être assemblé de diverses manières avec d'autres composants *compatibles*.
- Une autre définition :

  > "Entité logicielle pré-fabriquée, définie par une ou plusieurs interfaces, qui peut être "composée" de diverses façons à d'autres, dans différents contextes, pour réaliser de nouveaux composites" *Clemens Szyperski. Component Software. Acm Press 2002*

4.1 Classification historique, d'après D. Hoarau

- 1960s Components would be a good idea ... Mass produced software components (M.D. McIlroy 1967)
- 1970s ... 1980s : Modules, Objects ... executable et pipe (unix)
- 1991 ...Composands distribués CORBA 1.1
- 1993 COM : Microsoft Component Object Model. Empaquetage d'un programme dans un composant doté d'interfaces. Précurseur de .NET.
- 1995 CORBA 2.0, DCOM : Communication entre composants distribués - Middleware, Interopérabilité (Interfaces).
- 1997 JavaBeans, ActiveX : Construction de composés par assemblage de composants, shéma "objerver" et "MVC", application à l'interfaçage.

- 1997 ACME (Architecture Description Interchange Language), une tentative de langage minimal pour la description échangeable de descriptions d'architectures.
- 1998 SOFA (Software Applicances) : fournir une plate-forme pour le développement d'applications par "composition" d'un ensemble de composants.
  ADL : Langages de description d'architectures à base de composants.
- 1999 Enter prise JavaBeans (EJB), les composants distribués au dessus de Java.
- 2002 Fractal Component Model, Corba Component Model (CCM)
- 2002 ADL : ArchJava - Connect components - Pass Objects
- 2006 ... Langages de programmation et de modélisation par composants ... Evolution des langages à objets vers les composants ... Nombreuses visions et recherches.

4.2 Idées clé

1. développement par et pour la réutilisation,

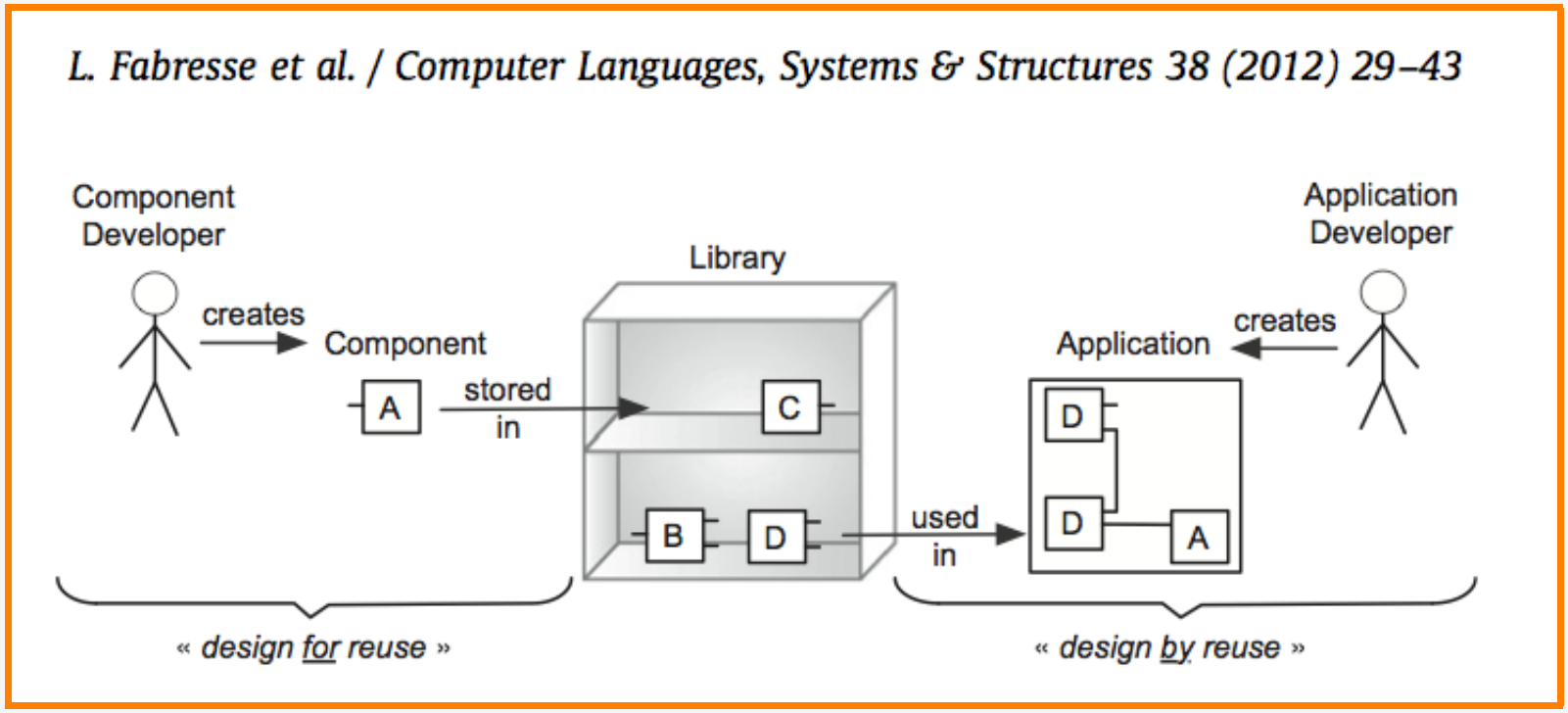

Figure (2) – *Développement pour et par la réutilisation*

2. encapsulation de services, décrit via des interfaces (fourni, requis) (d'utilisation, de configuration)

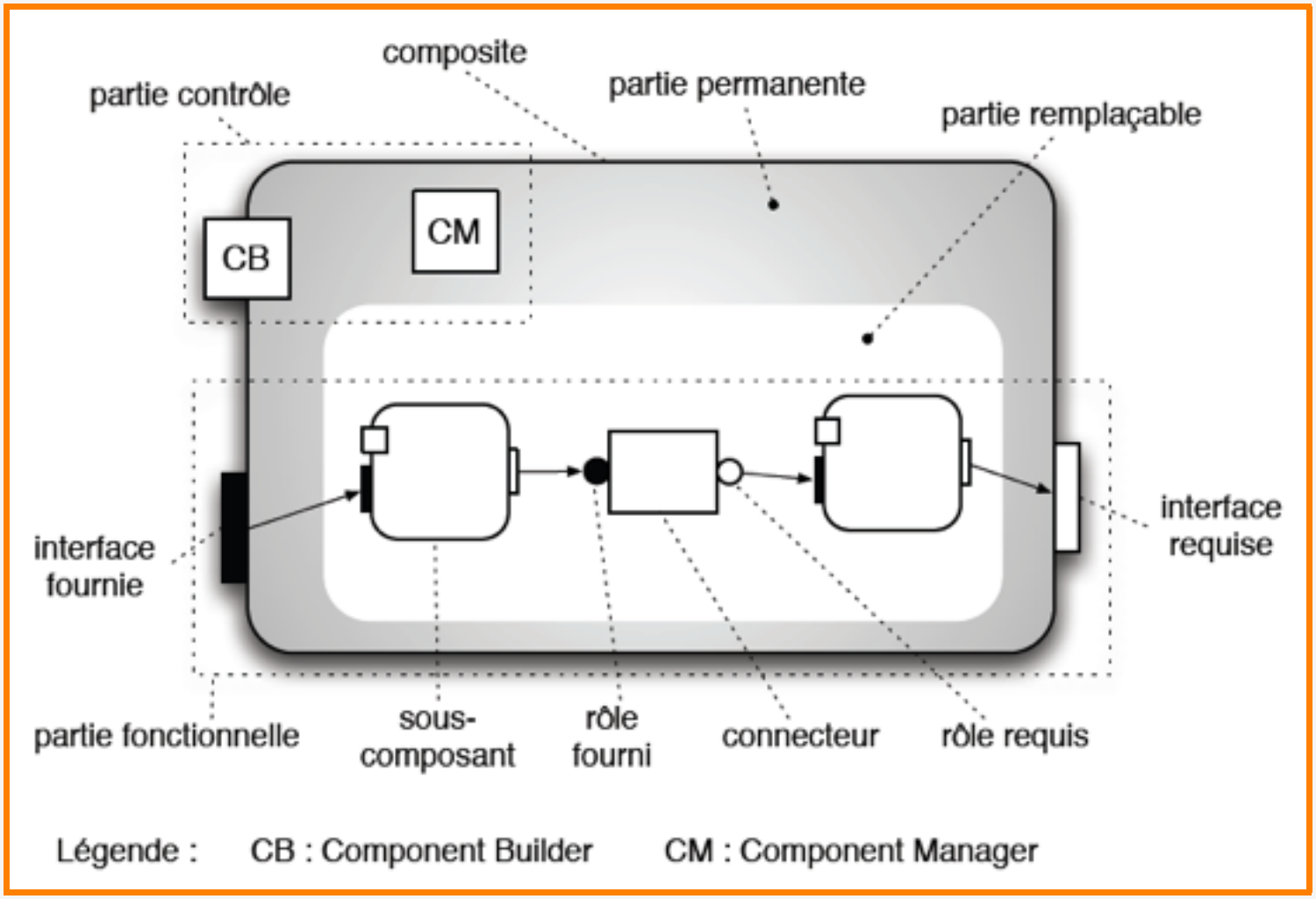

Figure (3) – *Expression du fourni et du requis, composition structurelle, exemple avec composant* `SOFA`

3. expression des architectures : composition structurelle et connections

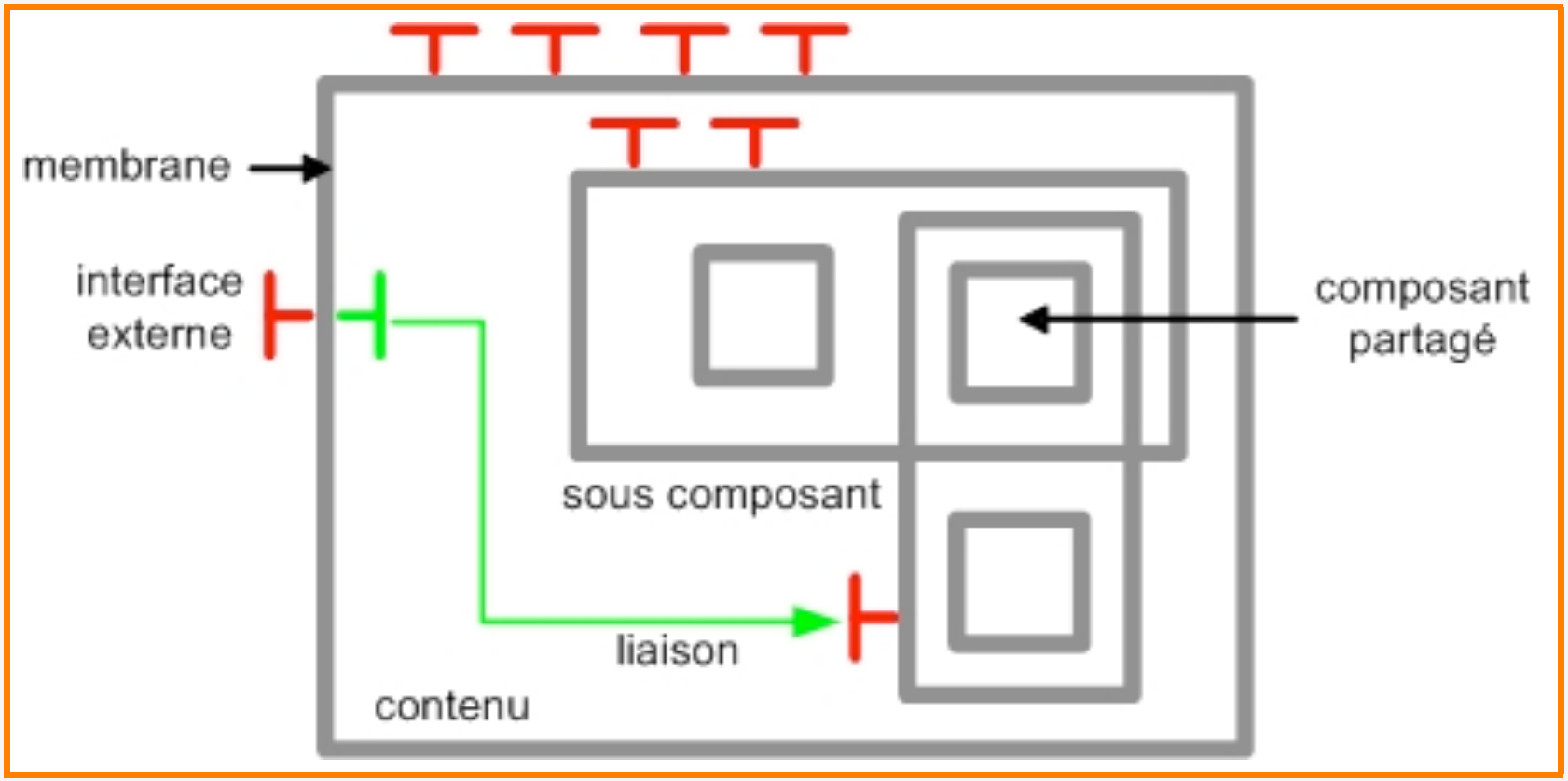

Figure (4) – *Autre exemple avec un composant* `Fractal` *(http ://wordpress.pragmaconsult.lu/wp-content/uploads/2008/07/fractal-component.jpg). Voir aussi http ://fractal.ow2.org/.*

Classification sémantique (d'après Spacek'13 - thèse de doctorat)

Domain	AUTOSTAR	BIP	BlueArX	CCM	COMDES II	CompoNETS	EJB	Fractal	KOALA	KobrA	IEC 61131	IEC 61499	JavaBeans	MS COM	OpenCOM	OSGi	Palladio	PECOS	Pin	ProCom	Robocop	RUBUS	SaveCCM	SOFA 2.0
General purpose				X		X	X	X		X		X	X	X		X		X						X
Specialized	X	X	X		X				X		X	X				X		X		X	X	X	X	

Figure (5) – *Le développement par composants est en phase d'émergence.*

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

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

   Exemples : UML Component Model, ACME, AeSop, Darwin, Rapide, SCA, Wright, xADL, SOFA, FRACTAL, AOKell, DiaSim...

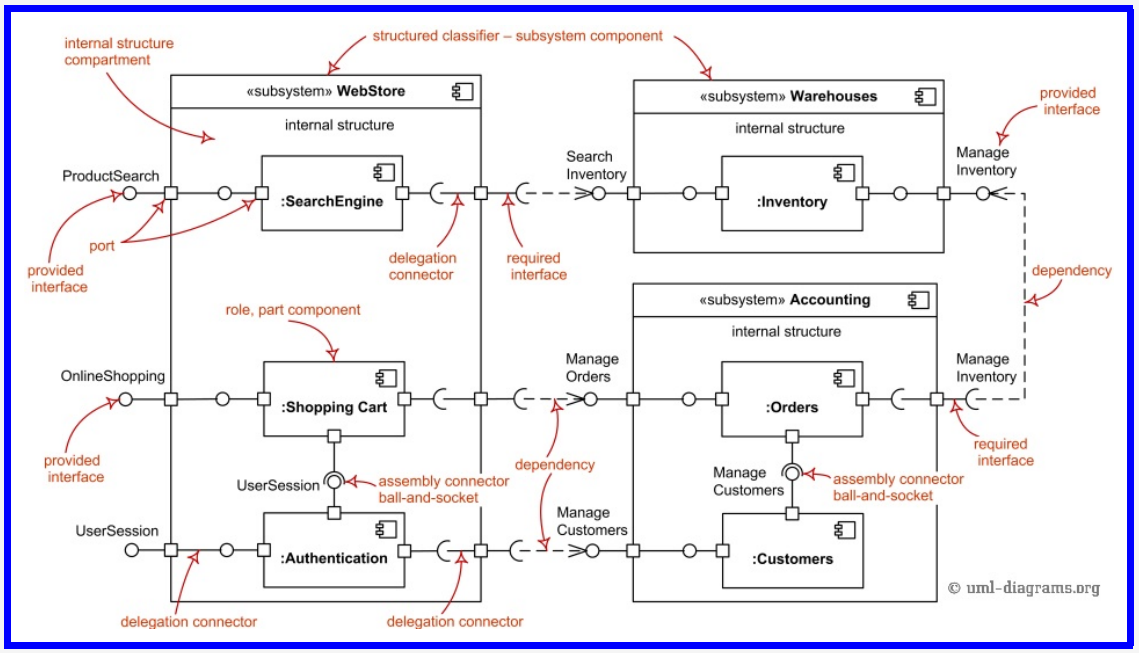

Figure (6) – *Exemple (basique) d'architecture (ici avec UML composents) [from http ://www.uml-diagrams.org/]*

(a) description de haut niveau (modélisation) de l'organisation d'un logiciel comme une collection de composants connectés.

(b) génération de tout ou partie (squelettes) du programme exécutable écrit dans un langage de programmation standard.

2. **L'approche Framework**

   Différents frameworks proposent un cadre pour les applications distribuées vues comme la collaboration d'un ensemble de composants.

   Le framework propose une gestion automatisée d'un ensemble de services non fonctionnels tels que l'interfaçage graphique, la distribution, la persistance ou les transactions.

   Les parties fonctionnelles des applications sont encodées dans les composants et les parties non fonctionnelles sont fournies par les conteneurs dans lesquels les composants peuvent être placés.

   Exemples : COM+, CORBA, EJB, DCUP SOFA, etc.

   Les services WEB entrent conceptuellement dans cette catégorie.

   Un ADL peut être associé au framework, exemple SOFA ou CCM (Corba Component Model).

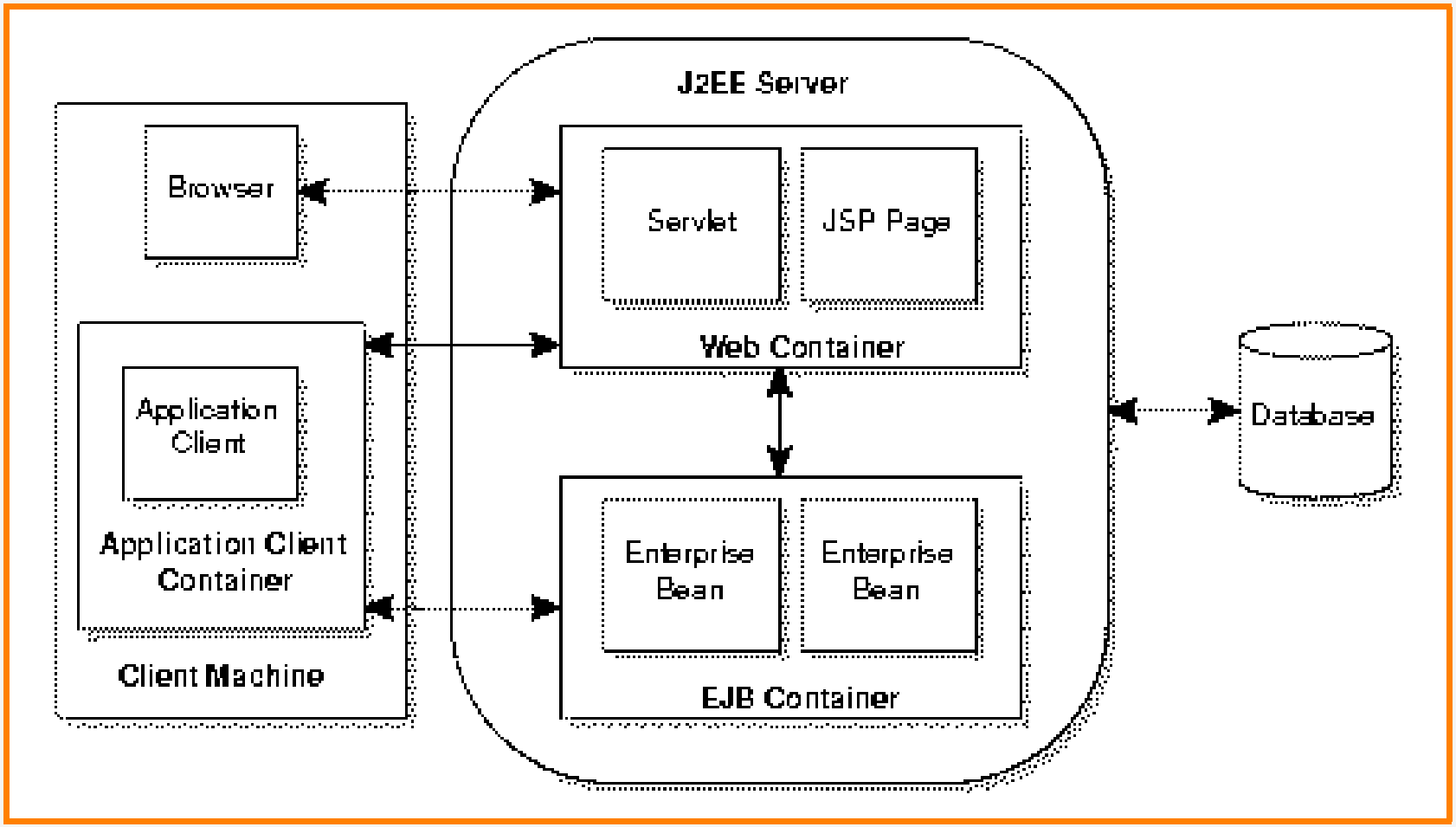

Figure (7) – *L'approche Framework : Conteneurs et Composants EJB (Framework JEE) ©Sun*

3. **L'approche Langage de Programmation par Composants (COLs - Component-Oriented Languages)**

   Propose de nouveaux langages qui sont à la fois des langages de descriptions d'architectures, des langages de modélisation (lien avec IDM) et des langages de programmation par composants ...

   Exemples : ACOEL, ArchJava, ComponentJ, SCL, COMPO, etc.

   ArchJava : premier langage de programmation offrant l'instruction **connect**.

   J. Aldrich, C. Chambers, and D. Notkin. **Archjava** : connecting software architecture to implementation. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, USA, 2002. ACM.

Figure 8 COL : composants, ports, connecteurs, interfaces, composites, protocoles d'interaction (requête-réponse, diffusion-abonnement (*publish/subscribe*) ...

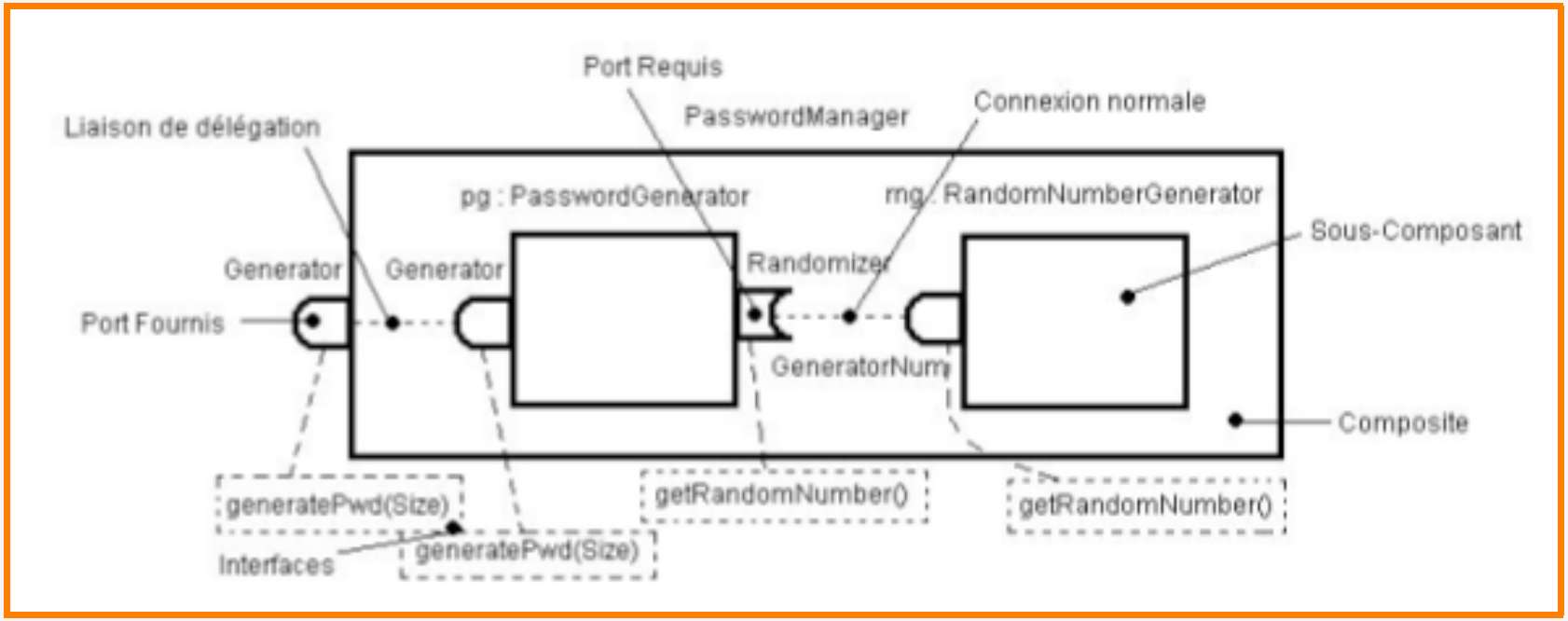

Figure (8) – *Concepts clés - Luc Fabresse - thèse de doctorat Université Montpellier-II 2008*

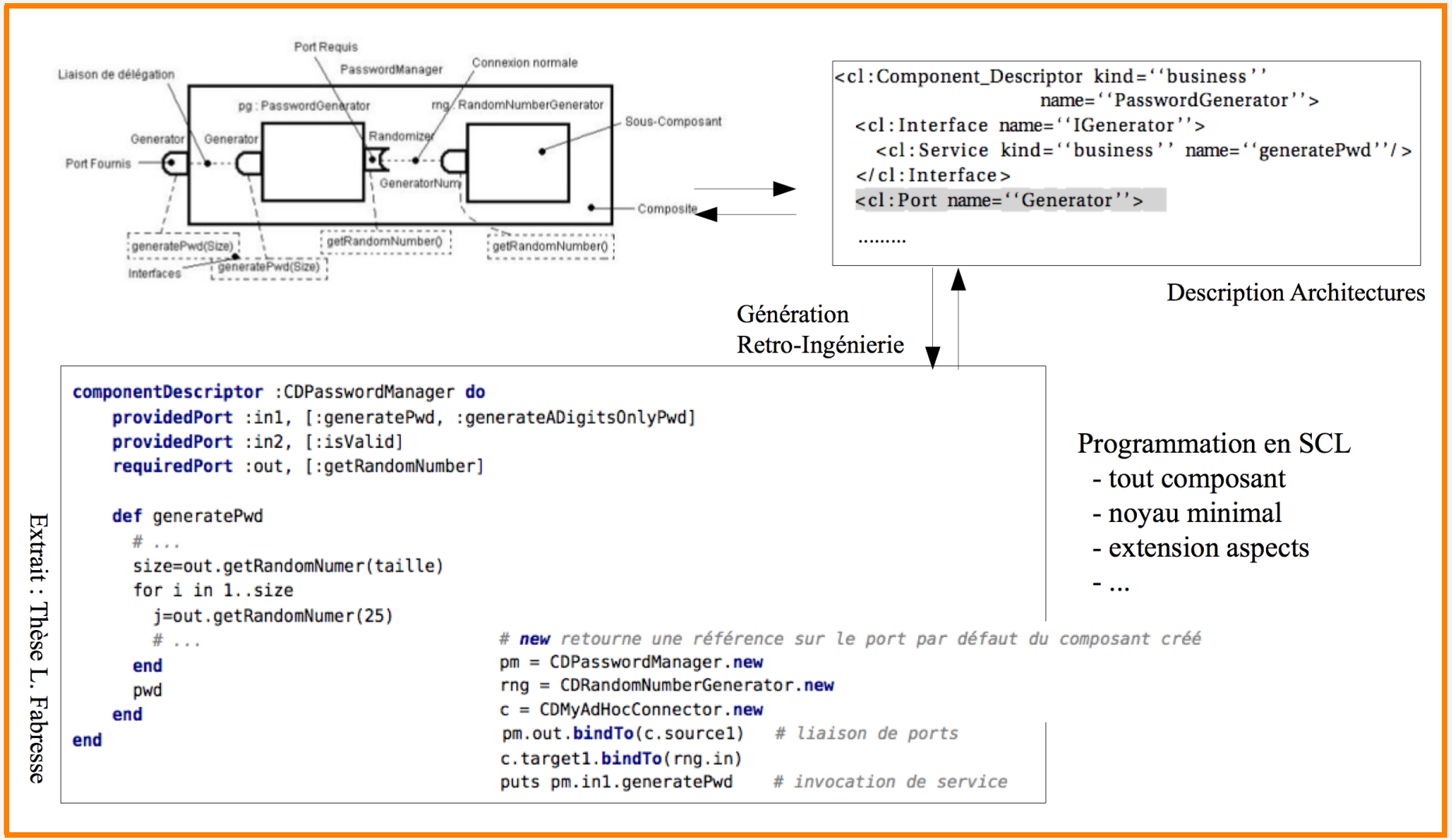

Figure (9) – *L'approche COL : Description d'architecture et Code des Composants en SCL*