

Approches de Re-architecturisation des systèmes existants

1) Migration des systèmes orientés objets existants vers des systèmes à base de composants

2) Restructuration des systèmes à base de composants

Abdelhak-Djamel Seriai

Maitre de conférences

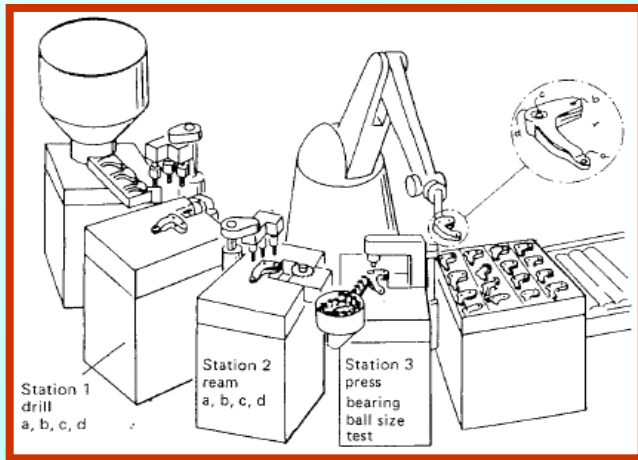
LIRMM/Université de Montpellier 2

Contact : seriai@lirmm.f

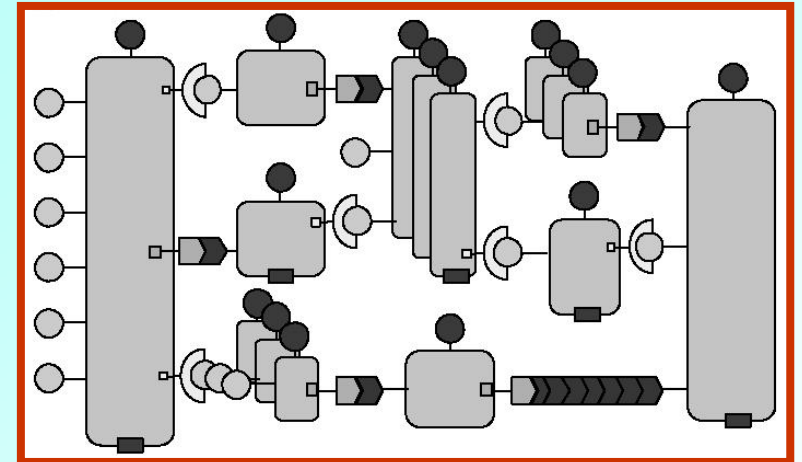
Introduction

□ Approche « composant logiciel»

Approche facilitant la réutilisation, l'évolution et la maintenance de logiciels selon une idée **analogue à celle des composants en industrie électronique ou mécanique**



Composants et assemblage mécanique

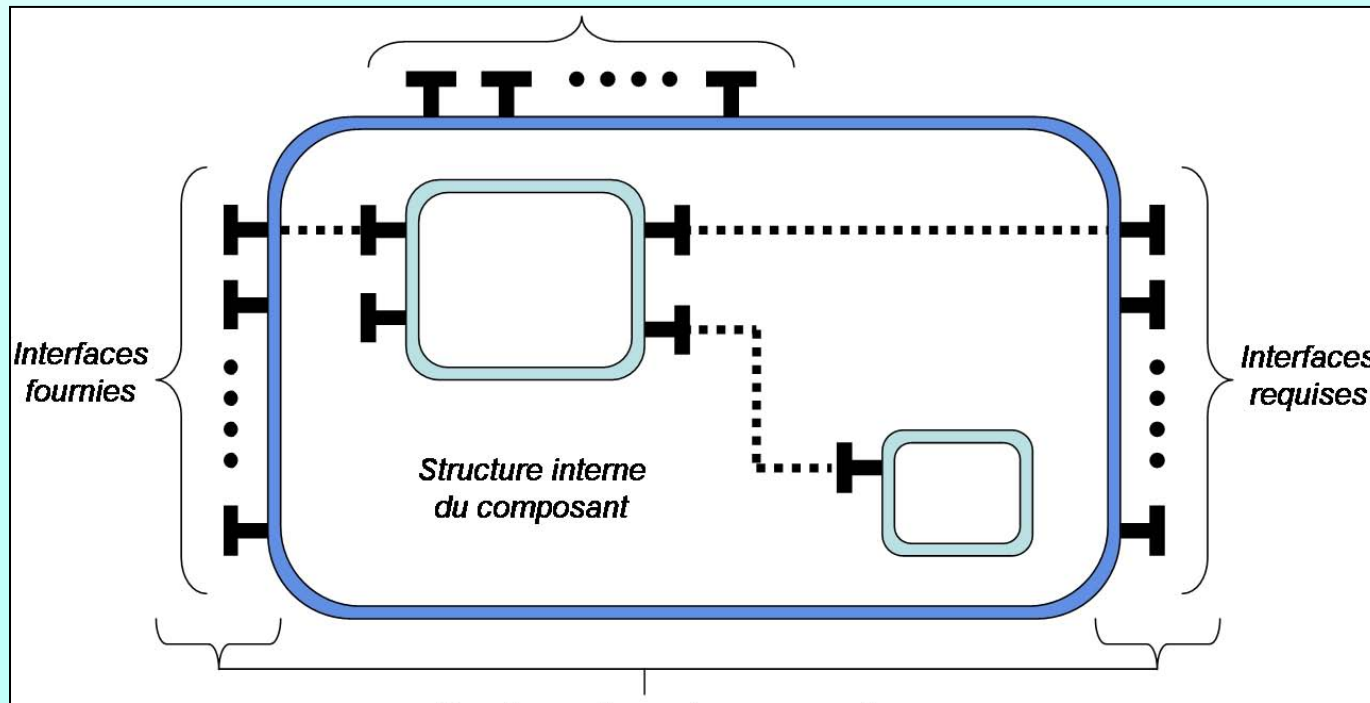


Composants et assemblage logiciel

Introduction

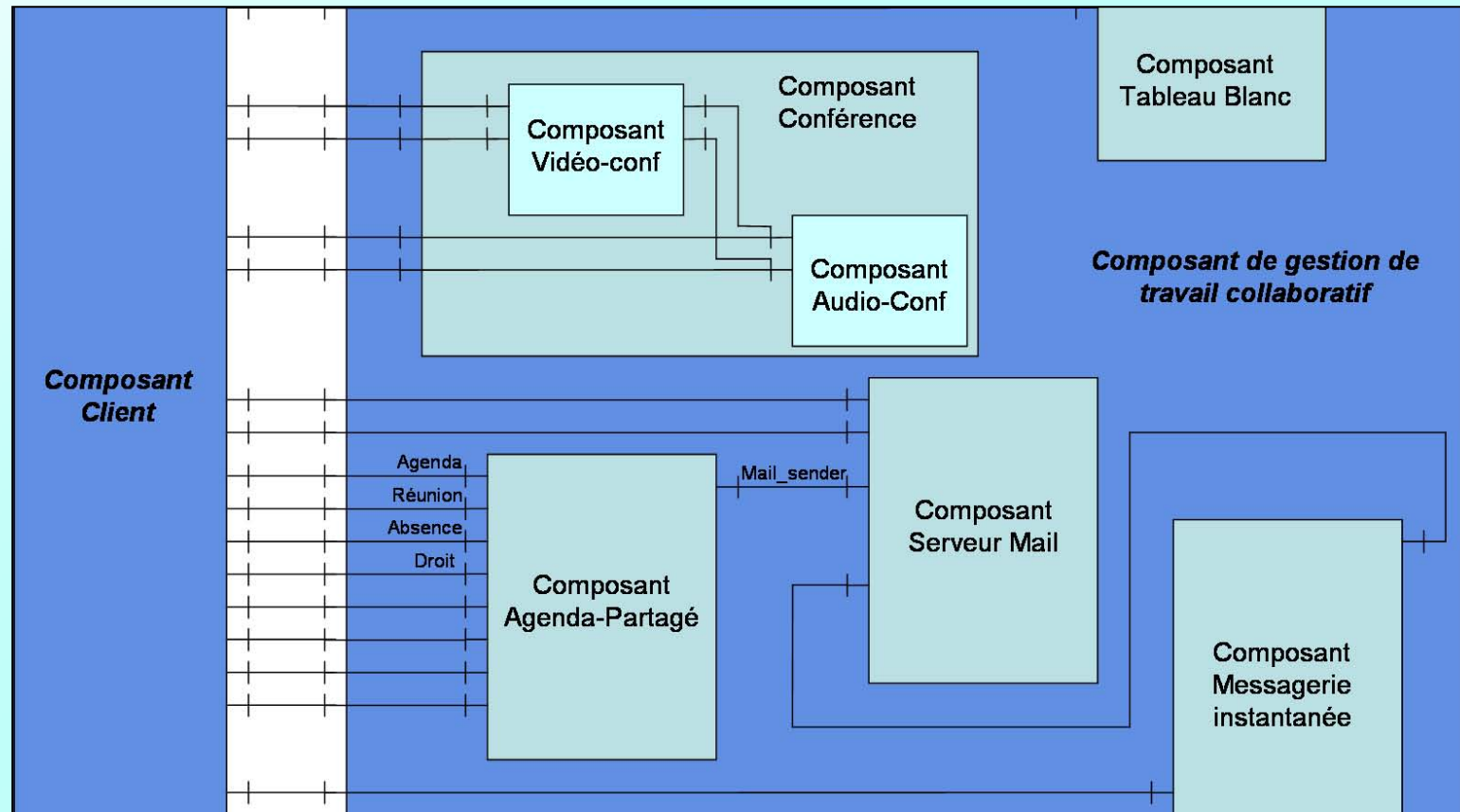
■ C'est quoi un composant ?

Une unité de composition possédant des interfaces spécifiées par contrat et des dépendances explicites avec le contexte. Il peut être déployé indépendamment et peut être composé par un tiers



Introduction

- Application conçue à base de composants
 - **Assemblage de composants logiciels**

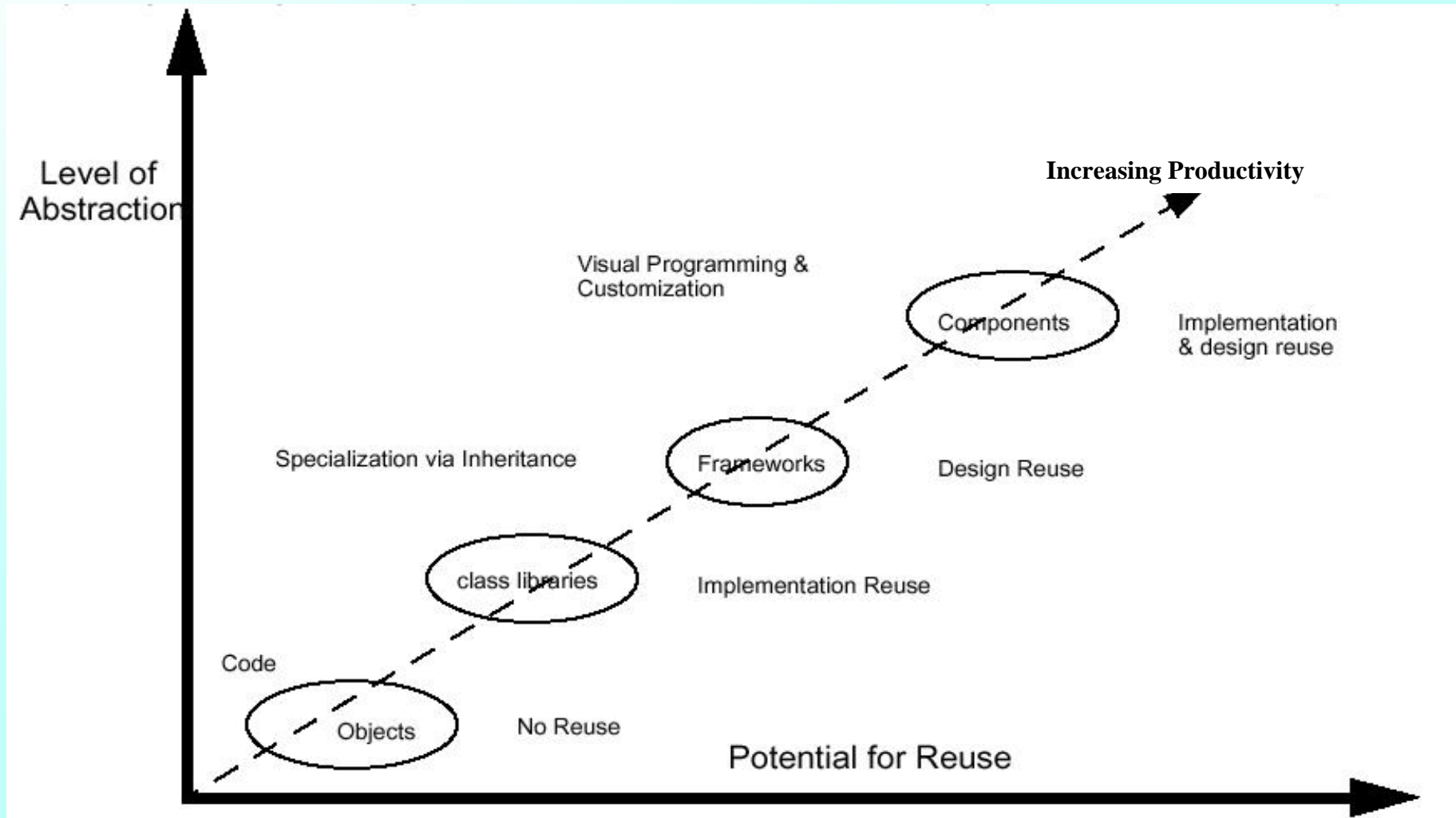


Introduction

■ Avantages des composants

- Facilite la réutilisation, l'évolution, la compréhension, l'intégration, etc.
 - Le composant est une unité de construction à gros grains
 - Encapsulation : Interfaces = seule voie d'accès, séparation interface-réalisation
 - Composabilité
 - Dépendances explicites (expression des ressources fournies et requises)
 - Composition hiérarchique (un assemblage de composants est un composant)
 - Capacité de description globale (langage de description architecturale)
 - Adaptation et reconfiguration

Introduction



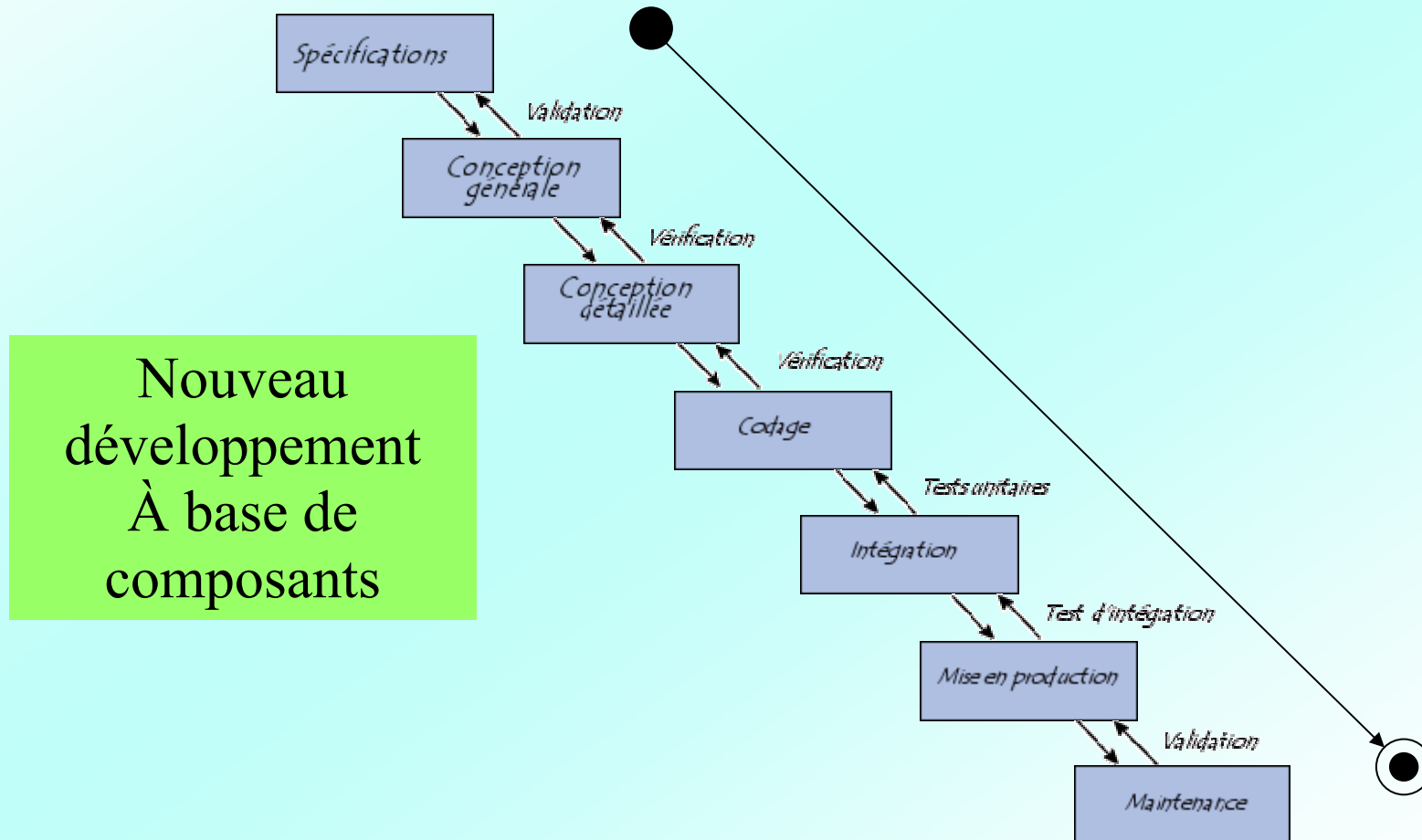
Développement de SBC

■ Cycle de vie d'un logiciel

- ✓ Définition des objectifs : définir la finalité du projet
- ✓ Analyse des besoins et faisabilité : recueil et la formalisation des besoins
- ✓ Conception générale : élaboration des spécifications de l'architecture générale
- ✓ Conception détaillée : définir chaque sous-ensemble du logiciel
- ✓ Codage (Implémentation ou programmation)
- ✓ Tests unitaires
- ✓ Intégration: s'assurer de l'interfaçage des différents éléments (modules)
- ✓ Qualification (ou *recette*): vérification de la conformité du logiciel aux spécifications initiales
- ✓ Documentation: produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs
- ✓ Mise en production
- ✓ Maintenance : toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel

Développement de SBC

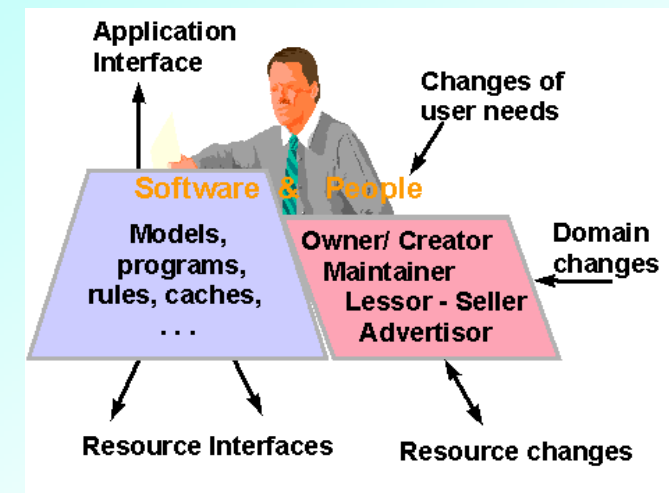
■ Cycle de vie d'un logiciel



Développement de SBC

❑ Les systèmes existants

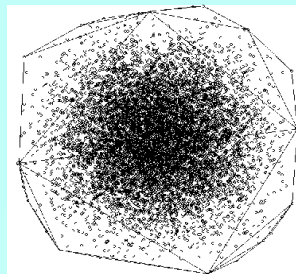
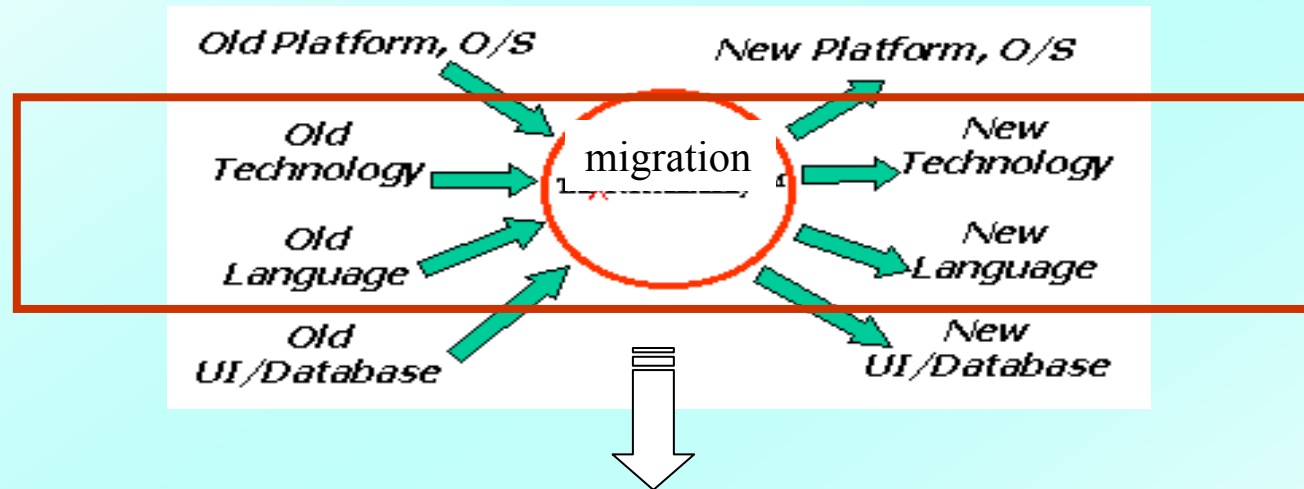
- Besoin de réutilisation
 - ✓ faciliter et diminuer les coûts des nouveaux projets par la capitalisation de l'existant
- Besoin de faciliter la maintenance
 - ✓ Absence de vues abstraites, donc difficulté de compréhension
 - ✓ Conception « petits grains » (les classes, ...)
 - donc, difficulté de séparation de préoccupations
- ✓ Phénomène d'érosion
 - Est concernée la documentation de la réalisation (le comment ?)
 - Est moins concernée la documentation des fonctionnalités (le quoi ?)



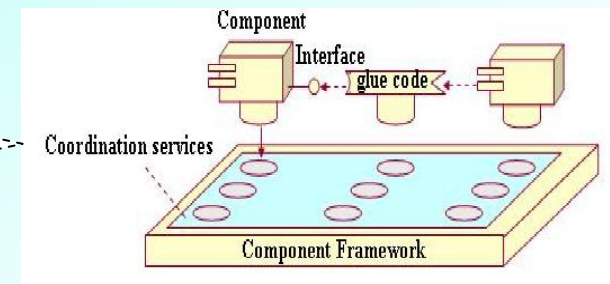
Développement de SBC

Migration vers les SBC : développement à partir de l'existant

- Faire évoluer l'existant



Systeme orienté objet

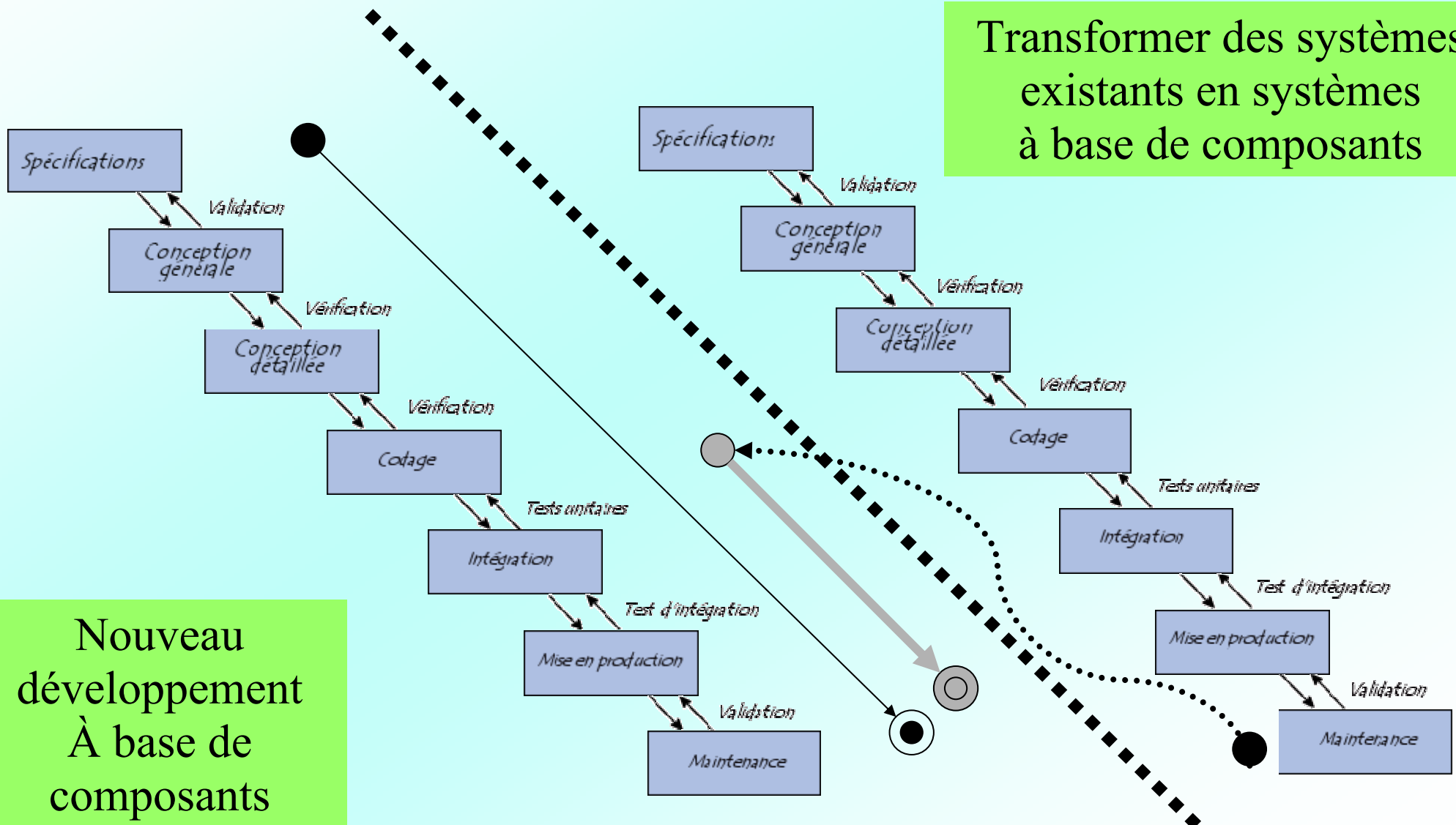


Systeme à base de composants

Développement de SBC

Migration vers les SBC : développement à partir de l'existant

Transformer des systèmes existants en systèmes à base de composants

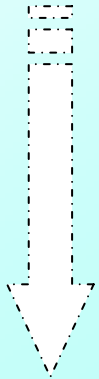


Nouveau développement
À base de composants

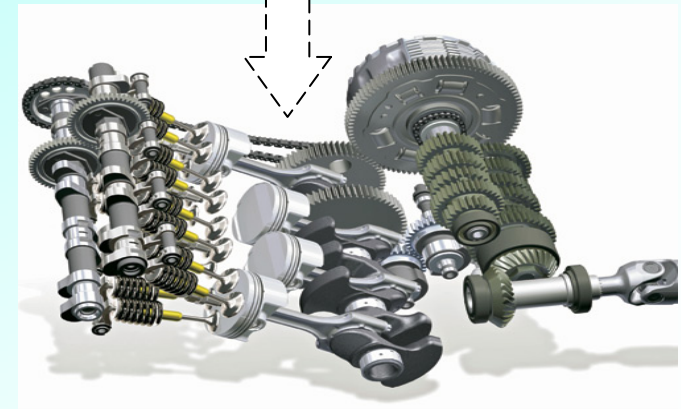
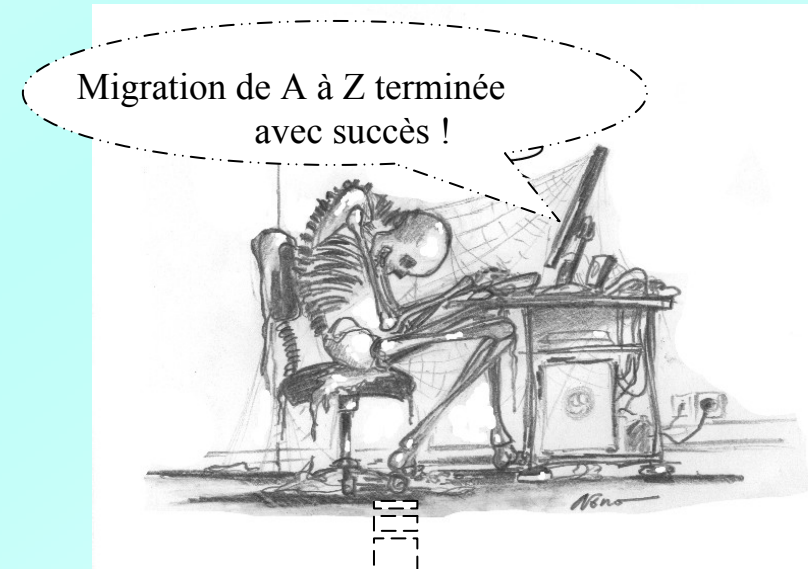
Développement de SBC

■ Migration manuelle difficile :

- Coûteuse
 - ✓ Temps
 - ✓ Experts
- Source d'erreurs

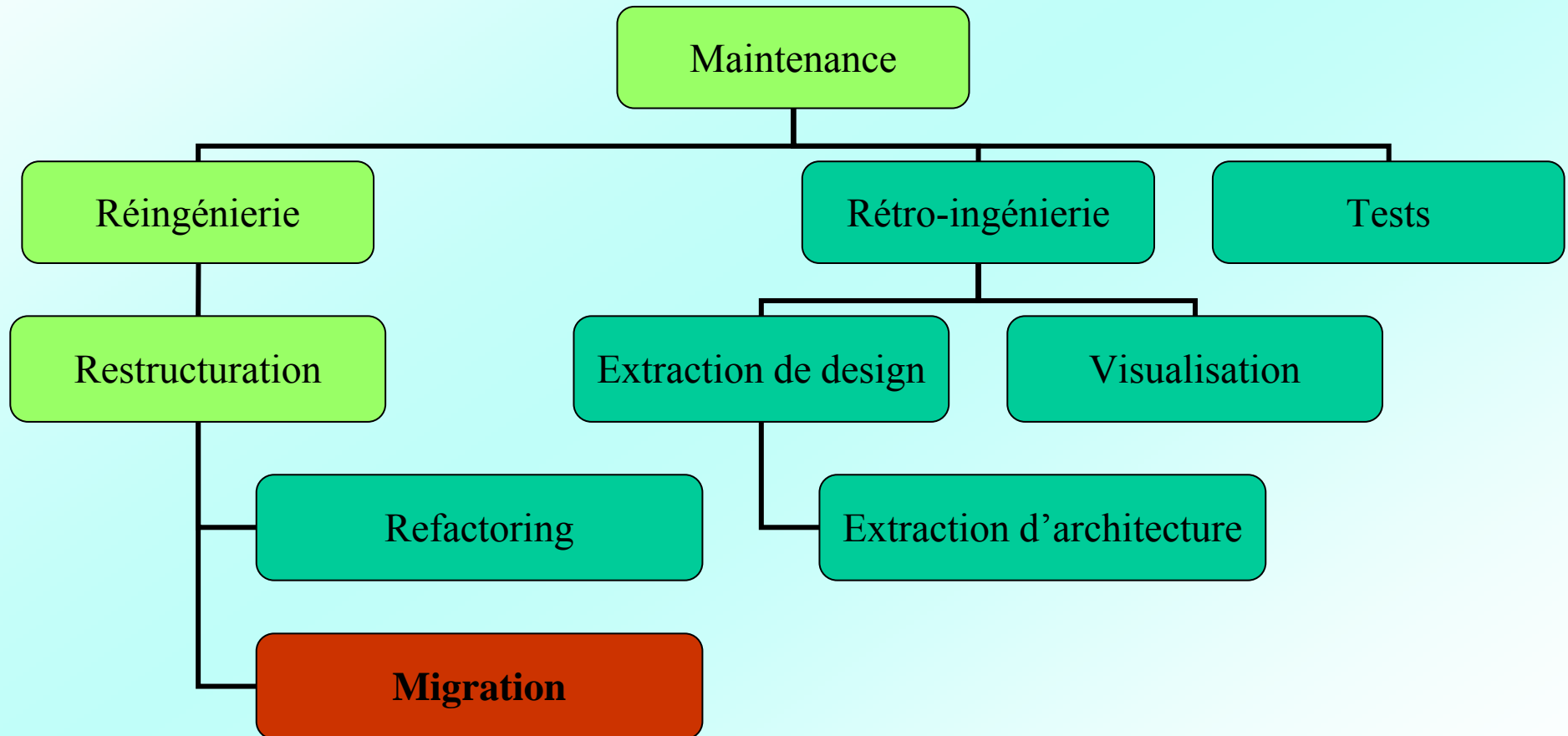


ROMANTIC : Re-engineering of Object-oriented systems by Architecture extraction and migration to Component based one



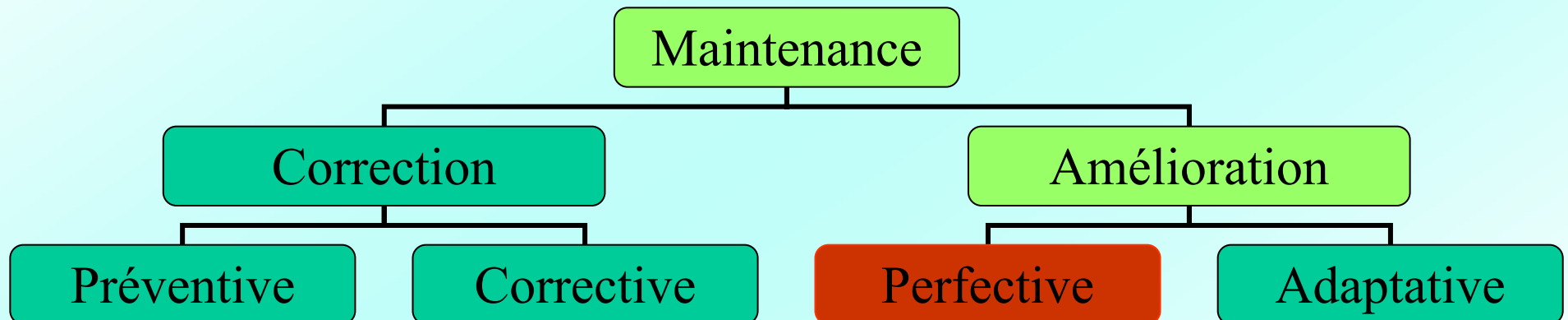
Développement de SBC

- La migration /les différents types d'évolution et de maintenance



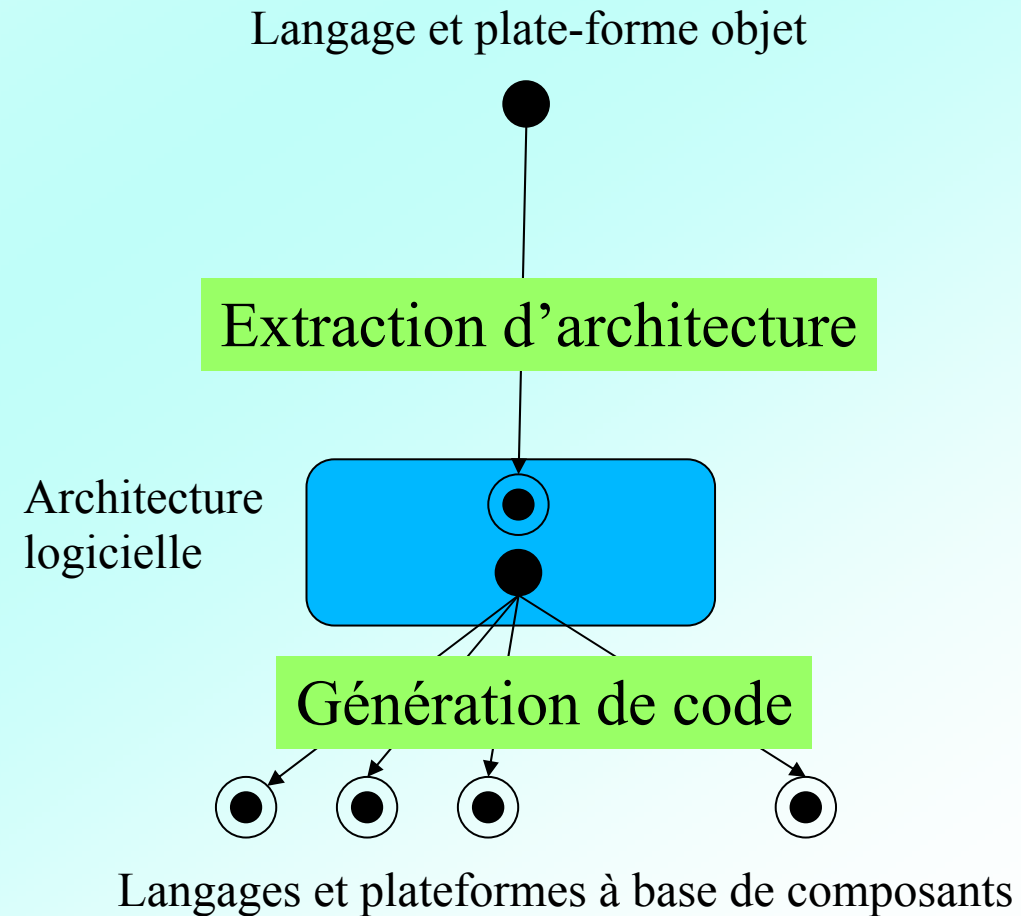
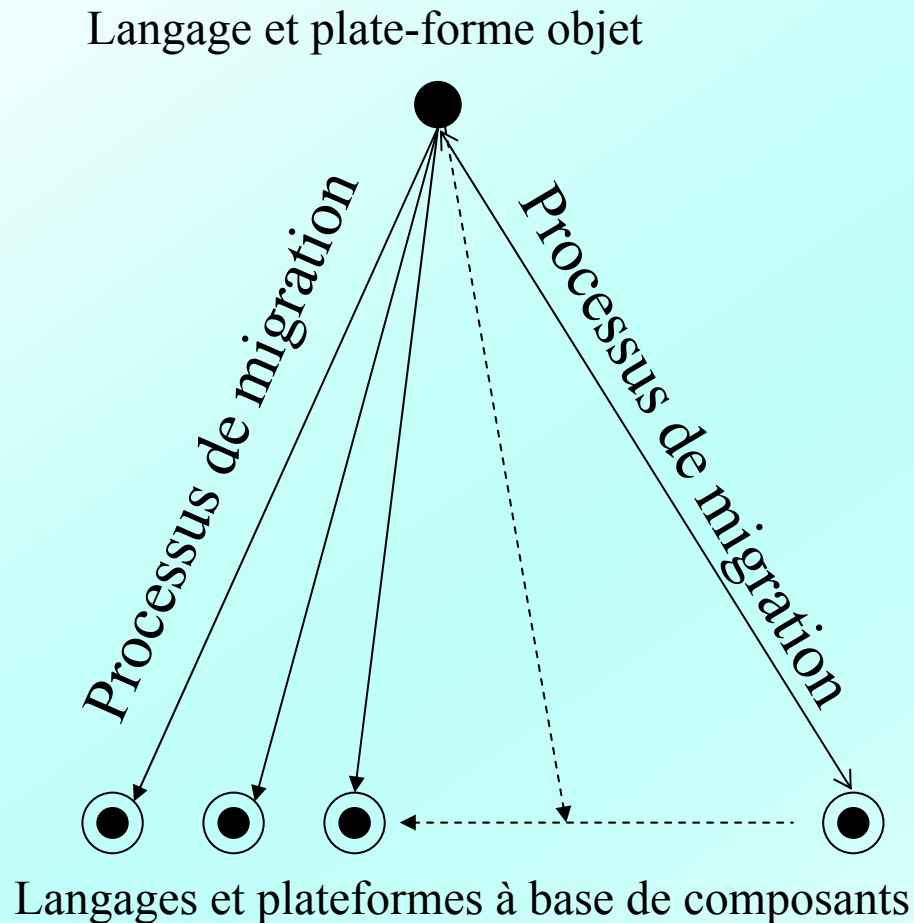
Développement de SBC

- La migration/les différents objectifs de l'évolution et de la maintenance



La migration centrée architecture

■ Migration point-à-point versus migration centrée architecture



La migration centrée architecture

■ Architecture

● *Abstraction d'un système*

➤ *Composants*

- Encapsule la partie métier
- Interfaces
 - Fournies
 - Requises

➤ *Connecteurs*

- Communications entre les composants

➤ *Configuration*

- Topologie des connections entre les composants à travers les connecteurs

La migration centrée architecture

■ Intérêts

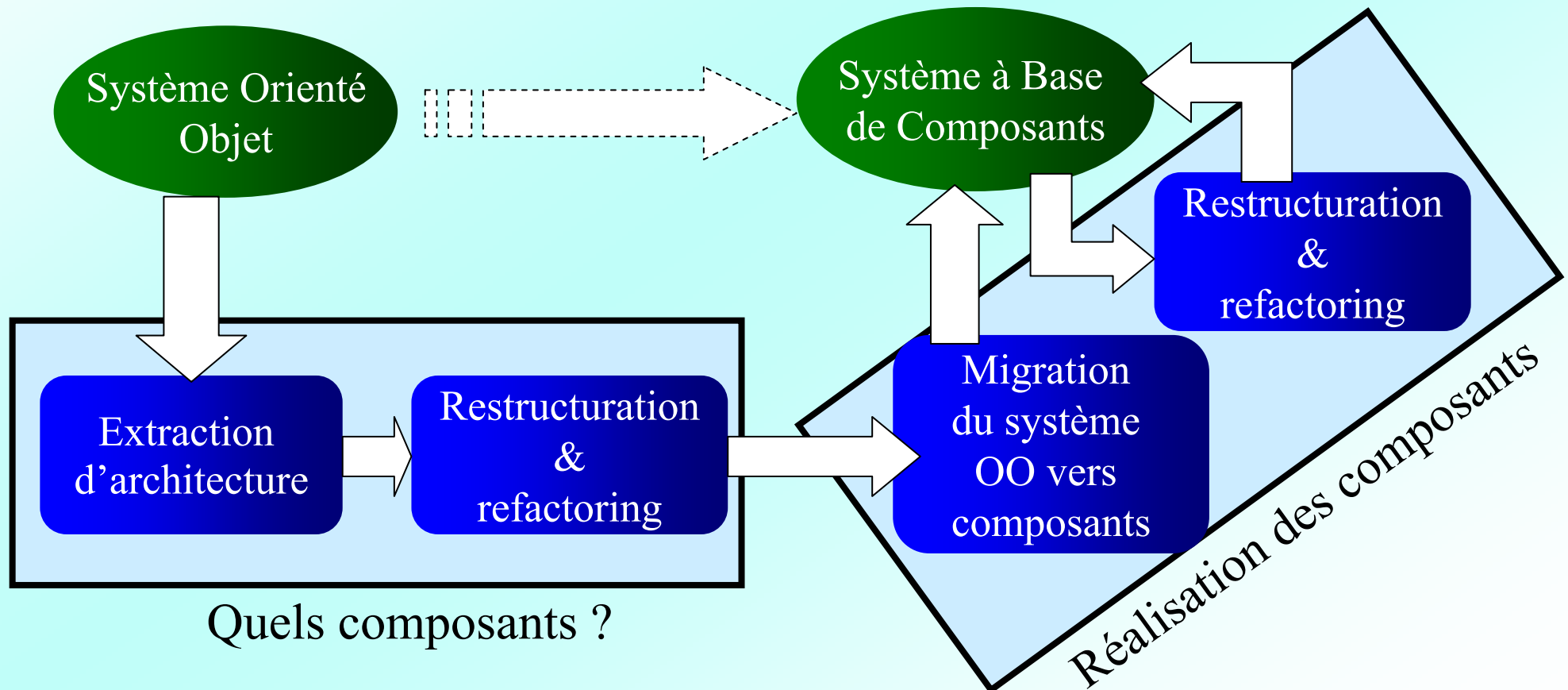
- Échanges entre les différents acteurs du cycle de vie
- Représentation du système :
 - Meilleure compréhension
 - Vérification des propriétés
 - Localisation des défauts
 - Réduction des risques dans les modifications
- Réutilisation

■ Etat des lieux

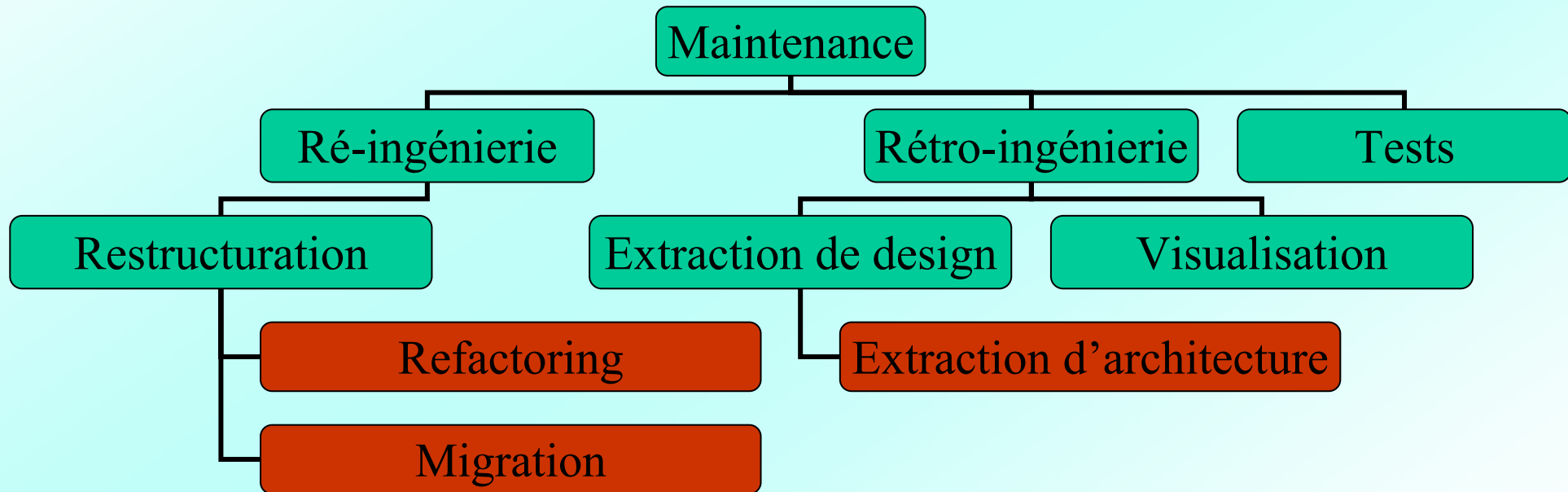
- Systèmes conçus sans représentation de l'architecture
- Systèmes avec une représentation incorrecte
 - Phénomène d'érosion :
 - Écart entre architecture conçue et implémentée.
 - Manque de synchronisation pendant les phases de maintenance

La migration centrée architecture

- **ROMANTIC** : Re-engineering of Object-oriented systems by Architecture extraction and migration to Component based ones

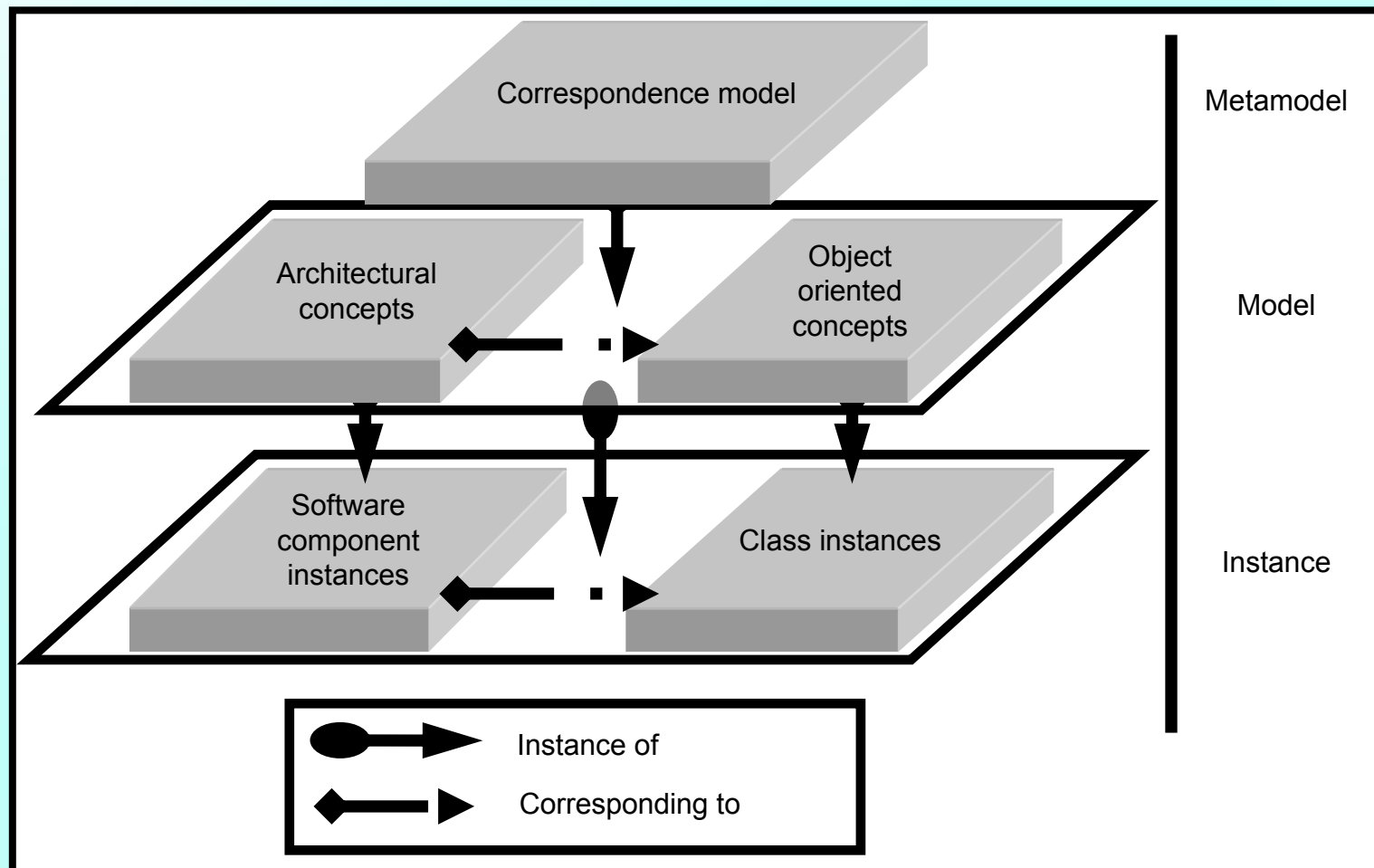


La migration centrée architecture



Les principes de l'extraction d'architecture

■ Modèle de correspondance orienté-objet/composant



Les principes de l'extraction d'architecture

■ Architecture

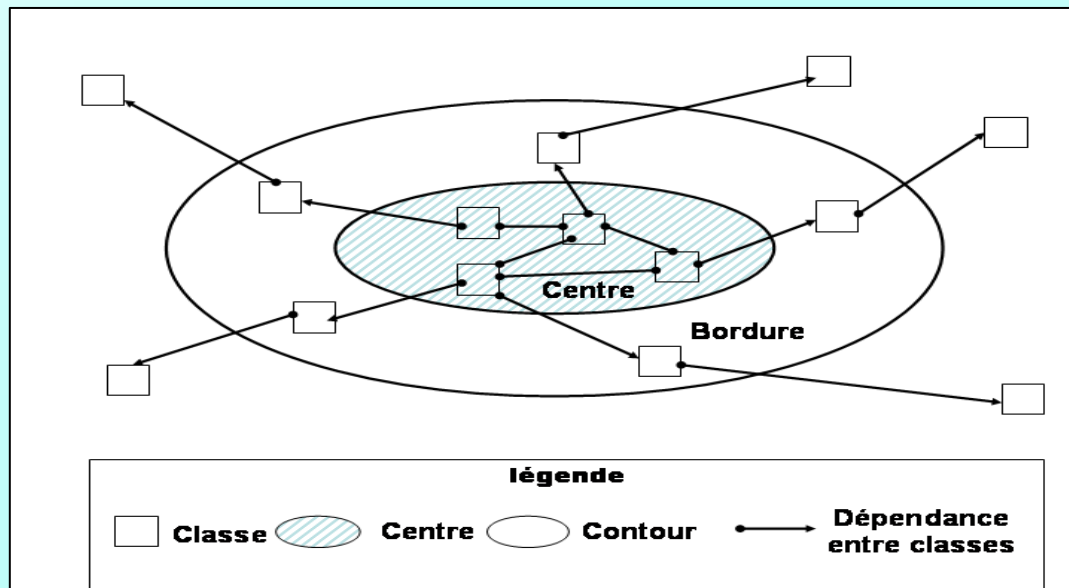
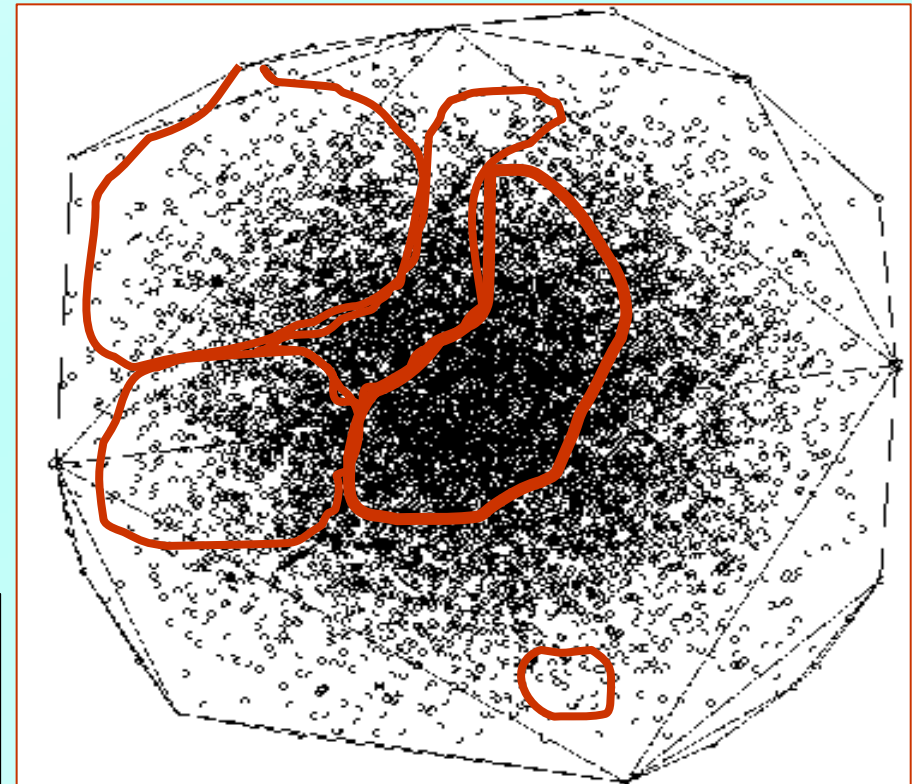
- Une partition des classes du système

■ Composant

- Un ensemble de classes

■ Connecteur

- Lien de dépendance entre classes de composants différents

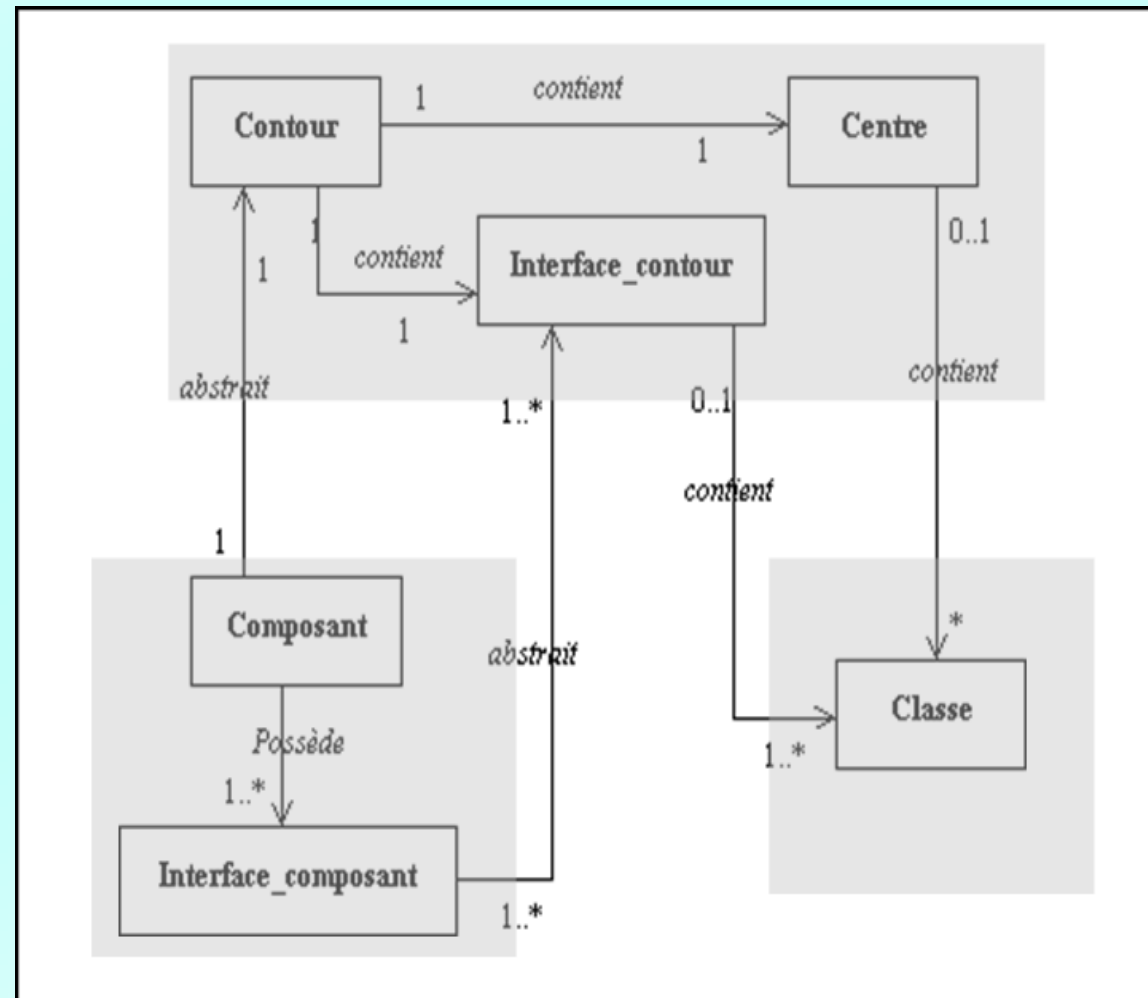


Les principes de l'extraction d'architecture

- **Contour :**
 - *Ensemble de classes*

- **Composant :**
 - *Abstraction d'un contour*

- **Architecture :**
 - *Partition des classes en contours*



Les principes de l'extraction d'architecture

■ Approche manuelle

- Même difficultés que la conception
- Fort besoin en expertise

■ Approche automatique

- L'espace des solutions

➤ Nombre d'architectures possibles:

$$\frac{(2 \cdot n)!}{(n + 1)! \cdot n!}$$

➤ Sélection aléatoire?

➤ La plupart des solutions possibles sont mauvaises

- Sémantique des composants
- Granularité des composants

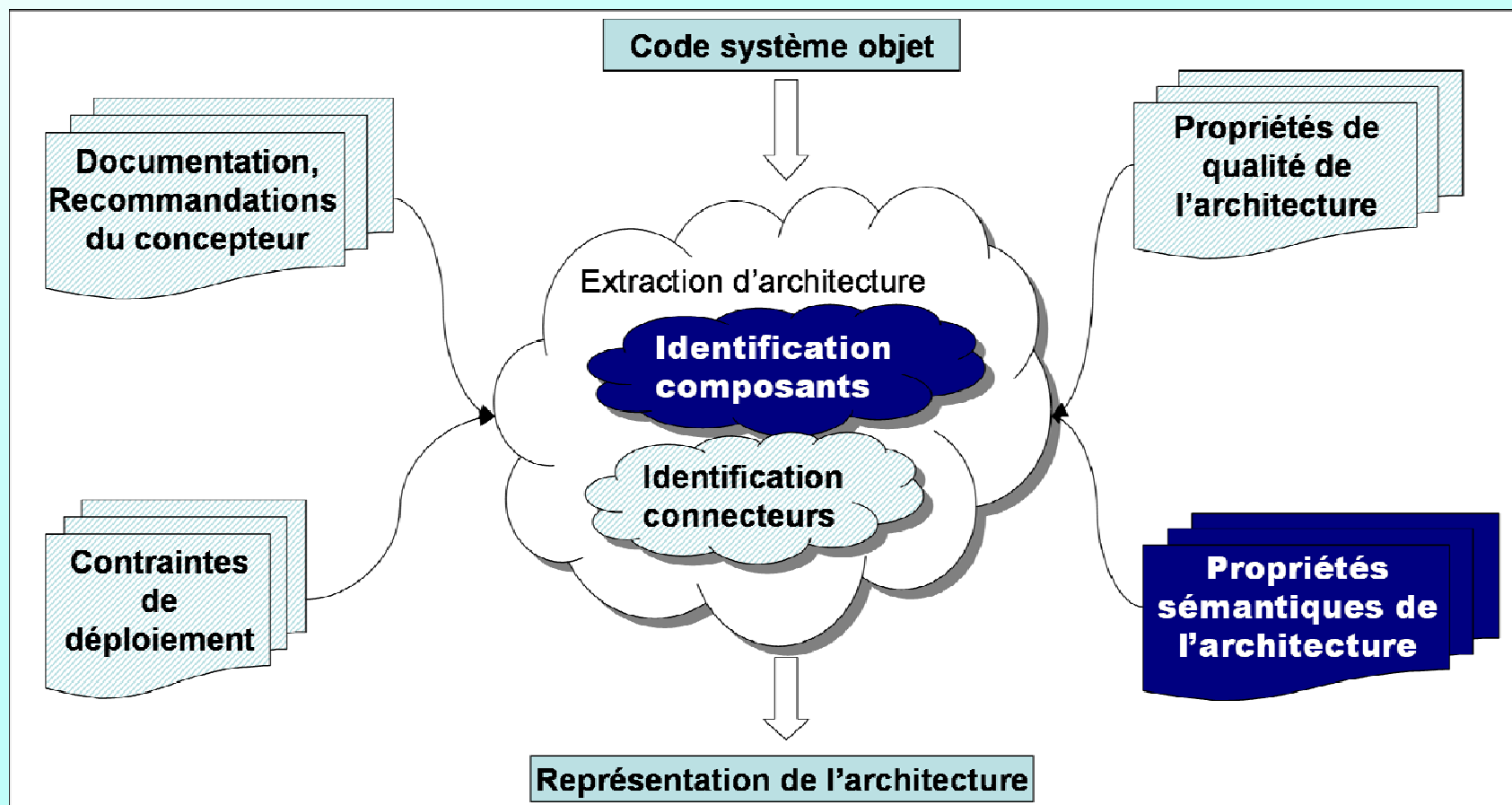
Les principes de l'extraction d'architecture

■ **Processus automatique**

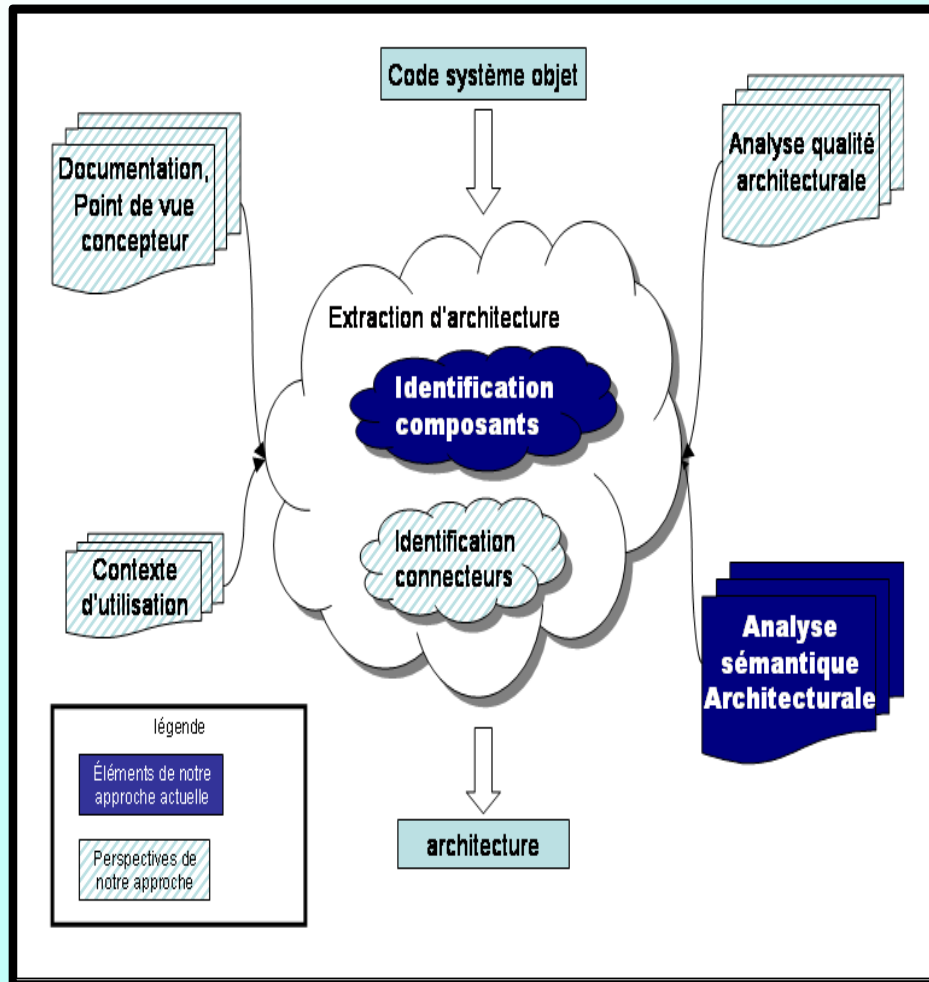
- Limite les besoins en expertise
- Exploration de l'espace de solution
- Besoin d'un ensemble de guides
 - Oriente le choix de la meilleure architecture
 - Dirige l'exploration
 - Sélectionne les solutions

Les principes de l'extraction d'architecture

Les guides de l'extraction

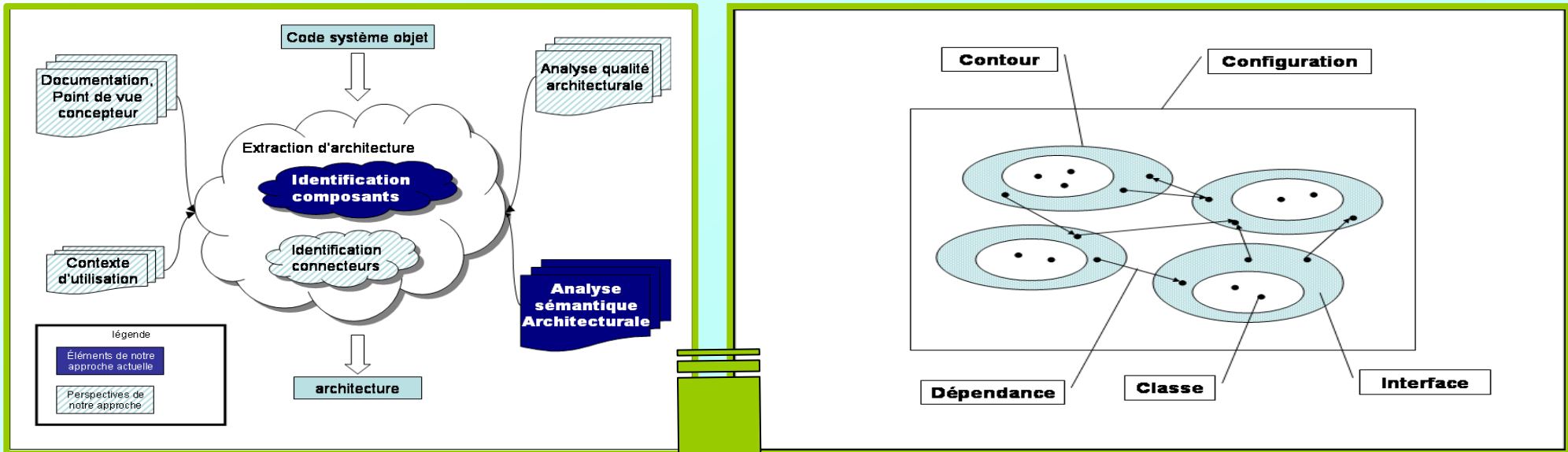


Les principes de l'extraction d'architecture



- ❑ Sémantique des éléments architecturaux
 - Construire des composants
- ❑ Qualité d'une architecture et de ses éléments
 - Construire des bons composants
- ❑ Éléments de conception
 - Construire des composants reflétant les services désirés

Les principes de l'extraction d'architecture



Définition d'une
fonction
d'évaluation de la
qualité

Trouver une instance du
modèle de correspondance
qui maximise cette
fonction

Processus de
construction
des
composants

Analyse et réification de la sémantique de la notion d'architecture

■ Définition

- La ou les structures du système ce qui inclut les composants logiciels, les propriétés externes de ces composants et les relations entre eux. (Kazman)
- Abstraction
 - Montre les interactions entre composants
 - Masque les informations purement internes

■ Les éléments architecturaux

- Les composants logiciels
- Les connecteurs
- La configuration

Analyse et réification de la sémantique de la notion d'architecture

■ Définitions généralement admises d'un composant

● **Szyperski** (Szyperski, 1998)

Une unité de composition possédant des interfaces spécifiées par contrat et des dépendances explicites avec le contexte. Il peut être déployé indépendamment et peut être composé par un tiers

● **Heinemann et Council** (Heinemann et al., 2001)

Un élément logiciel qui est conforme à un modèle de composant et peut être déployé indépendamment et composé sans modification selon un standard de composition

● **Luer** (Luer et al., 2002).

Un élément logiciel qui (a) encapsule une implémentation réutilisable d'une fonctionnalité, (b) peut être composé sans modification et (c) adhère à un modèle de composant.

Il distingue cette définition de celle d'un composant déployable qui est :

Un composant (a) pre-paquetagé, (b) distribué indépendamment, (c) facile à installer et désinstaller et (d) auto-descriptif

Analyse et réification de la sémantique de la notion d'architecture

■ Lien entre déployable et autonome

- Déployable :
 - Compréhensible
 - Adaptable
 - Autonome
 - Spécialisé

■ On ne retient pas l'adaptabilité et la compréhensibilité

- Trop liées à l'implémentation

■ Caractéristiques sémantiques du composant :

- Composable
- Autonome
- spécialisé

Analyse et réification de la sémantique de la notion d'architecture

■ Notre définition

● Un composant

*Un composant est un élément logiciel qui (a) est **composable** sans modification, (b) peut être distribué de manière **autonome**, (c) **encapsule une fonctionnalité**, et (d) qui adhère à un modèle de composant*

● Le modèle d'un composant (standard du composant)

*Un modèle de composant est la combinaison de (a) **un standard de composant** qui gouverne la construction de chaque composant et (b) **un standard de composition** qui régit comment organiser un ensemble de composants en une application et comment ces composants communiquent et interagissent entre eux. (Luer et al., 2002)*

- **Le modèle des composants regroupe les autres propriétés du composant**
 - **Adhésion à un standard de composition**
 - **Auto-description, pré-paquetage, facilité d'installation/désinstallation**

Analyse et réification de la sémantique de la notion d'architecture

■ Objectif

- Mesurer les caractéristiques d'un composant

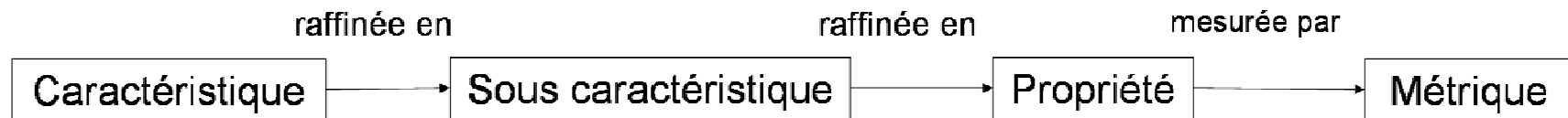
■ Problèmes :

- Absence de composants
 - Seulement un regroupement de classes

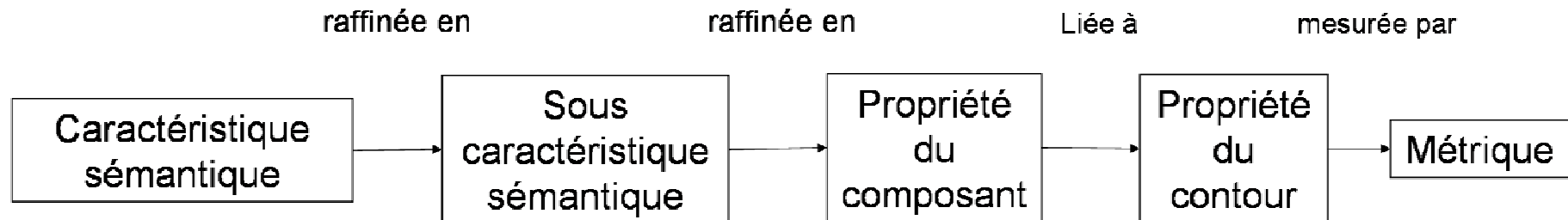
■ Processus en deux étapes :

- Comment mesurer les caractéristiques sur un composant?
 - Identification des propriétés mesurables des composants
 - Lien avec les caractéristiques
- Comment mesurer ces propriétés mesurables sur un contour?
 - Identification des propriétés mesurables du contour.
 - Liens avec les propriétés mesurables du composant.

Analyse et réification de la sémantique de la notion d'architecture,



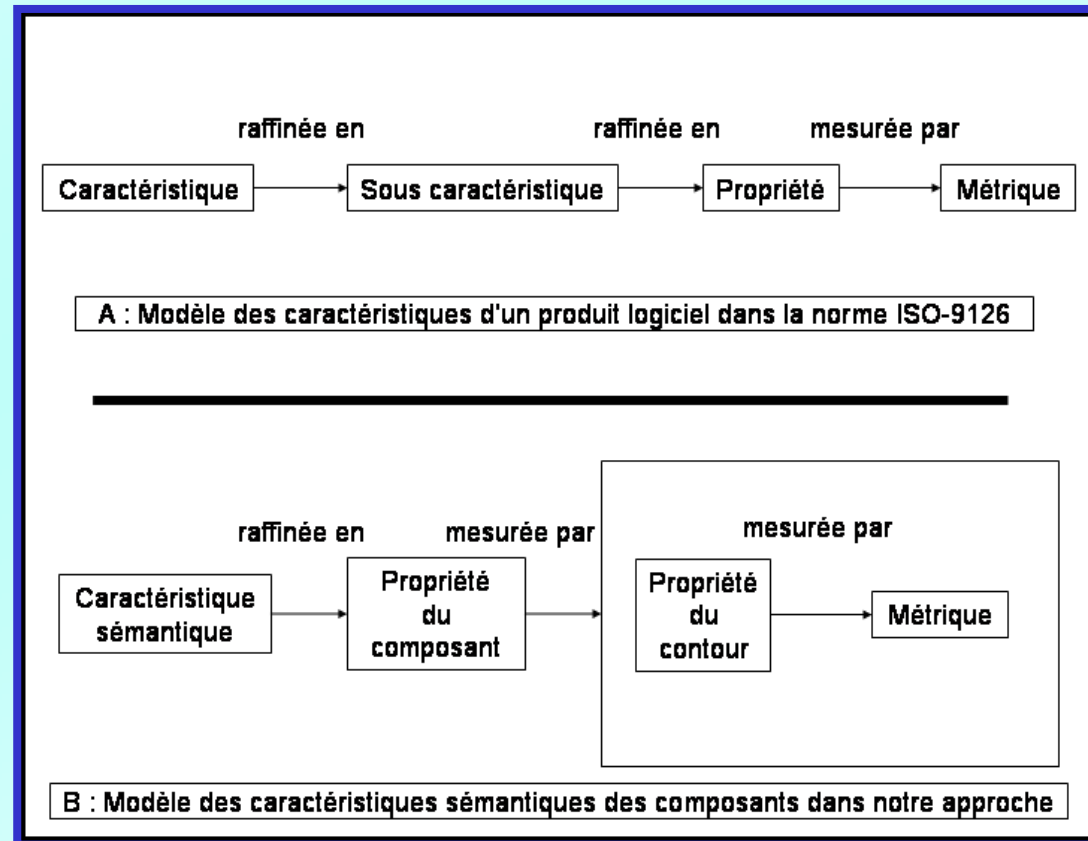
A : Modèle des caractéristiques d'un produit logiciel dans la norme ISO-9126



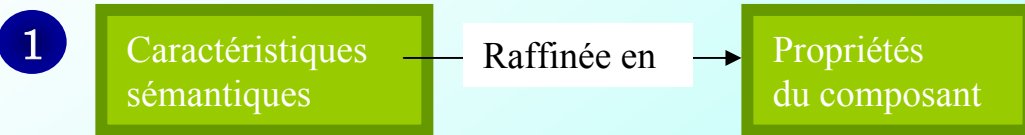
B : Notre modèle de mesure de la validité sémantique

Analyse et réification de la sémantique de la notion d'architecture

- Processus en trois étapes :
 1. Identification des liens entre **caractéristiques sémantiques** et **propriétés des composants**
 2. Identification des liens entre les propriétés des **composants** et celles des **contours**
 3. Choix de **métriques** pour la mesure des propriétés des **contours**



Analyse et réification de la sémantique de la notion d'architecture



Analyse

Une interface dont les services sont très cohérents fournit probablement une seule fonctionnalité

Un ensemble d'interfaces très cohérentes a plus de chance de fournir un nombre limité de fonctionnalités

Si le code du composant est très couplé, il fournit probablement un nombre réduit de fonctionnalités

Etc.

Propriétés de la structure du composant

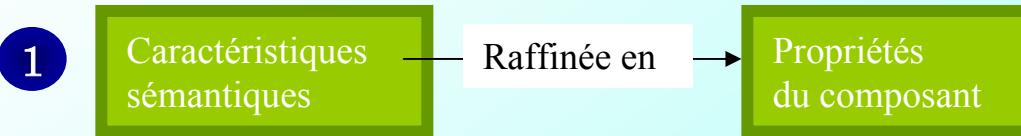
La moyenne des cohésions entre les services dans une interface

La cohésion entre les interfaces

Le couplage et la cohésion à l'intérieur du composant

Le nombre d'interfaces fournies

Analyse et réification de la sémantique de la notion d'architecture

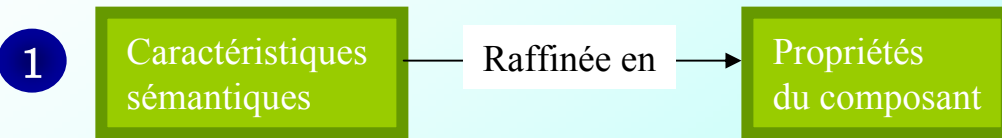


■ **Autonomie**

⊕ Analyse

- ▶ Un composant est complètement autonome s'il ne possède pas d'interface requise
- ⊕ Liens caractéristique *Autonomie* / propriétés de la structure de composant
 - ▶ L'*autonomie* peut être mesurée à travers la propriété «nombre d'interfaces requises»

Analyse et réification de la sémantique de la notion d'architecture



■ Composabilité

⊕ Analyse

► Un composant est considéré comme composable d'abord parce qu'il définit clairement les services qu'il fournit et ceux qu'il requiert, à travers des interfaces

► Un composant sera plus facile à assembler avec un autre si, dans chacune de ses interfaces, les services sont cohérents

⊕ Liens caractéristique *Composabilité* / propriétés de la structure de composant

► La propriété «cohésion des services dans chaque interface» permet de mesurer la composabilité

Analyse et réification de la sémantique de la notion d'architecture

■ Caractéristiques des composants vs. Propriétés structurelles des composants

	Nombre d'interfaces fournies	Nombre d'interfaces requises	Moyenne des cohésions des services de chaque interface	Cohésion entre les interfaces	Cohésion interne	Couplage interne
Composabilité			+			
Spécialisation	-		+	+	+	+
Autonomie		-				

+ : la caractéristique et la propriété sont co-variantes

- : la caractéristique et la propriété varient en sens inverse

Analyse et réification de la sémantique de la notion d'architecture

- Structure des composants vs. Structure des contours
- 5 propriétés mesurables sur le contour.
- Difficulté avec la mesure des interfaces fournis

contour		couplage entre la bordure et l'extérieur	cohésion de chaque classe de la bordure	cohésion des classes de la bordure	couplage des classes du contour	cohésion des classes du contour
nombre d'interface requis		+				
cohésion externe	entre les services dans une interface		+			
	entre interfaces			+		
couplage interne					+	
cohésion interne						+

Analyse et réification de la sémantique de la notion d'architecture

- Structure des composants vs. Structure des contours
- Calcul de la moyenne des cohésions des services de chaque interface du composant
 - ⊕ L'ensemble des interfaces du composant est en correspondance avec l'interface du contour
 - ⊕ La moyenne des cohésions de chaque classe de l'interface donne une bonne idée de la moyenne des cohésions des services dans chaque interface du composant
 - ⊕ La moyenne des cohésions de chaque classe de l'interface du contour pour mesurer cette propriété

Analyse et réification de la sémantique de la notion d'architecture

- Structure des composants vs. Structure des contours
 - Cohésion entre les interfaces du composant
 - ⊕ Associée à la propriété du contour mesurant la cohésion des classes de son interface
 - Cohésion/couplage interne du composant
 - ⊕ Liées à la cohésion/le couplage des classes du contour
 - Nombre d'interfaces fournies du composant
 - ⊕ Nous associe à chaque classe de l'interface du contour qui possède une méthode publique, une interface fournie du composant
 - Nombre d'interfaces requises par le composant
 - ⊕ Peut être évaluée en utilisant le couplage du composant avec l'extérieur
 - ⊕ Liée au couplage externe du contour

Analyse et réification de la sémantique de la notion d'architecture

■ Structure des composants vs. Structure des contours

		Propriété externe du contour	Propriétés internes du contour				
		Couplage entre le contour et l'extérieur	Nombre de classes de l'interface ayant une méthode publique	Moyenne des cohésions de chaque classe de l'interface	Cohésion des classes de l'interface	Couplage des classes du contour	Cohésion des classes du contour
Propriétés externes du composant	Moyenne des cohésions des services de chaque interface			+			
	Cohésion entre les interfaces				+		
	Nombre d'interfaces fournies		+				
	Nombre d'interfaces requises	+					
Propriétés internes du composant	Cohésion interne					+	
	Couplage interne						+

Analyse et réification de la sémantique de la notion d'architecture

■ Définition des métriques pour la mesure des propriétés du contour

■ Mesure de couplage

⊕ Besoin

- ▶ Les propriétés «couplage des classes du contour» et « couplage entre les classes du contour et l'extérieur» nécessitent une mesure de couplage

⊕ Propriétés requises pour la métrique de couplage

- ▶ Refléter uniquement les liens d'utilisation entre deux objets
- ▶ Permettre de mesurer l'intensité des liens entre deux objets
- ▶ Doit être relative et indépendante du langage de programmation

⊕ Constat

- ▶ Aucune métrique disponible dans la littérature ne satisfait tous ces besoins

⊕ Solution

- ▶ Définir notre propre métrique à partir de celles existantes.

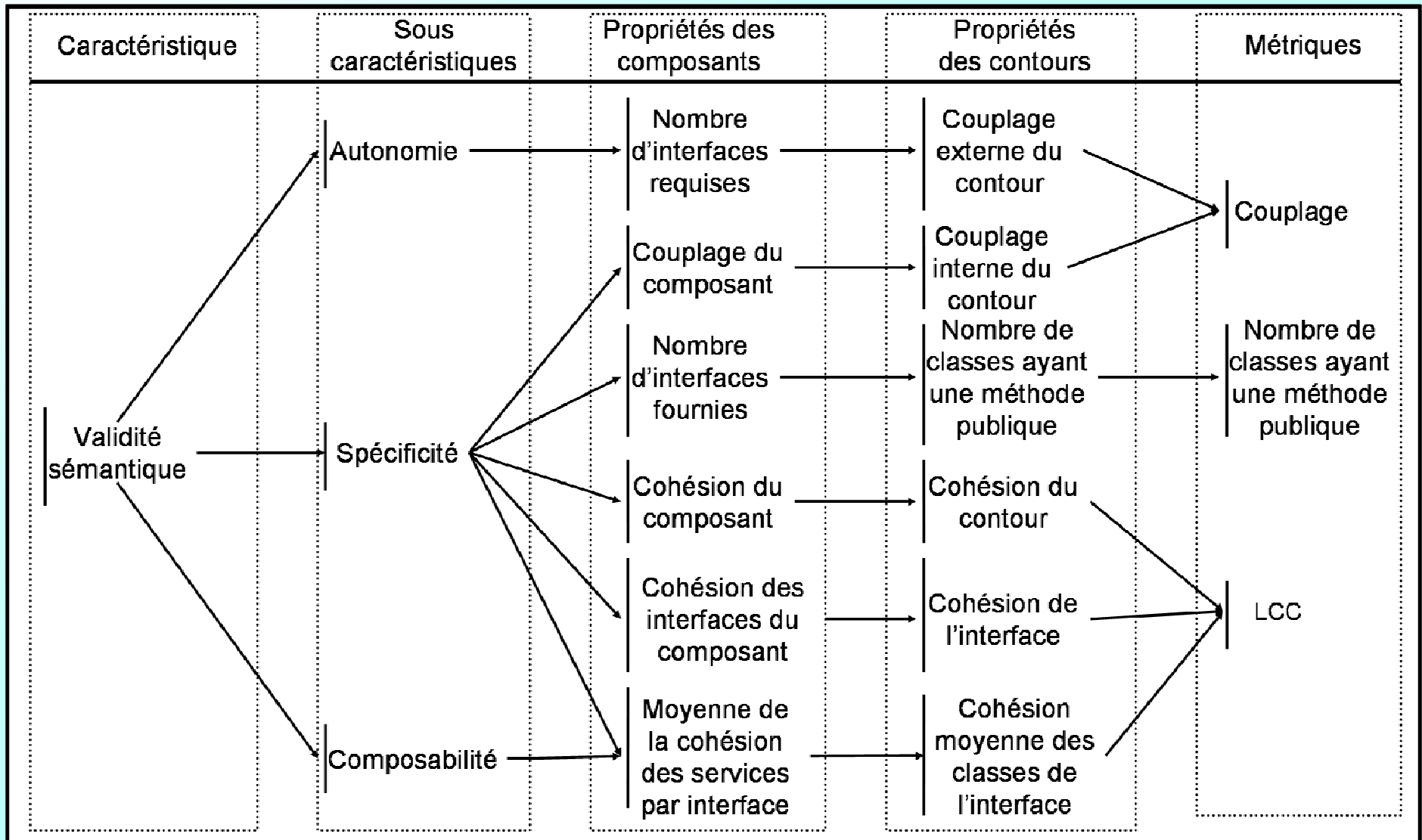
Analyse et réification de la sémantique de la notion d'architecture

- **Mesure de couplage (E est un ensemble de classes)**
 - **CM(E)** : nombre d'appels de méthodes entre classes de E, pondéré par le nombre de paramètres.
 - **CA(E)** : nombre d'attributs d'une classe de E qui ont pour type une autre classe de E.
 - **CP(E)** : nombre de paramètres dans une classe de E qui ont pour type une autre classe de E.
- **Le couplage de E est la moyenne des trois**
- **Le couplage de E avec l'extérieur est le complément.**

Analyse et réification de la sémantique de la notion d'architecture

- Définition des métriques pour la mesure des propriétés du contour
- Mesure de la cohésion (E ensemble de classes)
 - ⊕ LCC(E) : (Loose Class Cohesion, Bieman et Kang) mesure le nombre de paires de méthodes connectées directement ou indirectement.
 - ▶ Deux méthodes sont connectées si elles utilisent de manière directe ou indirecte des attributs en commun
 - ▶ Deux méthodes sont connectées indirectement s'il existe une chaîne de méthodes connectées qui les relient.

Analyse et réification de la sémantique de la notion d'architecture



Analyse et réification de la sémantique de la notion d'architecture

□ Mesure de la composabilité

$$C(E) = \frac{1}{|B|} \cdot \sum_{i \in B} LCC(i)$$



□ Mesure de l'autonomie

$$A(E) = \text{couplageExt}(E) = 100 - \text{Couplage}(E)$$

□ Mesure de la spécialisation

$$Spe(E) = \frac{1}{5} \cdot \left(\left(\frac{1}{|B|} \cdot \sum_{i \in B} LCC(i) \right) + LCC(B) + LCC(E) + Coup(E) \right)$$

□ Fonction d'évaluation de la sémantique des composants

$$S(E) = \sum_i \frac{1}{\lambda_i} (\lambda_1 \cdot C(E) + \lambda_2 \cdot A(E) + \lambda_3 \cdot Spe(E))$$

Analyse et réification de la qualité architecturale

■ Objectifs

- Mesurer les caractéristiques de qualité de l'architecture
- Ajout à la fonction objectif

■ Adaptation de mesures existantes

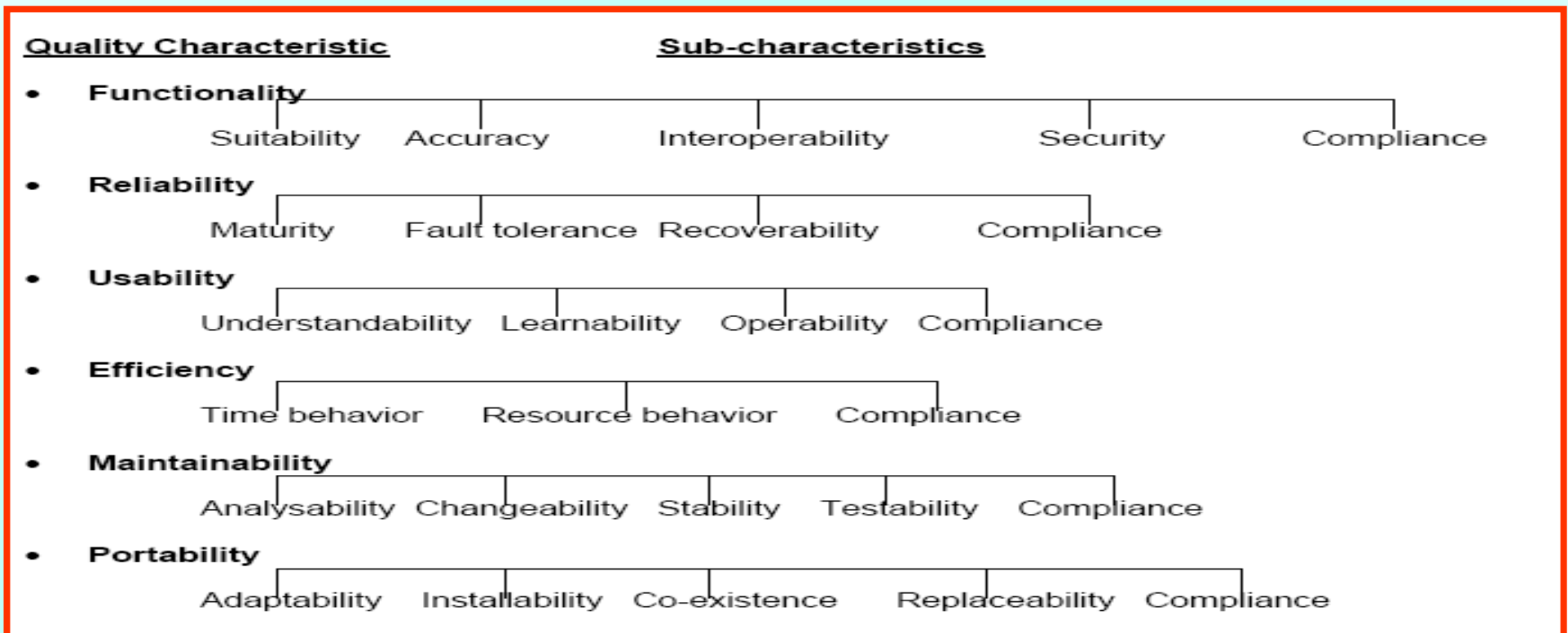
- Études des travaux de mesure de qualité
- Adaptation pour notre approche

Analyse et réification de la qualité architecturale

■ Qualité d'un produit ou service :

- Ensemble de caractéristiques qui lui permettent de remplir les besoins spécifiés ou impliqués. (ISO9126)

■ Qualité d'un logiciel :



Analyse et réification de la qualité architecturale

■ Caractéristiques de l'architecture

	suitability	maturité	tolérance au panne	operability	consommation en temps	consommation en ressource	stabilité	adaptabilité	total
nombres de composant		-1	1		-1	-1			-2
nombre de connecteurs		-1	1		-1	-1			-2
degré entrant des composants		-1	-1						-2
degré sortant des composants		-1	-1						-2
degré des connecteurs		-1	-1						-2
nombre de connexions entre 2 composants		-1	-1						-2
nombre de connecteurs entre 2 composants		-1	-1						-2
classe de connexité		1	-1						0
classes de 2-connexité		1	-1						0
classes de 2-arête connexité		1	-1						0
diamètre					-1				-1
nœuds jumeaux			1						1

Analyse et réification de la qualité architecturale

- On rejette celles qui:
 - Ne dépendent pas de l'architecture
 - Précision (Fonctionnalité)
 - Capacité d'apprentissage (Utilisabilité)
 - Nécessitent des informations indisponibles dans notre approche
 - Efficacité
 - Portabilité

- On analyse et on réifie
 - Fiabilité
 - Maintenabilité

Analyse et réification de la qualité architecturale

■ **Mesure de la maintenabilité**

- La maintenabilité mesure l'effort requis pour maintenir l'architecture
- La maintenabilité d'une architecture dépend de
 - La maintenabilité des composants
 - La maintenabilité de la configuration
- Mesure de la maintenabilité des composants et de la configuration
 - Basée sur la théorie de l'information
 - Basée sur les travaux de Jenkins
 - La maintenabilité d'un module décroît avec le nombre de dépendances

Analyse et réification de la qualité architecturale

■ Mesure de la maintenabilité

- Maintenabilité d'un composant
 - 2 phases de classes dans le composant
 - Une phase stable (degré < Kt)
 - Une phase instable (degré > Kt)
 - La proportion de chaque phase donne une mesure de la maintenabilité
- G un graphe
 - Les sommets sont les classes
 - Un arc entre a et b si a dépend de b
- Degré Kt :

$$\sum_{k'=0}^{k_t} k' \cdot n(k') = \sum_{k'=k_t}^{k_{\max}} k' \cdot n(k')$$

Analyse et réification de la qualité architecturale

■ **Mesure de la maintenabilité**

- **Maintenabilité de la configuration**
 - Idem composant
 - Mais les composants remplacent les classes

- **Maintenabilité de l'architecture**

$$M(C) = \frac{1}{2} \cdot \left(\frac{1}{|C|} \sum_{s \in C} \text{MComp}(s) + \text{MConf}(C) \right)$$

Analyse et réification de la qualité architecturale

■ **Mesure de la fiabilité**

- La fiabilité mesure la capacité du système à fournir le service demandé au moment donné
- Dépend de la complexité du système (Henry et Kafura, 1981)
- La complexité d'une architecture dépend
 - De la complexité des composants
 - Du poids de chaque composant dans l'architecture
- La complexité d'un composant dépend
 - Du couplage avec les autres composants
 - De la cohésion dans le composant

Analyse et réification de la qualité architecturale

■ Mesure de la fiabilité

- Complexité d'un composant

$$Complexite = Intra \cdot (Inter)^2$$

- Fiabilité d'un composant

$$RelS(s) = Complexite / 10^4$$

- Fiabilité d'une architecture

$$Rel(C) = \left(\sum_{k=0}^{k_{\max}} k \cdot n(k) \right)^{-1} \cdot \left(\sum_{s \in C} deg(s) \cdot RelS(s) \right)$$

Processus de partitionnement

■ Clustering hiérarchique

- Similarité : fonction d'évaluation de la sémantique
- Résultat :
 - un dendrogramme des classes
 - Une hiérarchie de clusters

■ On souhaite une partition des classes

■ Identification des composants

- Coupe dans le dendrogramme
- Objectif :
 - Une partition
 - Des contours qui atteignent les seuils


Processus de partitionnement

Data

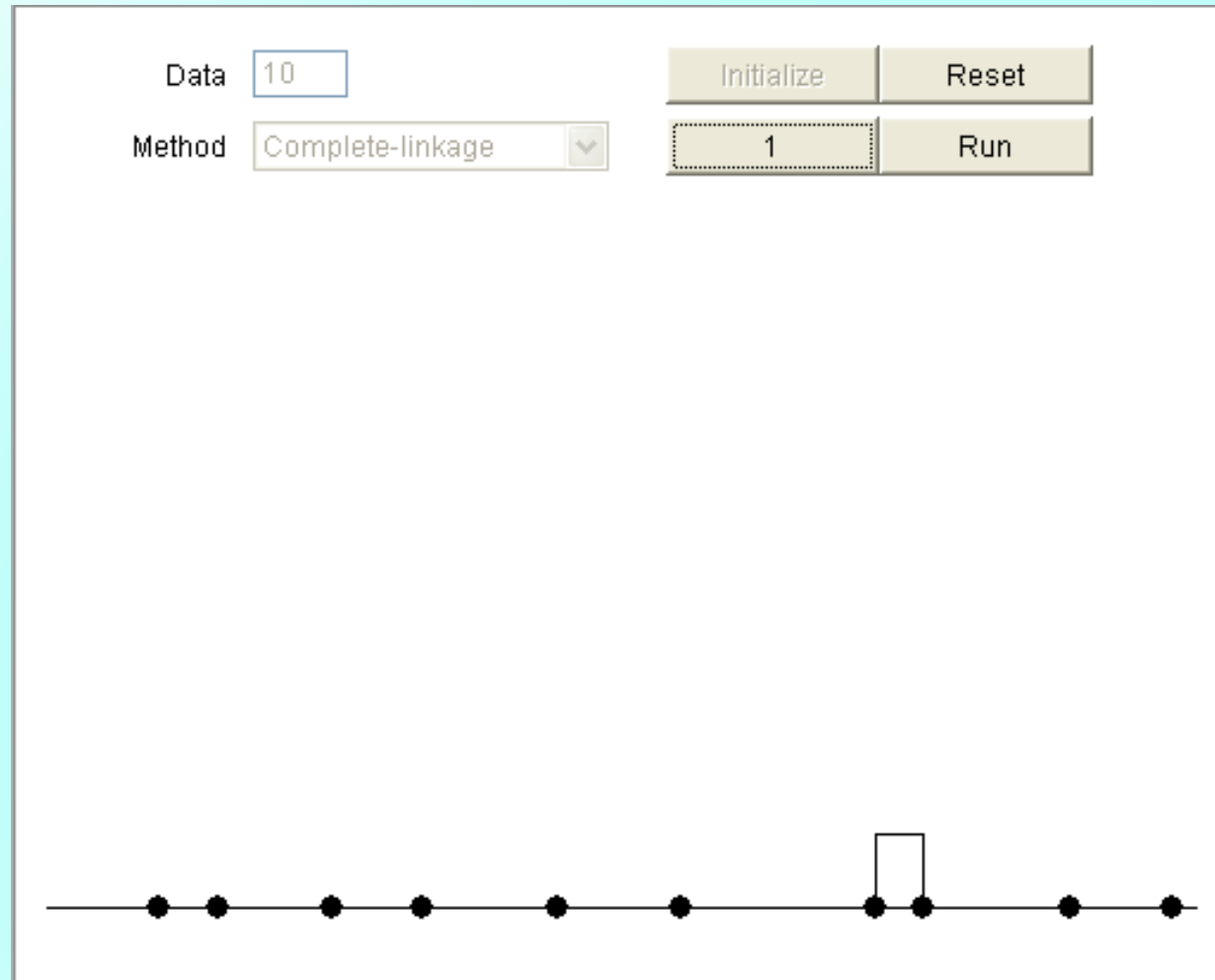
Method

Initialize Reset

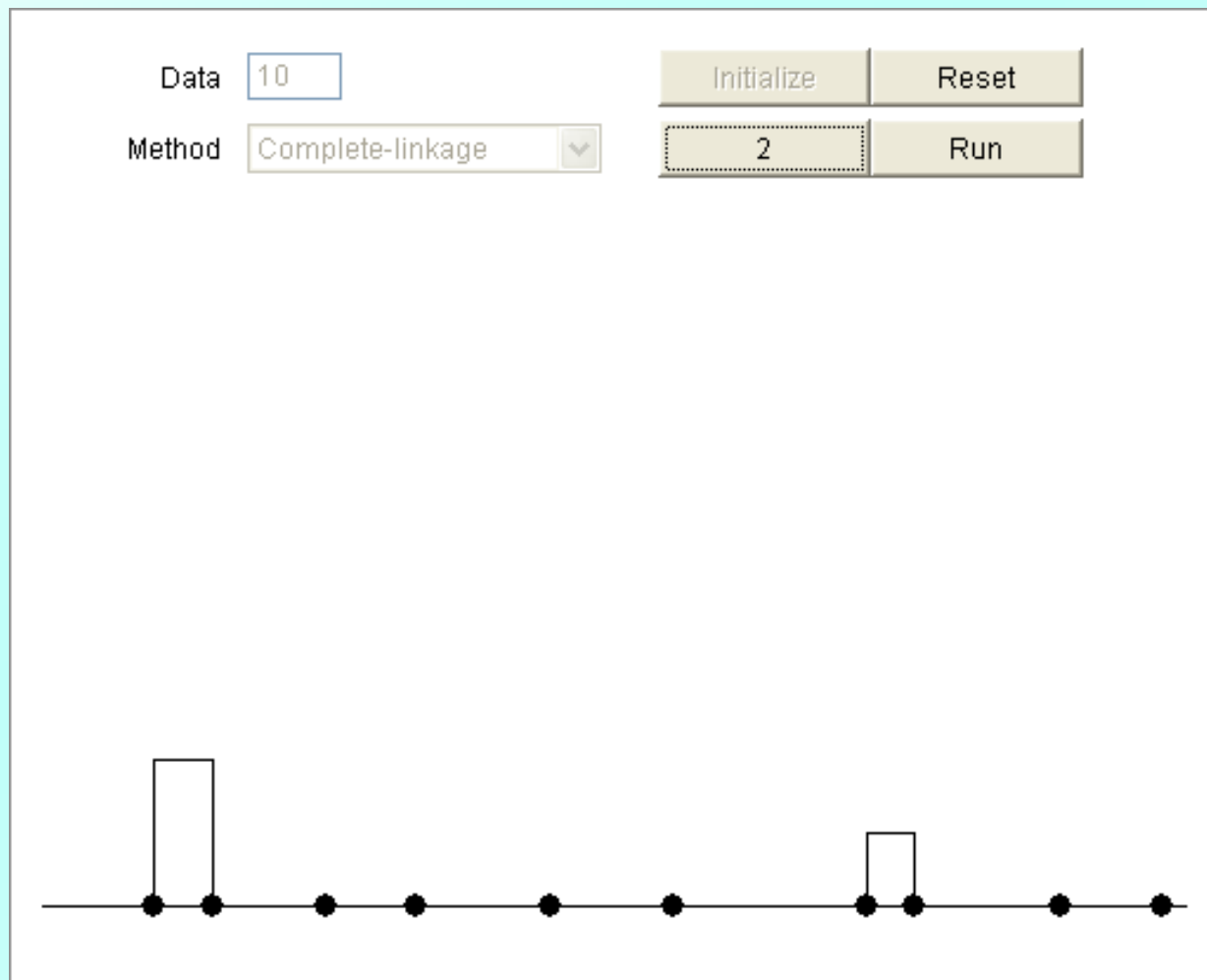
Step Run



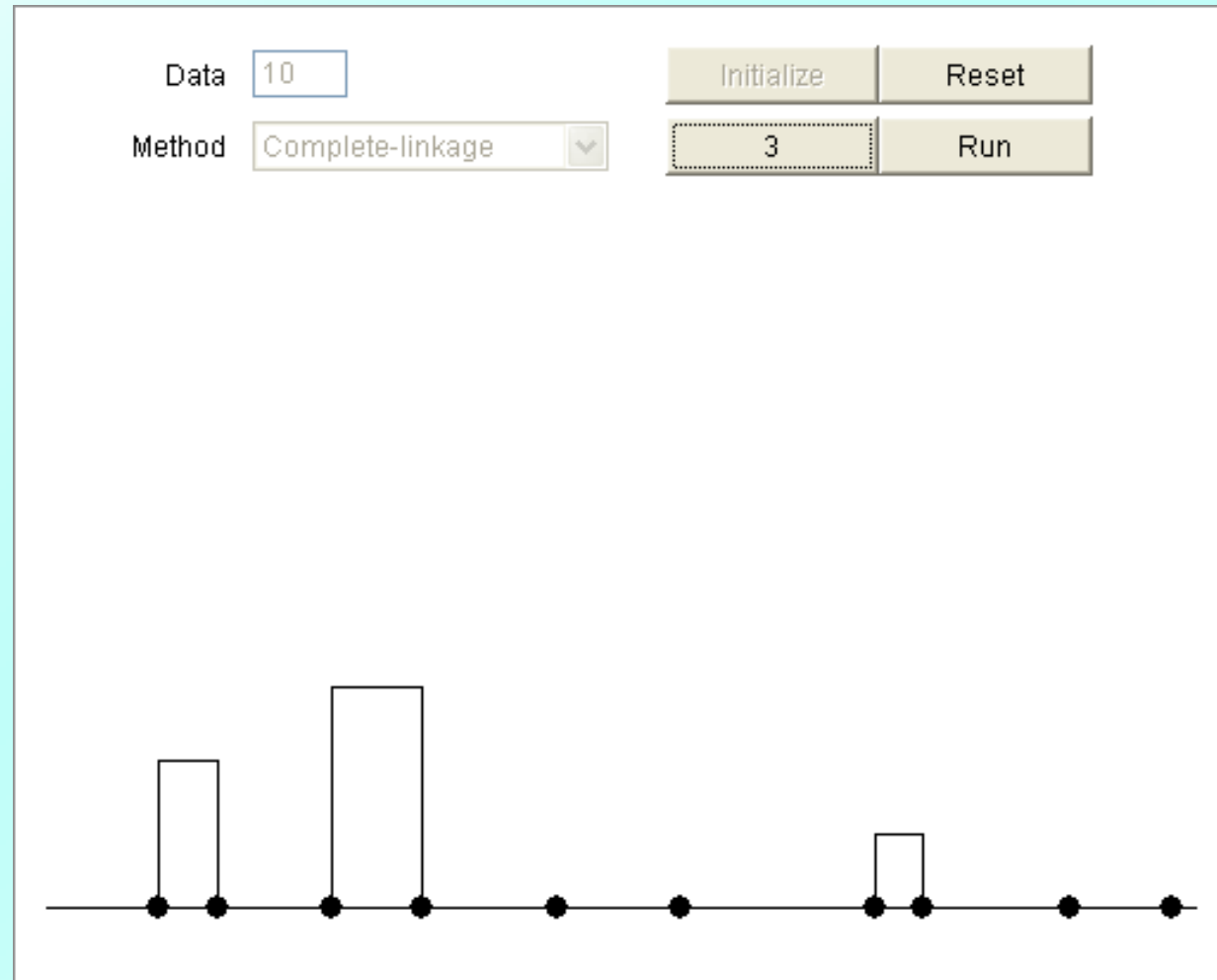
Processus de partitionnement



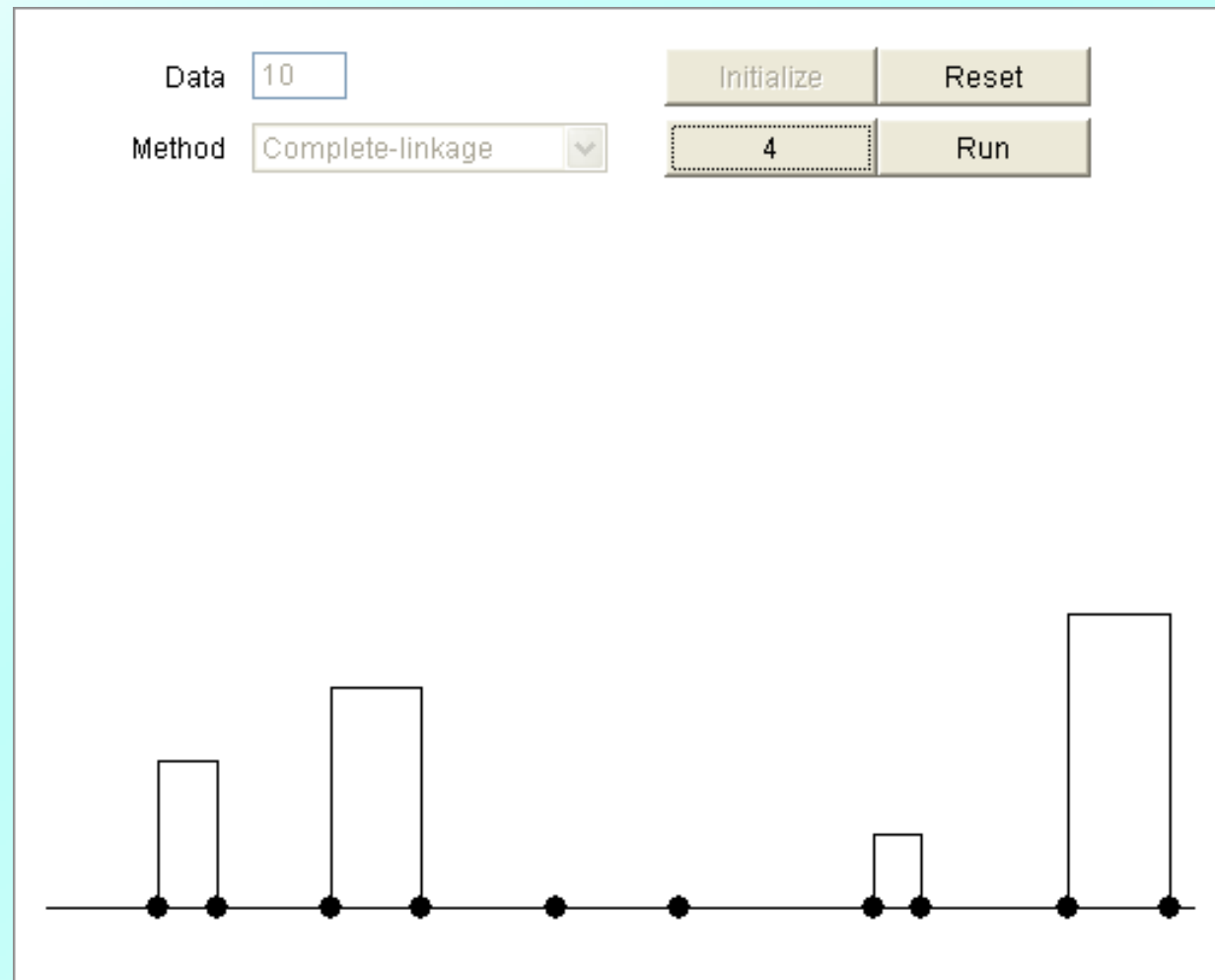
Processus de partitionnement



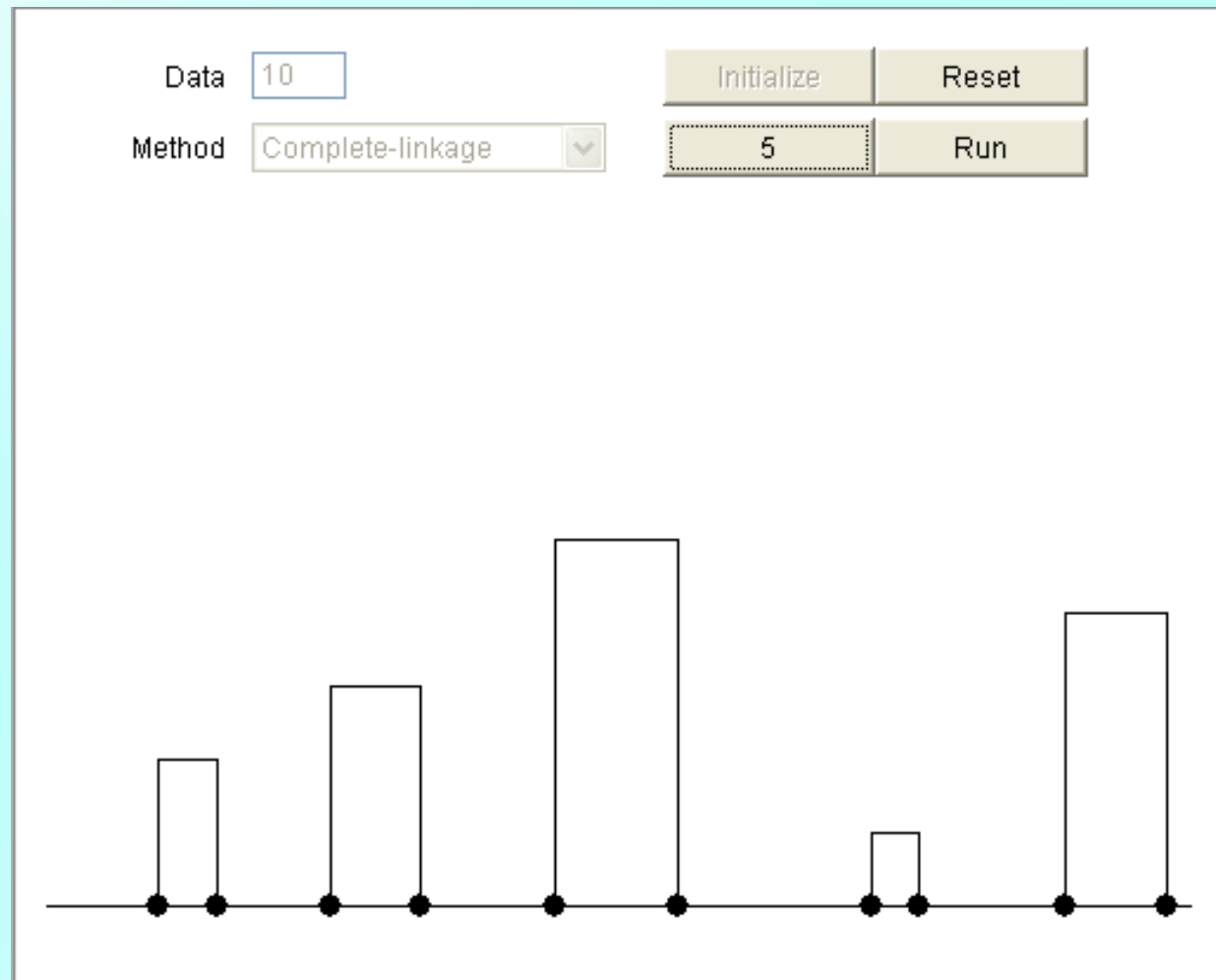
Processus de partitionnement



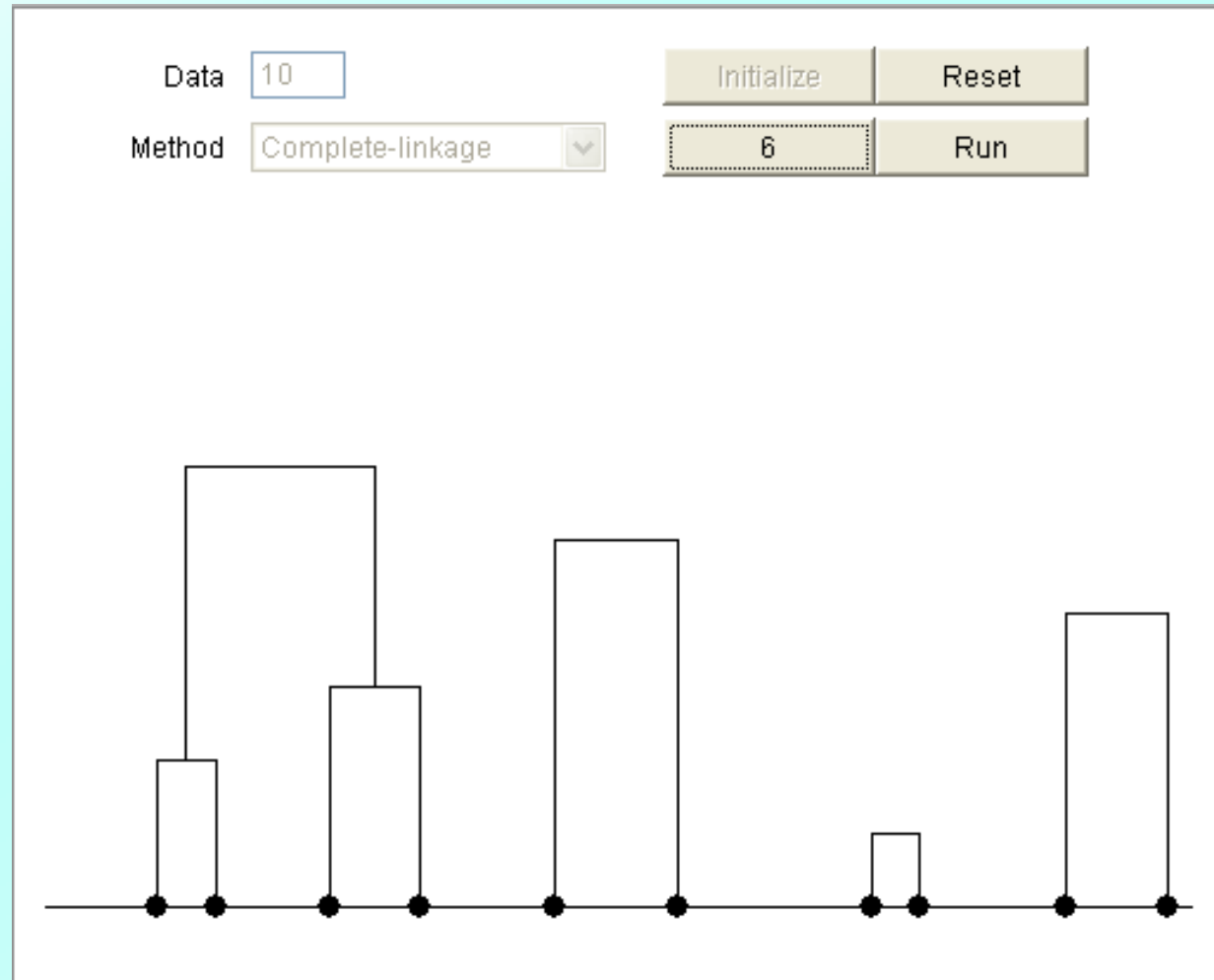
Processus de partitionnement



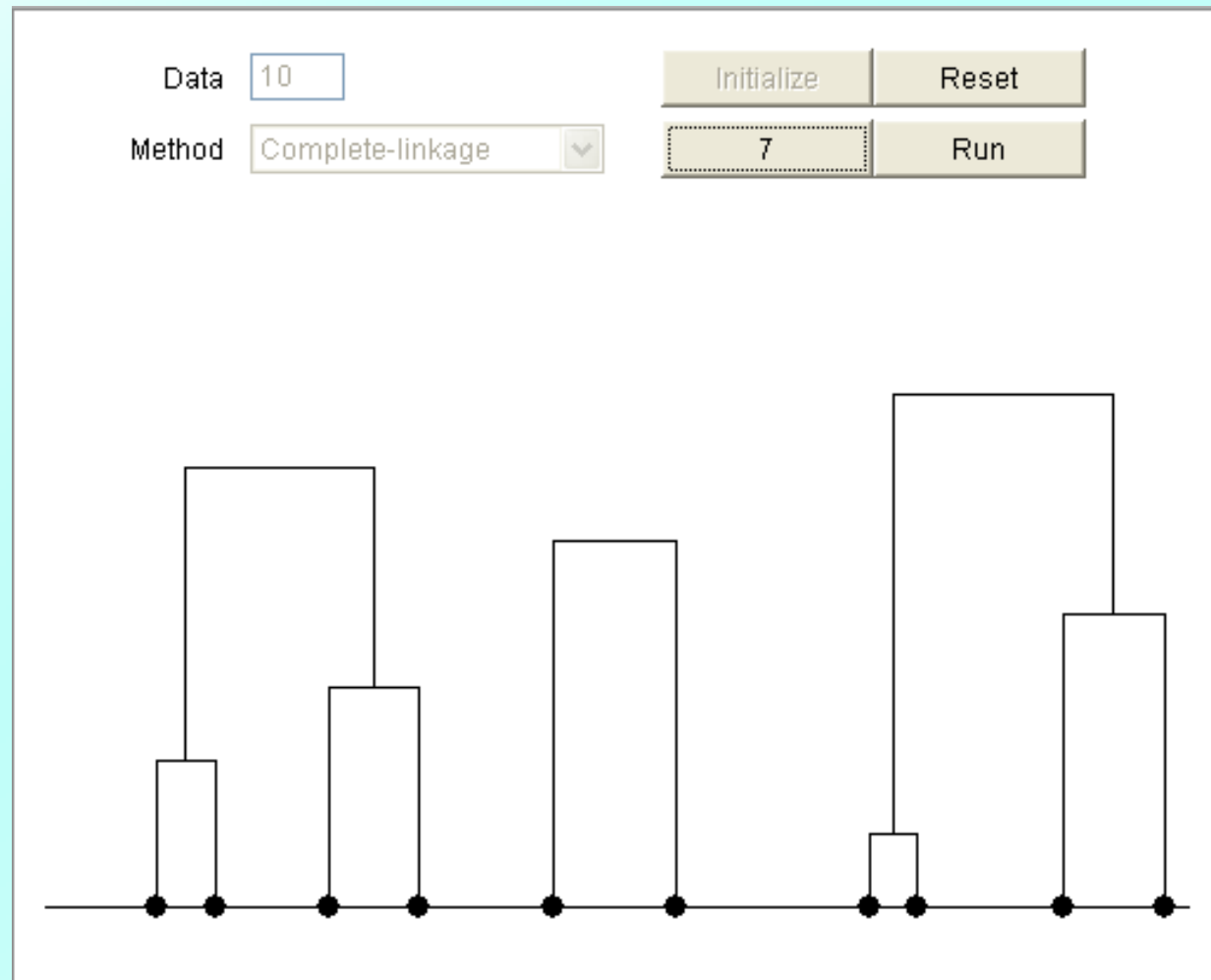
Processus de partitionnement



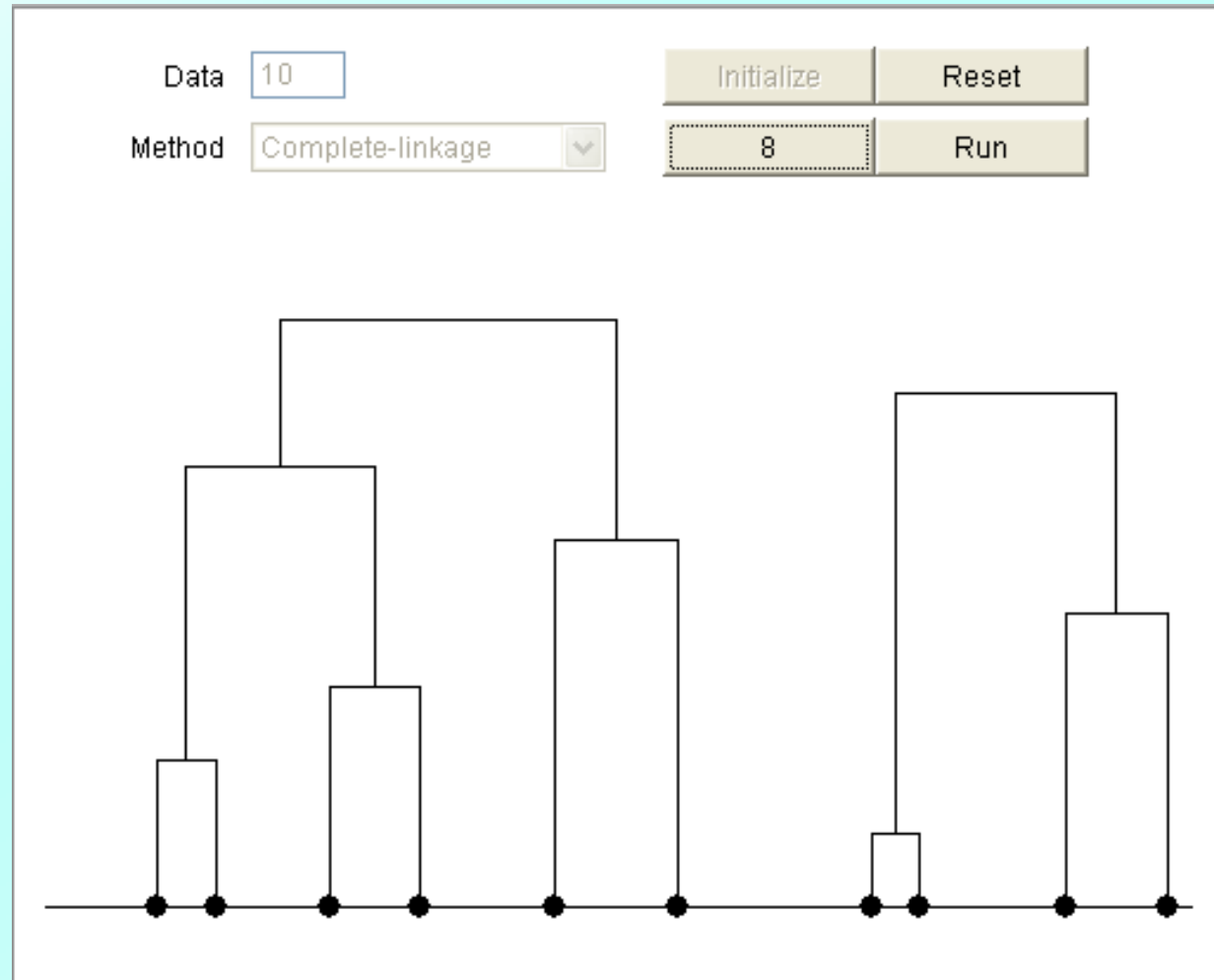
Processus de partitionnement



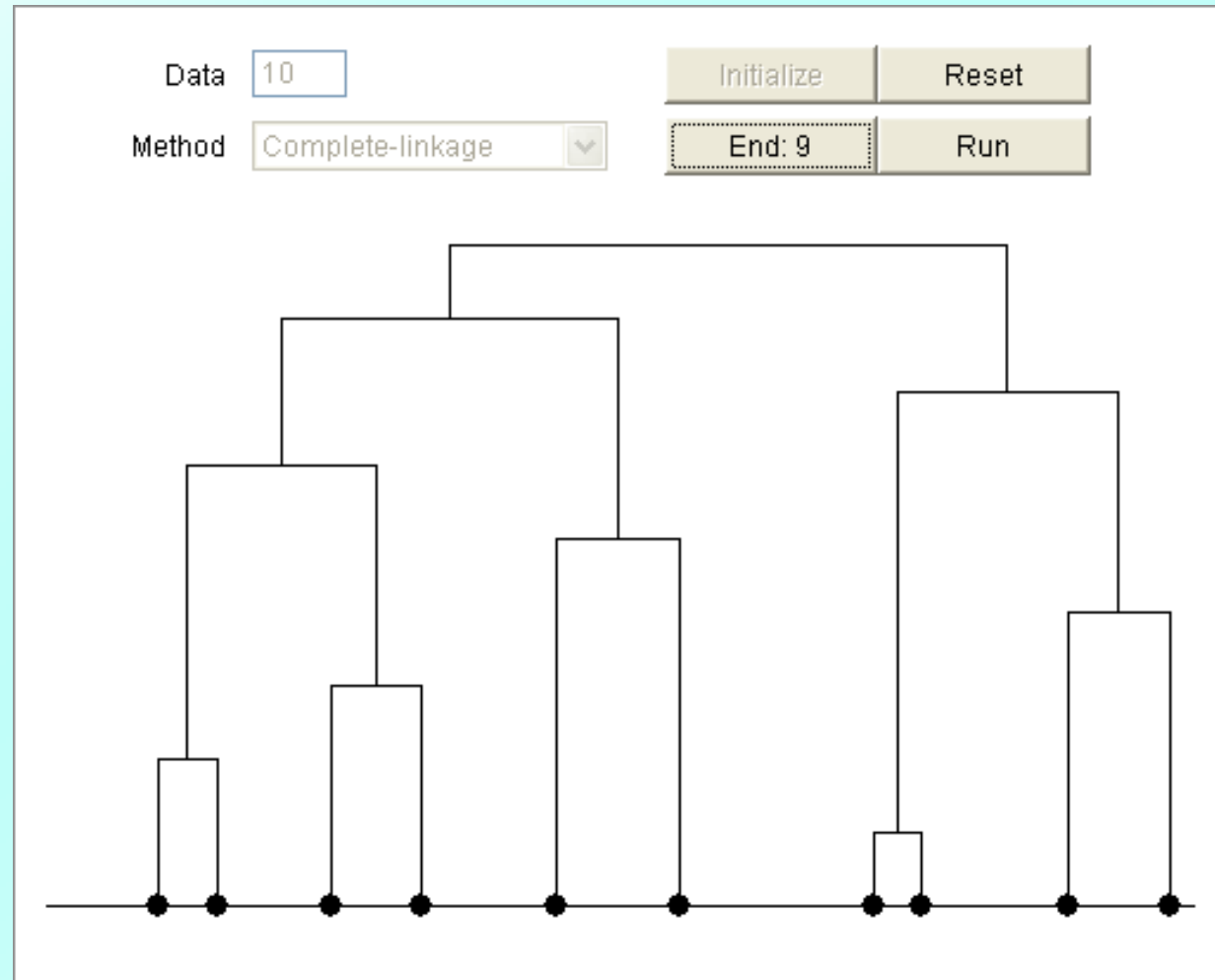
Processus de partitionnement



Processus de partitionnement



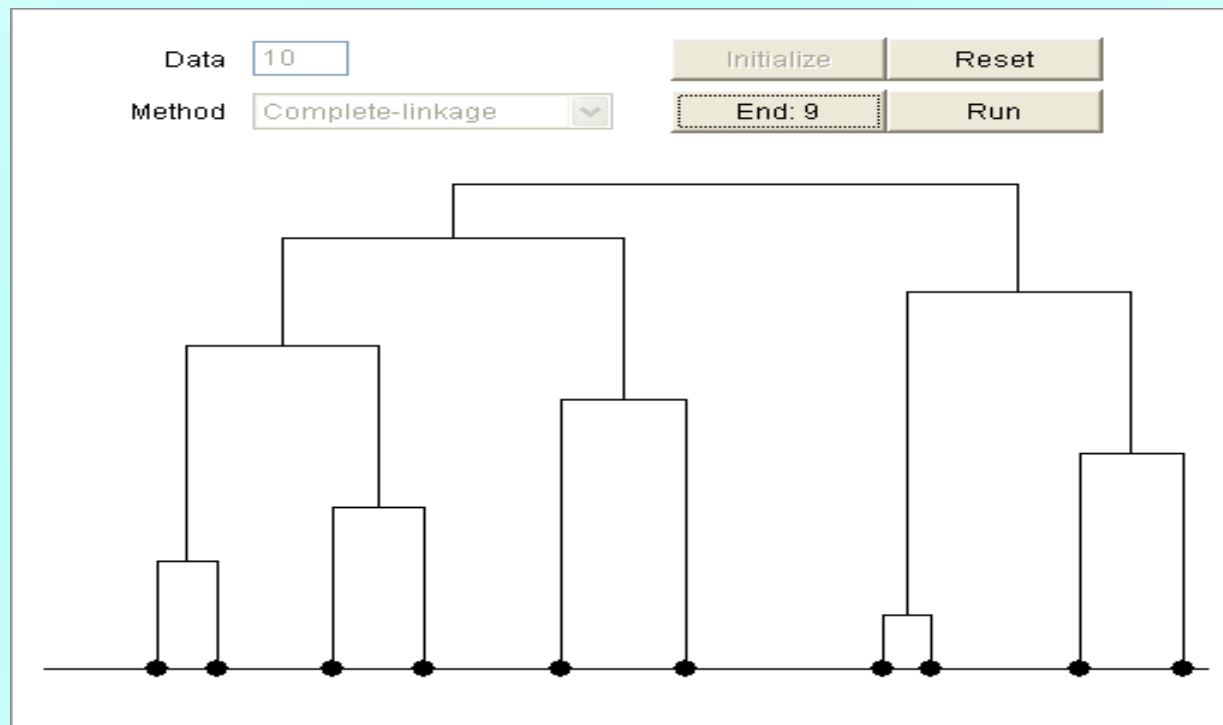
Processus de partitionnement



Processus de partitionnement

■ Identification des composants

- Parcours en profondeur du dendrogramme
- A chaque nœud
 - Si la fonction d'évaluation est meilleur que la moyenne des fils on a un composant



Processus de partitionnement

■ Approche par optimisation

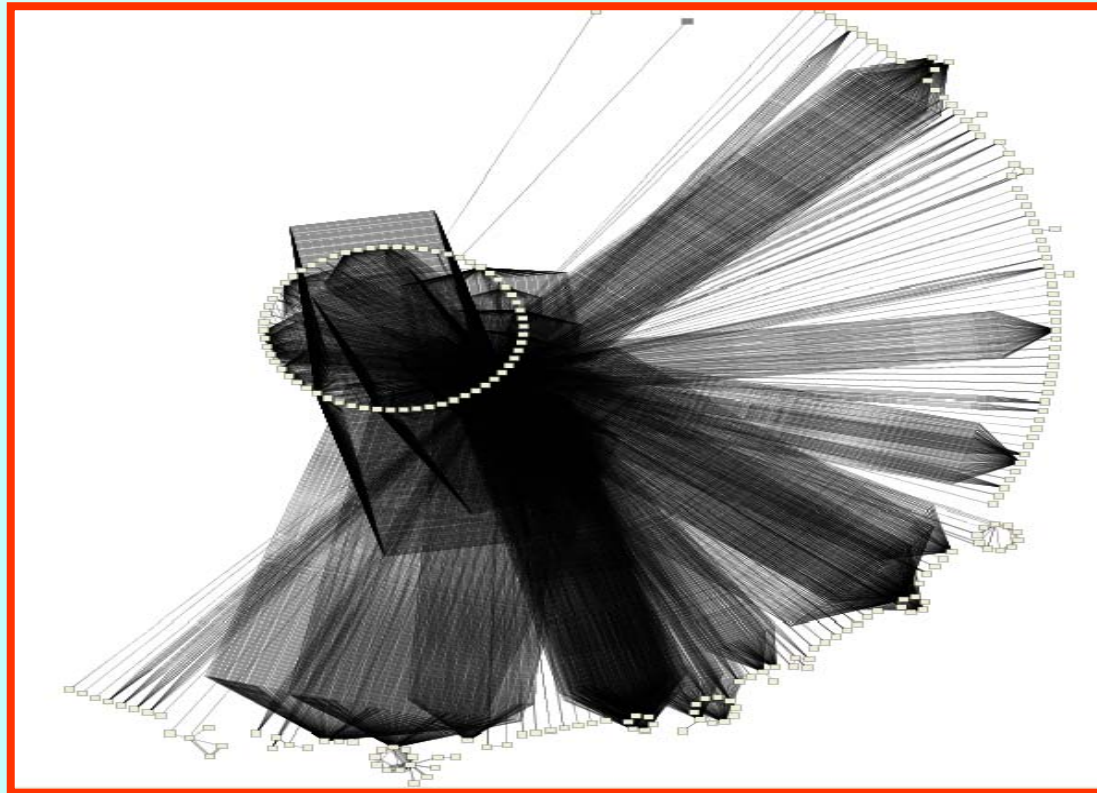
- Algorithme de Simulated Annealing
- Point de départ
 - Partition en composante fortement connexe
- Voisinage
 - 2 opérations
 - Déplacement d'une classe vers un cluster existant
 - Déplacement d'une classe vers un nouveau cluster
- Parcours de l'espace des solutions
- Si la nouvelle solution est moins bonne
 - On la conserve avec la probabilité : $e^{-\frac{\Delta E}{T}}$

Processus de partitionnement

■ Approche par optimisation

- Point de départ :

- Partition en composante fortement connexe



Processus de partitionnement

■ Approche par optimisation

- Point de départ :
 - Partition en composante fortement connexe
- Opérations
 - Fusion /éclatement de contours.
 - Déplacement d'un élément entre contours

Processus de partitionnement

■ Approche par optimisation

- Point de départ :
 - Partition en composante fortement connexe
- Opérations
 - Fusion /éclatement de contours.
 - Déplacement d'un élément entre contours
- Avantage :
 - Le résultat est situé par rapport à l'optimum
- Problèmes :
 - Complexité importante
 - Nécessité de développer un algorithme spécifique

Processus de partitionnement

■ **Processus d'exploration**

- Algorithme d'exploration
- Algorithme de recuit simulé
 - A chaque étape
 - On choisit une solution dans le voisinage de la solution actuelle
 - Si la voisine est meilleure elle devient le point de départ de l'étape suivante
 - Sinon elle peut être acceptée comme nouveau point de départ
 - Évite les minima locaux
 - Fonction de la température

Processus de partitionnement

■ **Processus d'exploration**

- Le point de départ
 - La solution de la hiérarchie de contrainte
- La température initiale
 - Calculer pour un taux d'acceptation de 80%
- Le schéma de refroidissement
 - Géométrique
 - À chaque étape la température décroît d'un facteur constant
 - Tend vers 1

Processus de partitionnement

■ **Processus d'exploration**

● Définition du voisinage

➤ Opérateurs de modification

- Fusion de deux ensembles de classes
- Déplacement d'une classe vers un autre ensemble
- Création d'un nouvel élément de la partition à partir d'une classe

➤ Choix de l'opérateur aléatoire

- Mais limité par les contraintes
 - Si deux classes doivent être dans le même ensemble

Processus de partitionnement

■ Processus d'exploration

- Acceptation d'un nouveau point de départ

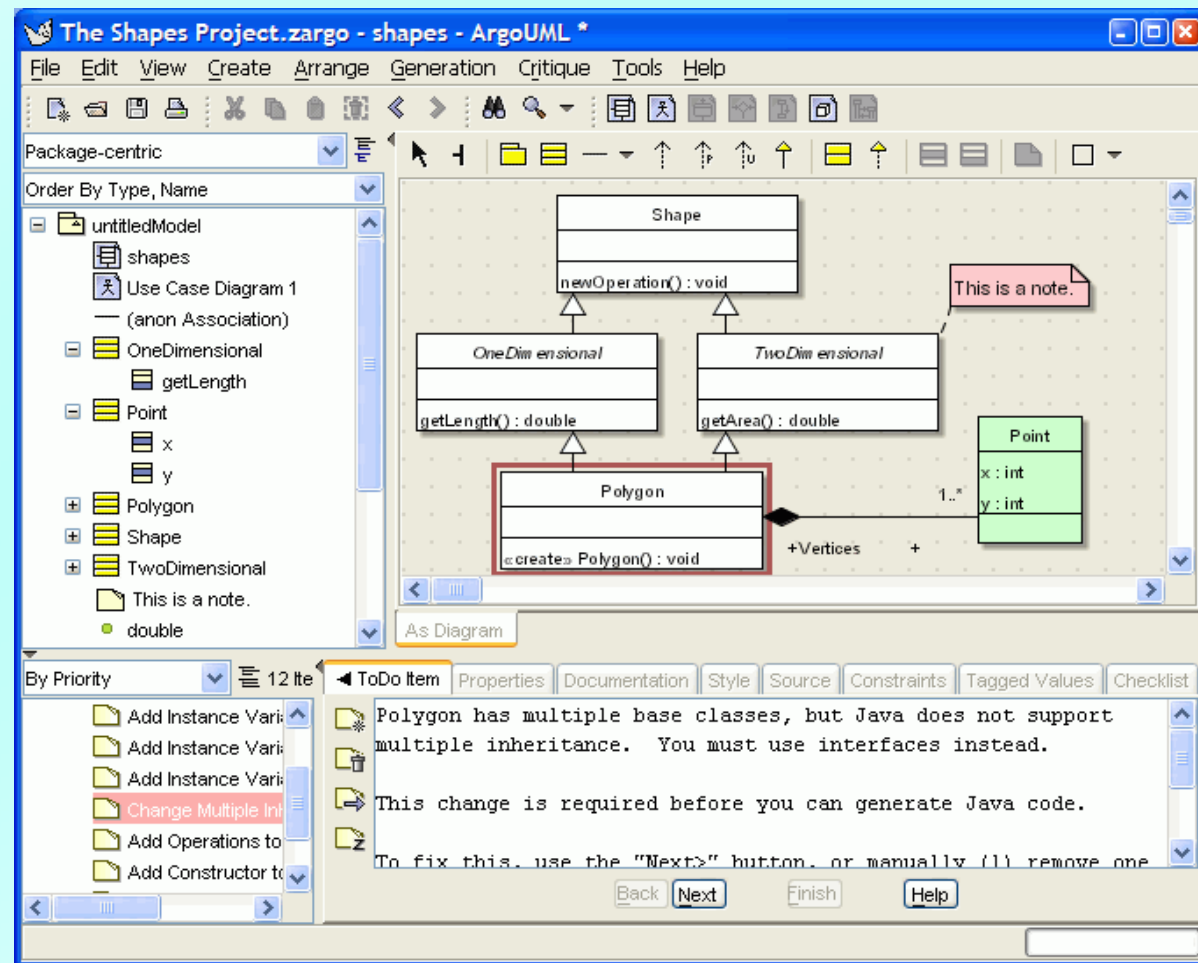
	$\Delta F > 0$	$\Delta F < 0$
$\Delta C > 0$	A	$p1(\Delta F, \Delta c, T)$
$\Delta C < 0$	$p2(\Delta F, \Delta c, T)$	$p3(\Delta F, \Delta c, T)$

- A : acceptation
- $p1 > p2 > p3$

Quelques cas d'étude

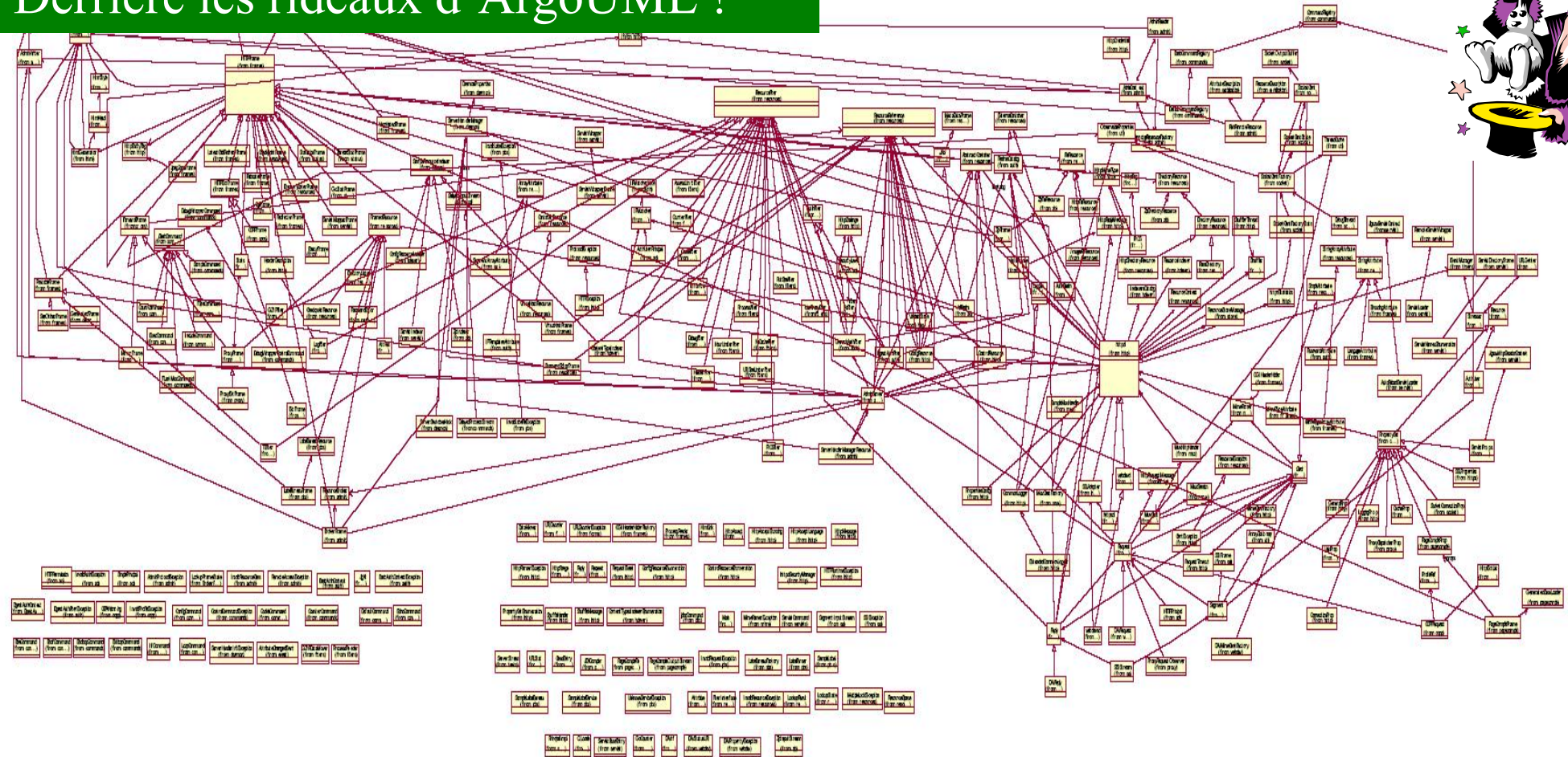
■ ArgoUML

- Un outil de modélisation en UML développé en JAVA.
- Contient 1210 classes.
- Plus de 55000 relations d'utilisation entre ces classes.

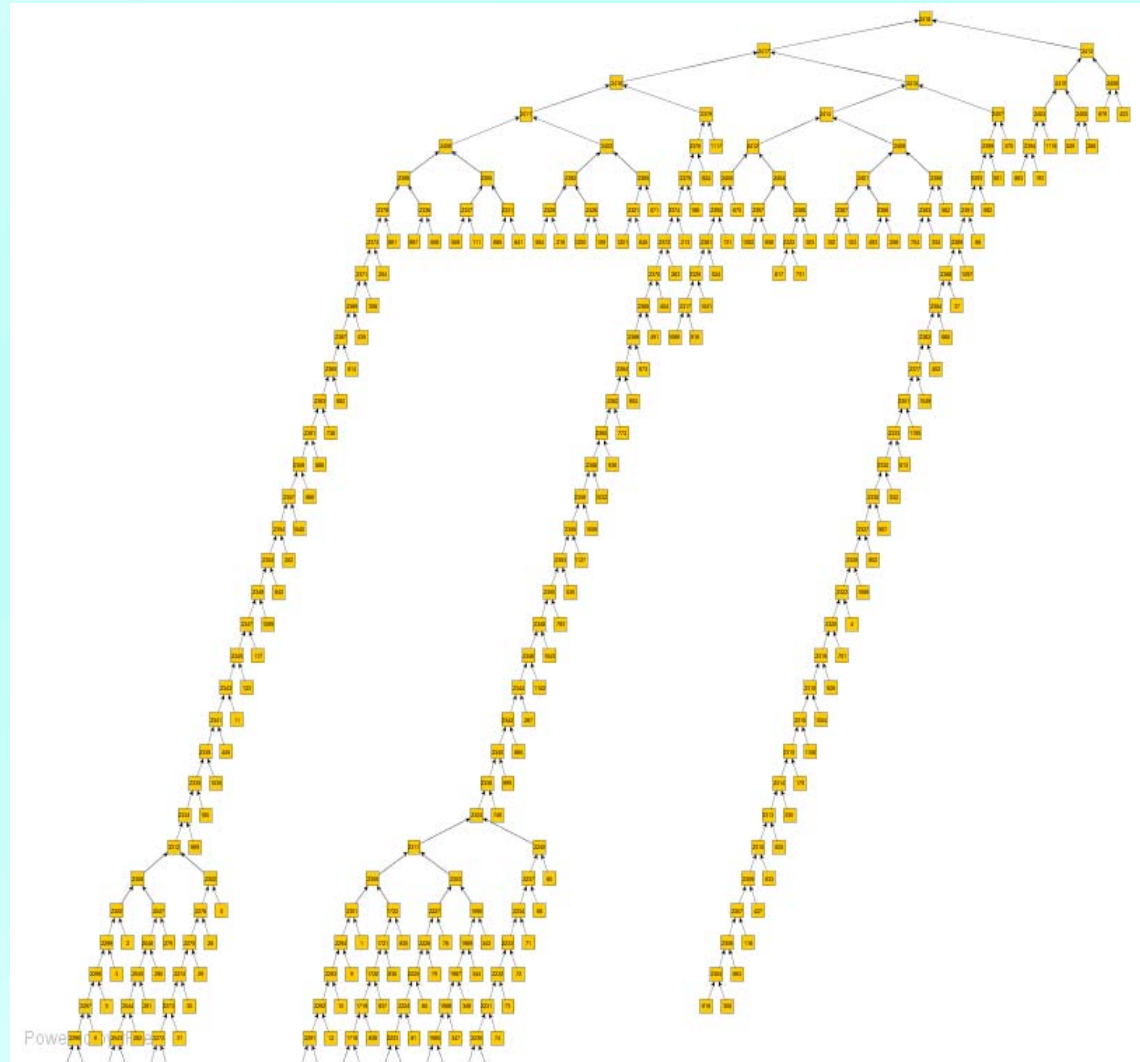
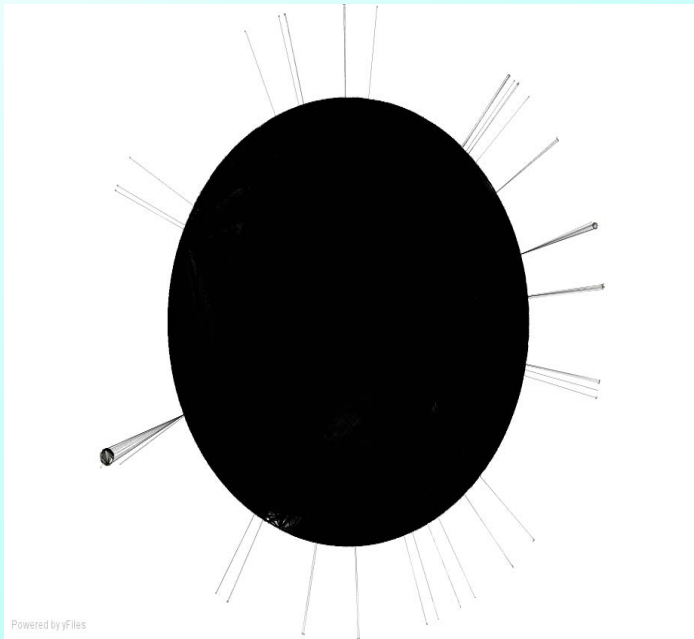


Quelques cas d'étude

Derrière les rideaux d'ArgoUML !



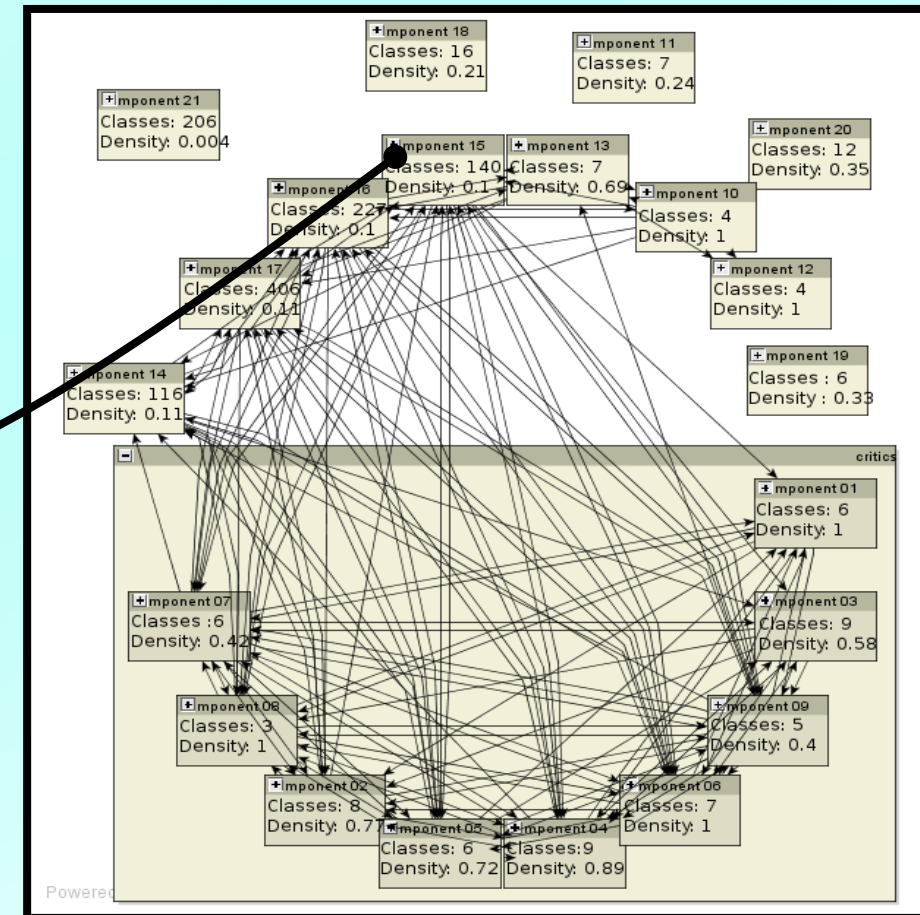
Quelques cas d'étude



Quelques cas d'étude

Treize composants identifiés

- Le nombre de classes par composant
 - ✓ Moyenne 57 classes/ composant
- ✓ Quelques composants identifiés
 - ✓ Composant « génération de code »
 - ✓ Composant « gestion de diagrammes »
 - ✓ Composant « IU »
 - ✓ Etc.



Quelques cas d'étude



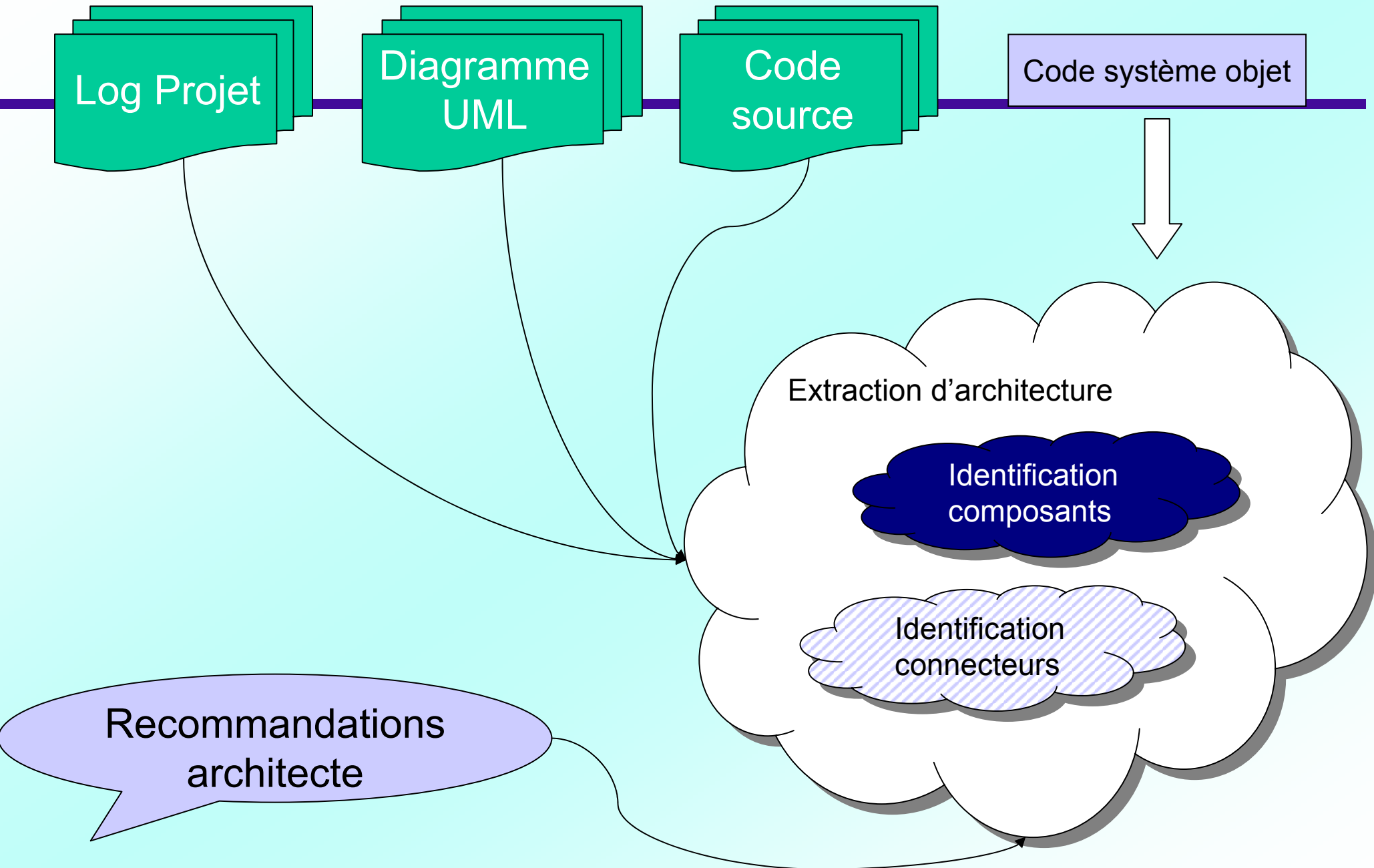
Documentation et recommandations

- Manque de prise en compte de l'architecte dans les guides précédents

- Utilisation de la documentation et des recommandations
 - Pour déterminer le point de départ de l'exploration
 - Pour réduire l'espace des solutions
 - Pour diriger l'exploration
 - Choix de la solution suivante à partir de la solution actuelle

Documentation et recommandations

- 3 documentations utilisées
 - Les diagrammes UML
 - Les commentaires et la sémantique contenu dans le code source
 - Les données de gestion de projet
 - Log CVS
- Les recommandations de l'architecte
 - Nombre de composants
 - Associations entre classes
 - Dissociations entre classes
 - Etc.



Documentation et recommandations

- Un processus interactif avec l'architecte
 - Si l'architecte est disponible

- Un processus en 4 étapes
 - Extraction des informations de la documentation
 - Validation des informations extraites par l'architecte
 - Modification des relations extraites
 - Suppression
 - Ajout
 - Recommandations de l'architecte

 - Calcul d'une hiérarchie de contraintes pour la solution

Extraction des informations

- A partir de chaque type de documentation disponible
- On extrait une hiérarchie de regroupement de classe
- Plus des informations supplémentaires suivant le type de documents
 - Cas d'utilisation du groupe de classes
 - Diagrammes UML
 - Mots clefs associés au groupe de classes
 - Code source

Validation par l'architecte

- Pour chaque hiérarchie obtenue
 - **Étape 1 : l'architecte peut**
 - La modifier
 - Ajouter/supprimer des informations
 - Mots clefs
 - Cas d'utilisation
 - Valider les informations sans modification
 - Ne rien faire
 - **Étape 2 : l'architecte peut ensuite**
 - Sélectionner une partition à partir d'une hiérarchie
 - Lancer la sélection par défaut puis
 - *valider le résultat*
 - *Invalider le résultat*
 - *Ne rien faire*

Extraction des informations

Automatique

Validation des informations extraites

Recommandations de l'architecte

Manuel

Création du réseaux de contraintes

Semi- Automatique

Recommandations de l'architecte

- Fournir une partition des classes
 - Avec des informations supplémentaires sur chaque groupe
- Préciser des groupes de classes
- Préciser des exclusions entre classes
- Donner un ensemble de composants associés à
 - Des cas d'utilisation
 - Des mots clefs
- Préciser un nombre de composant

Calcul d'une hiérarchie de contraintes

- Hiérarchie de contraintes à trois niveaux
 - Contraintes requises
 - Contraintes fortes
 - Contraintes faibles

- Les recommandations donnent des contraintes requises

- Les partitions sélectionnées ou validées par l'architecte (étape 2) sont des contraintes requises

- les partitions extraites de hiérarchies validées (étape 1) par l'architecte sont des contraintes fortes

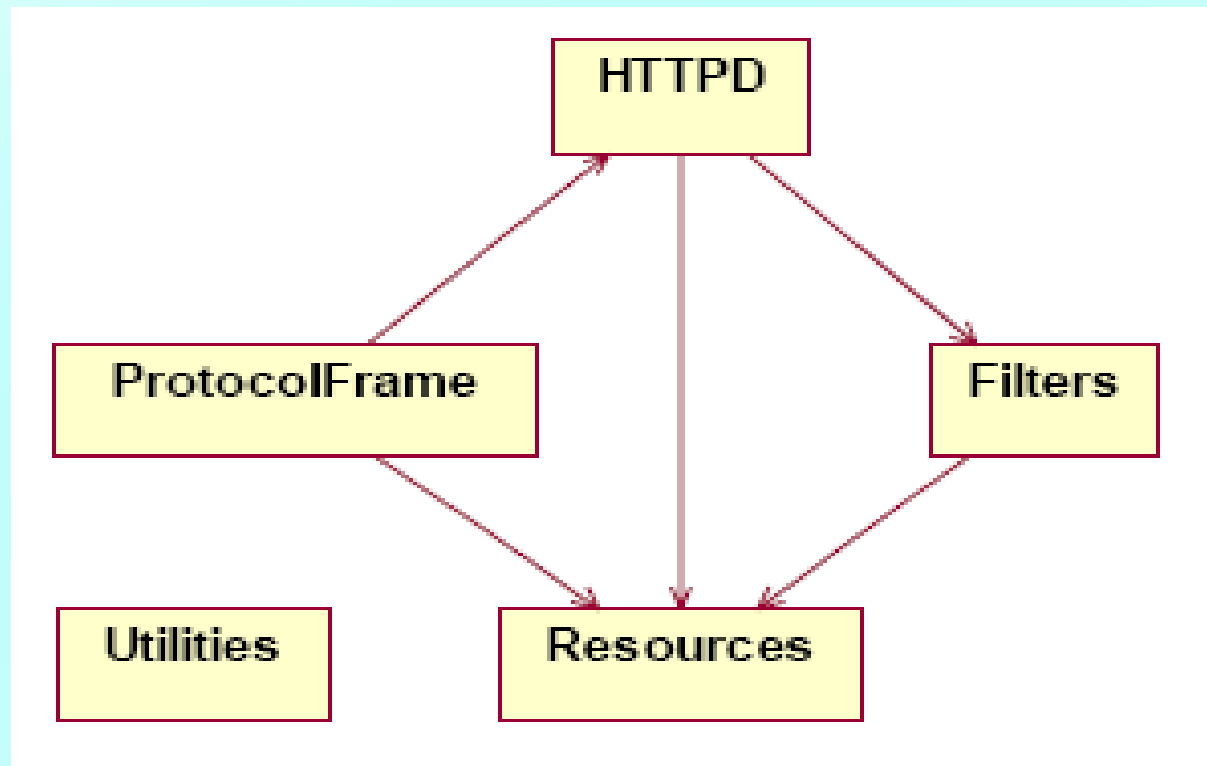
- Le reste constitue des contraintes faibles

Calcul d'une hiérarchie de contraintes

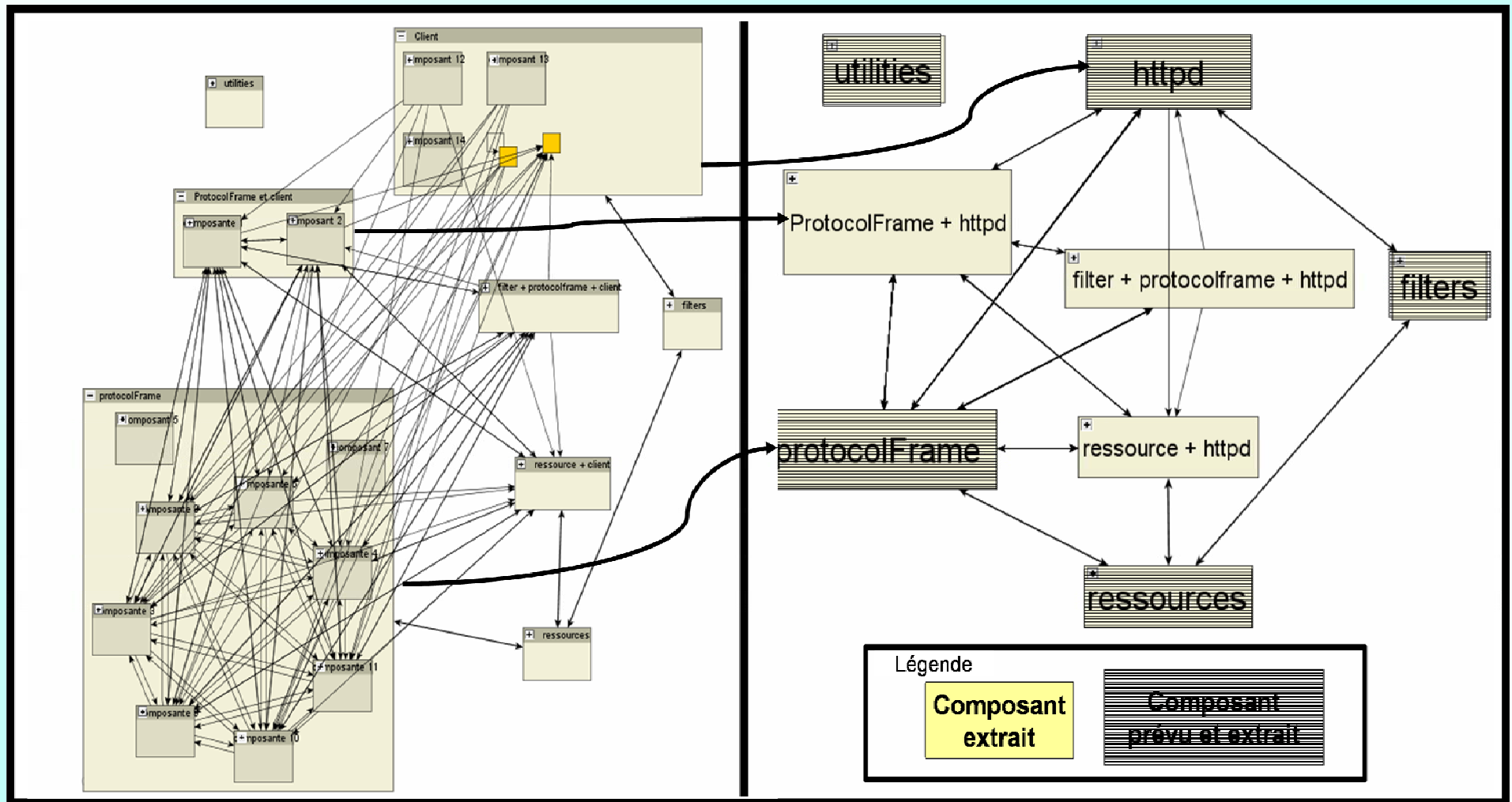
- En cas de conflit entre deux contraintes
 - Dans un même niveau
 - On interroge l'architecte
 - On conserve les deux contraintes par défaut
 - Dans des niveaux différents
 - On conserve la contrainte de plus haut niveau
 - On élimine la contrainte la plus faible

Cas d'étude : Jigsaw

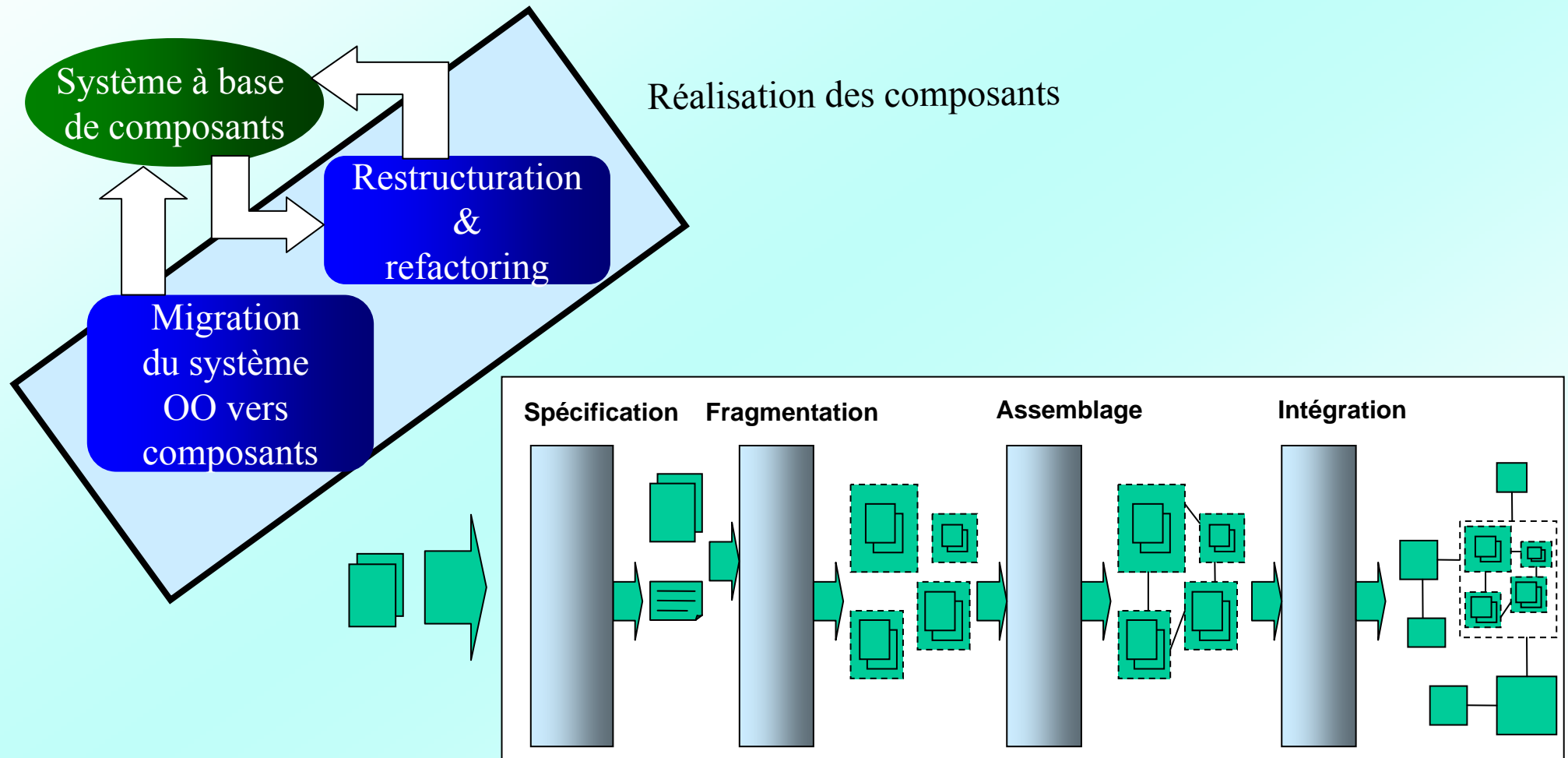
- **Java web serveur**
- **300 classes**
- **Architecture connue**
- **Utilisé comme cas d'étude par d'autres approches**



Cas d'étude : Jigsaw

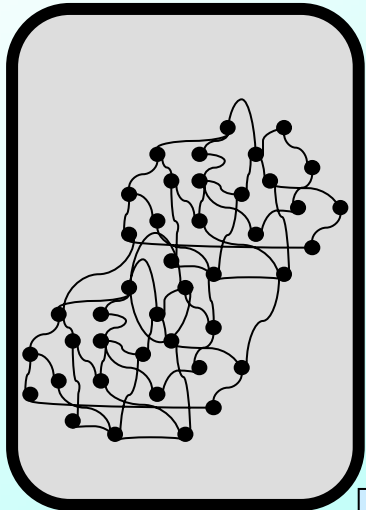


Restructuration du code objet

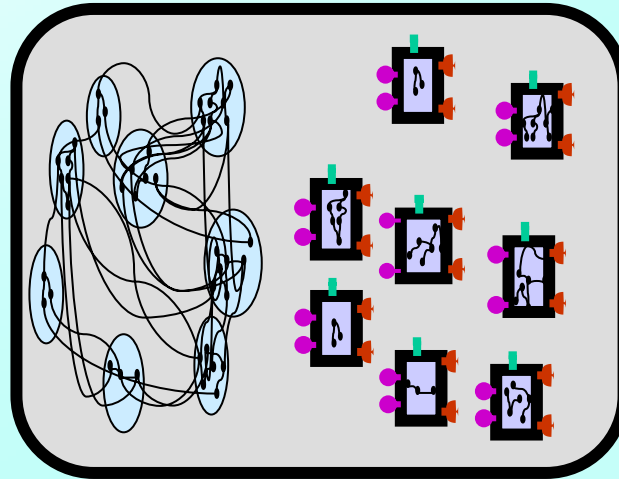


Restructuration du code objet

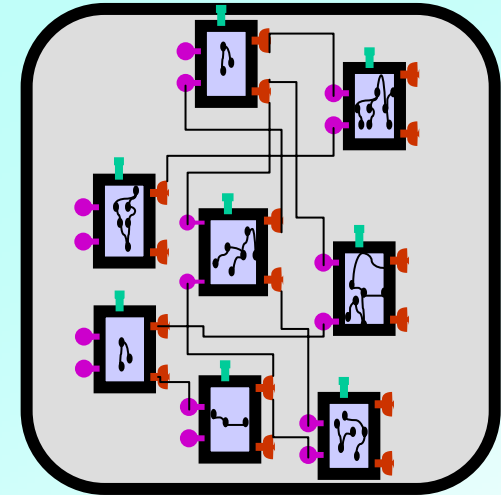
Les classes objets



Les contours des composants



Le système BC



Spécification
des composants
à générer

- Analyse et représentation du code sous forme de graphe
- Fragmentation du code
- Création des composants
 - ✓ Transformation de graphe
 - ✓ Intégrité & cohérence des composants

- Assemblage des composants générés
 - ✓ Interfaces non fonctionnelles
 - Ressources partagées

La suite ...

- Identification des connecteurs
- Application pour systèmes embarqués
- Extraction d'architecture par analyse dynamique
- Coévolution architecture/implémentation
- Migration vers les composants/migration vers les services (SOA)

La suite ...

■ Identification de composants réutilisables

