Introduction aux composants logiciels

M. Belguidoum

Université Mentouri de Constantine

Master2 Académique

Plan

Les limites de la programmation par objets

- **Pas d'expression des ressources nécessaires** : seules sont définies les interfaces fournies, non celles requises
- **Absence de vision globale de l'application** :
  - les principaux concepts sont définis au niveau d'un objet individuel
  - pas de notion de description globale de l'architecture
- **Difficulté d'évolution** : conséquence de l'absence de vision globale
- pour certaines infrastructures **absence de services (propriétés non fonctionnelles)** : les services nécessaires doivent être réalisés *à la main* (persistance, sécurité, tolérance aux fautes, etc.)
- **Absence d'outils d'administration (composition, déploiement)**
- **Conclusion**
  - charge importante pour le programmeur
  - incidence sur la qualité de l'application
  - une partie du cycle de vie n'est pas couverte

Composant vs objet

- plus haut niveau abstraction
- meilleure encapsulation, protection, autonomie : programmation systématique et vérifiable
- communications plus explicites : port, interface, connecteur
- connectables : schéma de connexion (ADL *Architecture Description Langage*) : *plan* applicatif
- séparation *métier*, technique
- meilleure couverture du cycle de vie : conception, implémentation, packaging, déploiement, exécution

Composant : définition

- 1ere apparition terme [McIlroy 68]
- 30 ans plus tard : Sun EJB, OMG CCM, MS .NET/COM+,
- plusieurs définitions plus au moins similaires
- recensement [Szyperski02]

Définition :[Szyperski02]

A component is a unit of composition with contractually specified interfaces and context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

Composant : définition

- **Définition** : module logiciel autonome
  - unité de déploiement (installation sur différentes plates-formes)
  - unité de composition (combinaison avec d'autres composants)
- **Propriétés** :
  - spécifie explicitement la ou les interface(s) fournie(s) (attributs, méthodes)
  - spécifie explicitement la ou les interface(s) requise(s) pour son exécution
  - peut être configuré
  - capable de s'auto-décrire
- **Intérêt** :
  - permettre la construction d'applications par composition de briques de base configurables
  - séparer les fonctions des fournisseurs et d'assembleur (conditions pour le développement d'une industrie des composants)

Les modèles de composants

Pour mettre en oeuvre des composants, il faut

- Un modèle à composants
- Une infrastructure à composants

Les modèles de composants

- Un modèle de composants définit
  - les entités
  - leur mode d'interaction
  - leur mode de composition
- Plusieurs niveaux de modèles
  - Abstrait (définition des entités, de leurs relations)
  - Concret (représentation particulière du modèle abstrait)

Infrastructure à composant

- met en oeuvre le modèle
- permet de
- construire
- déployer
- administrer
- exécuter des applications conformes au modèle

Infrastructure à composants

- De Nombreux modèles de composant
- construits au-dessus Java, C, C++
- EJB, Java Beans, CCM, COM+, JMX, OSGi, SCA
- Fractal, K-Component, Kilim, OpenCOM, FuseJ
- Jiazzi, SOFA, ArticBeans, PECOS, Draco, Wcomp, Rubus, Koala, PACC-Pin Bonobo, Carbon, Plexus, Spring
- au niveau analyse/conception : UML2

Conséquence de la multiplicité des modèles

- **multiplicité du vocabulaire** :
  - composant, bean, bundle
  - interface/liaison, port/connecteur, facette, puits, source
  - requis/fourni, client/serveur, export/import, service/référence
  - conteneur, membrane, services techniques, contrôleur
  - framework, serveur d'applications
- **exemples** :
  - Fractal : composant, interface, liaison, client/serveur
  - CCM : composant, facette, port, puits, source
  - UML2 : composant, fragment, port, interface
  - OSGi : bundle, package importé/exporté, service/référence
- un même terme peut avoir des significations différentes selon les modèles

Que doit fournir un modèle à composants ?

- **Encapsulation**
  - Séparation entre interface et réalisation
  - Interfaces = seule voie d'accès à un composant
  - Possibilité de définir des interfaces multiples (plusieurs vues d'un même composant)
- **Composition**
  - Dépendances explicites (expression des ressources fournies et requises)
  - Composition hiérarchique (un assemblage de composants est un composant)

Que doit fournir un modèle à composants ?

- **Description globale** : si possible exprimée formellement (langage de description)
- **Réutilisation et évolution** :
  - modèles génériques de composants
  - adaptation (interface de contrôle)
  - reconfiguration

Que doit fournir une infrastructure à composants ?

Couverture du cycle de vie

- Non limitée aux phases de développement et d'exécution
- **Administration**
  - **Déploiement** : installation et activation des composants
  - **Surveillance** : collecte et intégration de données, tenue à jour de la configuration
  - **Contrôle** : réaction aux événements critiques (alarme, surcharge, détection d'erreur)
- **Maintenance**
  - Maintien de la disponibilité de l'application
  - Évolution (redéploiement, reconfiguration) pour réagir à l'évolution des besoins et de l'environnement

Que doit fournir une infrastructure à composants ?

Services communs

- Propriétés non fonctionnelles
- Exemples :
  - Persistance
  - Transactions
  - Sécurité
  - QoS (*Quality of Service* : qualité de service)

Composition : notion d'architecture logicielle

- **Composant**
  - Unité de composition
  - Unité de déploiement
  - Remplit une fonction spécifique
  - Peut être assemblé avec d'autres composants
  - Donc, porte une description des interfaces requises et fournies
- **Connecteur**
  - Élément permettant d'assembler des composants en utilisant leurs interfaces fournies et requises
  - Remplit 2 fonctions : **liaison** et **communication**

Composition : notion d'architecture logicielle

- **Configuration**
  - Un assemblage de composants
  - Peut, par exemple, être elle-même un composant (selon le modèle)
- La différence entre un composant et un connecteur est une différence de fonction, non de nature (un connecteur est lui-même un composant)

Schéma de composition : Client/Serveur locale

- Application Client/Serveur Locale

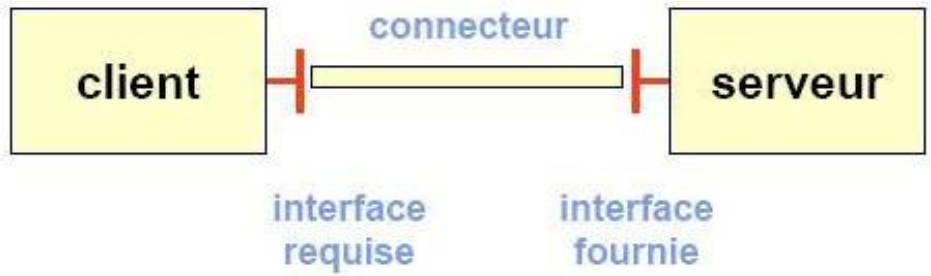


Schéma de composition : Client/Serveur locale évoluée

- Une autre vue de l'application client/serveur locale
- Le connecteur comme *composant*
- Le connecteur effectue une fonction plus évoluée

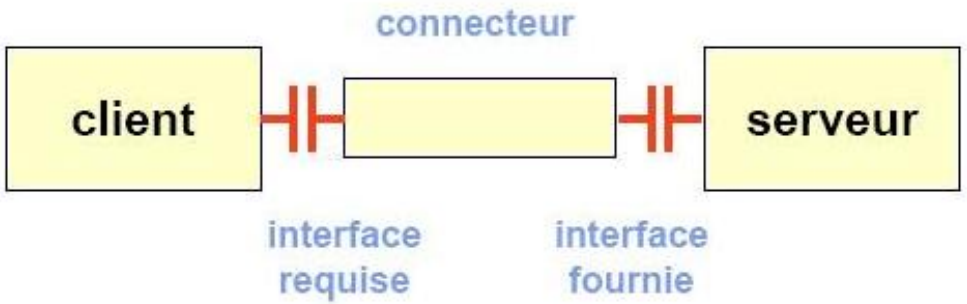


Schéma de composition : Client/Serveur répartie

- Application client-serveur répartie :
- Le connecteur comme *composant composite*
- Le connecteur gère la communication entre les deux composants répartis

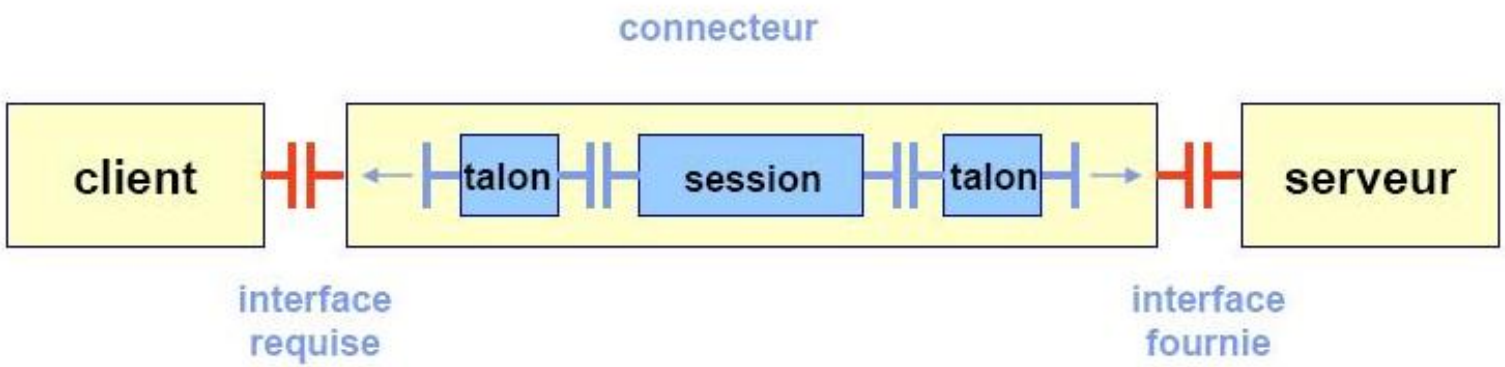
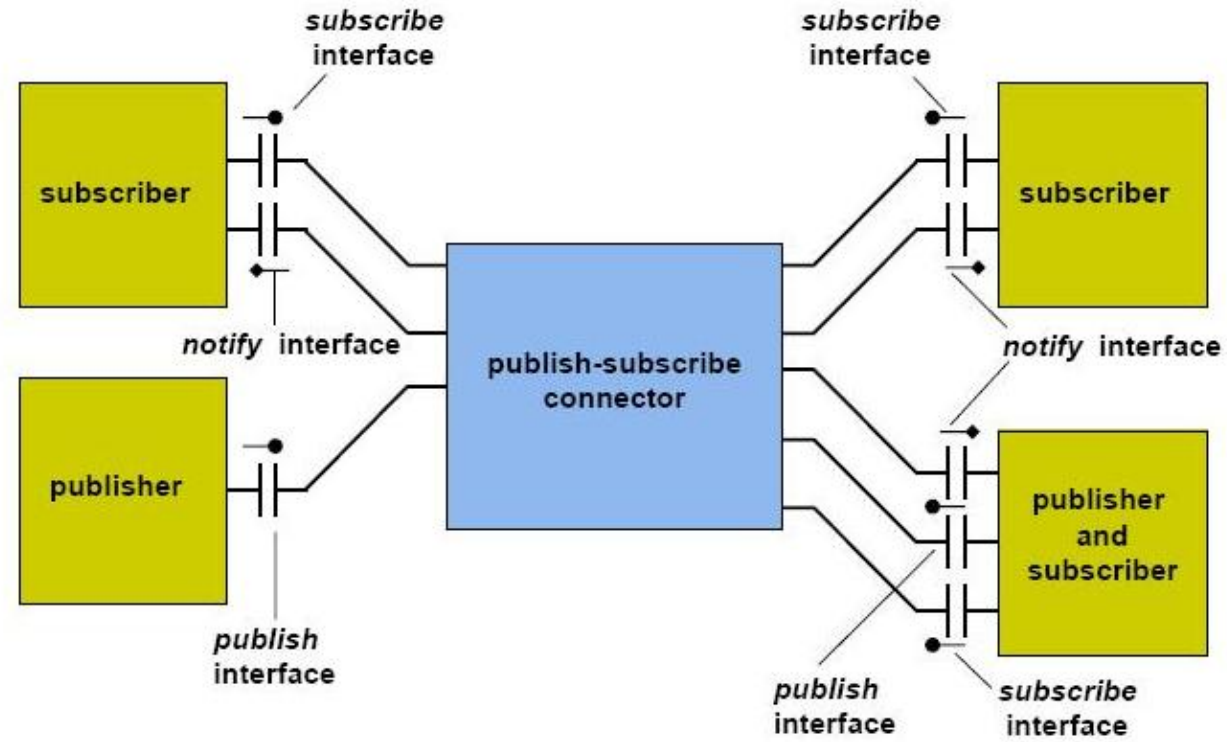


Schéma de composition : Publish/Subscribe

- Connecteur plus évolué : *publish-subscribe*



Les catégories des modèles de composants

- triptyque : **composant, interface, liaison**
  - un composant fourni et/ou requiert une ou plusieurs interfaces
  - une liaison est un chemin de communication entre une interface requise et une interface fournie
- triptyque : **composant, port, connecteur**
  - un composant fourni et/ou requiert une ou plusieurs ports
  - un connecteur implémente un schéma de communication entre des composants (client/serveur, diffusion, etc.)
  - un composant est relié à un connecteur via un ou plusieurs ports

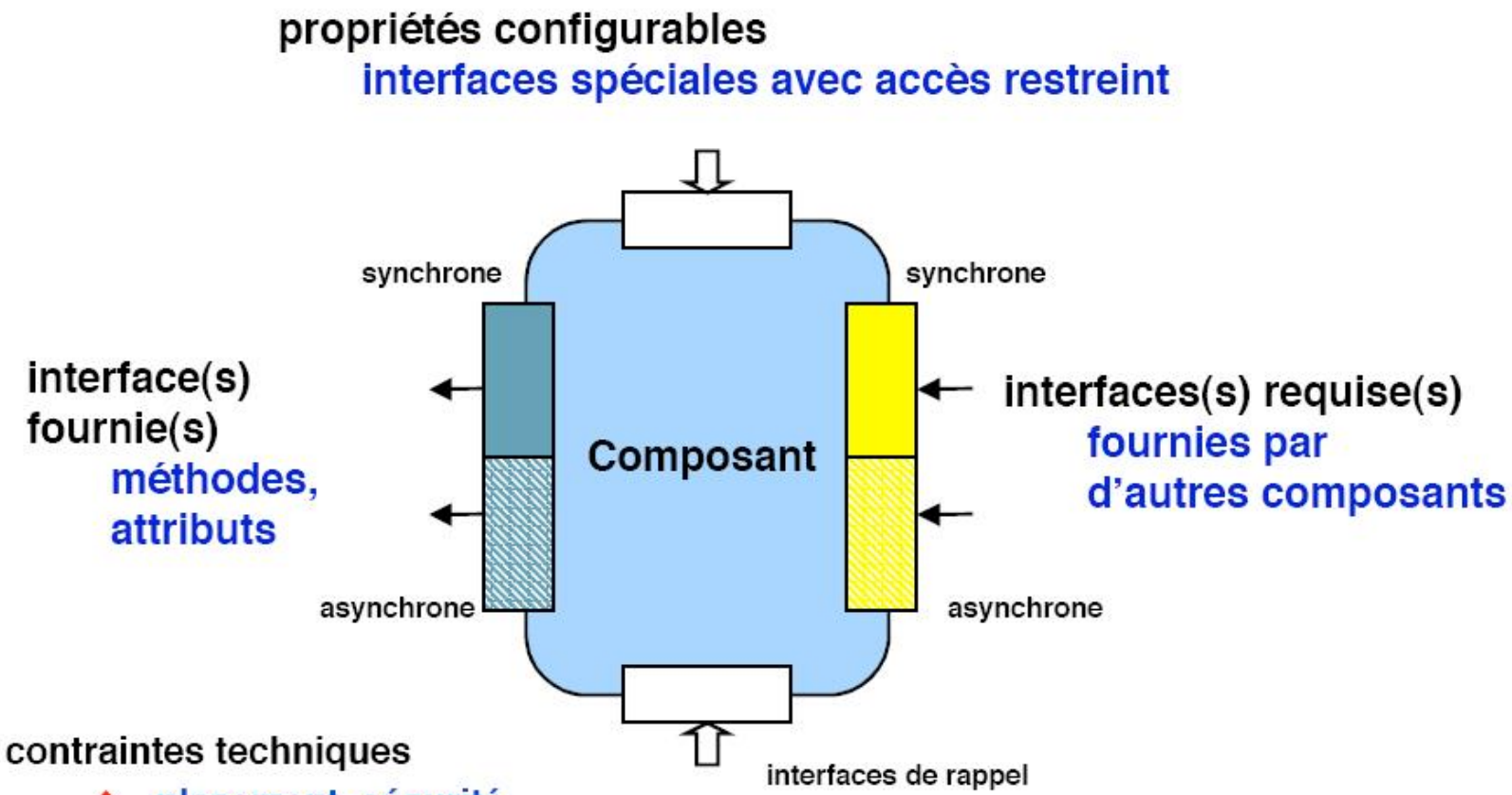
Comment décrire la composition ?

- Notion d'ADL (*Architecture Description Language*)
- Pas de standard reconnu (des tentatives) : ACME, ADML, Xarch, UML-2
- Vers l'utilisation de XML comme support commun (invisible) ?

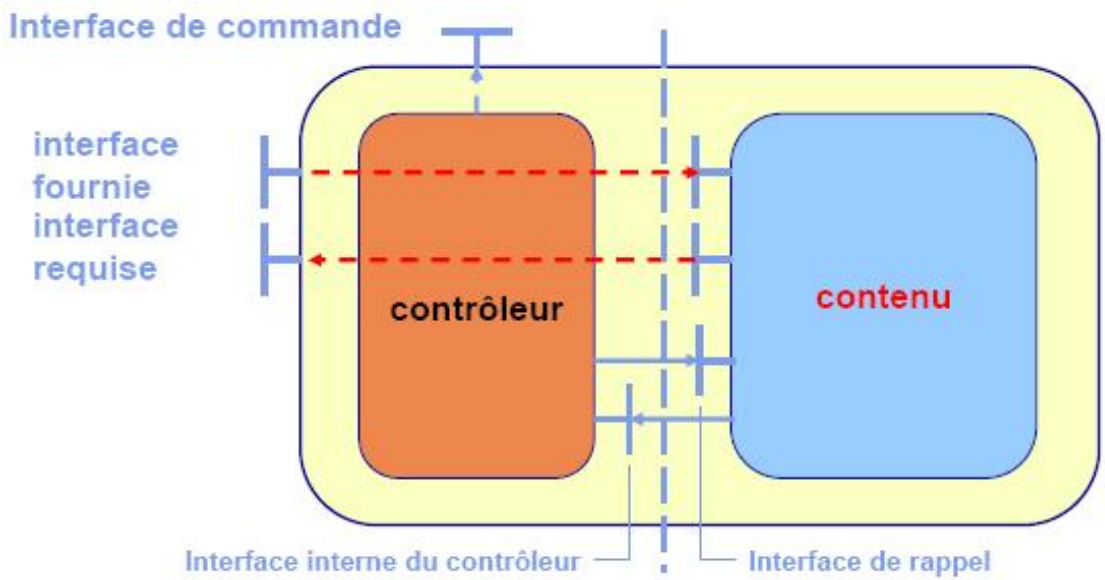
Elément d'un ADL

- **Description des composants**
  - Définition des interfaces : fournies et requises
  - Types des interfaces : synchrone et asynchrone
  - Signatures, contrats, annotations
- **Description des configurations**
  - Association interfaces fournies-interface requises
    - Directe (réalisée par édition de liens)
    - Via un connecteur : description, *rôle* = interface
- Composition hiérarchique (si le modèle l'autorise) : interfaces exportées/importées
- Interface graphique (visualisation, action)
- Aspects dynamiques

Le modèle générique des composants



Organisation logique d'un composant



La séparation entre contrôleur et contenu est motivée par la séparation des préoccupations : isoler la partie purement fonctionnelle (propre à l'application)

Fonctions d'un contrôleur

- Gestion du cycle de vie
  - Création, destruction
  - Activation, passivation
- Gestion des composants inclus : si le modèle comporte des composants composites
- Liaison : Connexion avec d'autres composants
- Médiation
  - Gestion des interactions avec d'autres composants
  - Médiation pour la fourniture de services externes
- Autres opérations réflexives :si le modèle le permet

Conteneur : Un canevas pour composants

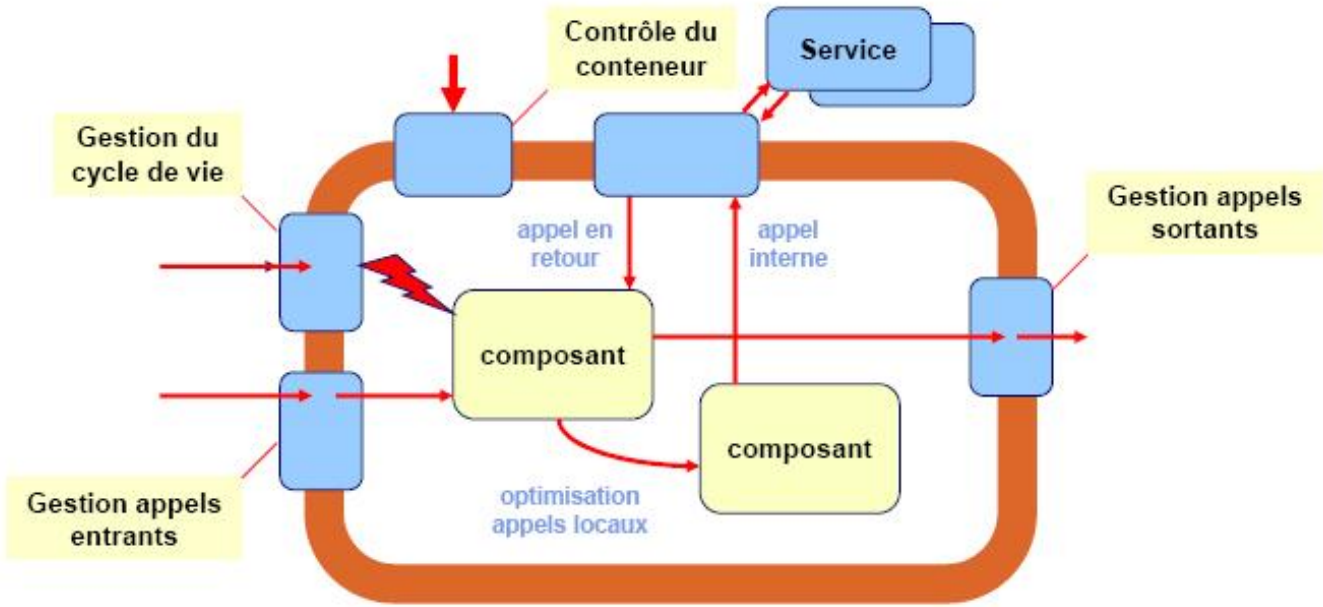
Infrastructures à conteneurs

- Canevas de base pour le support des composants
- Séparation des préoccupations
  - La partie **contenu** réalise les fonctions de l'application
  - La partie **conteneur** fournit les fonctions de l'infrastructure
- Fonctions du conteneur
  - Mise en oeuvre des fonctions du contrôleur (gestion, médiation)
  - Allocation des ressources aux composants
  - Réalisation des aspects Ṣnon fonctionnelsṪ (services communs)

Conteneurs et structures d'accueil

- **Conteneur**
  - encapsulation d'un composant (et ses composantes) prise en charge (masque) les services systèmes
  - nommage, sécurité, transaction, persistance, etc. prise en charge partielle des connecteurs invocations et événements techniquement par interposition (ou délégation)
- **Structures d'accueil**
  - Espace de déploiement du code du composant
  - Espace d'exécution des conteneurs et des composants Médiateur entre les conteneurs et les services systèmes

Le schéma générique d'un conteneur



Composition : notion d'architecture logicielle

Software Architecture

Software Architecture[a] encompasses :

- the organization of a software system
- the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements,
- the composition of these elements into progressively larger subsystems,
- the architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition.

a. http://www.sei.cmu.edu/architecture/start/community.cfm

Composition : notion d'architecture logicielle

Définition : [Clements et al., 1997]

L'architecture logicielle d'un programme ou d'un système d'information est la structure ou les structures d'un système ; elle comprend les composants logiciels, leur propriétés externes visibles et leurs relations.

Architecture logicielle : bénéfices

- **Compréhension** : abstraction de haut niveau de la structure du système
- Cadre pour la construction de l'application
  - identification des opportunités de réutilisation de modules logiciels
  - base pour la génération de code
- **Vérification**
- **Support de l'évolution du logiciel**
  - préservation des invariants
  - impact d'une modification

Complémentarité : composant et architecture

- **architecture** : construite à partir de composants, vision *top-down*
- **composants** : assemblés pour construire une architecture, vision *bottom-up*

Description d'architecture

- Ensemble des composants logiciels d'une application
  - identification des composants
  - designation des composants
- Structuration des composants : composition hierarchique (sous-composants)
- Interconnexion des composants
  - dépendances entre composants (fournit/requiert)
  - modes de communication (synchrone, asynchrone)
  - protocoles de communication

Conclusion

- Une vue générale sur les concepts de base des composants logiciels
- Les notion de composition et d'architecture logiciel
- Organisation logique des composants : contrôleur, contenu
- Toutes ces notions seront abordées dans le modèles Fractal

Références

- http://sardes.inrialpes.fr/people/krakowia/
- des figures et des transparents empruntés à S. Krakowiac, L. Seinturier, S. Bouchenak