Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Centre Universitaire de Mila
Institut des mathématiques et informatique
Département d'informatique, Master 1 : STIC
Laboratoire Modélisation des Systèmes Complexes et le Soft Computing

Programmation orientée composant

➢Dr Aouag. Mouna
➢Département d'informatique
aouag.mouna@centre_univ_mila.dz

Année Universitaire 2024/2025

- *Introduction*
- *Concepts de base*
- *Couches d'une architecture orientée composant*
- *Les avantages /Inconvénients de la POC*
- *Conclusion*

Introduction (1/4)

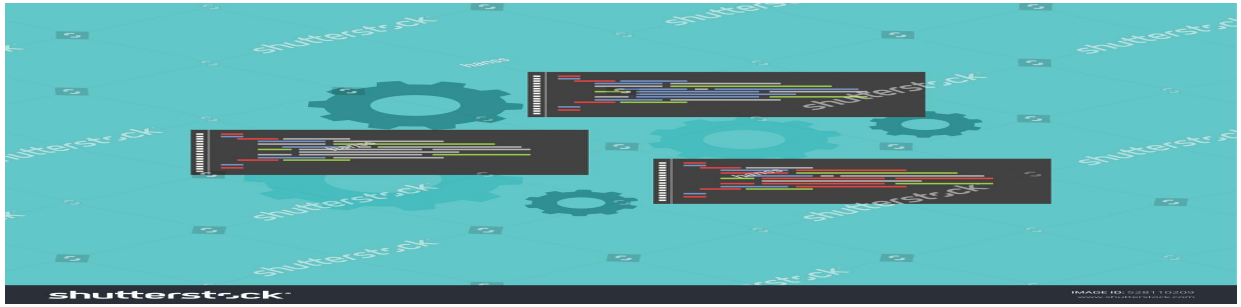
- La programmation orientée composants (POC) n'est jamais la démarche de programmation qui remplace la programmation orientée objet.
- La POC utilise une approche objet pour le développement, l'intégration, et le déploiement des applications à base de composants logiciel.

- POC est spécialement pratique pour faciliter le travail en équipes dont chaque équipe assure le développement d'une partie de l'application qui va ensuite être assemblé avec les autres partie pour construire une application complète. Chaque partie peut être développée selon la POC comme un composant autonome, persistant, réutilisable et remplaçable.

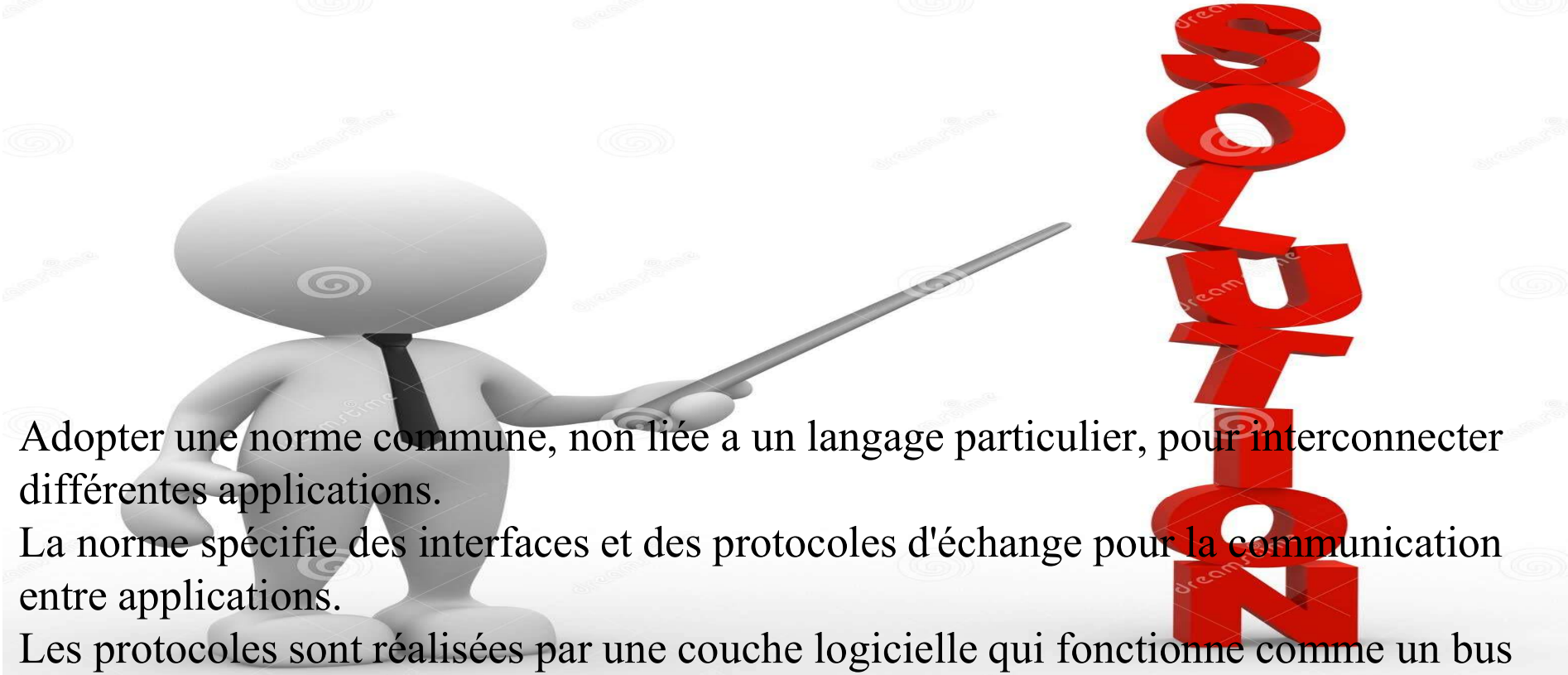
Introduction (2/4)

La programmation par composant est:

- indépendante de la POO
- évaluation de la POO:
  - Réutilisabilité accrue du code
  - Persistance des objets
  - Distribution du code et des objets (des objets peuvent être partagés par des applications différentes tournant sur des machines différentes).
  - Sécurité au niveau de l'exécution concurrente



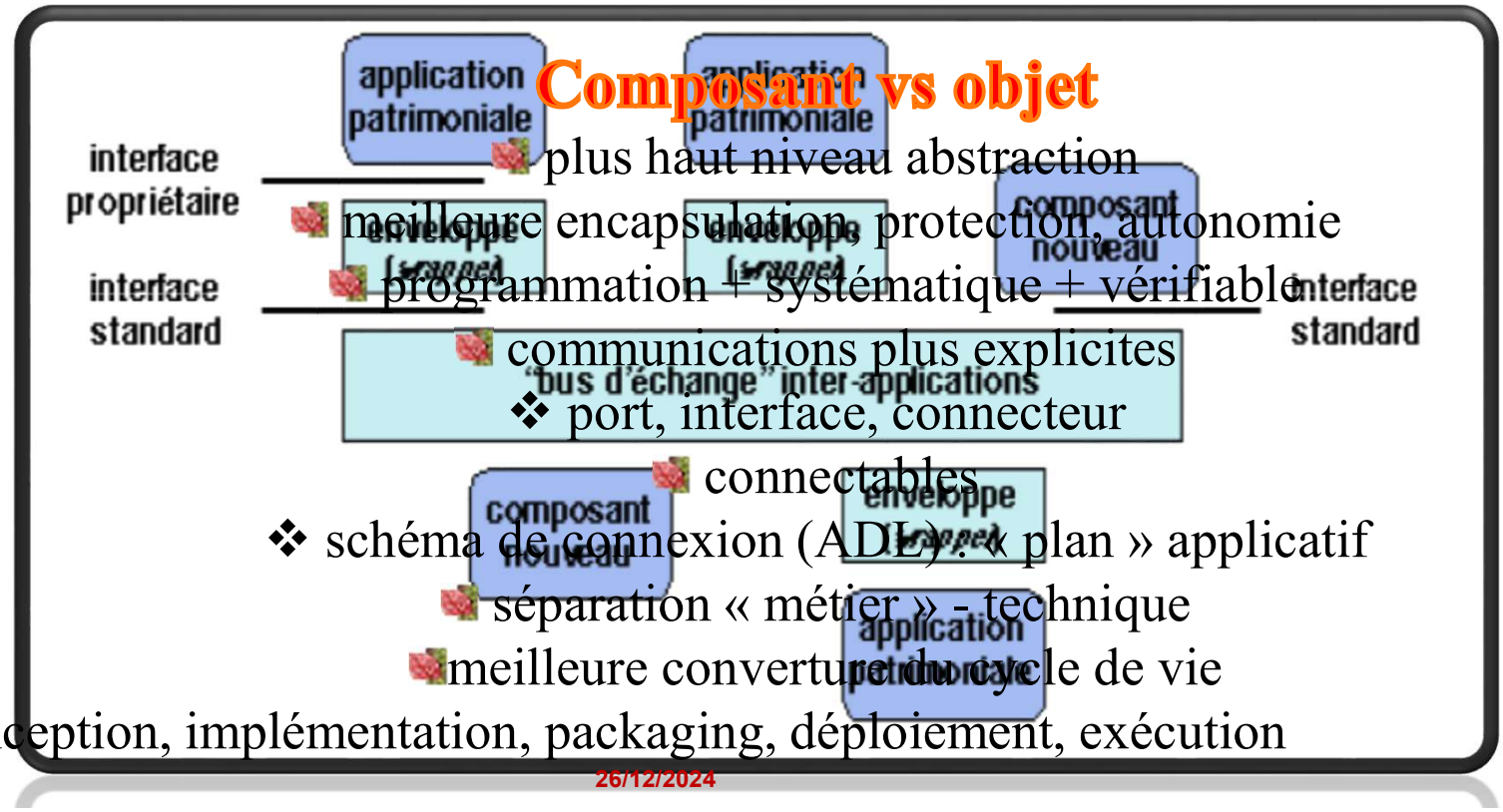
Introduction (3/4)



Adopter une norme commune, non liée a un langage particulier, pour interconnecter différentes applications.
La norme spécifie des interfaces et des protocoles d'échange pour la communication entre applications.
Les protocoles sont réalisées par une couche logicielle qui fonctionne comme un bus d'echanges entre applications: **c'est l' intergiciel (middleware)**

Introduction (4/4)

Utiliser des wrapper (envellope) = couche logicielle qui fait le pont entre l'interface originelle de l'application et une nouvelle interface conforme à la norme choisie.



Composant vs objet

- plus haut niveau abstraction
  - meilleure encapsulation, protection, autonomie
  - programmation + systématique + vérifiable
- communications plus explicites
  - ❖ port, interface, connecteur
  - connectables
  - ❖ schéma de connexion (ADL) : « plan » applicatif
- séparation « métier » - technique
- meilleure converture du cycle de vie
  - ❖conception, implémentation, packaging, déploiement, exécution

- *Introduction*
- *Concepts de base*
- *Modèles à Base de Composants Existants*
- *Les avantages /Inconvénients de la POC*
- *Conclusion*

2. Concepts de base(1/8)

2.1 Composant

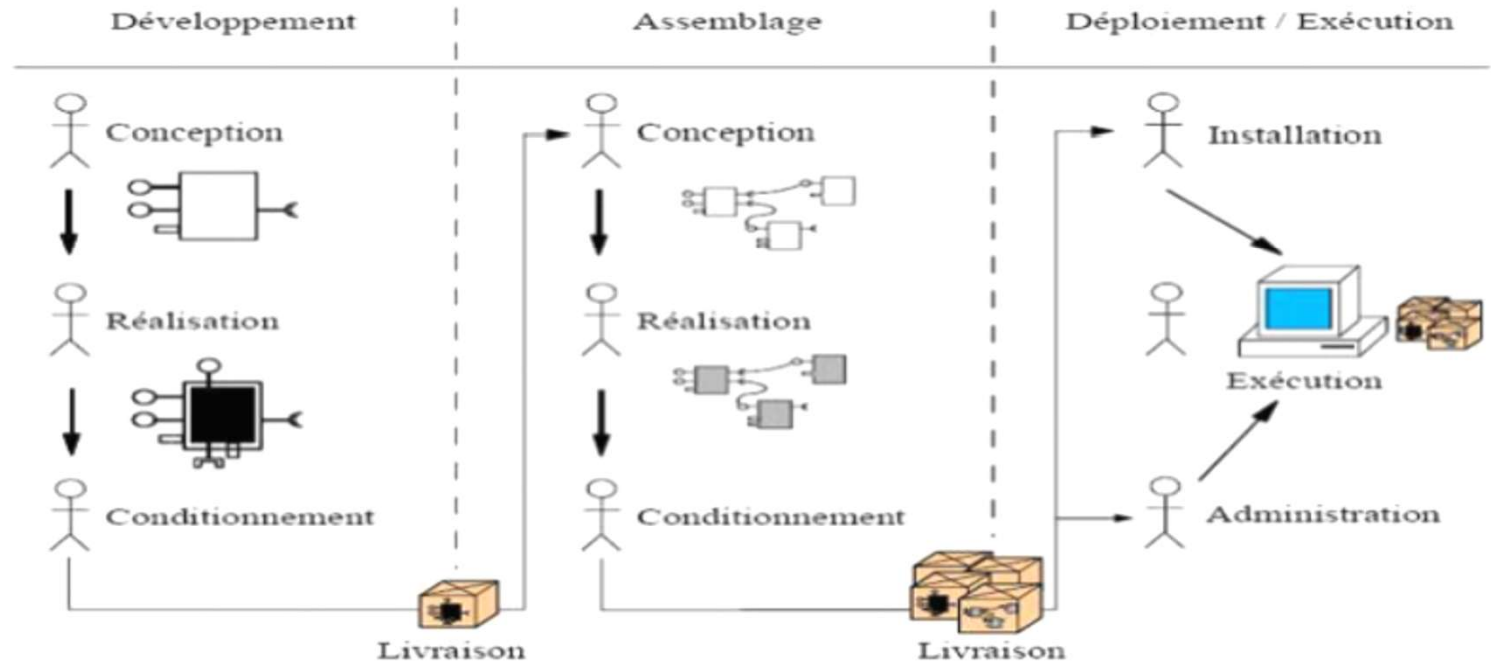


Figure 4.1- Cycle de vie d'une application à composant

2. Concepts de base(2/8)

2.2 Les éléments d'un Composant

Un composant logiciel d
- **Une interface** : à tra
sur la même application.
pour facilite aux client
avoir besoin de consulte
- **Des attributs** : ce sont
- **Une implémentation**
composant.

posants
une classe
ant sans
u

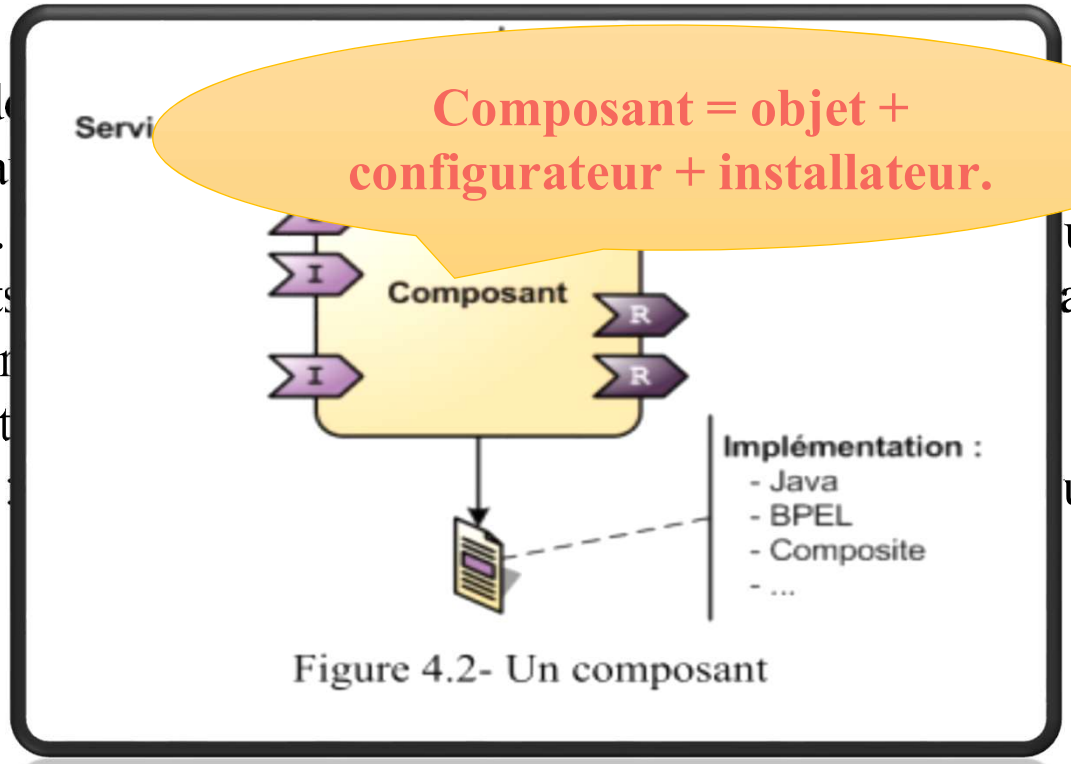


Figure 4.2- Un composant

2. Concepts de base(3/8)
2.3 Les Caractéristiques d'un Composant

Les caractéristiques d'un composant peuvent être regroupées en fonction de trois axes principaux :

Taille ,Granularité et Domaine : ou la fonction du composant est le type de rôle qui assure au sein de l'application, à savoir : **Fonctions techniques , Fonctions métiers** :

Généricité et L'abstraction

Classe de composant (équivalente à la notion de classe en OO)

- Définit l'implémentation du composant (logique fonctionnelle)
- Peut être vue à partir : **D'une vue externe et D'une vue interne**

Instance de composant (équivalente à la notion d'objet en OO)

- Obtenue à partir d'une classe de composants
- Fournit les fonctionnalités du composant à l'exécution
- Unique par rapport aux autres instances, peut avoir un état

2. Concepts de base(4/8)
2.3 Les Caractéristiques d'un Composant(suite)

Paquetage de composant

• Unité permettant de réaliser la livraison et le déploiement d'une classe de composant de manière indépendante

• Contient tout ce qui est nécessaire pour réaliser la création d'instances de la classe de composants

* Code binaire du composant

* Ressources nécessaires à son exécution (bibliothèques, images, fichiers de config)

Environnement d'exécution

• Fournit du support aux applications construites à partir du modèle à composants, lors de l'exécution

• Sorte de mini-système d'exploitation : gère les aspects divers (cycle de vie des instances, propriétés non fonctionnelles)

2. Concepts de base(5/8)
2.4 La programmation orientée composant (POC)

Consiste à concevoir un système complexe en terme **de réseau de composants** reliés entre eux par **des connecteurs**, qui collaborent pour réaliser les fonctionnalités désirées.

Connecteur : implémente un protocole spécifique de communication. Il permet de connecter des composants.

Composition : deux composants peuvent être composés pour former un composant plus complexe.

Fig2 : Plusieurs composants qui interagissent entre eux selon UML 2.0.

2. Concepts de base(6/8)
2.5 Couches d'une architecture orientée composant

A- Les technologies de composants

* Construire des entités qui répondent à certaines (petites ?!) contraintes qui vont leur permettre :

✓de bien s'intégrer à un environnement d'exécution

✓de bien s'intégrer avec d'autres entités de même type

✓environnement d'exécution =

* ce qui permet de traiter (exécuter) ces entités

* tout ce que l'environnement peut amener de services connus (gestion des mises à jour, un contexte commun à toutes ces entités, transaction, sécurité, …)

✓ entités = composants (souvent des instances ou des classes ou ...)

✓environnement d'exécution = conteneurs

✓Exemple de conteneurs : conteneur web (pour servlets, JSP), conteneur EJB (EJB) Et conteneur de programmes pour téléphones portables (MIDl et de Java ME)

2. Concepts de base(7/8)

B-Comment utiliser tous ces outils ?

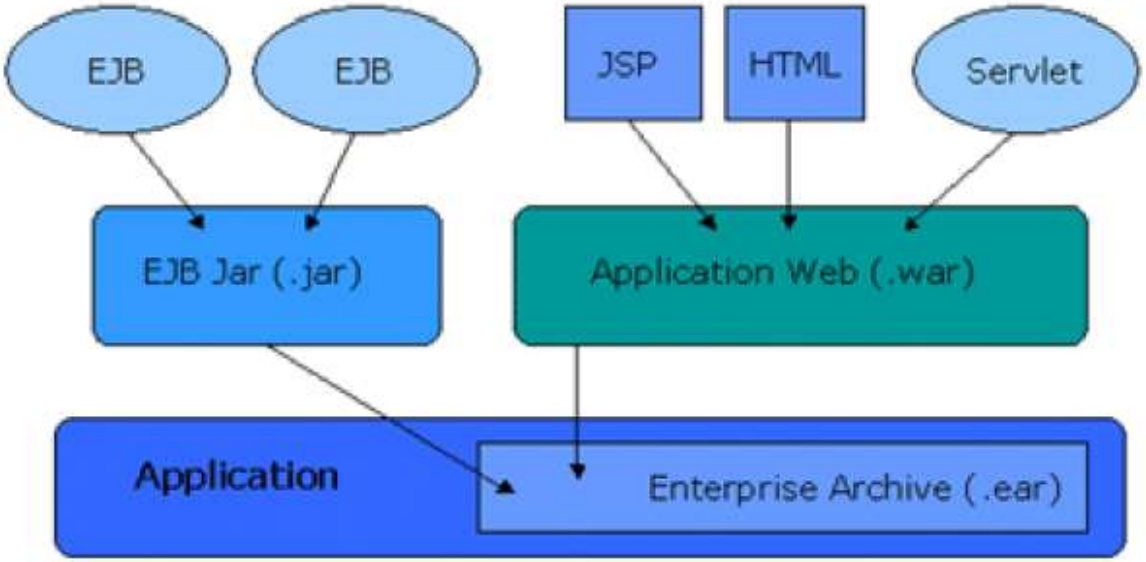
* Des règles de bons fonctionnements (bonnes écritures, bonnes utilisations, ...) ont été érigées
* Ce sont des « règles de conception » ou design pattern
* Design pattern = une solution à un problème (connu et récurrent) dans un certain contexte Cf. Gof Book Design Pattern Erich Gamma et al.

➢ Premières règles

❖Si l'interaction est simple (questions réponses) et nécessite pas d'architecture sécurisée importante (cryptage, transaction, …) utiliser seulement servlet + JSP.

❖Si l'interaction est intrinsèquement complexe et nécessite obligatoirement un service d'authentification, de confidentialité, Transactionel, utiliser les EJB (avec éventuellement servlet et JSP)

Architecture d'une application Java EE :



- 3 couches :
  - *Les composants.*
  - *Les modules regroupant les composants*
  - *Les applications regroupant les modules*

- Les modules et les applications correspondent physiquement à des fichiers d'archives : archive EJB JAR (.jar) pour un module EJB, archive WAR pour un module web, archive EAR pour une application.

- *Introduction*
- *Concepts de base*
- *Modèles à Base de Composants Existants*
- *Les avantages /Inconvénients de la POC*
- *Conclusion*

3. Modèles à Base de Composants Existants (1/2)

COM (Component Object Model, Microsoft):Résout le problème d'interopérabilité
Mais : ne propose pas une vue externe constante (les interfaces peuvent varier)
JavaBeans (Sun): Simplifier la construction d'applications grâce à la composition visuelle //**Mais :** Vise les applications non distribuées avec interface utilisateur

EJB 1 (Enterprise Java Beans, Sun): Vise les applications réparties en trois tiers (MVC).**Mais** Ne supporte pas que la partie contrôleur soit distribuée
CCM (CORBA Component Model, OMG)

- Définir l'architecture d'une application distribuée sous forme de composition d'instances de composants
- Utilisation d'un langage abstrait pour la description d'interfaces (IDL) + d'un langage CIDL pour décrire les implémentations) → Obligation de tout écrire en langage abstrait pour de compiler pour le langage cible
- Supporte les applications distribuées construits au-dessus **Java, C, C++**

3. Modèles à Base de Composants Existants(2/2)

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 (cadriciel), serveur d'applications
- Grande variabilité dans les propriétés attachées aux notions exemples

• **Fractal** : composant, interface, liaison, client/serveur
• **CCM** : composant, facette, port, puits, source
• **UML** 2 : composant, fragment, port, interface
• **OSGi** : bundle, package importé/exporté, service/référence

un même terme peut avoir des acceptations ≠ selon les modèles qualifier les notions (« connecteur au sens …») pas toujours facile de définir les équivalences

- *Introduction (Définitions)*
- *Concepts de base*
- *Modèles à Base de Composants Existants*
- *Les avantages /Inconvénients de la POC*
- *Conclusion*

4. Les avantages /Inconvénients de la POC (1/2)

A-Les avantages

Spécialisation: L'équipe de développement peut-être divisée en sous-groupes, chacun se spécialisant dans le développement d'un composant.

Sous traitance: Le développement d'un composant peut-être externalisé, à condition d'en avoir bien réalisé les spécifications au préalable.

Facilité de mise à jour: La modification d'un composant ne nécessite pas la recompilation du projet complet.

Facilité de livraison/déploiement: Dans le cas d'une mise à jour, d'un correctif de sécurité, ... alors que le logiciel à déjà été livré au client, la livraison en est facilitée, puisqu'il n'y a pas besoin de re-livrer l'intégralité du projet, mais seulement le composant modifié.

Choix des langages de développement: Il est possible, dans la plupart des cas, de développer les différents composants du logiciel dans des langages de programmation différents. Ainsi, un composant nécessitant une fonctionnalité particulière pourra profiter de la puissance d'un langage dans un domaine particulier, sans que cela n'influe le développement de l'ensemble du projet.

Productivité: La réutilisabilité d'un composant permet un gain de productivité non négligeable car elle diminue le temps de développement, d'autant que le composant est réutilisé souvent.

Bien que la programmation orientée composant soit une évolution de la POO elle commençait à faire à certaines limites face aux applications qui demandaient de plus en plus de ressource

4. Les avantages /Inconvénients de la POC (2/2)

B-Les Inconvénients

• La POC est réellement appréciable dans la conduite d'un projet de développement, cependant elle pose quelques désagréments.

• la POC est une méthode dont le bénéfice se voit surtout sur le long terme. En effet, lorsque l'on parle de réutilisation, de facilité de déploiement, c'est que le développement est sinon achevé, du moins bien entamer. Mais factoriser un logiciel en composants nécessite un important travail d'analyse. La rédaction des signatures des méthodes devra être particulièrement soignée, car modifier une signature nécessitera de retravailler toutes les portions de codes du projet qui font appel au composant, et l'on perdrait alors les bénéfices de l'indépendance des briques logicielles.

• En un mot, si la POC industrialise le développement, la phase de conception du logiciel prendra un rôle encore plus important.

Le fait de ne pas connaître l'implémentation d'un composant (à moins d'avoir accès à la source), peut également gêner certains chefs de projets qui veulent garder un contrôle total sur leur logiciel.

• Un autre problème qui se pose également c'est avec les systèmes distribuées à large échelle il faudra mettre en place des mécanismes qui permettront aux applications de s'adapter à ces environnements.

- *Introduction (Définitions)*
- *Concepts de base*
- *Modèles à Base de Composants Existants*
- *Les avantages /Inconvénients de la POC*
- *Conclusion*

Conclusion et perspectives

Ce paradigme porte sur l'architecture logicielle plutôt que sur le code.
Il s'agit d'externaliser le code fonctionnel d'une application afin de le rendre utilisable dans d'autres applications.
De nos jours, une même application peut par exemple être déployée en client léger, en intranet et en extranet mobile, la notion de composant devient alors fondamentale car seule l'interface de présentation change, la logique métier reste identique.
Réciproquement, un ensemble de composants graphiques développés pour une application peuvent être réutilisables dans d'autres applications n'ayant pas la même logique métier.
Un composant n'est rien d'autre qu'un code compilé – versionné directement réutilisable.
Les avantages sont nombreux en terme de livraison / déploiement / mise à jour.
Cette notion doit donc être prise en compte dès la conception d'une application car un code trop imbriqué (tout dépend de tout) ne peut plus être externalisé par la suite.
Le paradigme événementiel permet entre autre d'isoler des composants, ils n'ont en effet pas besoin de connaître les modules dépendants pour les contacter, ils diffusent simplement un événement.

Références Bibliographiques

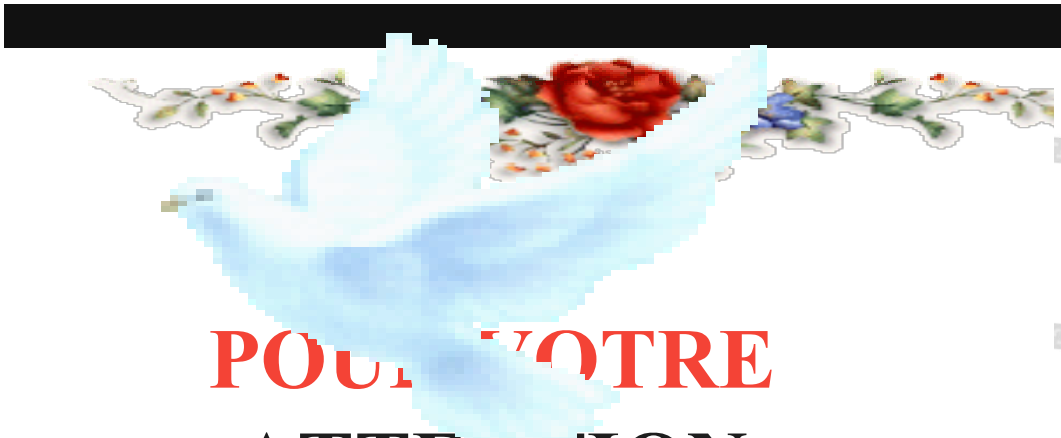
[Maiga;2010]: Abdou Maiga,2010, Composition et architectures par composants, Département de génie informatique et de génie logiciel École Polytechnique de Montréal

[Bentahar;2022]: Atef bentahar,2022,cours:Middlewares orientés composant, Université Batna 2

[Loucif ;2022]: Loucif Hemza,2022, Cours 2éme année Master Informatique, Programmation Orienté Composants (POC), Département : Informatique, Université de M'sila,

[Mballo,2009]: Marie Héléne Mballo, Diplôme d'étude approfondie 2009, Mécanisme multicritère de découverte de services dans les grilles de calcul, Université Cheikh Anta Diop de Dakar

- **Cours :** Licence Informatique 3éme Année, Université de Picardie jules verne https://home.mis.u-picardie.fr/~furst/docs/Java_Beans.pdf



POUR VOTRE ATTENTION

VOS QUESTIONS