Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques

Cours: Approche à base de composants et concurrence

Filière: MR-GL2

Chapitre 1: Introduction à l'approche par composants

Réalisé par:
Dr. Sakka Rouis Taoufik

https://github.com/srtaoufik/Cours-Comp-Concurrence-MR2

1

I. Historique

1) La crise du logiciel :

- Il y a eu une crise de conscience dans les années 70, appelée la crise du logiciel, dû à un tournant décisif.

- C'est à cette époque que :

  - le coût de construction du logiciel est devenu plus important que celui de la construction du matériel.

  - la taille et la complexité des logiciels croissent plus vite que les ressources que l'on est capable de consacrer à leur développement.

2

I. Historique

2) La solution proposée : la réutilisation

- Ne pas réinventer la roue. Se concentrer sur les 20% de code qui représentent la valeur ajoutée de l'application.
- L'approche par composant: un nouveau paradigme de développement «!post-objets!» qui répond à certaines faiblesses de l'approche orientée objet.
- Elle vise la réutilisation par un assemblage aisé et sain des composants.

- Ceci permet l'accroissement de la productivité et de la qualité des logiciels

II. Définition

- Module logiciel autonome pouvant être installé sur différentes plates-formes.
- Une unité de calcul ou de données qui peut être atomique ou composite et qui possède des interfaces décrivant les points d'interaction du composant avec l'environnement.
- Briques configurables pour permettre la construction d'une application par composition
- Composants célèbres: Java Beans, Entreprise Java Beans, composants Corba

III. Avantages

- **Spécialisation :** L'équipe 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

III. Avantages

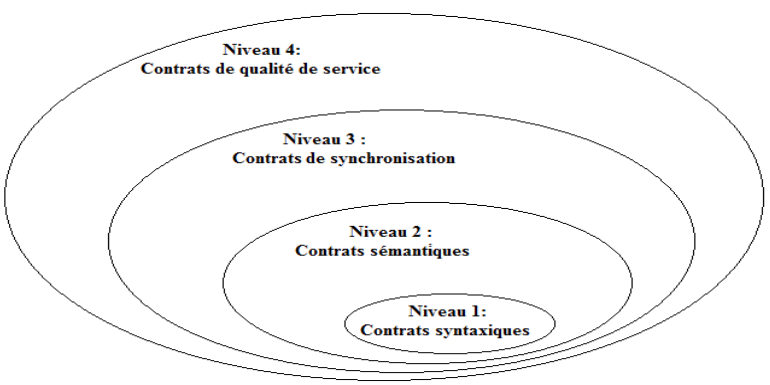
- **Choix des langages de développement :** Il est possible, dans la plupart des cas, de développer les différents composants du logiciel dans des langages de programmation différents.
- **Productivité :** La réutilisabilité d'un composant permet un gain de productivité non négligeable car elle diminue le temps de développement, d'autant plus que le composant est réutilisé souvent

IV. Inconvenants

- Factoriser un logiciel en composants nécessite un travail important dans :
  - **phase d'analyse**
  - **phase de conception architecture**.
- Un assemblage sain de composants exige **la vérification des propriétés (contrats)** liées à la cohérence des interfaces interconnectées.

V. Vérification d'assemblage de composants

➔ A. Beugnard et al. (1999) ont établi une classification des contrats des composants selon quatre niveaux:



V. Vérification d'assemblage de composants

1) Les contrats syntaxiques:

permettent de vérifier la conformité entre les signatures des méthodes des interfaces. La signature d'une méthode peut comporter les éléments suivants :

- nature de méthode : Opération de construction, consultation et modification.
- paramètres formels : Pour chaque paramètre, trois informations à prendre en considération à savoir son type, sa position et sa nature logique (in, out et in/out).

V. Vérification d'assemblage de composants

2) Les contrats sémantiques:

les propriétés qui peuvent être spécifiées avec des pré-conditions, des post-conditions et des invariants.

Ces propriétés sont liées à une méthode, qui est vue comme une unité élémentaire.

Les travaux sur la conception par contrats (Meyer, 1992) se placent à ce niveau.

Une telle propriété sémantique peut être décrite en utilisant un langage de contraintes de type OCL.

V. Vérification d'assemblage de composants

3) Les contrats comportementaux

Il s'agit ici des propriétés concernant les interactions (la synchronisation, la concurrence, la coopération) des éléments architecturaux (composants, ports, connecteurs etc.).

Le comportement global ou partiel des éléments architecturaux peut être décrit par:

- des PSMs (Protocol State Machine cas d'UML2.0)
- des algèbres de processus (CSP pour l'ADL Wright).
- des taches concurrentes (rendez-vous de tasks Ada)

La vérification des contrats comportementaux (de synchronisation) est souvent confiée à des model-checkers.

V. Vérification d'assemblage de composants

4) Les contrats qualitatifs

couvrent les propriétés non-fonctionnelle telles que:

la qualité de service:

- fiabilité,
- disponibilité,
- sécurité,
- etc.

la gestion des ressources,

le temps de réponse.

VI. Architecture à base de composants

La conception architecturale occupe une position clef et critique dans le processus de développement des systèmes à base de composants.

L'architecture logicielle fournit une description de haut niveau de la structure d'un système.

VI. Architecture à base de composants

La présence d'une représentation de l'architecture logicielle du système étudié:

+ favorise le passage de l'étape de conception à l'étape d'implémentation.

+ facilite la localisation des erreurs résiduelles. (lors de l'étape de maintenance corrective)

+ favorise l'extensibilité du logiciel lors de l'étape de maintenance évolutive.

VI. Architecture à base de composants

- La communauté scientifique a développé plusieurs ADLs (Architecture Description Language) permettant la description plus ou moins précise des architecturales à base de composants.
- Les ADL les plus connus: **Wright (Allen, 1997), ACME (Garlan, 2000), Rapide (Luckham, 1995), Unicon (Shaw, 1995), C2 (Taylor, 1996), Darwin (Magee, 1995)** et **AESOP (Garlan, 1994).**

- Généralement, tout ADL doit supporté les 3 concepts de base: **Composant, Connecteur et Configuration**.
- Il peut fournit des concepts supplémentaires: propriétés spécifique, types, contraintes, etc.

VI. Architecture à base de composants

1) Composant

- une unité de calcul ou de stockage des données
- possède un ensemble d'interfaces, appelés **ports**, qui définissent les points d'interaction entre cet élément et son environnement.

La taille d'un composant peut varier d'une application à une autre.

Les modèles de composants hiérarchiques comme Wright, Acme et UML2.0 acceptent le concept de composant composite (un composant contenant à son tour une configuration de composants et de connecteurs).

VI. Architecture à base de composants

2) Connecteur

- modélise les interactions entre les composants et les règles qui régissent ces interactions.

- définit un ensemble d'interfaces, appelés **rôles** qui permettent d'identifier les participants à l'interaction.

Par exemple, un connecteur «Pipe» possède un rôle d'écriture et un rôle de lecture.

VI. Architecture à base de composants

3) Configuration

- décrit la structure complète d'un système sous forme d'un graphe connexe regroupant des composants et des connecteurs.

- Le graphe est obtenu en associant les ports des composants avec les rôles des connecteurs adéquats, en vue de construire l'application.

Par exemple, les ports des composants de type "Filtre" sont associés aux rôles des connecteurs de type "Pipe" à travers lesquels ils lisent et écrivent des flux de données.

VI. Architecture à base de composants

4) Concepts supplémentaires

Propriétés: La capacité de spécifier une propriété d'un élément architectural. Par exemple, L'ADL Acme supporte le concept Property qui permet la spécification d'une propriété sous la forme d'un triplet (nom, type et valeur).

Types: Le typage permet l'encapsulation des fonctionnalités et des éléments internes en vu d'être réutilisables.

Contraintes: expriment des restrictions placées sur les composants, les connecteurs et les configurations. Une contrainte sur un composant peut par exemple servir à borner les valeurs de ses propriétés.

19