



Introduction aux Composants Logiciels

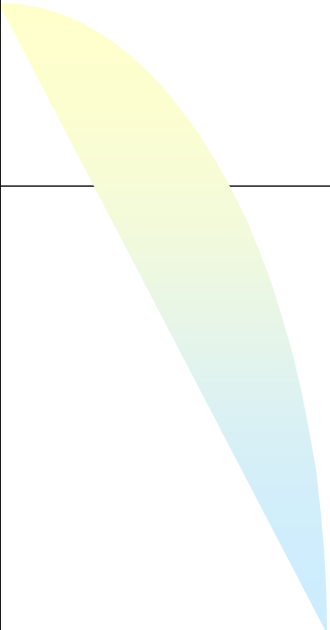
Christian Pérez

LIP/INRIA

Année 2010-11

Plan

- Introduction aux composants logiciels
 - Pourquoi des composants logiciels
 - Notions de composants logiciels
 - Conclusion
- Survol du modèle de composant de CORBA
- Survol de Fractal



Motivations: pourquoi des composants logiciels



Motivations

- Critères économiques :
Développer des logiciels est une tâche chère et complexe
 - Augmenter la productivité des développeurs
 - Concurrence internationale/ mondialisation
 - Réduire la complexité des logiciels
 - Diminuer les coûts de développements
 - Diminuer les coûts de maintenance
 - Réduire le temps de mise sur le marché
 - Réutiliser plutôt que redévelopper
 - Capitaliser le savoir faire d'une entreprise
 - Réutiliser des codes réalisés dans l'entreprise
 - En bref : Passer du stade artisanal au stade industriel

Limites du développement sur mesure

- **Avantage**
 - Très spécifique, très bien adapté quand ça marche
- **Inconvénients**
 - Très coûteux
 - Temps de développement considérable
 - Solution sous-optimale par manque de spécialistes
 - Il faut tout maîtriser : la partie « métier » et les parties « techniques »
 - Manque d'interopérabilité
 - Nécessite de redévelopper
 - Même des parties de programmes déjà écrites dans des contextes différents
- **Exemples**
 - Logiciels pour les administrations ou les grands comptes

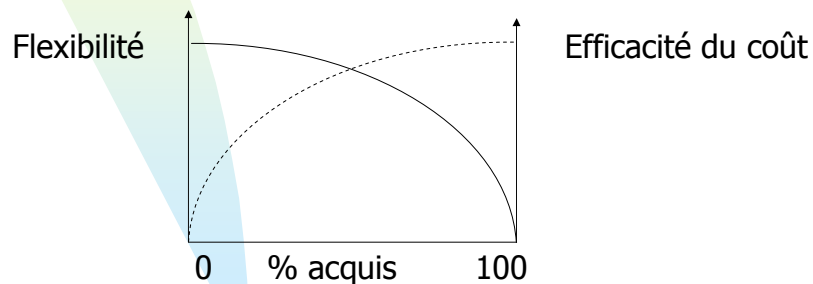
Limite des logiciels standard

- **Avantages**
 - Temps de mise sur le marché plus court
 - Juste à configurer
 - Produit acquis
 - maintenance, évolution à la charge du « vendeur »
- **Inconvénients**
 - Réorganisation locale du « business » (adaptation)
 - Les concurrents ont aussi accès aux fonctionnalités
 - Adaptabilité aux besoins limitée
- **Exemple**
 - Excel

Entre les deux:

□ Principe:

- Acquérir ce qui est disponible et développer ce qui est spécifique



Le chaînon manquant ?

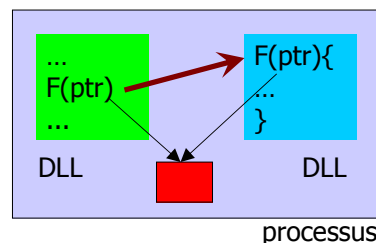
- La notion de composant est très largement répandue...
 - Composants électroniques
 - Composants d'une voiture (depuis Henry FORD)
 - Composants d'un meuble, d'une construction
 - On fait même de la cuisine « d'assemblage »
- Des composants partout ... sauf en informatique !
 - On reste ainsi au stade artisanal ...
- Et pourtant ...c'est un ancien rêve de l'informatique !
 - Les premiers travaux datent des années 60

Notion fondamentale: la composition

- Les composants sont une vision qui met en avant l'opération de composition
 - Assembler des composants pour réaliser un logiciel plus complexe
- Rôle principal d'un composant : être composé !
 - Son fonctionnement interne importe peu
- Conséquence: Les composants prônent
 - la préfabrication (indépendante d'une application)
 - la réutilisation (dans plusieurs applications)

Une première « définition »

- *Un composant est une unité binaire indépendante de production, d'acquisition et de déploiement qui interagit via un canevas (framework).*
- Exemple
 - « Composants Logiciels »
 - fonctions dans une DLL
 - Canevas
 - un processus
 - Supports d'interaction
 - registre/mémoire
- Contre exemple: les macros en C



Quelle place pour les composants ?

- **Marché**
 - Masse critique à atteindre
 - Qualité des composants
- **Supériorité face à d'autres solutions**
 - Facilité de la composition
 - Unité de déploiement (« d'installation »)
 - Analogie: composants ~ lego !
- **On n'a jamais vu d'objet vendu/acheté/déployé !**

Les succès du passé

- **Les systèmes d'exploitation**
 - « Composants logiciels » :
 - Les applications
 - Supports d'interaction
 - le système de fichier (format de fichier commun), les pipes
 - OLE, COM
- **Les applications à plug-in**
 - Photoshop/Gimp
 - IE/Mozilla, ...

Analyse de ces succès

- Mise à disposition de canevas
 - Disposant de fonctionnalités importantes
 - Différents « composants » peuvent
 - s'exécuter en même temps
 - provenir de différents vendeurs
 - Abstraction pour le client
 - juste des propriétés à configurer sur les « composants »

Composants vs standards

- Les composants ont besoin de standards
 - C'est la base de l'interopérabilité
- Standard ~ accord sur un modèle commun

- En général, créé par les besoins
 - Innovation -> produit -> standard -> marché ↗
 - Exemple: Microsoft, SUN
 - Contre exemple: OMG: standard -> produit !

Un standard ou des standards ?

- Un seul standard ~ un seul modèle de connexion
 - facile en théorie ☺
 - Mais est-il libre ? Accepté de tous ?
- Plusieurs en pratique ☹
 - Exemple: courant et prises électriques, télévision, tel portables
- Solution: adaptateur = passerelle entre standards
 - Exemple:
 - Adaptateur 110V/220V
 - passerelle CORBA/COM définie par l'OMG
 - Difficile si les expressivités sont différentes

En résumé

- Besoin économique
- Composant = outil pour faciliter
 - La réutilisation
 - La composition
 - Le déploiement
- Ce n'est pas une idée neuve
 - Mais c'est quoi alors ?

...



Notion de composants logiciels

Composant \neq objet !

- Attention: les composants et les objets sont différents (voir plus loin) **MAIS** ils utilisent un même vocabulaire avec des sens différents !
- Exemple: interface
 - Objet: Interface \sim Classe totalement abstraite
 - Composant: Une spécification d'un point d'accès

Définition par les propriétés

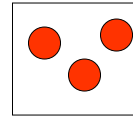


- Un composant est défini par ses propriétés:
 - 1. *c'est une unité de déploiement indépendant*
 - 2. *c'est une unité de composition par une tierce entité*
 - 3. *un composant n'a pas d'état persistant*
- Remarques
 - dépl. ind. ⇒ bien séparé par rapport à l'environnement
 - unité de dépl. ⇒ pas de déploiement partiel
 - tierce entité ⇒ non accès aux détails
 - pas d'état ⇒ notion de copie non défini
 - Soit le composant est disponible soit il ne l'est pas

Et les objets ?

- Propriétés d'un objet
 - 1. *Un objet est une unité d'instanciation; il a un identifiant unique*
 - 2. *Un objet a un état; cet état peut être persistant*
 - 3. *Un objet encapsule son état et son comportement*
- Remarques
 - Unité d'instanciation ⇒ ne peut être partiellement instancié
 - État change, pas son identifiant
 - Plan pour construire un objet: en général, une classe

Relations entre composants et objets



- Un composant peut être constitué d'un ensemble d'objets
- Un composant peut ne contenir aucun objet
- Un objet ne peut pas contenir de composant

- Composants ~ interfaces, réutilisabilité, comportement à l'exécution
- Approche objet ~ langage de programmation
- Composant et objets se complètent car ils abordent des questions différentes

Couleurs de boîtes

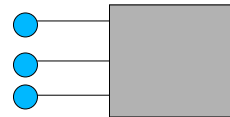


- Boîte blanche : tout est visible
 - Analogie en C++ : structure
- Boîte noire : rien n'est visible à part l'interface
 - Analogie en C++ : classe sans variable publique
- Quelle est la bonne couleur pour les composants ?
 - En pratique: boîte grise
 - On aimerait que rien ne soit visible mais pour diverses raisons (performance, ...) ce n'est pas le cas.

Définition d'un composant logiciel

- *Un composant logiciel est une **unité de composition** qui a, par **contrat**, spécifié uniquement ses **interfaces** et ses **dépendances** explicites de contextes. Un composant logiciel peut être **déployé** indépendamment et est sujet à **composition** par des tierces entités.*
- Définition de Szyperski et Pfister, 1997.

Interfaces

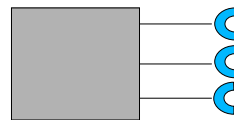


- Un composant peut contenir plusieurs interfaces
 - Exemple: distributeur de boissons
- Interface ~ point d'accès
 - Nom d'un point d'accès
 - Définit description et protocole du point d'accès
 - N'offre pas d'implémentation
- Conséquence: composant peut être changé
- Interface décrite dans un langage: par ex IDL
 - Seulement propriétés fonctionnelles
 - Interface importée/exportée vers/de l'environnement

Contrats

- Explicite le comportement d'une interface
- Exemple:
 - Contrainte de fonctionnement
 - Pré-condition
 - Post-condition
- Problème ouvert:
 - Comment spécifier les propriétés non fonctionnelles
 - Sécurité, transaction, performance (temps réel), ...

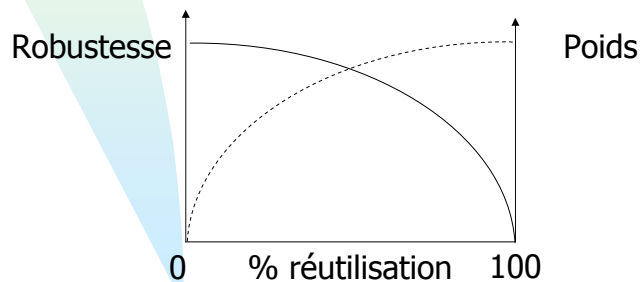
Dépendances de contextes explicites



- Contextes inclus les contextes de composition et de déploiement
- Objectif: vérifier si un composant est déployable
- Problème: plusieurs mondes (COM, CORBA ,...) qui contiennent plusieurs environnement (OS, ...)
 - Problème difficile à résoudre

Poids d'un composant logiciel

- Compromis entre
 - Interfaces importantes -> réutilisation et dépendance ↗ mais poids ↗
 - Interfaces petites -> réutilisation et dépendances ↘
- Remarque: maximiser la réutilisation minimise l'utilisation



Standard et norme

- La question de la taille d'un composant peut être simplifiée par des normes
 - Objectif: Standardiser par domaine afin de simplifier dépendances
 - Délégation d'une partie des fonctionnalités
 - Exemple: STL en C++, POSIX Thread
- Mais on retrouve le même problème si trop de standards !

Comment partitionner une application ?

- En général, la partition dépend de 2 considérations:
 - les besoins
 - le catalogue des composants disponibles
- Même si le catalogue est petit, l'approche composant force
 - une architecture modulaire
 - une gestion explicite des dépendances
- Question : taille d'un composant ?
- Unité traditionnelle : bibliothèque, classe, module
 - Pas de solution générale

Conclusi
on

Conclusion

- ❑ Les composants logiciels abordent des problèmes non pris en compte jusqu'à présent
- ❑ Ils présentent une vision particulière de comment développer des logiciels
- ❑ Ils ne disent pas comment programmer : l'approche orientée objet est complémentaire
- ❑ Analogie avec l'approche orienté objet et la programmation structurée

Bibliographie

- ❑ *Software Component : Beyond object oriented programming*, par C. Szyperski, Addison-Wesley, Reading, MA, 1998
- ❑ *Design patterns, element of Reusable Object-Oriented Software*, par E. Gamma, R. Helm, R. Johnson et J. Vlissides, Addison-Wesley, Reading, MA, 1995
- ❑ *Developing and Integrating Enterprise Components and Services*, série d'articles dans «Communication of the ACM », octobre 2002, volume 45:10