Architecture logicielle : quelques éléments

Licence professionnelle IDSE
2012-2013
http://anubis.polytech.unice.fr/iut/2012_2013/lp/idse/gl/management

Mireille Blay-Fornarino
blay@unice.fr

Architecture logicielle

- L'architecture informatique définit la structuration d'un système informatique (i.e. matériel et logiciel) en termes de composants et d'organisation de ses fonctions.

François Trudel, ing., M.Sc.A.
Président Fondateur
francois.trudel@agilexx.com
université de Sherbrook

L'ARCHITECTURE LOGICIELLE EN PRATIQUE

Qu'est-ce qu'une architecture logicielle ?

- Contrairement aux spécifications produites par l'analyse fonctionnelle
  - le modèle d'architecture ne décrit pas ce que doit réaliser un système informatique mais plutôt comment il doit être conçu de manière à répondre aux spécifications.
  - L'analyse fonctionnelle décrit le « quoi faire » alors que l'architecture décrit le « comment le faire »

Description d'une architecture logicielle ?

La définition de l'architecture logicielle consiste à:

- Décrire l'organisation générale d'un système et sa décomposition en sous-systèmes ou composants
- Déterminer les interfaces entre les sous-systèmes
- Décrire les interactions et le flot de contrôle entre les sous-systèmes
- Décrire également les composants utilisés pour implanter les fonctionnalités des sous-systèmes
  - Les propriétés de ces composants
  - Leur contenu (e.g., classes, autres composants)
  - Les machines ou dispositifs matériels sur lesquels ces modules seront déployés

Pourquoi une architecture logicielle [Garlan 2000]

- **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une vue de haut-niveau de leur structure et de leurs contraintes. Les motivation des choix de conception sont ainsi mis en évidence
- **Réutilisation** : favorise l'identification des éléments réutilisables, parties de conception, composants, caractéristiques, fonctions ou données communes.
- **Construction** : fournit un plan de haut-niveau du développement et de l'intégration des modules en mettant en évidence les composants, les interactions et les dépendances. Elle doit permettre aux développeurs de travailler sur des parties individuelles du système en isolation
- **Évolution** : met en évidence les points où un système peut être modifié et étendu. La séparation composant/connecteur facilite une implémentation du type « plug-and-play»
- **Analyse** : offre une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la cohérence, test de conformité, analyse des dépendances
- **Gestion** : contribue à la gestion générale du projet en permettant aux différentes personnes impliquées de voir comment les différents morceaux du casse-tête seront agencés. L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Pourquoi une architecture logicielle [Garlan 2000]

- **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une **vue de haut-niveau** de leur structure et de leurs contraintes. Les **motivation des choix de conception** sont ainsi mis en évidence
- **Réutilisation** : favorise l'identification des éléments réutilisables, parties de conception, composants, caractéristiques, fonctions ou données communes.
- **Construction** : fournit un plan de haut-niveau du développement et de l'intégration des modules en mettant en évidence les composants, les interactions et les dépendances. Elle doit permettre aux développeurs de travailler sur des parties individuelles du système en isolation
- **Évolution** : met en évidence les points où un système peut être modifié et étendu. La séparation composant/connecteur facilite une implémentation du type « plug-and-play»
- **Analyse** : offre une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la cohérence, test de conformité, analyse des dépendances
- **Gestion** : contribue à la gestion générale du projet en permettant aux différentes personnes impliquées de voir comment les différents morceaux du casse-tête seront agencés. L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Pourquoi une architecture logicielle [Garlan 2000]

- **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une vue de haut-niveau de leur structure et de leurs contraintes. Les motivation des choix de conception sont ainsi mis en évidence
- **Réutilisation** : favorise **l'identification des éléments réutilisables**, parties de conception, composants, caractéristiques, fonctions ou données communes.
- **Construction** : fournit un plan de haut-niveau du développement et de l'intégration des modules en mettant en évidence les composants, les interactions et les dépendances. Elle doit permettre aux développeurs de travailler sur des parties individuelles du système en isolation
- **Évolution** : met en évidence les points où un système peut être modifié et étendu. La séparation composant/connecteur facilite une implémentation du type « plug-and-play»
- **Analyse** : offre une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la cohérence, test de conformité, analyse des dépendances
- **Gestion** : contribue à la gestion générale du projet en permettant aux différentes personnes impliquées de voir comment les différents morceaux du casse-tête seront agencés. L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Pourquoi une architecture logicielle [Garlan 2000]

- **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une vue de haut-niveau de leur structure et de leurs contraintes. Les motivation des choix de conception sont ainsi mis en évidence
- **Réutilisation** : favorise l'identification des éléments réutilisables, parties de conception, composants, caractéristiques, fonctions ou données communes.
- **Construction** : fournit un **plan de haut-niveau** du développement et de l'intégration des modules en mettant en évidence les composants, les *interactions et les dépendances*. Elle doit permettre aux développeurs de **travailler sur des parties individuelles du système en isolation**
- **Évolution** : met en évidence les points où un système peut être modifié et étendu. La séparation composant/connecteur facilite une implémentation du type « plug-and-play»
- **Analyse** : offre une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la cohérence, test de conformité, analyse des dépendances
- **Gestion** : contribue à la gestion générale du projet en permettant aux différentes personnes impliquées de voir comment les différents morceaux du casse-tête seront agencés. L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Pourquoi une architecture logicielle [Garlan 2000]

- **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une vue de haut-niveau de leur structure et de leurs contraintes. Les motivation des choix de conception sont ainsi mis en évidence
- **Réutilisation** : favorise l'identification des éléments réutilisables, parties de conception, composants, caractéristiques, fonctions ou données communes.
- **Construction** : fournit un plan de haut-niveau du développement et de l'intégration des modules en mettant en évidence les composants, les interactions et les dépendances. Elle doit permettre aux développeurs de travailler sur des parties individuelles du système en isolation
- **Évolution** : met en évidence les points où un système peut être modifié et étendu. La **séparation composant/connecteur** facilite une implémentation du type « plug-and-play»
- **Analyse** : offre une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la cohérence, test de conformité, analyse des dépendances
- **Gestion** : contribue à la gestion générale du projet en permettant aux différentes personnes impliquées de voir comment les différents morceaux du casse-tête seront agencés. L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Pourquoi une architecture logicielle [Garlan 2000]

- **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une vue de haut-niveau de leur structure et de leurs contraintes. Les motivation des choix de conception sont ainsi mis en évidence
- **Réutilisation** : favorise l'identification des éléments réutilisables, parties de conception, composants, caractéristiques, fonctions ou données communes.
- **Construction** : fournit un plan de haut-niveau du développement et de l'intégration des modules en mettant en évidence les composants, les interactions et les dépendances. Elle doit permettre aux développeurs de travailler sur des parties individuelles du système en isolation
- **Évolution** : met en évidence les points où un système peut être modifié et étendu. La séparation composant/connecteur facilite une implémentation du type « plug-and-play»
- **Analyse** : offre une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la **cohérence**, **test de conformité**, analyse des **dépendances**
- **Gestion** : contribue à la gestion générale du projet en permettant aux différentes personnes impliquées de voir comment les différents morceaux du casse-tête seront agencés. L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Pourquoi une architecture logicielle [Garlan 2000]

- **Compréhension** : facilite la compréhension des grands systèmes complexes en donnant une vue de haut-niveau de leur structure et de leurs contraintes. Les motivation des choix de conception sont ainsi mis en évidence
- **Réutilisation** : favorise l'identification des éléments réutilisables, parties de conception, composants, caractéristiques, fonctions ou données communes.
- **Construction** : fournit un plan de haut-niveau du développement et de l'intégration des modules en mettant en évidence les composants, les interactions et les dépendances. Elle doit permettre aux développeurs de travailler sur des parties individuelles du système en isolation
- **Évolution** : met en évidence les points où un système peut être modifié et étendu. La séparation composant/connecteur facilite une implémentation du type « plug-and-play»
- **Analyse** : offre une base pour l'analyse plus approfondie de la conception du logiciel, analyse de la cohérence, test de conformité, analyse des dépendances
- **Gestion** : contribue à la gestion générale du projet en permettant aux différentes personnes impliquées de voir comment les différents morceaux du casse-tête seront agencés. L'identification des dépendance entre composants permet d'identifier où les délais peuvent survenir et leur impact sur la planification générale

Critères de Qualité Logicielle

- Interopérabilité
- Portabilité
- Compatibilité
- Validité
- Vérifiabilité
- Intégrité
- Fiabilité
- Maintenabilité
- Réutilisabilité
- Extensibilité
- Efficacité
- Autonomie
- Transparence
- Composabilité
- Simplicité

François Trudel, ing., M.Sc.A.
Président Fondateur
francois.trudel@sigmoc.com
université de Sherbrooke

L'ARCHITECTURE LOGICIELLE EN PRATIQUE

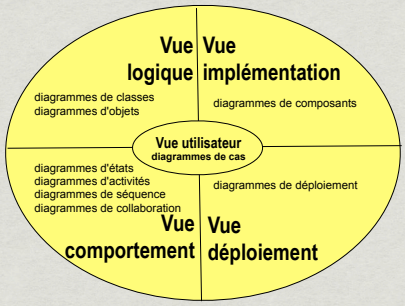
Rôle d'un Architecte Logiciel

- Le rôle de l'architecte logiciel est:
  - 1) de **définir** une architecture logicielle qui satisfasse les **contraintes** du système
  - 2) de la **communiquer**.

François Trudel, ing., M.Sc.A.
Président Fondateur
francois.trudel@bginex.com
université de Sherbrook

L'ARCHITECTURE LOGICIELLE EN PRATIQUE

Modèle d'architecture

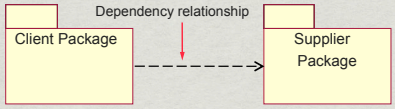


Modéliser avec UML

- Les vues (structurelles) d'une architecture logicielle
  - **Vue logique.** Description logique du système décomposé en sous-systèmes (modules + interface)
    - UML : diagramme de paquetages
  - **Vue d'implémentation.** Description de l'implémentation (physique) du système logiciel en termes de composants et de connecteurs
    - UML : diagramme de composants
  - **Vue de déploiement.** Description de l'intégration et de la distribution de la partie logicielle sur la partie matérielle
    - UML: diagramme combiné de composants et de déploiement

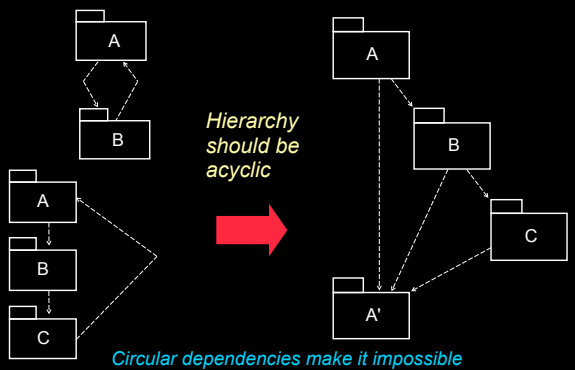
La vue logique

- Aspects statiques et dynamiques
- Les éléments
  - Les objets, Les classes
  - Les collaborations, Les interactions
  - Les paquetages : organisation des éléments en groupes logiques



Il faut essayer de maximiser la cohésion au sein des paquetages (éléments liés) et minimiser le couplage entre eux

Avoiding Circular Dependencies



Circular dependencies make it impossible to reuse one package without the other.

Rational the software development company

La vue de réalisation

- Permet de visualiser les modules dans l'environnement de développement.
- Un bon support, le diagramme de composants d'UML.

Diagramme de composants UML 2.0

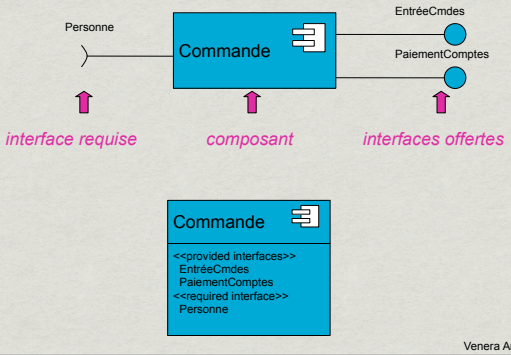
- Offre une vue de haut niveau de l'architecture du système
- Utilisé pour décrire le système d'un point de vue implémentation
- Permet de décrire les composants d'un système et les interactions entre ceux-ci
- Illustre comment grouper concrètement et physiquement les éléments (objets, interfaces, etc.) du système au sein de modules qu'on appelle composants

Modèle à composants

- Unité modulaire avec des interfaces bien définies qui est remplaçable dans son environnement
- Unité autonome au sein d'un système
  - A une ou plusieurs interfaces fournies et requises
  - Sa partie interne est cachée et inaccessible
  - Ses dépendances sont conçues de telle sorte que le composant peut être traitée de façon aussi autonome que possible

Foutse Khomh

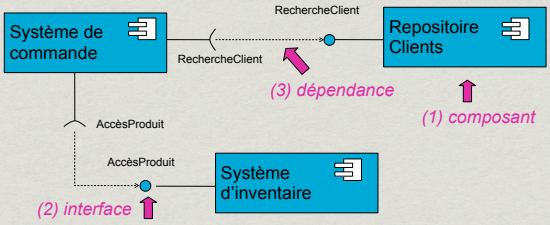
Modèle à composants : notation



Venera Arnaoudova

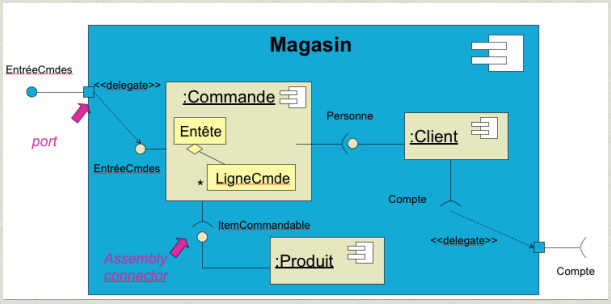
Modèle à composants : notation

- Composants et relations – notation
  - Une flèche de dépendance permet de mettre en relation des composant via les interfaces requises et fournies



Venera Arnaoudova

Modèle à composants : vue interne

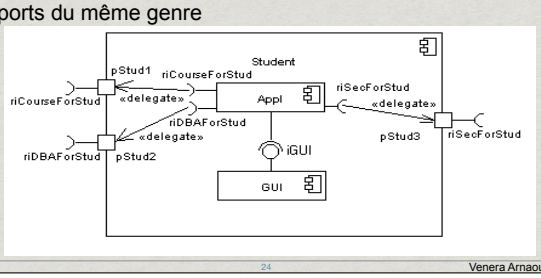


Venera Arnaoudova

Modèle à composants : notation

Délégation

- Rattacher le contrat externe d'un composant interne à la réalisation
- Représente la transmission des signaux
- Il ne doit être défini qu'entre les interfaces requises ou des ports du même genre



Venera Arnaoudova

La vue de déploiement

- Un diagramme de déploiement représente la façon dont déployer les différentes éléments d'un système
  - Les ressources matérielles et l'implantation du logiciel dans ces ressources
  - Les éléments
    - Les noeuds
    - Les modules
    - Les programmes principaux

Exemple de diagramme de déploiement

- Un diagramme de déploiement propose une vision statique de la topologie du matériel sur lequel s'exécute le système
- Un diagramme de déploiement montre les associations (connexions) existant entre les noeuds du système
- Un diagramme de déploiement ne montre pas les interactions entre les noeuds

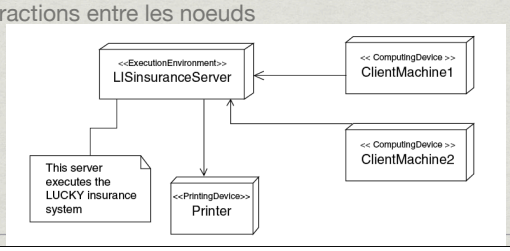
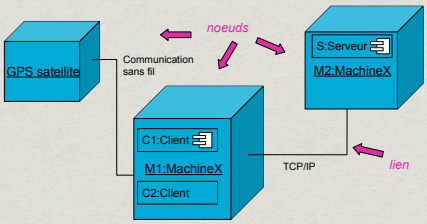


Diagramme de déploiement et communication



Connexion entre noeuds

- Une connexion est une connexion physique reliant deux noeuds entre-eux.
- Elle indique en général la méthode utulisée : ex TCP/IP

- Exemples de connexion :
  - une connexion Ethernet,
  - une ligne série,
  - un bus partagé,
  - …

Artefact

- Un artefact est la spécification d'un élément physique qui est utilisé ou produit par le processus de développement du logiciel ou par le déploiement du système.
  - élément concret : fichier, exécutable ou table d'une base de données....
- Un artefact peut être relié à d'autres artefacts par notamment des liens de dépendance.

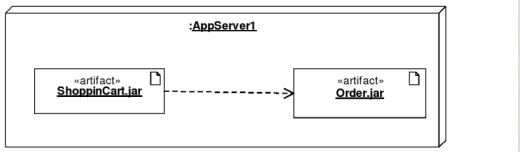
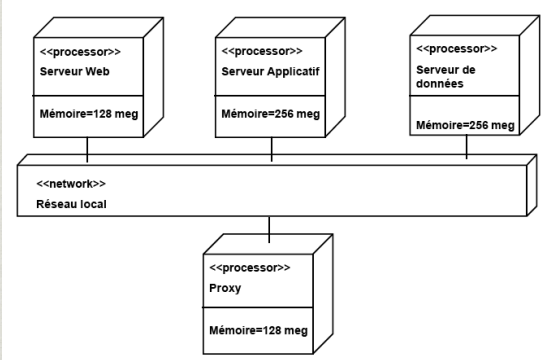


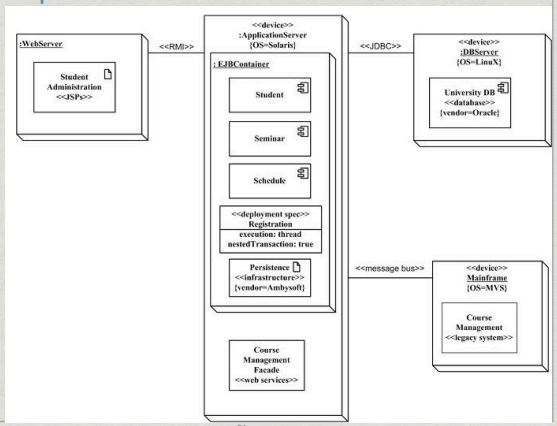
Figure 8.9: Représentation du déploiement de deux artefacts dans un nœud. La dépendance entre les deux artefacts est également représentée.

Exemple de déploiement et contraintes



Exemple

http://www.agilemodeling.com/artifacts/deploymentDiagram.htm



Définir un diagramme de déploiement

1. Identifier le cadre : une application ou l'ensemble du système?
2. Prenez en compte les caractéristiques techniques essentielles.
   - Quels systèmes pré-existants doivent être intégrés? Par interaction?
   - Quelle qualité est attendue ? Mise en place de redondance?
   - Qui/quoi doit interagir avec le système? Par quels moyens? (Internet, exchanging data files, ...)? Comment le système sera monitoré? Quelle sécurité? (firewall, sécurité hardware,...)
3. Identifier le style de l'architecture de distribution.
4. Identifier les noeuds et leurs connexions.
   - OS? Connexions (RMI, SOAP, ...).
5. Distribuer le logiciel sur les noeuds.

Exemples de propriétés à prendre en compte

- Persistance
  - granularité
  - volume
  - durée
  - mécanisme d'accès
  - fréquence d'accès (création / suppression, mise à jour, lire)
- Fiabilité
  - Mécanisme de communication inter-processus
  - latence
  - Synchronicité
  - taille des messages
  - Protocole

Exemples de propriétés à prendre en compte

- Interfaces avec d'autres systèmes
  - latence
  - durée
  - mécanisme d'accès
  - fréquence d'accès
- Sécurité
  - granularité des données
  - granularité des utilisateurs
  - règles de sécurité
  - privilèges

Développer un modèle architectural

- Commencer par faire une esquisse de l'architecture
  - En se basant sur les principaux requis des cas d'utilisation ; décomposition en sous-systèmes
  - Déterminer les principaux composants requis
  - Sélectionner un style architectural
- Raffiner l'architecture
  - Identifier les principales interactions entre les composants et les interfaces requises
  - Décider comment chaque donnée et chaque fonctionnalité sera distribuée parmi les différents composants
- Considérer chacun des cas d'utilisation et ajuster l'architecture pour qu'il soit réalisable
- Détailler l'architecture et la faire évoluer

Développer un modèle architectural

- Décrire l'architecture avec UML
  - Tous les diagrammes UML peuvent être utiles pour décrire les différents aspects du modèle architectural
  - Trois des diagrammes UML sont particulièrement utile pour décrire une architecture logicielle
    - Diagramme de packages
    - Diagramme de composants
    - Diagramme de déploiement

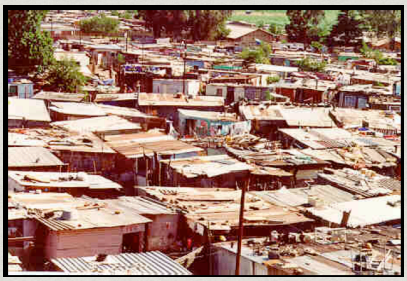
Styles d'Architecture Logicielle

- L'architecture logicielle, tout comme l'architecture traditionnelle, peut se catégoriser en styles.
- Un système informatique pourra utiliser plusieurs styles selon le niveau de granularité ou l'aspect du système souhaité.

Quelques styles d'Architecture Logicielle

- **Centré sur les Données**
  - Base de données
  - Blackboard
- **Flots de Données**
  - Par lots
  - Tuyaux et Filtres
- **Hiérarchique**
  - En couches
- **Invocation implicite**
  - Orientée Événements
  - Model-View-Controller

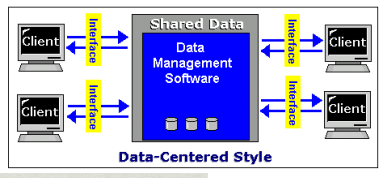
Le style le plus répandu

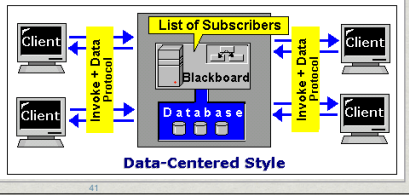


Architecture centrée données

- Entrepôt de données centralisée qui communique avec un certain nombre de clients.
- **Objectif :** maintenir l'intégrité des données
- Utilisée dans le cas où des données sont partagées et fréquemment échangées entre les composants
- On distingue deux sous-types: référentiel et tableau noir
  - Référentiel: un client envoie une requête au système en demandant d'exécuter une action nécessaire (par exemple des données d'insertion)
  - Blackboard: le système informe les abonnés intéressés par les changements. Une Architecture de style Blackboard est similaire à l'observateur, modèle de conception (Gamma et al., 1995).

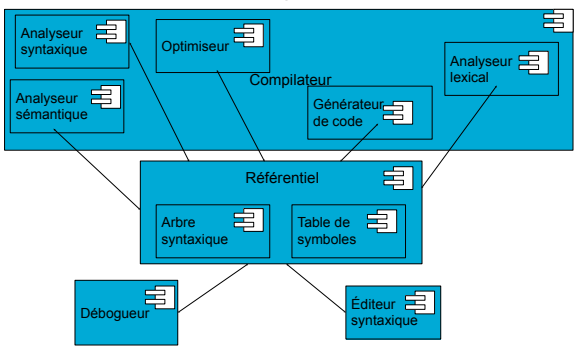
Repository vs Blackboard





Architecture avec référentiel

- Environnement de programmation



Centrée données : avantages et difficultés

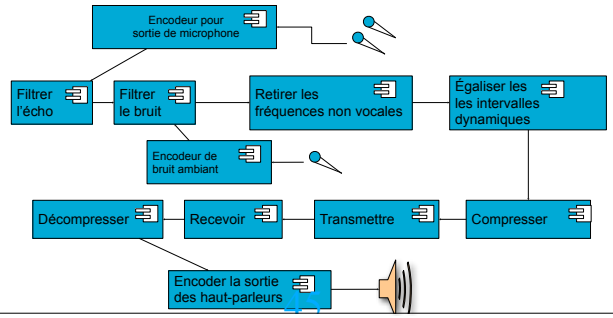
- Indépendances des clients les uns des autres
  - on peut ajouter ou retirer des clients
- ➡Mais attention aux optimisations qui créent un couplage fort.
- Point à aborder:
  - La cohérence des données - synchronisation des lectures / écritures
  - La sécurité des données, le contrôle d'accès
  - Point de défaillance unique
  - Passage à l'échelle (réplication vs complexité)

Architecture flots de données

- Succession de transformations des données d'entrée.
- **Objectifs** : réutilisation et évolutivité.
- 2 types : séquentiel ou pipeline
  - **style séquentiel** : chaque étape s'exécute jusqu'à la fin avant la prochaine étape commence
    - Par exemple Tubes UNIX en ligne de commande
  - **style pipeline :** certaines étapes peuvent fonctionner simultanément

Architecture flot de données

- Système de traitement du son



Flots de données : avantages et inconvénients

✓Faible complexité des interactions entre les composants : traitement en boîtes noires.

- Pas pour des applications interactives.
- performance et efficacité : gestion de buffers affectant l'efficacité de la mémoire.

➡ Base des workflows scientifiques utilisés sur les grilles de calcul.

Architecture multi-couches

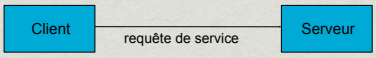
- Organisation hiérarchique du système en un ensemble de couches
- Des interfaces bien définies entre les couches
- Chaque couche agit comme un
  - Serveur : Fournisseur de services de couches "supérieures":
  - Client: consommateur de services de couche (s) »ci-dessous"
- Les connecteurs sont des protocoles de la couche d'interaction
- **Objectifs :**
  - Réduire la complexité,
  - Améliorer la modularité, réutilisabilité, maintenabilité
- Différents critères de stratification: notamment abstraction

Style Client-Server Style

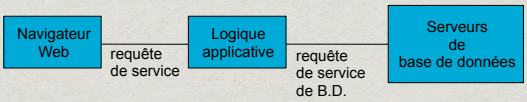
- Les composants sont les clients et les servers
- Les serveurs ne connaissent pas le numéro ou l'identité des clients
- Les clients connaissent l'identité du serveur
- Les connecteurs sont basés sur les protocoles basés sur RPC

Architecture n-niveaux

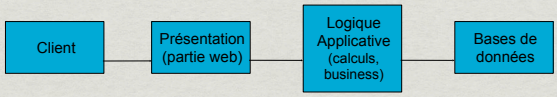
- Architecture 2-niveaux (client-serveur ou client lourd)



- Architecture 3-niveaux (client léger)

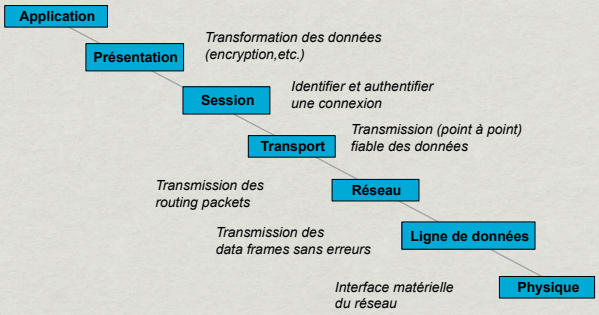


- Architecture 4-niveaux



Architecture multi-couches

- Système fermé
  - Reference Model of Open Systems Interconnection (OSI model)



Architecture multi-couches

- ✓ Conception et testabilité séparée des couches
- ✓ Cohésion des couches
- ✓ Faible couplage et abstraction :
  - ✓ les couches inférieures ne devraient rien savoir des couches supérieures
  - ✓ connexions autorisées entre couches uniquement via les API
- ✓ Réutilisabilité des couches inférieures : des solutions génériques réutilisables
- ✓ Flexibilité : ajout de nouveaux services construits sur les service; une modification affecte au plus les couches adjacentes.
- \- Performances
- \- Pas toujours applicables
- \- Saut entre couches

Invocation implicite

- Levée d'un événement au lieu de l'invocation explicite de la méthode
  - "Auditeurs" s'inscrivent aux évènements qui les intéressent et les méthodes associées
  - "Les annonceurs" ne sont pas conscients des effets des événements produits : aucune hypothèse sur le traitement en réponse à des événements
- Deux types de connecteurs
  - L'invocation est soit explicite ou implicite, en réponse à des événements

Invocation implicite

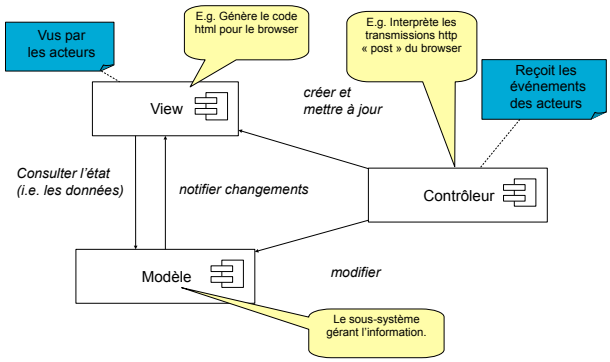
- ✓ la réutilisation des composants
- ✓ Evolution du système
  - ✓ Tant au système de construction en temps & run-time
- \- Structure du système non-intuitive :
  - \- contrôle des calculs donné aux Système
  - \- Quelles réactions à un événement ?
  - \- Dans quel ordre, les traitements?

➡ Evolution vers les bus à messages et le complex event processing

Architecture Modèle-Vue-Contrôleur (MVC)

- Séparer la couche interface utilisateur des autres parties du système (car les interfaces utilisateurs sont beaucoup plus susceptibles de changer que la base de connaissances du système)
- Composé de trois types de composants
  - Modèle : rassemble des données du domaine, des connaissances du système. Contient les classes dont les instances doivent être vues et manipulées
  - Vue : utilisé pour présenter/afficher les données du modèle dans l'interface utilisateur
  - Contrôleur : contient les fonctionnalités nécessaires pour gérer et contrôler les interactions de l'utilisateur avec la vue et le modèle

Architecture Modèle-Vue-Contrôleur



Architecture Modèle-Vue-Contrôleur

- Modèle : noyau de l'application
  - Enregistre les vues et les contrôleurs qui en dépendent
  - Notifie les composants dépendants des modifications aux données
- Vue : interface (graphique) de l'application
  - Crée et initialise ses contrôleurs
  - Affiche les informations destinées aux utilisateurs
  - Implante les procédure de mise à jour nécessaires pour demeurer cohérente
  - Consulte les données du modèle
- Contrôleur : partie de l'application qui prend les décisions
  - Accepte les événement correspondant aux entrées de l'utilisateur
  - Traduit un événements (1) en demande de service adressée au modèle ou bien (2) en demande d'affichage adressée à la vue
  - Implémente les procédures indirectes de mise à jour des vues si nécessaire

Architecture Modèle-Vue-Contrôleur

- Avantages : approprié pour les systèmes interactifs, particulièrement ceux impliquant plusieurs vues du même modèle de données. Peut être utilisé pour faciliter la maintenance de la cohérence entre les données distribuées
- Inconvénient : goulot d'étranglement possible

- D'un point de vue conception
  - Diviser pour régner : les composants peuvent être conçus indépendamment
  - Cohésion : meilleure cohésion que si les couches vue et contrôle étaient dans l'interface utilisateur.
  - Couplage : le nombre de canaux de communication entre les 3 composants est minimal
  - Réutilisabilité : la vue et le contrôle peuvent être conçus à partir de composants déjà existants
  - Flexibilité : il est facile de changer l'interface utilisateur
  - Testabilité : il est possible de tester l'application indépendamment de l'interface

Styles d'Architecture Logicielle : conclusion

- Pas toujours immédiat d'identifier un style d'architecture
- Ils servent à comprendre et à communiquer.
  - Une architecture centrée donnée peut être encapsulée dans un composant indépendant.
  - Les couches d'une architecture peuvent correspondre à des composants.
  - Les composants dans une architecture pipeline peuvent eux même être mis en oeuvre comme des composants indépendants avec leur propre architecture
  - les architectures client / serveur CORBA peuvent être décrites comme une architecture à couches.

Conclusion

Il y a beaucoup de diagrammes
•Il est important de bien saisir leur articulation
•UML se prête bien àla représentation de l'architecture

Biblio sur les architectures logicielles

- LOG4430 :
  Architecture logicielle et conception avancée, Foutse Khomh
- VERIFICATION AND VALIDATION FOR QUALITY OF UML 2.0 MODELS, BHUVAN UNHELKAR, PHD
- UML (Diagramme de composants,Diagramme de déploiement)
- http://www.emse.fr/~boissier/enseignement/aco/pdf/UML.Deploiement.4pp.pdf
- http://laurent-audibert.developpez.com/Cours-UML/html/Cours-UML051.html
- http://www.iict.ch/Tcom/Cours/OOP/Livre/UML13.pdf
- Modularization and Software Architectures, http://www.softwareresearch.net/fileadmin/src/docs/teaching/WS06/SE1/SE1_lect12.pdf

Biblio diagrammes de composants

- COMPONENT DIAGRAM in UML 2.0, Veronica Carrega
- http://www.ibm.com/developerworks/rational/library/dec04/bell/