Styles architecturaux des applications

Au dessus du système d'exploitation et de ses primitives, on trouve l'univers des "applicatifs". Cet univers est complexe et présente des logiciels de toute nature et de toute structure.

Qu'est-ce qu'un style architectural ?

Comme lm'orsqu'on construit un bâtiment public, qui a une certaine fonction, on peut le construire de plusieurs manières. Certains "styles" ne correspondent pas à toutes les situations : par exemple le style "Pavillon du Vexin" s'adapte mal à la construction d'un collège.

Les styles architecturaux définissent des grandes masses, des grands ensembles qui inscrivent l'application dans un certain jeu de possibilités.

Les grands styles connus

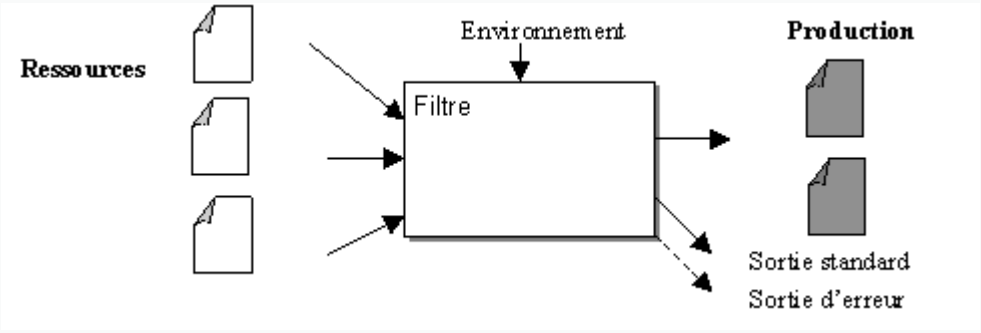
Le style "filtre"

Un filtre est un logiciel qui transforme une entrée en une sortie. Ce filtre est assimilable à une commande système. Il est difficile de dissocier dans les filtres ceux qui constituent une extension du système d'exploitation et ce qui est qualifiable d'applicatif. En général, les deux aspects :

- Déclenchement par ligne de commande texte.
- Exécution linéaire de type "script" avec terminaison.
- Pas d'interaction avec l'utilisateur une fois lancé.

Militent en général pour la qualification de ce logiciel dans la gamme des "outils", si sa destination n'est pas strictement liée à des fonctions système.

Le filtre est une définition assez générale : l'entrée peut être multiple, la sortie également. Par exemple, un compilateur Java peut en quelques sortes être qualifié de filtre : une fois lancé, il exécute son travail de production jusqu'au boût.



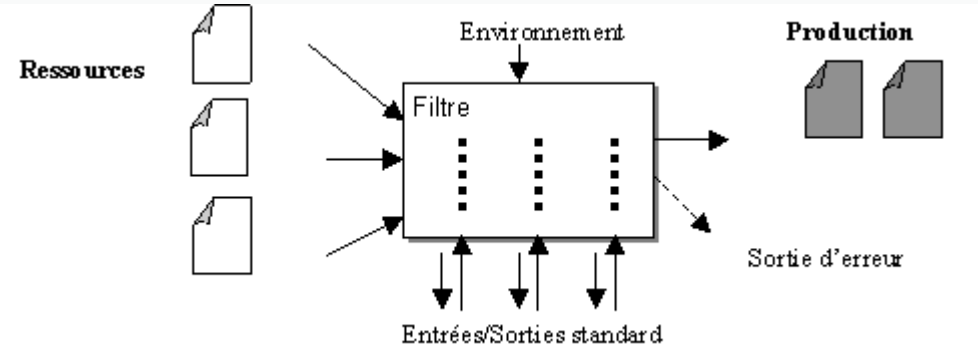
Conséquences de ce modèle :

- Style basé sur les traitements
- La programmation procédurale convient assez bien, il y a peu d'avantages à utiliser un modèle objet pour ce type d'applicatifs.
- Les langages scriptés sont intéressants pour la mise au point et la maintenance des applicatifs.

- Principalement monotâche (ou monoprocessus)
- Traitement linéaire, avec peu de structuration des fonctions.

Le style "filtre interactif" ou "procédure"

Ce style est une amélioration du précédent et permet un dialogue interactif avec l'utilisateur pendant l'exécution de la procédure, par exemple, pour proposer une conduite du déroulement ou des alternatives de traitement. Fonctionnellement, il n'y a que peu de différence entre un filtre non interactif et un filtre interactif : on peut très bien faire entrer toutes les réponses demandées interactivement par des paramètres en ligne de commande. La distinction est purement ethno-ergonomique : écrire une ligne de commande longue et complexe n'est pas pratique et est sujette à de nombreuses erreurs. Or certaines erreurs permettent un déclenchement erroné du programme, avec une production inadéquate qu'il faut nettoyer. Le filtre procédure permet de guider l'utilisateur plus humainement, pour lui faire prendre des décisions une par une.

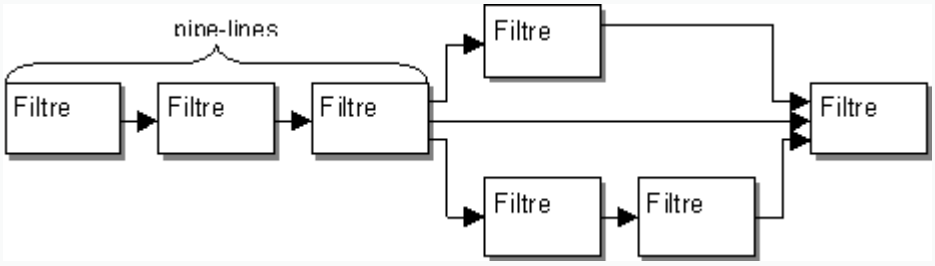


Conséquence du modèle :

- Mise en place d'un dialogue "console" avec réception de commandes ou données intermédiaires.
- Plus difficile à utiliser dans un cadre "entièrement automatique"
- Les différentes options de fonctionnement si elles découlent du dialogue, ramènent l'architecture à un cas de flux (ci-après), à ceci-près qu'une seule séquence d'algorithmes est déclenchée.
- Style basé sur les traitements.
- Langages procéduraux, comme les filtres simples.
- Principalement monotâche (ou monoprocessus)

Le style "flux"

Le style flux est une extension du type filtre. Un tel applicatif consiste en l'assemblage de filtres. Les sorties et les entrées sont combinés pour effectuer un processus de production plus complexe qu'un filtre simple. Le principe de réutilisation d'une sortie dans une entrée est un "pipe-line", pour référer à un assemblage de "tubes emboîtés" les uns dans les autres.

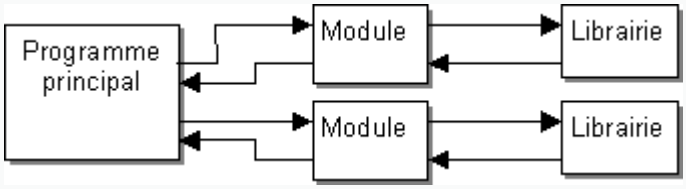


Architecture en flux

Le style "Appels-Retour"

Le style Appels-Retours correspond à une organisation très classique des "programmes procéduraux structurés". On les trouve dans des applicatifs interactifs lorsqu'on parle de "menu de commandes" et non interactifs lorsque certaines fonctionnalités sont réutilisables entre plusieurs applicatifs, ou lorsque le code du programme est trop complexe pour être sérieusement développé et maintenu en un seul morceau.

On trouve des versions compactes (code exécutable compilé en un bloc, exécutable sur un seul environnement), mais aussi des versions plus distribuées, utilisant des principes d'appels "distants" de procédures ou de fonctions, s'exécutant parfois sur d'autres machines que celle qui a démarré le programme (RMI, RPC, Corba, etc.).



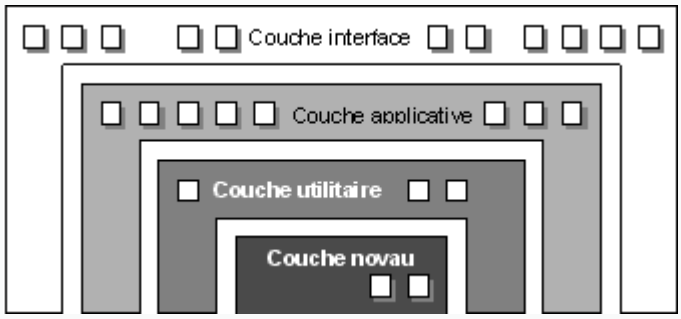
Architecture en appels-retours

Conséquences du modèle :

- La programmation procédurale convient bien, mais elle peut être "encapsulée" dans une présetnation et une segmentation objet.
- L'organisation structurée permet d'accéder à une complexité intermédiaire du logiciel, mais limitée cependant par manque de principes d'encapsulation et une testabilité limitée.
- Réutilisation effective de certaines librairies qu'il faut alors pensée plus "générique".
- Principalement monomode, monotâche, sauf cas particuliers.
- Distribuabilité, dans les langages où c'est pratique.

Architecture en couches

Nous reparlons ici évidemment de l'architecture en couches évoquée dans le chapitre général sur les architectures logicielles. L'architecture en couche est une généralisation de l'architecture précédente, lorsque les différents niveaux d'appels de fonction prennent une sémantique "homogène".



Architecture en couches

Conséquence du modèle :

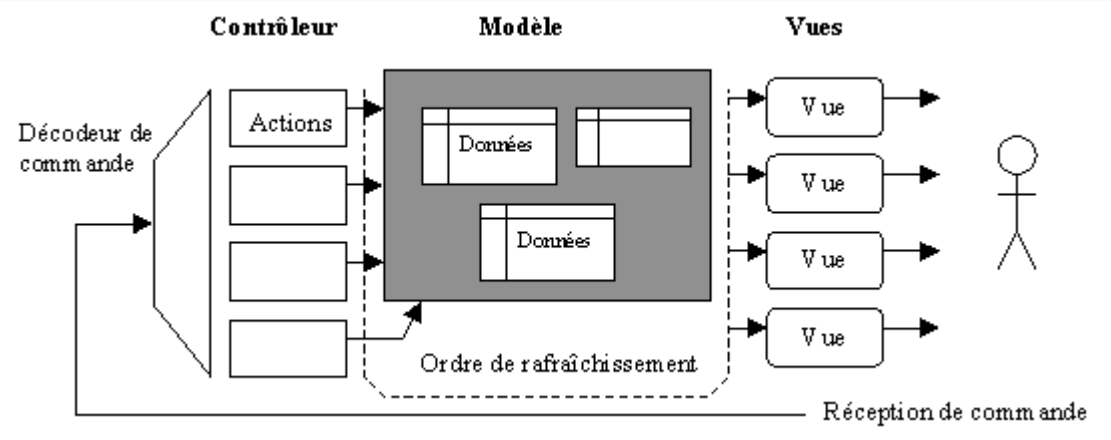
- On peut commencer à élaborer des applications complexes.
- Bonne réutilisabilité des différents "points d'entrée".
- Constitue une infrastructure de développement riche et puissante, accumulant les "boîtes à outils", et accroissant progressivement la productivité e développement.
- Indépendant de la nature procédurale ou objet.

Les architectures centrées "données"

L'architecture modèle-vue-controleur

Cette architecture dissocie trois éléments distincts constituant trois aspects complémentaires d'un procédé logiciel :

- Le problème est représenté par des données. Ce modèle peut être statique dans les cas les plus simples, mais évolutif (en taille, pas en structure) en général.
- Le modèle peut être visualisé par l'utilisateur, sous un ensemble de formes appelées "vues".
- L'évolution du modèle suppose l'exécution de certaines "commandes" qui transforment ou modifient le modèle (donc l'état de l'objet représenté) ce qui nécessitera probablement une remise à jour de la vue.



Architecture Modèle-Vue-Contrôleur

Conséquences du modèle :

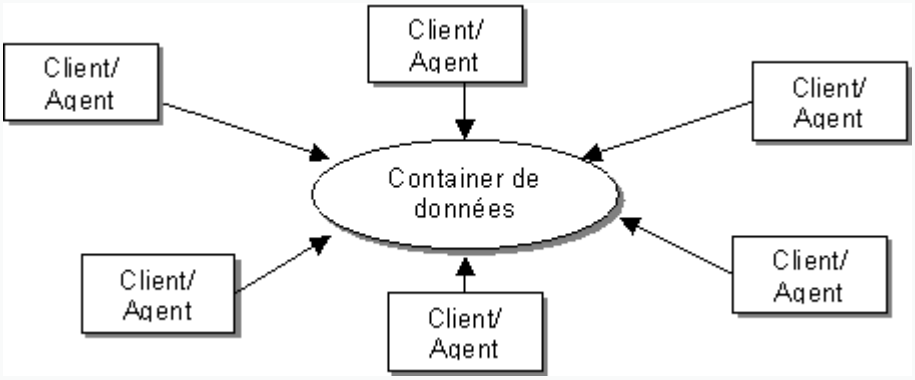
- Convient bien à une application monoutilisateur graphique (traitement événementiel de la réception de commandes).
- Adapté à une conception orienté objet, mais également compatible avec un développement procédural.

L'application de gestion de données centralisée

Le modèle MVC introduit un concept de centrage de l'analyse sur le modèle de données. C'est lui qui devient la matérialisation du problème à traiter, les autres aspects, vues et contrôleurs sont des aspects plus techniques et "standardisablese" que la conception du modèle central. C'est en général sur ce modèle central que se concentre la méthodologie UML.

Le modèle MVC représente bien une application "monoutilisateur", telle que des applications bureautique classiques ou des applicatis scientifiques de laboratoire. Lorsque le modèle de données est une représentation "centrale", collectivement utilisée, il faut

étendre ce modèle en centralisant le modèle de données.



Architecture centralisée

Cette architecture introduit une architecture technique trés connue : l'architecture "client-serveur".

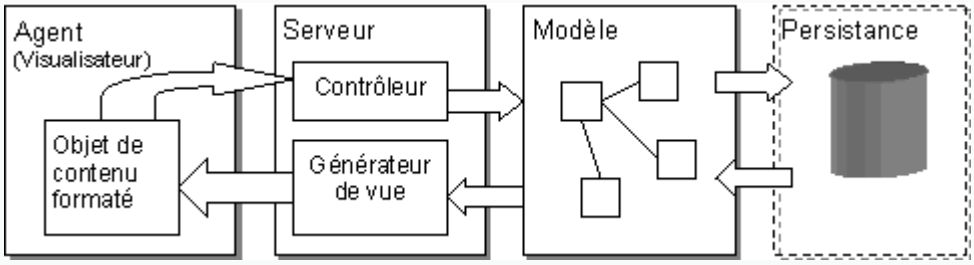
Le "client" réceptionne les commandes les valide et les transmet au serveur et réalise les vues.

Le serveur exécute les actions du contrôleur et maintient le modèle de données.

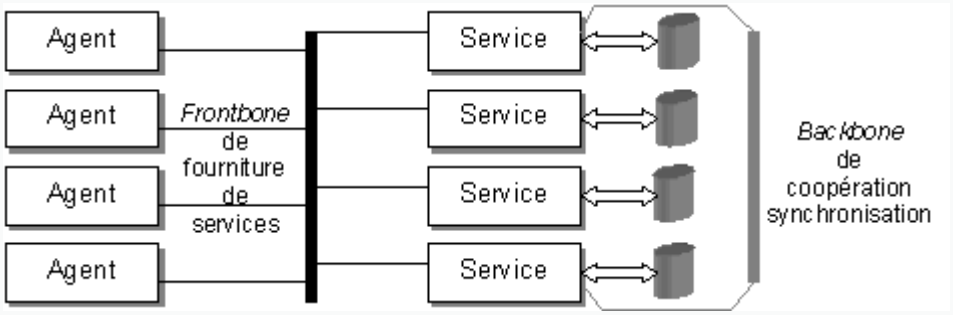
Conséquences de ce modèle :

- Possibilité d'augmenter la puissance fonctionnelle de l'application par augmentation du nombre d'opérateurs.
- Programmation nécessairement multitâche pour la partie serveur, qui doit exécuter simultanément les appels des différents clients.
- Nécessité d'un arbitrage et d'une gestion de "concurrence".
- Les clients peuvent être des applications MVC dont on a déporté le modèle (par la méthode des appels distants vues dans le modèle appels-retours).
- Nécessité d'une connectivité pour faire fonctionner l'ensemble.

L'architecture centrée sur les contenus



L'architecture Agents/Services


Agent
Agent
Agent
Agent
Frontbone
de
fourniture
de
services
Service
Service
Service
Service
Backbone
de
coopération
synchronisation