UE 1 : Architecture logicielle

- Cours 4 :

  Appliquer des styles architecturaux et des patterns
  - Les styles d'architecture
  - Les patterns d'architecture

Introduction

Deux types de produits logiciels réutilisables

- les familles de produits
  - Architectures logicielles spécifiques du domaine
  - Familles d'application pour un domaine commun
  - Classification par fonction
- les patterns d'architecture
  - Styles d'architecture : classes de systèmes basés sur la forme de la solution
  - Patterns et styles : principes de conception réutilisable

Choix d'une architecture

- Le choix dépend de exigences fonctionnelles et non fonctionnelles du logiciel
- Choix favorisant la stabilité : l'ajout de nouveaux éléments sera facile et ne nécessitera en général que des ajustements mineurs à l'architecture.
- Choix influencé par certains « modèles connus » de décomposition en composants et de mode d'interactions .

Propriétés des patterns pour l'architecture logicielle

- Les patterns documentent des bonnes pratiques existantes et construites sur une expérience de conception éprouvée et testée.
- Les patterns identifient et spécifient des abstractions qui sont au-dessus du niveau des objets, des classes et des composants.
- Les patterns fournissent un vocabulaire commun et une compréhension partagée pour des concepts de conception.
- Les patterns sont un moyen de documentation des architectures logicielles.

Style architectural

Un style architectural fait référence à la forme générale d'un système.

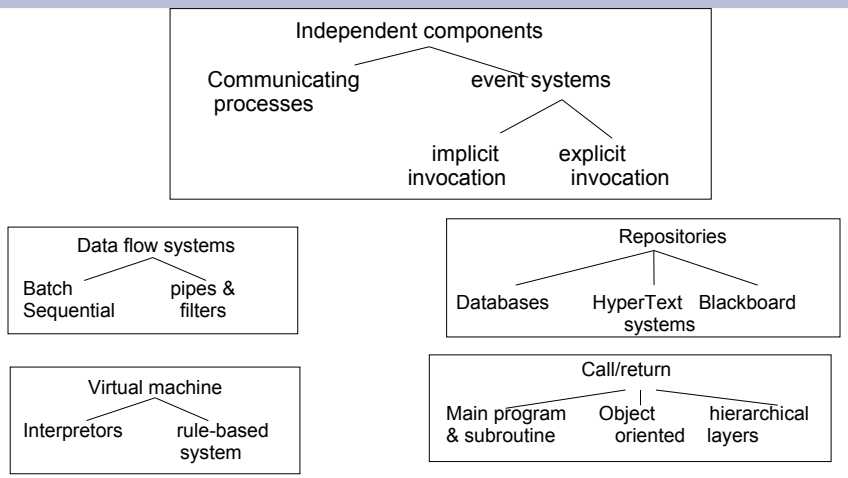
Choisir le style approprié est important parce que les décisions de conception sont faites dans le contexte de ce style.

Les éléments d'un style
- Les blocs de construction de base : les éléments clés et comment les composants et les services sont nommés
- Les connexions entre les blocs de base : communication entre les composants
- Les règles de configuration des composants et connecteurs et des contraintes qui définissent comment les éléments peuvent être intégrés pour former le système.
- La famille de solutions
- Les contextes et situations de problèmes dans lesquels le style est le plus utile.

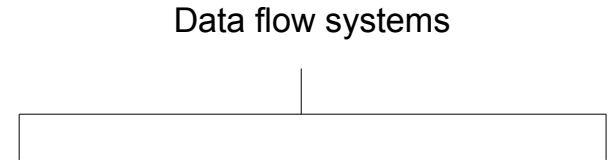
Style architectural : famille et langage

- Un style architectural caractérise une famille de systèmes reliés par des propriétés structurelles et sémantiques partagées.
- Plus spécifiquement un style architectural fournit un langage de conception spécialisé pour une classe spécifique de systèmes.

Taxonomie des styles ou idiomes
(d'après Shaw et Garlan 1994)



Style 1 : Architecture en flot de données



Batch sequential

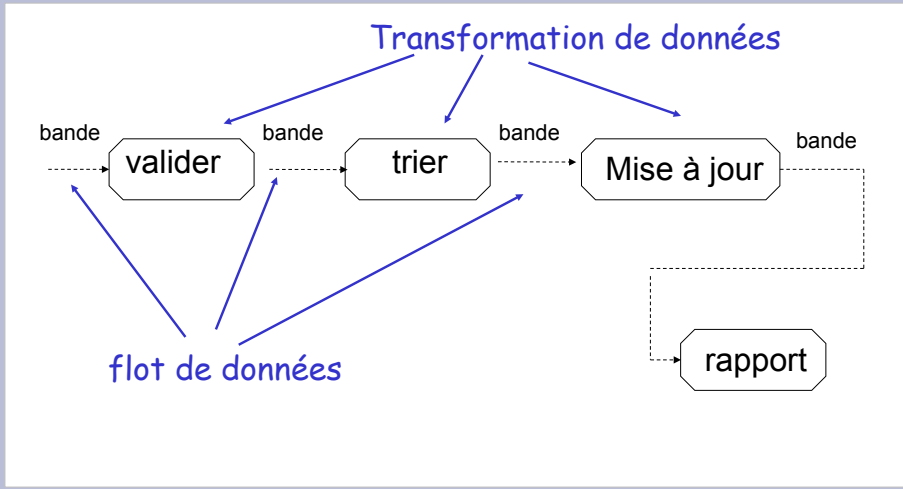
Boucle de contrôle

maj base de données

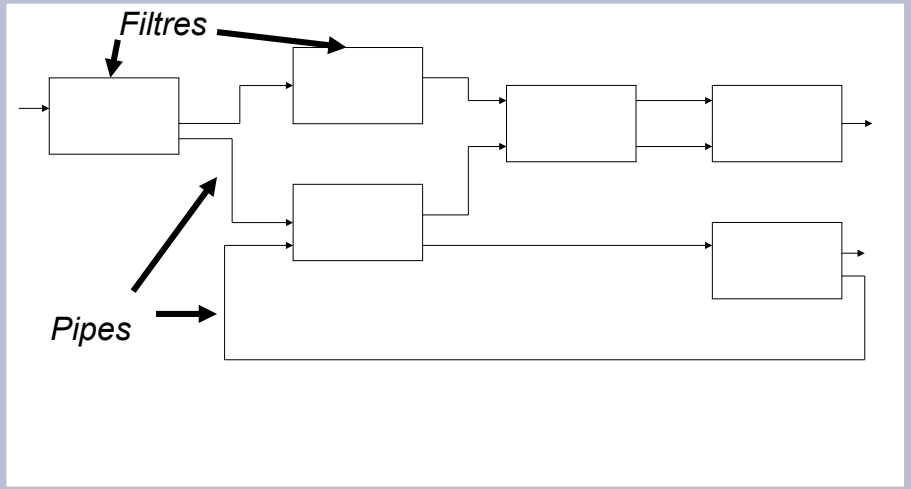
Pipes and Filters

Script shell Unix

Flot de données : batch séquentiel



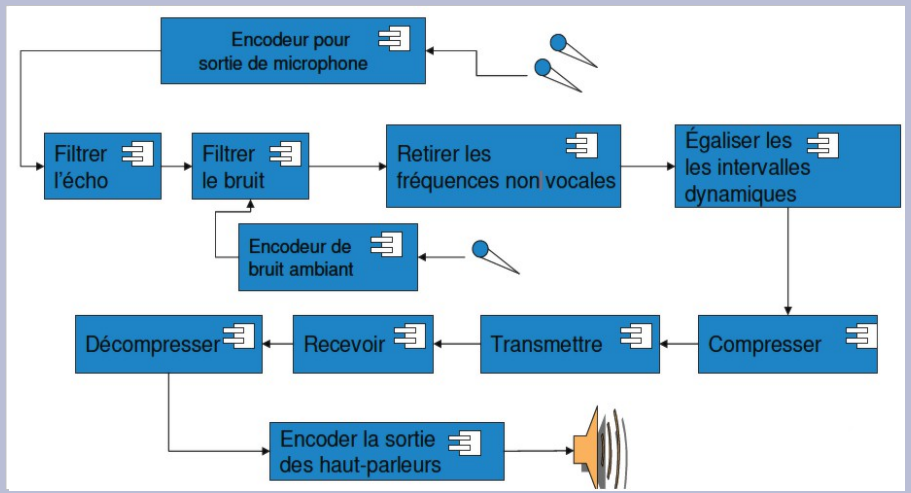
Style Flot de données : Pipes and Filters



Architecture pipeline

- Convient bien aux systèmes de traitement et de transformation de données
- Composant = filtre ; connecteur = canal
- Filtre
  - Traite de façon indépendante et asynchrone
  - Reçoit ses données d'un ou plusieurs canaux d'entrée, effectue le traitement des données et envoie les données de sortie sur un ou plusieurs canaux de sortie
  - Fonctionnement en concurrence
- Canal
  - Unidirectionnel au travers duquel circule un flot de données
  - Synchronisation et utilisation d'une zone tampon parfois nécessaire pour assurer un bon fonctionnement entre filtre producteur et filtre consommateur

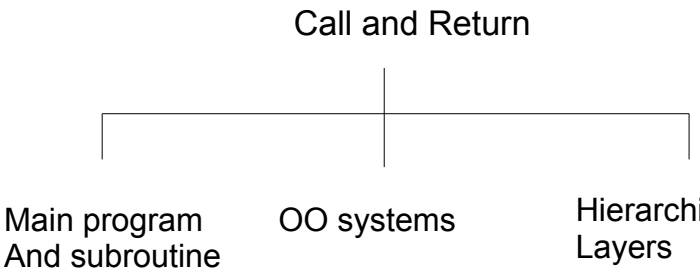
Exemple pipe & filter : système de traitement du son



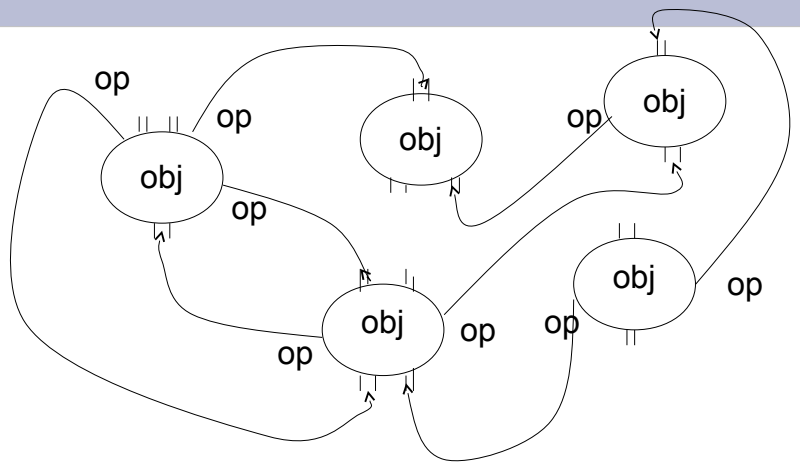
Avantages et inconvénients de l'architecture pipeline

- Avantages
  - Bon pour le traitement en lot
  - Très flexible
  - Analyse facilitée : performance, synchronisation
  - Se prête bien à la décomposition fonctionnelle d'un système
- Inconvénients
  - Mauvais pour le traitement interactif
- D'un point de vue conception
  - Diviser pour régner
  - Cohésion fonctionnelle des filtres
  - Couplage : entrée/sortie
  - Abstraction : détails internes des filtres cachés
  - Réutilisabilité : réutilisation des filtres dans d'autres contextes
  - Réutilisation : insertion de filtres existants dans le pipeline

Style 2 : Architecture « call-and-return »



Abstraction de données / Objets



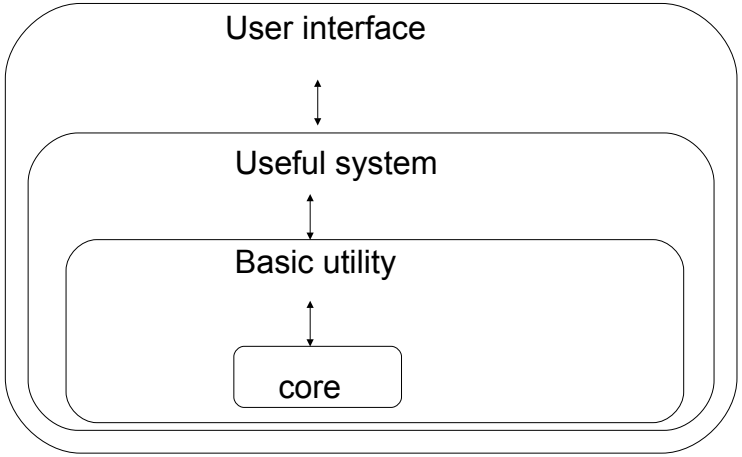
Avantages de l'approche objets

- Décomposition du problème
  - Correspondance naturelle avec les entités du monde réel
  - L'héritage permet le partage de définition
- Maintenance et réutilisation
  - Décroît le couplage
  - Accroît la réutilisation
- Protection des représentations internes
  - Encapsulation permet l'intégrité des données et des états

Désavantages de l'approche objets

- Maintenance
  - requiert plus de structure (un niveau d'objets trop plat)
- Effets de bord
  - Accès multiples des objets à une seule ressource
- Identité
  - Pour importer besoin des noms objets/méthodes
- Héritage
  - Souvent non intuitif

Le style à couches



Avantages des couches

- Portabilité
  - Chaque couche constitue une machine abstraite
- Réutilisabilité :
  - les couches inférieures peuvent être conçues de façon à offrir des solutions génériques réutilisables
- Abstraction :
  - on n'a pas à connaître les détails d'implémentation des couches inférieures
- Réutilisation
  - Les couches inférieures peuvent être conçues de façon à offrir des solutions génériques réutilisables.
  - Des interfaces standards peuvent être créées

Désavantages des couches

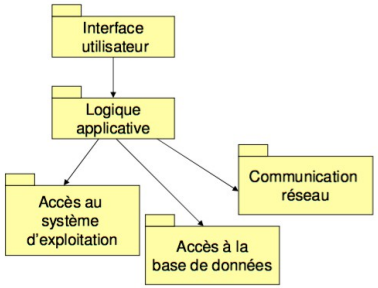
- Performance
  - requiert plus de structure (un niveau d'objets trop plat)
- Peut être le mauvais modèle
  - Si on veut contrôler plus profondément
- Difficile de trouver les bonnes abstractions
  - Danger de connexion forte des couches
- Connexion des couches
  - Ruine souvent le modèle

Modèles en 5 couches

Décomposition logique d'une application en 5 couches « successives » :

- Présentation
  - Gère et assure l'affiche de l'interface graphique utilisateur
- Contrôleur
  - Invocation des appels de service, gère les sessions …
- Services
  - Gère les services métiers qui enchaînent des règles métiers et des appels à la couche Domaine
- Domaine
  - Concentrée sur le métier de l'entreprise, recense les objets métiers
- Persistance
  - Service de stockage des données

Exemple d'architecture multi-couches

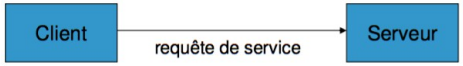


Architecture n-niveaux

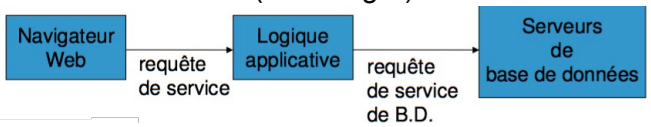
- Pour les systèmes distribués
  - Comparable à une architecture par couches... dont les couches seraient distribuées
- Par abus de langage, la notion de tier a pris le sens de couche distribuée
- Composants : chaque niveau est représenté par un composant qui joue le rôle de client et/ou de serveur
- Connecteurs : relient un client à un serveur. Connexion asymétrique. Doit supporter les communication distantes (*e.g.*, requêtes RPC, HTTP, TCP/IP)
- Exemples : Client-serveur,Trois niveaux,Quatre niveaux

Exemples : architecture n-niveaux

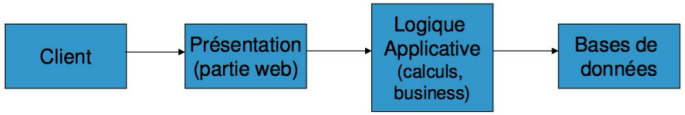
- Architecture 2-niveaux (client-serveur ou client lourd)



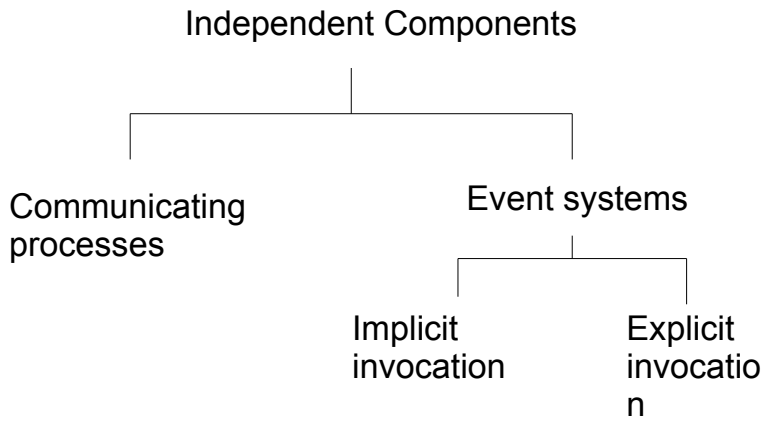
- Architecture 3-niveaux (client léger)



- Architecture 4-niveaux



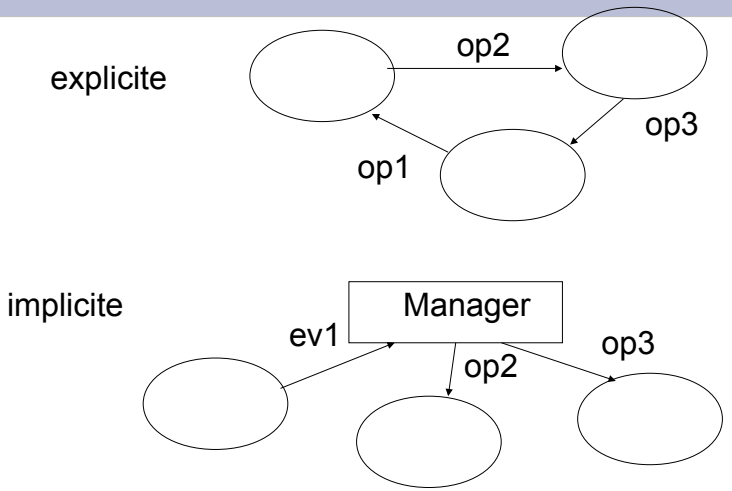
Style 3 : Composants indépendants



Processus communicants

- Composants
  - Processus indépendants
- Connecteurs : envoi de message
  - point à point
  - asynchrone et synchrone
  - RPC etc.

Evénements : invocation implicite vs. explicite



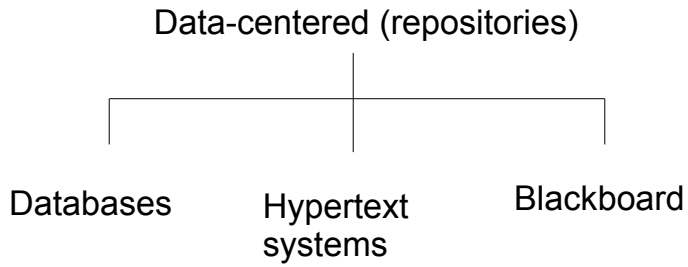
Avantages : systèmes d'événement

- Décomposition du problème
  - Calcul et coordination sont séparés
- Maintenance et réutilisation
  - Pas de dépendance de nom statique
  - Ajout facile de nouveaux objets (inscription)
  - Facilité d'intégration
- Performance
  - Invocations parallèles

Désavantages : systèmes d'événement

- Décomposition du problème
  - Pas de contrôle sur l'ordre d'invocation
- Maintenance et réutilisation
  - Requiert des « pages jaunes » centralisées pour « who knows what »
- Performance
  - Indirection/communication -> pénalité de performance

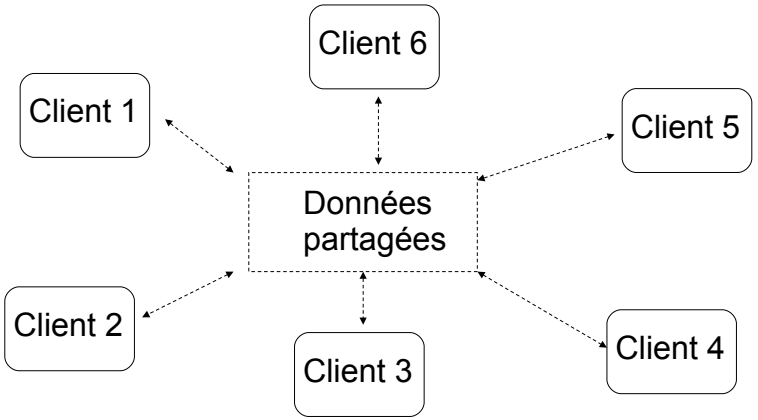
Style 4 : Architecture centrée sur les données



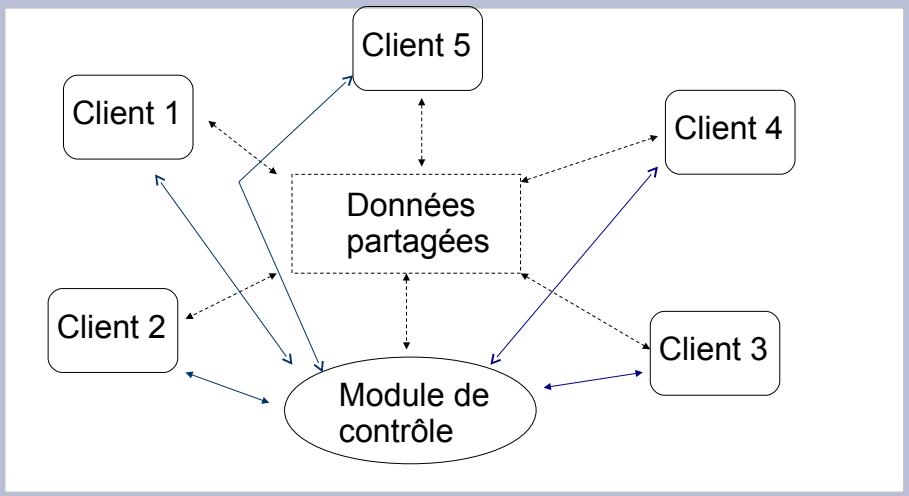
Architecture avec référentiel

- Ce style d'architecture est utilisé dans le cas où des données sont partagées et fréquemment échangées entre les composants.
- Deux types de composants :
  - Les référentiels de données constituent le médium de communication entre les accesseurs de données.
- Un connecteur relie un accesseur à un référentiel
  - Rôle de communication mais aussi possiblement de coorination, de conversion et de facilitation.
- Variantes
  - Style référentiel
  - Style tableau noir

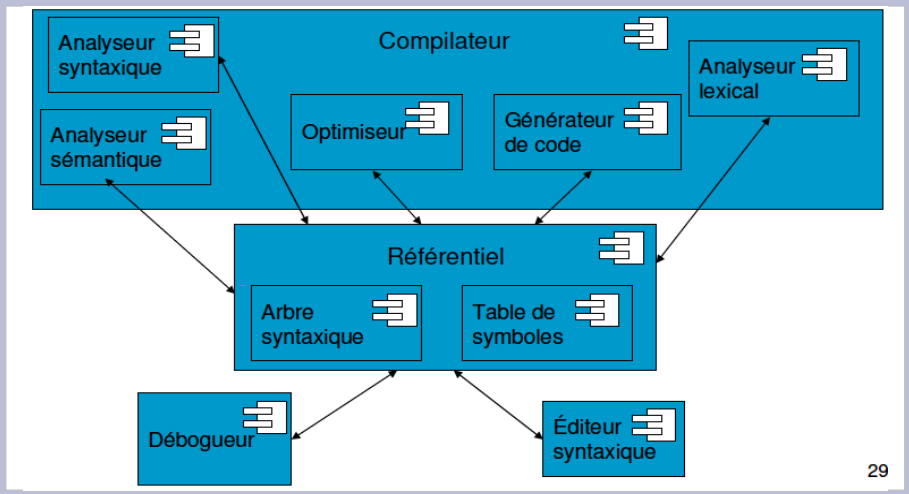
Style centré sur les données (style référentiel)



Style centré sur les données (blackboard)



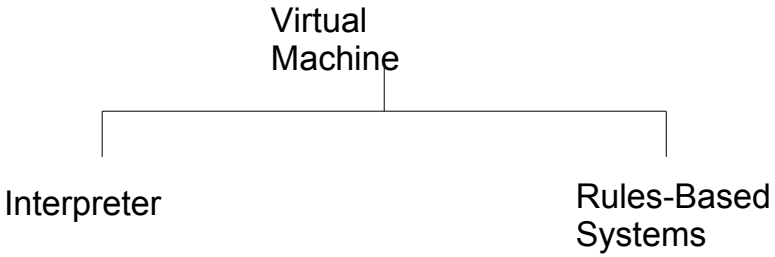
Architecture avec référentiel : environnement de programmation



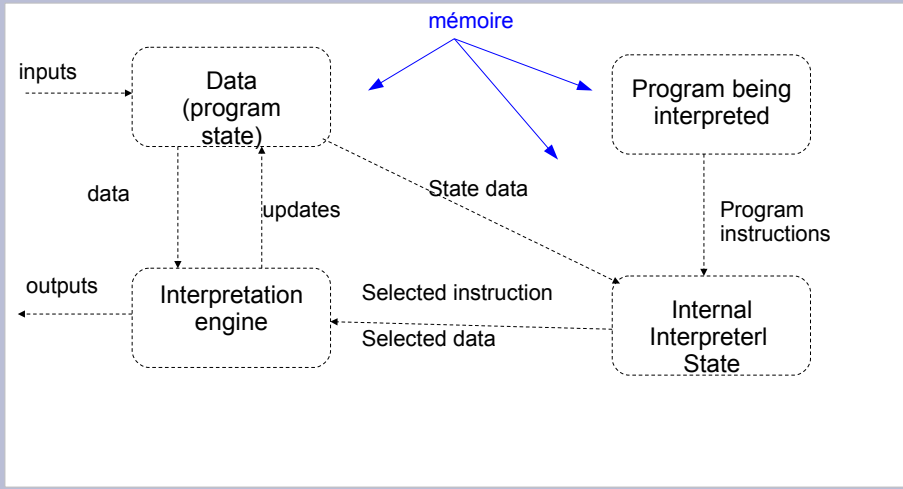
Avantages et inconvénients de l'architecture avec référentiel

- Avantages
  - Bien pour les applications impliquant des tâches complexes sur des données nombreuses et changeant souvent.
  - Une fois le référentiel bien défini, de nouveaux services peuvent facilement être ajoutés.
- Inconvénient
  - Le référentiel peut facilement constituer un goulot d'étranglement

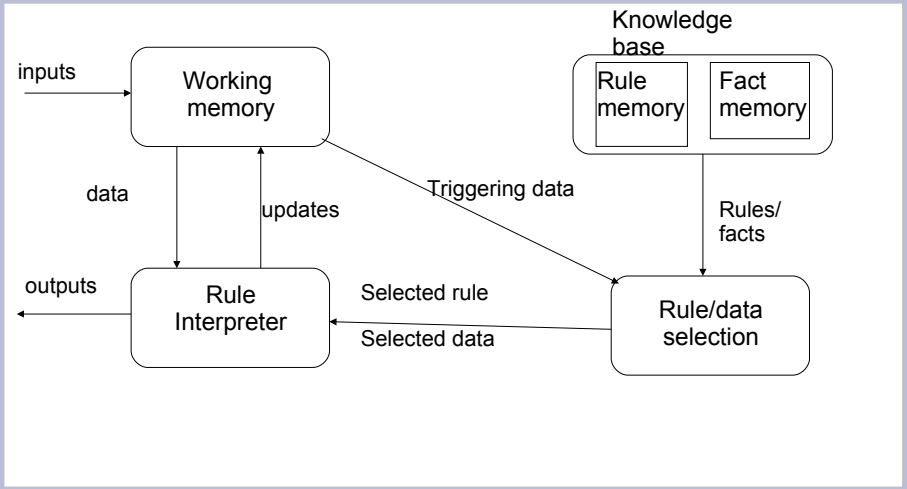
Style 5 : Architecture machine virtuelle



Interpréteur



Système à base de règles



Autres architectures familières

- Les processus distribués
  - Caractérisés par leurs aspects topologiques (organisation en anneau, en étoile)
  - Caractérisés aussi par le type de communication
- Domain-specific software architectures (DSSA)
  - Architecture de référence pour des domaines spécifiques
  - Structure adaptée à une famille d'application (avioniques, commande et contrôle, gestion de tournée)
- Systèmes état-transition
  - Organisation courante des systèmes réactifs

Architectures hétérogènes

Combinaison des styles d'architecture de 3 sortes :

- « locationnaly heterogenous » (des branches d'un programme principal ont un répertoire de données partagé)
- « hierarchically heterogenous » : le composant d'un style quand il est décomposé est structuré selon les règles d'un style différent.
- « Simultaneoustly heterogenous » (plusieurs styles possibles)

Les patterns d'architecture

- Interviennent dans un haut niveau de description
- Souvent découverts à partir de gros projets
- De nombreux catalogues existent
- Correspondance avec les styles d'architecture

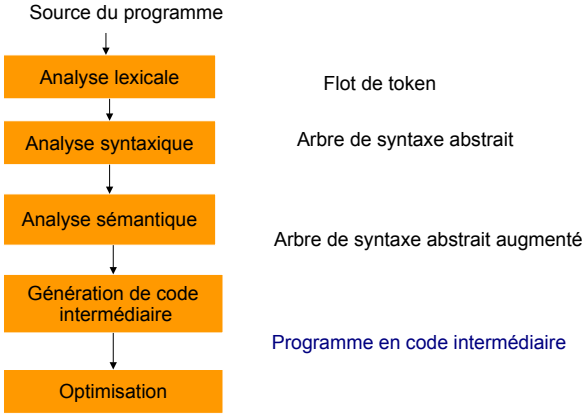
Schéma de présentation des patterns

- nom
- exemple
- contexte
- problème
- solution
- structure (CRC cards)
- dynamique (scénarios)
- implémentation
- exemple résolu
- variantes
- usage connus
- conséquences
- voir aussi
- crédits

Un premier exemple : Pipe and Filter

- Chaque composant possède un ensemble de données en entrée et un ensemble de données en sortie.
- Un composant lit un flot de données en entrée et produit un flot de données en sortie.

Exemple de pipe & filter : compilateur



Le pattern d'architecture Pipe & Filter

- Le problème : construire des systèmes pour transformer et calculer des flots de données.
- La solution : la tâche globale est naturellement décomposée en plusieurs étapes de calcul séquentielles.
- La structure :

Class Filter	Collaborators
Responsibility • Gets input Data. • Performs a function,on its input data. • Supplies output data.	Pipe

Class Pipe	Collaborators
Responsibility - Transfers Data. - Buffers Data. - Synchronizes active neighbours.	• Data Source • Data Sink • Filter

Class Data Source	Collaborators
Responsibility Delivers input to processing pipeline	Pipe

Class Data Sink	Collaborators
Responsibility Consumes output	Pipe

Le catalogue de Buschmann

	Achitectural Patterns	Design Patterns	Idioms
From Mud to Structure	Layers Pipe and Filters Blackboard		
Distributed Systems	Broker Pipes and Filters Microkernel		
Interactive Systems	MVC PAC		
Adaptable Systems	Microkernel Reflection		

Suite du catalogue

	Achitectural Patterns	Design Patterns	Idioms
Structural Decomposition		Whole-part	
Organization of Work		Master-Slave	
Management		Command Processor View Handler	
Access Control		Proxy	
Communication		Publisher-Subscriber Forwarder-Receiver Client-Dispatcher-Server	
Resource Handling			Counted Pointer

Catalogue de Design Patterns [GoF]

purpose →
scope ↓

	Creational	Structural	Behavioral
class	Factory Method	Adapter (class)	Interpreter Template Method
object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State, Strategy, Visitor

Classification selon l'intention pour les patterns en Java

Interface	*Responsibility*	*Construction*	*Operation*	*Extension*
Adapter(S)	Singleton(C)	Builder(C)	Template Method(B)	Decorator (S)
Facade (S)	Observer(B)	Factory Method(C)	State (B)	Iterator (B)
Composite(S)	Mediator(B)	Abstract Factory(C)	Strategy (B)	Visitor (B)
Bridge(S)	Proxy(S)	Prototype (C)	Command (B)	
	Chain of Responsibility(B)	Memento(B)	Interpreter(B)	
	Flyweight(S)			