Patrons d'analyse et de conception

Khalid Benali

M2 SID & M 2ACSI 2005 2006
Université Nancy 2

Bibliographie

- Design pattern (GOF ≅ "Gang OF Four", Gamma et al.)
- System of patterns (Buschman et al.)
- Applying UML and Patterns (Larman)
- Analysis Patterns (Fowler)
- Sur le WEB:
  - Cetus Links : http://www.cetus-links.org/

Matériel pédagogique utilisé

- Transparents de Franck Morer, Calgary
- Transparents de David L. Levine, St Louis
- Transparents de P. Molli, Nancy
- Transparents de Philippe Perrin, EDF

Rappel

Méthode de Conception Orientée Objet et example (in english)...

Cours de Master 1 MIAGE
Khalid Benali
Université Nancy 2

Un exemple

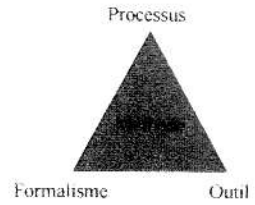
- Un jeu de dés
- Le joueur lance 10 x 2 dés
- Si le total fait 7, il marque 10 points à son score
- En fin de partie, son score est inscrit dans le tableau des scores.

Le Rational Unified Process (RUP)

- Les conseils (guide/démarche) sur les étapes à franchir pour réaliser une analyse ou une conception s'appellent le procédé (process) d'analyse ou de conception
- RUP = Processus développé par Rational (Grady Booch, Ivar Jacobson, James Rumbaugh.)

UML

- Une notation graphique ou formalisme est utilisée pour exprimer une analyse ou une conception
- UML = Unified Modeling Language
  Langage Unifié de Modélisation
- UML = Une dizaine de formalismes ou notations graphiques

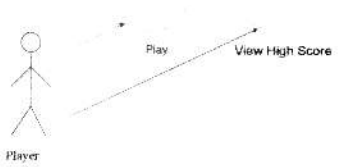


Use case View

- D'abord les use cases dans la « use case view » de Rational Rose

- Un cas d'utilisation est une interaction typique
  entre un utilisateur
  et un système informatique
- Deux cas d'utilisation typiques pour un traitement de texte:
  « mettre en gras un certain texte »
  « indexer un document »
- Un cas d'utilisation
  capture une fonction visible pour l'utilisateur
  peut être simple ou complexe
  a un but précis pour l'utilisateur

Use case View



Use case View

- Ensuite la description des use cases par des diagrammes d'activités (tjs dans la use case view « sous » le use case décrit)

Use case View

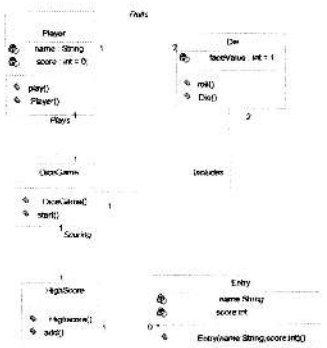
Roll Dice

Store Score

Logical View

- Description des classes dans la «logical view»

Logical View



Logical View

- Description des use cases de réalisation dans la logical view et description des relations dans un diagramme de use cases nommé Réalisations

Logical View

«realize»

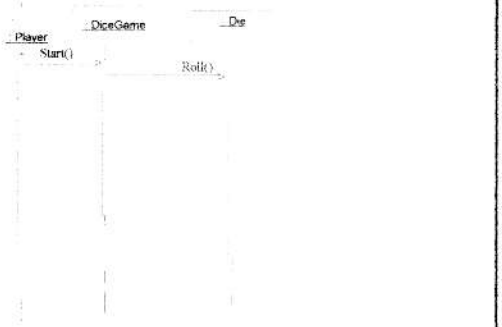
Play

Play

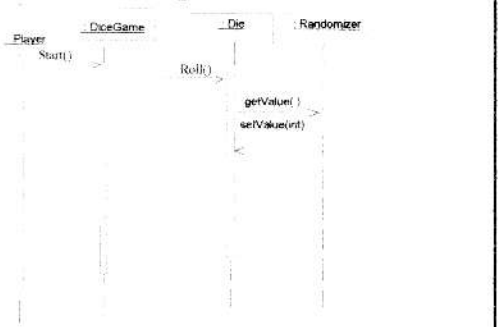
Logical View

- Description des interactions entre objets pour la mise en œuvre des use cases de réalisation.

Logical View



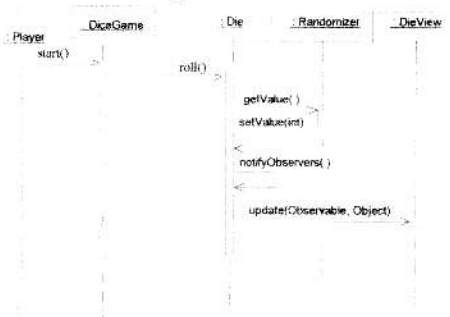
Logical View



Logical View

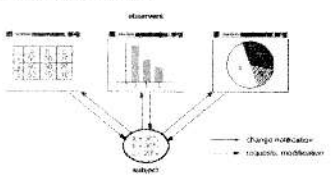
- Une nouvelle classe apparaît « Randomizer »
- Une nouvelle méthode de Die apparaît « setValue »
- Que fait on du résultat du lancer de dé ?

Logical View



Observer (Observateur)

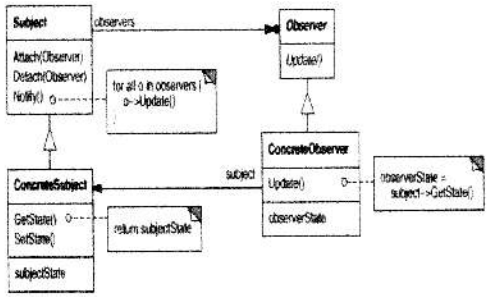
- Dépendance Un-à-Plusieurs entre objets: des changements d'un objet devront être automatiquement notifiés aux observateurs



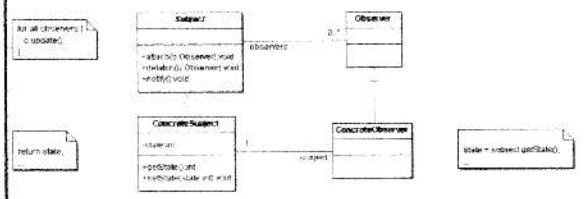
Observer: Applicabilité

- Un changement d'un objet nécessite le changement d'un ensemble « indéfini » d'autres objets
- Un objet devrait être capable d'en notifier d'autres qui peuvent ne pas être connus au début

Observer: Structure



Observer: Structure (en UML)



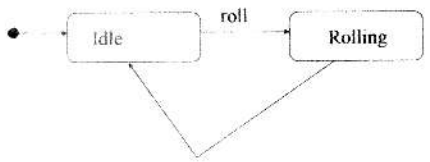
Observer: Conséquences

- Couplage abstrait entre un sujet et des observateurs
- Supporte la communication par diffusion
- Difficile à maintenir

Logical View

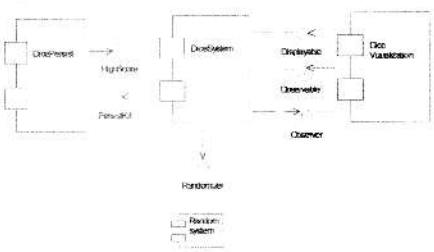
- Description des états et des transitions d'état pour les classes importantes dans la logical view (Die par exemple ci-après).

Logical View



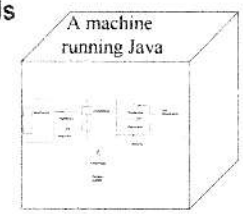
Component View

- Description des composants logiciels implémentant les classes



Deployment View

- Description des composants matériels sur lesquels tournent les composants logiciels



RUP en un clin d'œil

- 4 vues qui se complètent les unes les autres et qu'on augmente au fur et à mesure de l'avancement dans le projet
  - Use case view
  - Logical view
  - Component view
  - Deployment view

Les Patterns (ou Patrons)

Historique des patterns
Définition et typologie des patterns
Les patterns d'analyse
Les patterns de conception
Les patterns de programmation
UML et la représentation des patterns
Patterns, composants, frameworks et toolkits
Conclusion

Historique des patterns

- A l'origine, en 1964, Christopher Alexander décrit des patterns dans le domaine de l'architecture (Description en langage naturel (~10 pages par pattern))

Original definition (C. Alexander)

```
IF you find yourself in CONTEXT
 for example EXAMPLES,
 with PROBLEM,
 entailing FORCES
THEN for some REASONS,
 apply DESIGN FORM AND/OR RULE
 to construct SOLUTION
 leading to NEW CONTEXT and OTHER PATTERNS
```

Description d'un problème récurrent et des grandes lignes d'une solution générale à ce problème

Exemple : Pattern No 112 d'Alexander

- Nom : Transition d'entrée
- Quoi : L'entrée crée une transition du monde extérieur vers un univers intérieur, plus privé.
- Pourquoi : L'entrée dans un bâtiment influence la façon dont on va se sentir à l'intérieur.

- Quand : Pour la conception de systèmes d'accès et d'entrée pour les maisons, les cliniques, les magasins, les églises, etc.
- Comment : Créer un espace de transition entre la rue et la porte d'entrée. Marquer le cheminement dans l'espace de transition par un changement de lumière, un changement de direction, etc.
- Patterns en relation : vue Zen (No 134), mur de jardin (No 173).

Définitions

- Pattern :
  Description d'un problème récurrent et des grandes lignes d'une solution générale à ce problème (Alexander).

  Description d'une micro-architecture permettant la résolution d'un problème dans un contexte donné. Il est abstrait.

  Son implémentation doit se faire à chaque cas concret d'utilisation.

- Idiome (ou pattern de programmation) :
  Résultat de la définition, dans un langage donné, d'une méthode d'implémentation d'un trait absent dans ce langage
- Par exemple, l'héritage multiple doit être implémenté par un idiome en Smalltalk alors qu'en C++ il est présent.

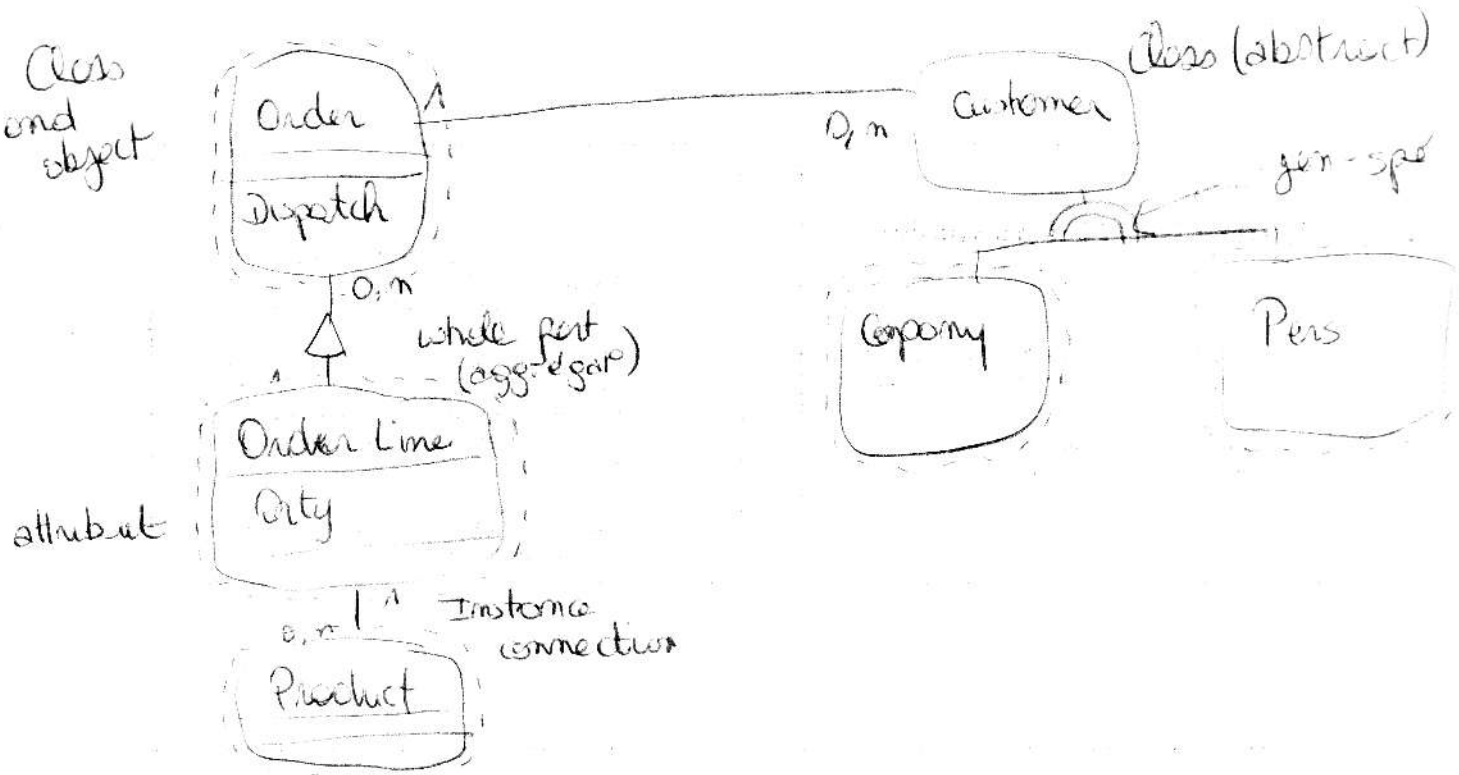
Typologie des patterns

- **Les patterns pour l'analyse**
- **Les patterns pour la conception**
- **Les patterns pour la programmation**

Les patterns pour l'analyse

- **Strategies and Patterns for building Object Models**
- **Analysis Patterns**
- **Data Models Patterns**

Coad / Yourdon object-oriented analysis model



Les patterns pour la conception

Les patterns pour la conception sont aussi appelés Design Patterns :

- GoF
- Siemens Group
- ...

Les patterns pour la programmation ou idiomes

- Idiomes pour le C++ (Coplien, ...)
- Idiomes pour Smalltalk (K. Beck)
- Idiomes pour Java

Les patterns pour l'analyse

Les Object-Oriented patterns de Coad

- Peter Coad a défini en 1995 des stratégies (146) et des patterns (31) pour construire des modèles objet.
- Coad Object-Oriented Pattern = Sextuplet

(Name , Structure, typical object interactions, Examples, Combinations, notes)

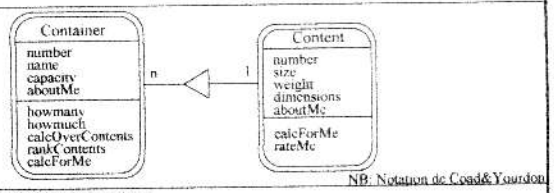
Coad Object-Oriented Pattern :

- name : nom du pattern
- structure : Représentation graphique de la structure statique des classes mises en œuvre dans le pattern.
- typical object interactions: Spécifications des services des objets
- examples : énumération d'exemples
- combinations : liste des patterns pouvant être mis en relation
- notes : Explication supplémentaire nécessaire à la bonne compréhension du pattern.

- Le pattern fondamental
  #1. "Collection-Worker"
- Les 12 patterns "Transaction"
  #2. "Actor-Participant"
  #3. "Participant-Transaction"
  #4. "Place-Transaction"
  #5. "Specific Item-Transaction"
  #6. "Transaction-Transaction Line Item"
  #7. "Transaction-Subsequent Transaction"
  #8. "Transaction Line Item-Subsequent Transaction Line Item"
  #9. "Item-Line Item"
  #10. "Specific Item-Line Item"
  #11. "Item-Specific Item"
  #12. "Associate-Other Associate"
  #13. "Specific Item-Hierarchical Item"
- Les 6 patterns "Agrégation"
  #14. "Container-Content"
  #15. "Container-Container Line Item"
  #16. "Group-Member"
  #17. "Assembly-Part"
  #18. "Compound Part-Part"
  #19. "Packet-Packet Component"
- Les 5 patterns "Project"
  #20. "Plan-Step"
  #21. "Plan-Plan Execution"
  #22. "Plan Execution-Step Execution"
  #23. "Step-Step Execution"
  #24. "Plan-Plan Version"
- Les 7 patterns "Interaction"
  #25. "Peer-Peer"
  #26. "Proxy-Specific Item"
  #27. "Publisher-Subscriber"
  #28. "Sender-Pass Through-Receiver"
  #29. "Sender-Lookup-Receiver"
  #30. "Caller-Dispatcher-Caller Back"
  #31. "Gatekeeper-Request-Resource"

Exemple de pattern de Coad

#14. « Container – Content » Pattern — aggregate patterns

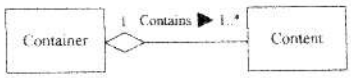


NB: Notation de Coad&Yourdon

- Typical object interactions
  howmany → calcForMe
  howmuch → calcForMe
  calcOverContents → calcForMe
  rankContent → rateMe
- Examples
  - **Containers:** airport,aircraft, aisle,bank,bin, building, cabinet, folder, garage, hangar, hospital, locker, room, safe, strore, warehouse.
  - **Container-Content:** aircraft-cargo;aircraft-passenger;building-room; catalog-catalog item;store-item;warehouse-loading dock.
  - **Combinations:** container-content(again);container-container line item; group-member;assembly-part.

  Also, when « container » or « content » is a participant, place or specific item: actor-participant;participant-transaction;place-transaction;...

#14. « Container – Content » Pattern — aggregate patterns



NB : Notation UML

- Typical object interactions
  howmany → calcForMe
  howmuch → calcForMe
  calcOverContents → calcForMe
  rankContent → rateMe
- Examples
  - **Containers:** airport,aircraft, aisle,bank,bin, building, cabinet, folder, garage, hangar, hospital, locker, room, safe, strore, warehouse
  - **Container-Content:** aircraft-cargo;aircraft-passenger;building-room; catalog-catalog item;store-item;warehouse-loading dock.
  - **Combinations:** container-content(again);container-container line item; group-member;assembly-part

  Also, when « container » or « content » is a participant, place or specific item: actor-participant;participant-transaction;place-transaction;...

Les patterns d'analyse de Fowler

Description des patterns présentés en 1997:

- Titre : nom du pattern
- Section : « Type » du pattern
- Principe de modélisation : Expression de règle de bonne conduite pour solutionner le problème posé.
- Description : description des différents contextes où le pattern est utile
- Structure : Utilisation de modèles pour représenter la structure statique, les interactions, les cycles de vie.
- Exemples : description d'exemples tirés de la réalité des entreprises.

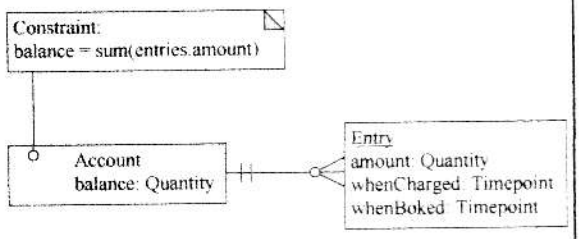
- Quelques «Analysis Patterns» de Fowler
  - Party
  - Organization Hierarchies
  - Organization Structure
  - Accountability
  - Party Type Generalizations
  - Hierarchic Accountability
  - Transaction
  - Quantity
  - Measurement
  - Observation
  - Protocol
  - Enterprise Segment
  - Account ...

- Quelques «Support Patterns» de Fowler
  - Two-Tier Architecture
  - Three-Tier Architecture
  - Presentation and Application Logic
  - Database Interaction...
- NB : Les Support Patterns de Fowler sont en fait des design patterns.

Exemple de pattern d'analyse de Fowler

- Titre : "Account" (compte)
- Section : "Inventory and Accounting"
- Principe de modélisation : Pour mémoriser l'histoire des modifications d'une valeur, utiliser un compte pour cette valeur.
- Description : Dans de nombreux domaines, il est important de conserver en mémoire non seulement la valeur de quelque chose mais aussi les détails de chaque modification affectant cette valeur. Le solde, qui représente la valeur courante du compte, est le résultat de toutes les entrées associées au compte

- Structure :



- ...

Les Data models patterns de Hay

- Présentation en 1996 de modèles de données E-A de haut niveau représentatifs des domaines d'activités des entreprises.
  - The Enterprise and Its World
  - Things of the Enterprise
  - Procedures and Activities
  - Contracts
  - Accounting
  - The Laboratory
  - Material Requirement Planning
  - Process Manufacturing

The Enterprise and Its World

Parties
Employee Assignments
Organizations
Addresses
Geographic Locations
...

Things of the Enterprise

Products and Product Types
Inventory
Structure
...

Procedures and Activities

Dividing Activities
Work Orders
labor Usage
Actual Asset Usage
Kinds of Works Orders
  Maintenance Works Orders
  Production Orders
  Projects
...

Contracts

Purchase Orders and Sales Orders
User Specifications
Contract Roles
Employment Contracts
Marketing Regions and Districts
Deliveries Of Products and Services
...

Accounting

Basic Bookkeeping
- Revenue
- Expense
- Investment
- ...
- Cost of Good Sold
- ...

Time and Budgets

The Laboratory

Samples, Tests and Observations

...

Tests as Activities

Material Requirement Planning

Planning Finished Products

...

The Manufacturing Planning Model

The Planning Model

Process Manufacturing

Structure and Fluid paths

Flows

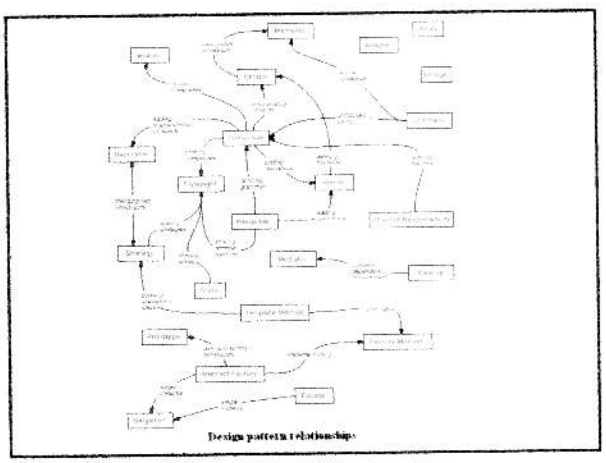
Processes

Monitoring Processes
- Tags and Measuring Points
- The Laboratory
- ...

Les patterns pour la conception

Les «Design Patterns» du GoF

- Catalogue de patterns de conception défini par [Gamma 95]
- Tentative d'établissement de liens entre patterns pour fournir des éléments cohérents d'architecture
- (GoF) Design pattern = Quadruplet (pattern name, problem, solution, consequences)



Design pattern relationships

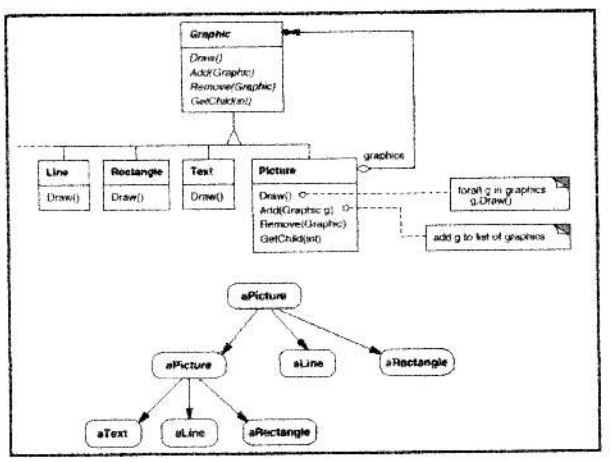
GoF Pattern

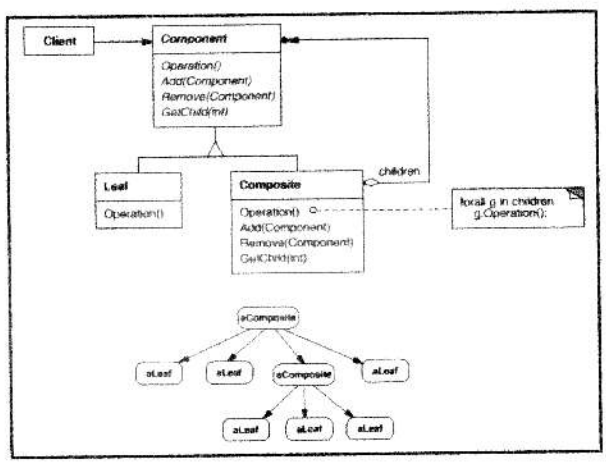
pattern name, problem, solution, consequences

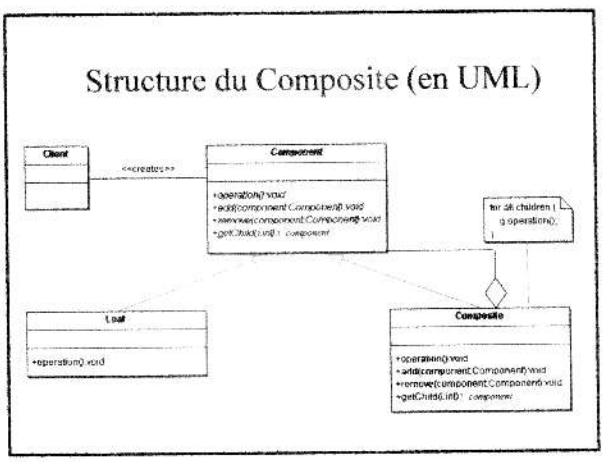
- pattern name : nom du pattern en clair
- problem : description des conditions d'application. Contient la description du contexte
- solution : description des éléments (objets, relations, responsabilités, collaborations) permettant de concevoir la solution au problème
- consequences : description des résultats de l'application du pattern sur le système. Contient la description des effets induits par cette application

Exemple de pattern de conception de GoF : le pattern Composite

- Intention :
  - Composition d'objets en des structures arborescentes pour représenter des hiérarchies composant/composé.
  - Traitement unique des objets individuels et de leurs combinaisons.
- Motivation :
  - Les applications graphiques (éditeurs de dessin, systèmes de représentation schématique) permettent à l'utilisateur de construire des diagrammes complexes à partir de simples composants.
  - Ce pattern décrit l'utilisation de la composition récursive.
- Indication d'utilisation :
  - Représenter des hiérarchies de l'individu à l'ensemble
  - Traiter uniformément tous les objets de la structure composite







- Constituants :
  - Composant : il déclare l'interface des objets entrant dans la composition. Il implémente le comportement par défaut de l'interface commune à toutes les classes. Il déclare une interface pour accéder à ses composants enfants et les gérer.
  - Feuille : elle définit le comportement d'objets primitifs dans la composition.
  - Composite : Il définit le comportement des composants dotés d'enfants. Il stocke les composants enfants. Il implémente les opérations liées aux enfants dans l'interface Composant.
  - Client : il manipule les objets de la composition à l'aide de l'interface Composant.

- Collaborations :
  - Les clients utilisent l'interface de la classe Composant pour manipuler les objets de la structure. Si l'objet est une Feuille, la requête est traitée directement. Si c'est un Composite, il transfère la requête à ses composants.
- Conséquences :
  - Définition de hiérarchie de classes.
  - Traitement à l'identique des structures composites et des objets individuels.
  - Facilité d'ajouter de nouveaux composants.

- Implémentation :
  Plusieurs points de vue à prendre en compte (gestion explicite des parents, partage de composants, ...)
- Exemples de code :
  ...
- Utilisations remarquables :
  MVC, ET++
- Modèles apparentés :
  Chaînes de Responsabilité, Décorateur, Poids Mouche, Itérateur, Visiteur

Les «Design Patterns» du Siemens Group

- [Buschmann 96] définit 14 sections :

  (Name, Also known as , Example, Context, Problem , Solution , Structure , Dynamics, Implementation, Example resolved, Variants, Knows uses, Consequences ,See also)

- Name : nom du pattern
- Also known as : synonymes du pattern décrit
- Example : un exemple du monde réel montrant l'existence du problème et le besoin de création de ce pattern.
- Context : les situations dans lesquelles le pattern peut être appliqué.
- Problem : le problème résolu par le pattern, faisant apparaitre ses forces et ses limites.
- Solution : le principe de solution fondamental choisi. Chaque composant participant au pattern est décrit à l'aide des cartes CRC.
- Structure : spécification détaillée des aspects structurels du pattern (utilisation du "Class Diagram" d'OMT)

- Dynamics : scénarios typiques du comportement du pattern, illustré à l'aide de OMSC (Object Message Sequence Charts).
- Implementation : guide pour l'implémentation du pattern. Ce ne sont que des suggestions. L'illustration des implémentations est faite en C++, Smalltalk et Java.
- Example resolved : discussion sur un (des) aspects important(s) pour résoudre le problème, aspect(s) pas encore couvert(s) dans les sections Solution, Structure, Dynamics, Implementation.
- Variants : description brève de variantes ou de spécialisation du pattern.
- Knows uses : cas d'utilisation du pattern référencés et tirés de systèmes existant.

- Consequences : les bénéfices fournis par le pattern et ses limites potentielles.
- See also : références aux patterns résolvant des problèmes similaires, et (ou) aux patterns pouvant aider à raffiner celui décrit.

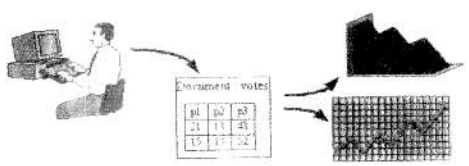
Exemple de pattern du Siemens Group : PAC: Presentation-Abstraction-Control

- Ce pattern définit une structure pour des logiciels interactifs basée sur une hiérarchie d'agents copératifs. Le modèle PAC structure l'architecture d'un système interactif en trois constituants : la présentation, l'abstraction et le contrôle.

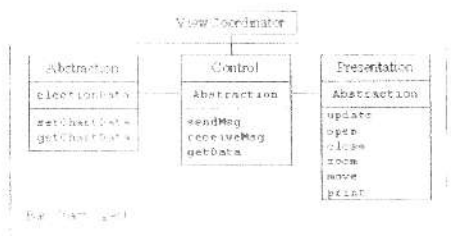
- La présentation définit l'image du système, c'est-à-dire son comportement en entrée comme en sortie vis-à-vis de l'utilisateur.
- L'abstraction désigne les concepts et les fonctions du système.
- Le contrôle maintient la cohérence entre les facettes présentation et abstraction.

- Exemple : Considérons une application pour gérer les élections. Cette application posséde un tableur pour la saisie des données et des graphiques pour représenter les données. L'utilisateur interagit avec le logiciel via une interface graphique.

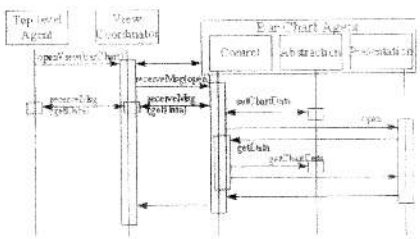
Exemple PAC



Exemple PAC



Exemple PAC



Les patterns de programmation ou idiomes

Les idiomes de Coplien pour C++

- C++ est très riche en idiomes car c'est un langage de bas niveau.
- Principaux domaines : la gestion de la mémoire, la réification d'algorithmes, ...
- Un exemple d'idiome : la forme canonique d'une classe
- A utiliser (au moins) dès que l'on souhaite supporter l'affectation ou le passage par valeur et que l'objet contient des références à d'autres objets.

- La "forme canonique" d'une classe X

un constructeur par défaut
X::X ()

un copy-constructor
X::X (const X&)

un opérateur d'affectation
X::operator= (const X&)

un destructeur
X::~X ( )

Les idiomes de Beck pour Smalltalk

- Composed Method
- Constructor Method
- Constructor Parameter Method
- Shortcut Constructor Method
- Conversion
- Converted Method
- Converted Constructor Method
- Query Method
- Comparing Method
- Reversing Method
- Method Object
- Execute Around Method
- Debug Printing Method
- Method Comment
- Message
- Choosing Message
- Decomposing Message
- Intention Revealing Message
- Intention Revealing Selector
- Dispatched Interpretation
- Double Dispatch
- Mediating Protocol
- ...

Conclusion

La réutilisation est un domaine en plein essor.

Première capitalisation des compétences
- des équipes de programmeurs (via les "Idiomes"),
- d'architectes de systèmes (via les "Architectural Patterns", «Design Patterns")
- et d' Analyste-Concepteurs (via les "Analysis Patterns")

dans le domaine du logiciel mais aussi dans l'ingénierie des systèmes d'information.

Des activités nouvelles apparaissent:
par exemple le «Pattern Mining» (Coplien).

Retour sur l'exemple

- Un jeu de dés
- Le joueur lance 10 x 2 dés
- Si le total fait 7, il marque 10 points à son score
- En fin de partie, son score est inscrit dans le tableau des scores.

Conception

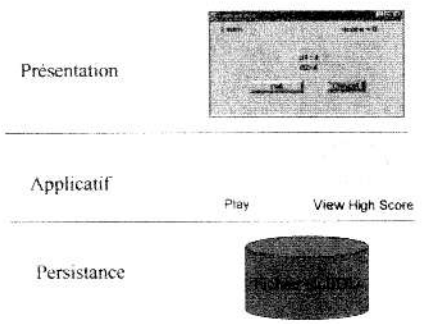
Gérer la partie interface graphique

Gérer la persistance des « highscores »

⇒ Définir une architecture

⇒ Rajouter les classes techniques permettant d 'implanter cette architecture !

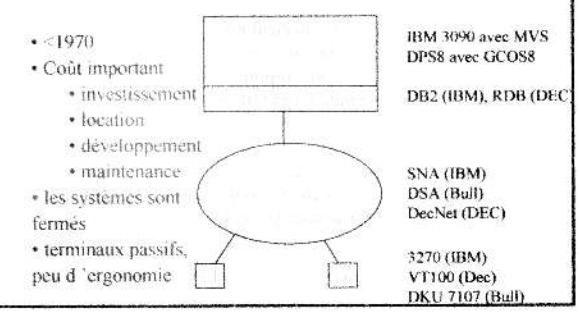
Conception de l'architecture



Décomposition de l 'architecture des systèmes d 'informations en niveaux

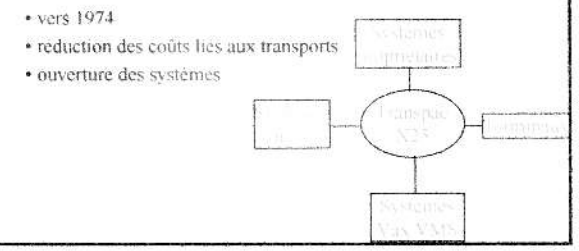
- Les systèmes d 'informations doivent pouvoir gérer
  - un grand nombre d 'informations complexes
  - et de nombreux utilisateurs.
- Cela a été traité différemment selon les époques et les technologies existantes.

Architecture des SI de type mainframe



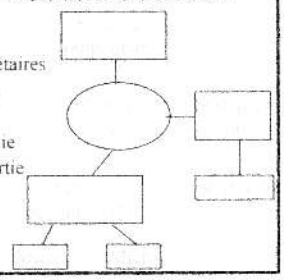
Evolution des architectures originelles

- Arrivée des réseaux partagés (Transpac)



Evolution des architectures originelles

- Arrivée des SGBD Relationnels et des micro-ordinateurs

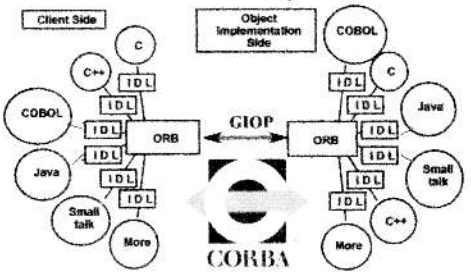


- alternative aux systèmes propriétaires
- les terminaux deviennent actifs
- on parle de « poste client »
  - amélioration de l 'ergonomie
  - capacité de traitement répartie
  - applications bureautiques

Conséquences

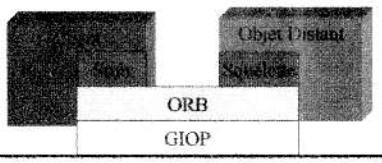
- Passage d 'une architecture à une architecture
  - mono-technologie → multi-technologie
  - propriétaire → multi-vendeur
  - fermée → ouverte
  - → (standardisée ?)
- Micro-ordinateur
  - Un des centres du SI
  - accès transparent aux BDs distantes
- Réduction des coûts
  - investissement
  - Maintenance
- Nouveau coût
  - intégration
- Nouveau Marché
  - middleware

Un exemple de middleware: CORBA (Common Object Request Broker Architecture)



CORBA
Spécifications définies par l 'OMG (Object Management Group)

- Indépendant des langages
- Langage de définition d 'interface : IDL
- Middleware : ORB (Object request Broker)
- Transport : IIOP = GIOP sur TCP/IP



Un autre exemple de middleware: les Web Services

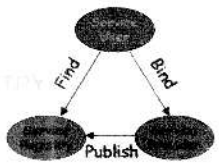
Collection de technologies permettant aux opérations maison d'être accessibles via Internet.

Caractéristiques:

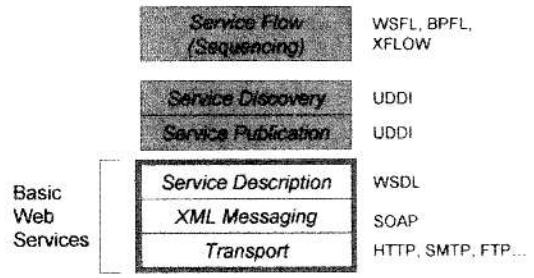
- Basés sur XML
- Basés sur l'envoi de Message
- Indépendant de tout Langage
- Dynamiquement localisés
- Accédés à travers internet
- Faiblement couplés
- Utilisent des protocoles standards
- NOT "self-describing".

Web Services Components

- Service Provider
  - Offers services to clients
  - Advertises the existence of these services by in a
- Service Registry
  - Supports publishing and location services
  - Similar to Internet's DNS
- Service User/Requestor
  - a service using the Registry
  - to a service on the Service Provider



Web Services Stack



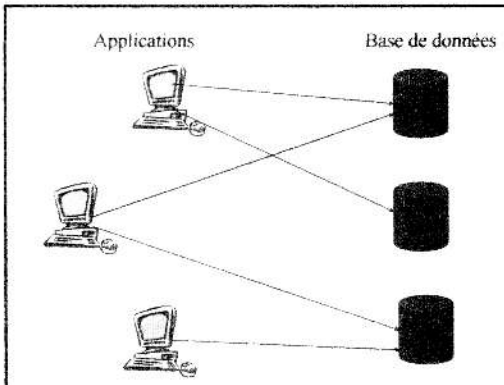
Evolution des architectures originelles

- Les SI doivent être flexibles pour accompagner les changements.
- Pour une gestion plus simple les SI sont décomposés en niveau :
  - L 'architecture deux niveaux est la plus répandue mais elle est limitée puisque l 'interface utilisateur est liée aux données.
  - L 'architecture trois niveaux a été mise en place dans les années 70 pour répondre à ce problème.

L 'architecture deux niveaux (1)

Le système est divisé en 2 niveaux:

- La base de données
- Plusieurs applications, qui accèdent à la base de données.



L 'architecture deux niveaux (2)

Avantages

- La plupart des organisations ont des données demandant un contrôle centralisé et une maintenance uniforme

L 'architecture deux niveaux (3)

Inconvénients

- Les applications sont développées pour des utilisations spécifiques
- Difficultés à changer la structure de la base de données
- La base de données ne rend toujours pas bien compte de la réalité de l 'entreprise
- Les données sont souvent réparties sur plusieurs bases de données.

L 'architecture trois niveaux (1)

Ce type d 'architecture divise le système en 3 niveaux:

- Les applications
- Le domaine
- La base de données

Le domaine peut être défini comme un ensemble de librairies

L 'architecture trois niveaux (2)

- L 'ajout de ce niveau domaine permet:

  - de représenter la véritable sémantique de l 'entreprise

  - de fournir une séparation entre les applications et la base données

  - de réutiliser les différents objets définis à ce niveau

  ⇒ structure et emplacement de la BD peuvent être modifiés sans répercussion pour les applications existantes

L 'architecture trois niveaux (3)

L 'emplacement du niveau domaine

- On peut :
  - 1) rester dans un environnement «physique » Client/Serveur, et placer le domaine sur le client,
  - 2) ou bien passer à une architecture « physique » 3 niveaux et placer le niveau domaine à part (sur une ou plusieurs machines).

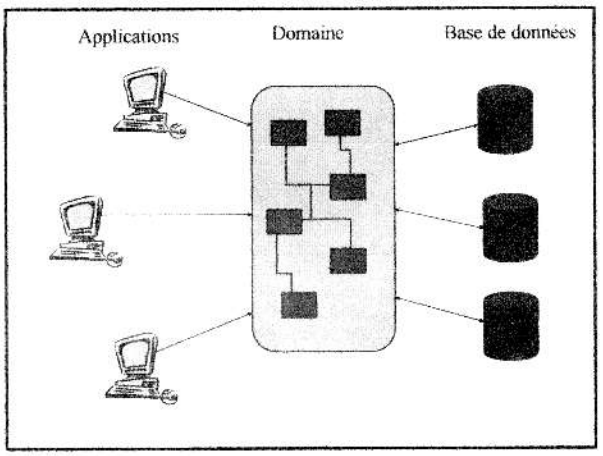
L 'architecture trois niveaux (4)

- Le choix de placer le domaine sur le client peut poser problème:

  - ce dernier aura beaucoup de choses à gérer et devra être puissant

  - chaque changement devra être répercuté sur le client

- Le choix d 'ajouter un niveau domaine :
  - limite le nombre de machines et de systèmes à maintenir



Présentation et logique d'application (1)

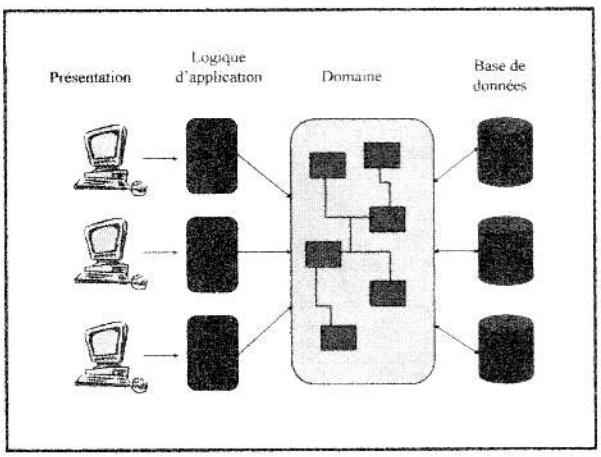
Séparation du niveau *application* en deux :

* *la présentation :*

elle gère uniquement *l'interface utilisateur* à l'aide d'informations fournies par la *logique d'application*, indépendamment du *domaine*

* *la logique d'application :*

- elle accède au *domaine*, prépare tout pour le niveau *présentation*
- elle peut être développée par une série de façades gérant la conversion de types et permettant de simplifier un modèle compliqué.



Avantages et inconvénients de la séparation

- Inconvénients :
  - travail supplémentaire pour construire les différents niveaux
  - parfois une baisse de performance
  - la *présentation* doit être définie avant la construction de la *façade*
- Avantages :
  - l'*interface* et la *façade* peuvent être développées séparément
  - l'*interface* devient indépendante du *domaine* car la *façade* s'occupe des interactions
  - plusieurs *interfaces* sont possibles pour une même *façade*
  - les tests de l'*interface* et de la *façade* sont séparés

Façades et environnement client/serveur

- On place une *façade* à la fois sur le client et sur le serveur
- La *façade client* s'occupe :
  - des opérations pour la *présentation*
  - de la communication avec la *façade serveur*
  - du chargement et de la sauvegarde d'opérations
- *La façade serveur s'occupe :*
  - des interactions avec le *domaine*
  - de la communication avec la *façade client*
  - du chargement et de la sauvegarde d'opérations
- *Principal avantage :* en cas de demandes multiples au serveur, les *façades* permettent à l'aide de méthodes de transférer un lot de demandes au serveur. Ceci évite une baisse de performance.

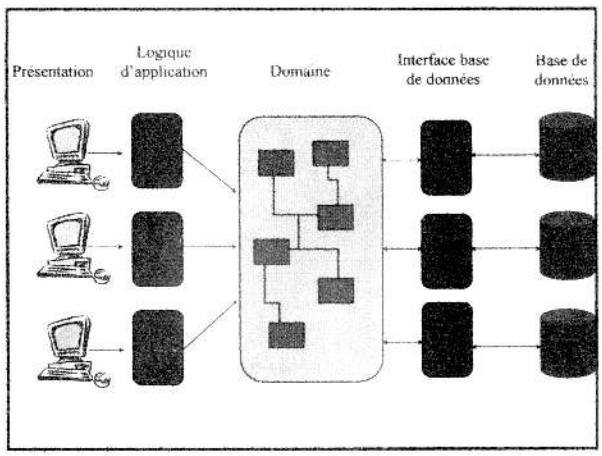
Interaction avec une base de données directement liée au domaine

Cette solution est peu utilisée car elle pose de nombreux problèmes :

- les classes du *domaine* devraient pouvoir se construire elles-mêmes à partir de la *base de données* pour représenter la réalité de l'entreprise
- les extractions de données peuvent être critiques si elles se font sur plusieurs *bases de données* à la fois

Interaction avec une base de données : Niveau Interface base de données

- Ce niveau est similaire à la *logique d'application* : il permet la séparation entre le *domaine* et la *base de données*.
- Il s'occupe du chargement du *domaine* avec les données de la *base*, et de la mise à jour de la *base de données* quand un changement survient au niveau du *domaine*.



SYNTHESE (1)

- L 'architecture deux niveaux est bien adaptée pour les petits systèmes.
- L 'architecture trois niveaux permet de nombreuses améliorations et est bien supportée par les technologies objets.
- La séparation du niveau application permet :
  - la réutilisation de la logique d 'application pour différentes interfaces.
  - la facilité de tests.
  - la gestion des performances pour des systèmes client/serveur.
  - des équipes de développement spécialisées
- Le niveau interface base de données est particulièrement utile pour des sources de données complexes.

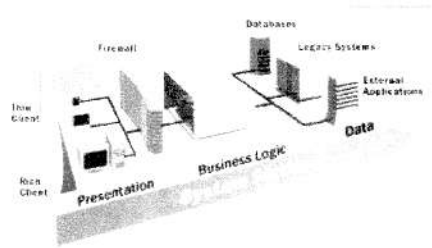
SYNTHESE (2)

- **Niveau domaine :**
  - modèle direct des objets de l 'entreprise.
  - Indépendant des applications et des sources de données.
- **Niveau logique d 'application :**
  - simplification du domaine pour une application
  - ne contient aucun code de l 'interface utilisateur mais fournit un ensemble de façades du domaine pour l 'interface utilisateur.
  - *Convertit les types du domaine en types exigés* par la présentation.

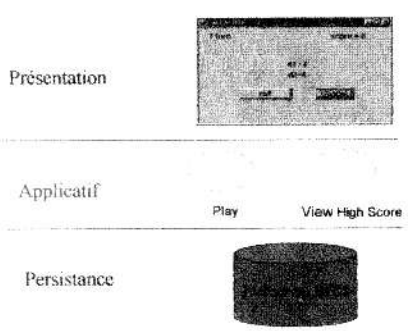
SYNTHESE (3)

- **Niveau présentation :**
  - formate les informations de la logique d 'application.
  - S 'occupe uniquement de l 'interface utilisateur.
  - N 'a aucune connaissance du domaine.
- **Niveau interface base de données :**
  - responsable de l 'échange d 'informations entre les sources de données et le domaine
  - fournit un ' interface broker ' simple au domaine pour émettre les demandes.
  - Communique avec le domaine et les sources de données.
  - Divisée en sous-systèmes en fonction du type de sources de données utilisé.

Systèmes d'Information Distribués: Architecture multi-niveaux



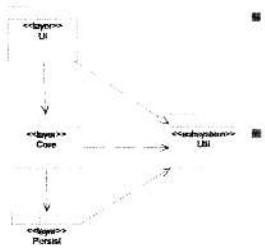
Architecture en couches



Architecture en couches...

- Une architecture possible, d 'autres existent (voir « A system of patterns » de Buschmann)
- Les couches doivent être le plus indépendantes possible
- « Découpler » les couches en s 'appuyant sur interfaces + classes abstraites (design patterns)

Découpage en « package » logiques

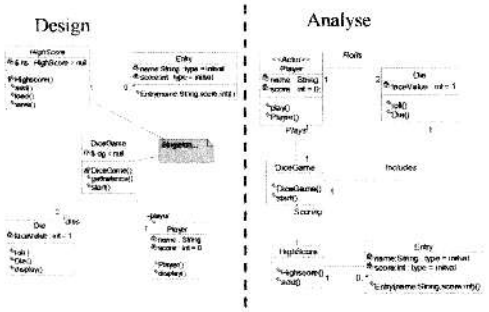


- Mapper l 'architecture sur des packages « layer »
- Exprimer les dépendances

Layer « core »

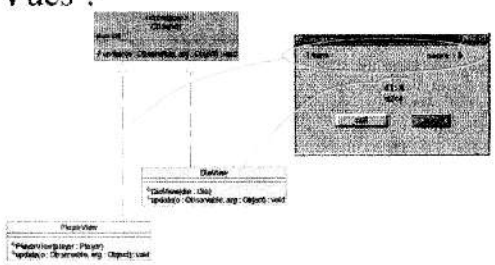
- Les classes représentant la logique de notre application.
- En fait, les classes d 'analyses revisitées en vue de la réalisation

Core « Layer »:1er diagramme

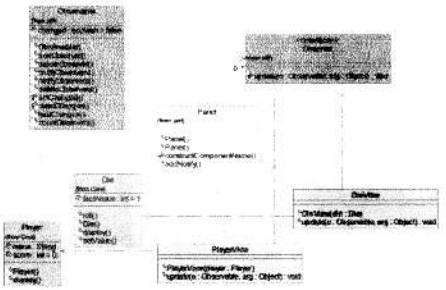


Découplage interface graphique:MVC

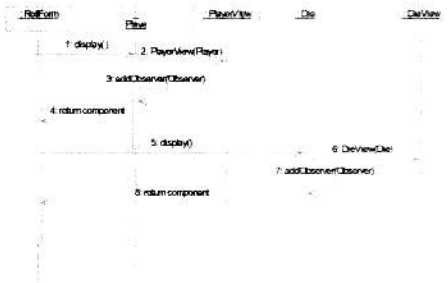
Vues ?



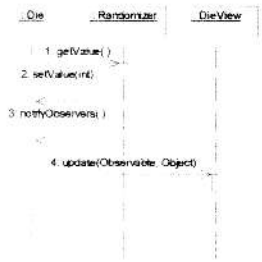
Attention...



MVC en action: 1) mise en place

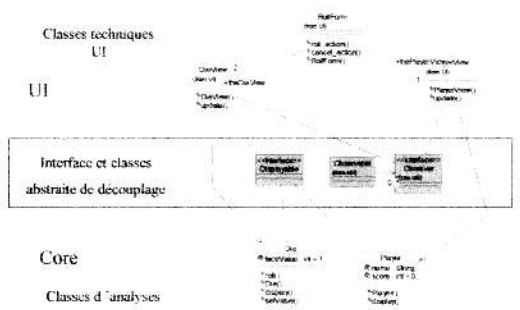


MVC en action: 2) changement d 'état



Propagation des changements d 'états vers les objets graphiques

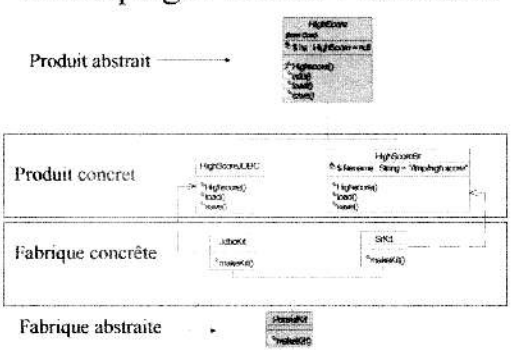
Architecture en couche...



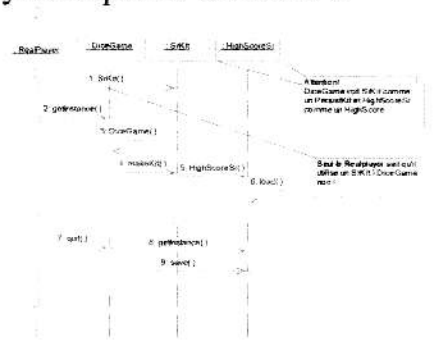
Layer « Persist »

- Classes techniques de persistances
- Assurer l 'indépendance Core/Persist
  - pouvoir changer de « persistent engine »
- Par exemple:
  - Persistance par « Serialisation »
  - Persistance via une base de données relationnelle (JDBC).

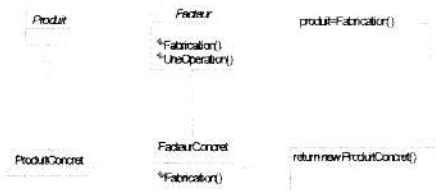
Découplage : Pattern Fabrication



Dynamique de « Persist »



Structure



...

- A vous de terminer la conception et le codage d'une solution au jeu de dés utilisant à bon escient les différents patrons présentés (et d'autres ?)

Design patterns de Gamma : Classification et utilisation

	Créateurs	**Structuraux**	**Comportementaux**
Classe	Factory Method	Adapter(class)	Interpreter Template Method
Objet	Abstract Factory Builder Prototype Singleton	Adapter(objet) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Resp. Command Iterator Mediator Memento Observer State Strategy Visitor

Quelques exemples de

- Creational Patterns
- Structural Patterns
- Behavioural Patterns

Creational Patterns

Patrons de création :

Abstraire le processus d'instanciation.

Rendre indépendant de la façon dont les objets sont créés, composés, assemblés, représentés.

Encapsuler la connaissance de la classe concrète qui instancie.

Cacher ce qui est créé, qui crée, comment et quand.

Principes

AbstractFactory : on passe un paramètre à la création qui définit ce qu'on va créer

Builder : on passe en paramètre un objet qui sait construire l'objet à partir d'une description

FactoryMethod : la classe sollicité appelle des méthodes abstraites ...il suffit de sous-classer

Prototype : des prototypes variés existent qui sont copiés et assemblés

Singleton : instance unique

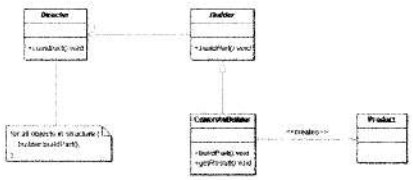
Utilisation du Builder

On utilise le Builder lorsque :

l'algorithme pour créer un objet doit être indépendant des parties qui le compose et de la façon de les assembler

le processus de construction permet différentes représentations de l'objet construit

Structure du Builder (en UML)



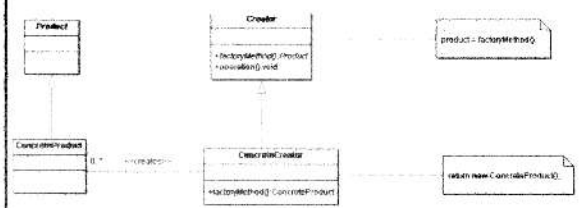
Utilisation du FactoryMethod

On utilise le FactoryMethod lorsque :

une classe ne peut anticiper la classe de l'objet qu'elle doit construire

une classe délègue la responsabilité de la création à ses sous-classes, tout en concentrant l'interface dans une classe unique

Structure du FactoryMethod (en UML)



Structural Patterns

Patrons de structure :

Comment les objets sont assemblés

Les patrons sont complémentaires les uns des autres

Principes

Adapter : rendre un objet conformant à un autre

Bridge : pour lier une abstraction à une implantation

Composite : basé sur des objets primitifs et composants

Decorator : ajoute des services à un objet

Facade : cache une structure complexe

Flyweight : petits objets destinés à être partagés

Proxy : un objet en masque un autre

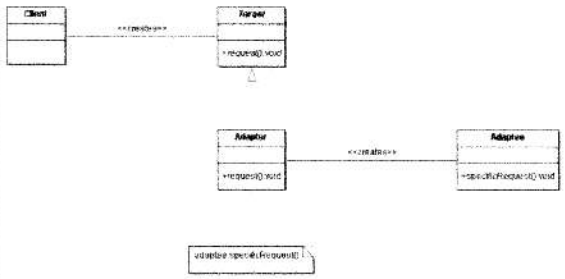
Utilisation de l'Adapter

On utilise l'Adapter lorsque on veut utiliser :

une classe existante dont l'interface ne convient pas

plusieurs sous-classes mais il est est coûteux de redéfinir l'interface de chaque sous-classe en les sous-classant. Un adapter peut adapter l'interface au niveau du parent.

Structure de l'Adapter (en UML)



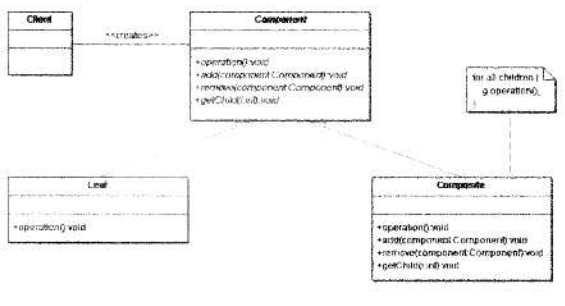
Utilisation du Composite

On utilise Composite lorsque on veut :

représenter une hiérarchie d'objets

ignorer la différence entre un composant simple et un composant en contenant d'autres (interface uniforme)

Structure du Composite (en UML)



Utilisation du Proxy

On utilise le Proxy lorsqu'on veut référencer un objet par un moyen plus complexe qu'un pointeur...

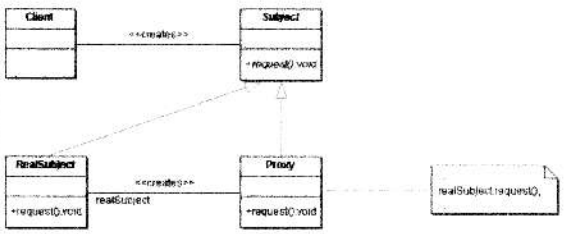
remote proxy : ambassadeur

protection proxy : contrôle d'accès

référence intelligente

persistence
comptage de référence

Structure du Proxy (en UML)



Behavioural Patterns

Patrons de comportement pour décrire :

des algorithmes

des comportements entre objets

des formes de communication entre objet

Principes

Chain of Responsibility

Command

Interpreter

Iterator

Mediator

Memento

Observer

State

Strategy

Template Method

Visitor

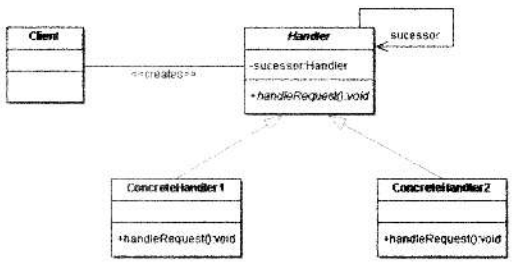
Utilisation de la Chain of Responsibility

On utilise Chain of Responsibility lorsque :

plus d'un objet peut traiter une requête, et il n'est pas connu a priori

l'ensemble des objets pouvant traiter une requête est construit dynamiquement

Structure de la Chain of Responsibility (en UML)



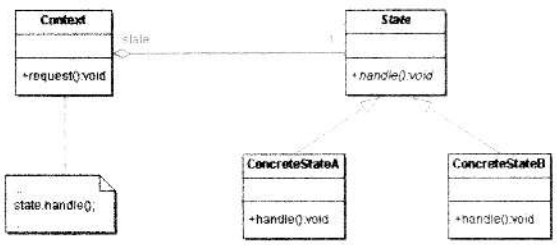
Utilisation de State

On utilise State lorsque :

Le comportement d'un objet dépend de son état, qui change à l'exécution

Les opérations sont constituées de parties conditionnelles de grande taille (case)

Structure de State (en UML)



Utilisation de Strategy

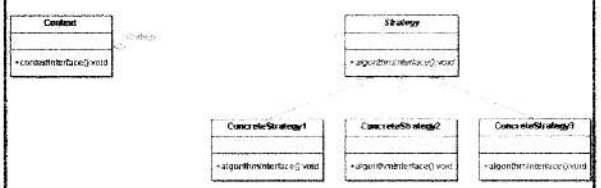
On utilise Strategy lorsque :

de nombreuses classes associées ne diffèrent que par leur comportement. Stratégie offre un moyen de configurer une classe avec un comportement parmi plusieurs.

on a besoin de plusieurs variantes d'algorithme.

un algorithme utilise des données que les clients ne doivent pas connaître.

Structure de Strategy (en UML)



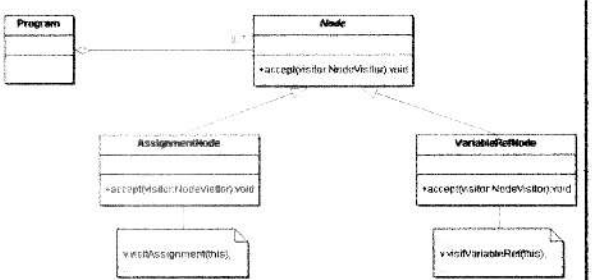
Utilisation de Visitor

On utilise Visitor lorsque :

une structure d'objets contient de nombreuses classes avec des interfaces différentes et on veut appliquer des operations diverses sur ces objets.

les structures sont assez stables, et les opérations sur leurs objets évolutives.

Structure de Visitor (en UML)



Conclusion

Les patterns proposent des abstractions qui n'apparaissent pas "naturellement" en observant le monde réel :

Composite : permet de traiter uniformément une structure d'objets hétérogènes

Strategy : permet d'implanter une famille d'algorithmes interchangeables

Ils améliorent la flexibilité et la réutilisation (capitalisation d'expériences)