GL - Patterns et Frameworks

Approches pour la réutilisation
Frameworks
Patterns

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Deux approches pour l'extensibilité

- L'approche **boite blanche** (transparente) pour obtenir l'extensibilité se base sur les concepts des langages à objets comme l'héritage et la liaison dynamique. Les fonctionnalités existantes sont réutilisées et étendues par l'héritage des classes de base du framework et la surcharge des méthodes prédéfinies.
- L'approche **boite noire** supporte l'extensibilité par la définition des interfaces des composants qui peuvent être enfichés dans le framework par la composition d'objets. Les fonctionnalités existantes sont réutilisées par la définition des composants qui sont conformes aux interfaces, l'intégration de ces composants dans le framework.

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Boite blanche et boite noire

- L'approche **boite blanche** nécessite une connaissance approfondie de la structure interne du framework. Les systèmes obtenus ainsi sont très intimement couplés aux détails de la hiérarchie d'héritage.
- L'approche **boite noire** est basée sur la composition et délégation plutôt que l'héritage. Il en résulte une réutilisation plus facile et une plus grande extensibilité. Ces frameworks sont par contre plus difficiles à développer car ils demandent aux développeurs une définition anticipée d'interfaces et de méthodes pour couvrir les cas d'utilisation potentiels.

BTD/GL/P&F

3

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Object-oriented Application Framework (1)

- Les performances des machines et la puissance des réseaux ont considérablement augmentées dans la dernière décennie, tandis que la conception et la réalisation des logiciels complexes sont restées coûteuses et sources d'erreurs.
- Le coût et l'effort à fournir sont en grande partie dus à la découverte perpétuelle et la réinventions de concepts de base et de composants.
- En particulier, la hétérogénéité croissante des architectures matérielles, la diversité des systèmes d'exploitation et de plates-formes de communication rendent difficile la construction d'applications correctes, portables, performantes et peu coûteuses à partir de rien.

BTD/GL/P&F

4

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Object-oriented Application Framework (2)

- **Object-oriented application frameworks** constituent une approche prometteuse pour proposer des conceptions et des implémentations de logiciels dans le but de réduire le coût et d'augmenter la qualité du logiciel.
- Le framework (cadre) est une application semi-finie qui peut être spécialisée pour produire une application adaptée.
- En contraste avec l'approche objet classique, dans laquelle la réutilisation était basée sur une bibliothèque de classes, les cadres (frameworks) visent des entités applicatives particulières (comme la téléphonie mobile) et des domaines applicatifs (IHM embarquée du pilote).

BTD/GL/P&F

5

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Object-oriented Application Framework (3)

- Les frameworks comme MacApp, AT++, Interviews, ACE, Microsoft MFC et DCOM, RMI de JavaSoft et les implémentations de CORBA jouent un rôle croissant dans le développement actuel des logiciels.
- Le gain initial des frameworks provient de la modularité, la réutilisabilité, l'extensibilité et de l'inversion du contrôle qu'ils fournissent aux développeurs.
- La modularité est accrue par l'encapsulation des détails d'implémentation derrière des interfaces stables.

BTD/GL/P&F

6

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Object-oriented Application Framework (4)

- La modularité du framework aide à augmenter la qualité du logiciel par la localisation de l'impact des changements de la conception et de l'implémentation ce qui réduit l'effort demandé pour comprendre et maintenir un logiciel existant.
- Les interfaces stables proposées par des frameworks augmentent la réutilisation par la définition de composants génériques qui peuvent être réappliquées pour créer de nouvelles applications.
- La réutilisation du framework sert de levier à l'effort fourni lors de l'acquisition des connaissances en évitant la nouvelle conception et la revalidation des solutions de base pour des besoins récurants.

BTD/GL/P&F

7

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Object-oriented Application Framework (5)

- La réutilisation des composants du framework conduit à l'augmentation substantielle de la productivité et à une meilleure qualité, performance, fiabilité et interopérabilité.
- L'extensibilité du framework est obtenue grâce aux méthodes qui peuvent être spécialisées sans remettre en cause la stabilité des interfaces. Ces méthodes découplent la stabilité des interfaces de la variation nécessaire lors de l'instanciation de l'application dans un contexte particulier.
- L'extensibilité du framework est essentielle pour permettre adaptation de nouveaux services et attributs.

BTD/GL/P&F

8

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Object-oriented Application Framework (6)

- L'architecture d'exécution du framework est caractérisée par l'inversion du contrôle. Cette architecture permet aux éléments canoniques de l'application d'être spécialisées par des objets gestionnaires d'événements qui sont évoqués par le mécanisme du framework qui assure la distribution réactive.
- Quand un événement apparaît, le distributeur du framework réagit en évoquant les méthodes des objets gestionnaires qui effectuent des traitements spécifiques. L'inversion du contrôle permet au framework (et non à chaque application) de déterminer quel est l'ensemble de méthodes spécifiques qui doivent être activées en réponse aux événements externes (comme des messages de fenêtres venant des utilisateurs).

BTD/GL/P&F 9

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Des frameworks les plus courants

- Interfaces graphiques
- ORB (object request broker) qui facilite la communication entre objets locaux et distants.

BTD/GL/P&F 10

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Typologie des frameworks

- **Framework d'infrastructure des systèmes** pour obtenir des systèmes portables et performants comme des systèmes d'exploitation et de communication, des framework d'interfaces utilisateurs et outils de traitement de langages. Ces frameworks sont principalement utilisés de façon interne dans les équipes de développement et ne sont pas diffusés vers les clients.
- **Frameworks de middleware** sont utilisés généralement pour bâtir des applications distribuées en intégrant des composants. Ils augmentent la modularité, la réutilisation et par là la fiabilité d'infrastructure à comportement semblables dans différents environnements distribués.
- **Frameworks d'applications** pour les entreprises couvrent un ensemble croissant de domaines d'applications comme des finances , la productique, les télécommunications,… En comparaison avec les deux autres types de frameworks ces derniers sont à la fois beaucoup plus coûteux à développer ou à acquérir, mais génèrent également un retour sur investissement car ils supportent le développement direct d'applications finalisées pour les clients. Ceci contraste avec les deux premiers qui ne concernent qu'indirectement les clients finaux.

BTD/GL/P&F

11

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Avantages et inconvénients (1)

- Les frameworks en conjonction avec des patterns et des bibliothèques de classes peuvent augmenter de façon significative la qualité et réduire l'effort de développement.
- Pour cela il faut savoir que : Si la difficulté de développement d'un logiciel complexe, celle du framework correspondant le sera également.
- La connaissance de développement reste encore et toujours dans les têtes des experts. L'apprentissage de l'utilisation d'un framework est longue (6-12 mois) mais l'investissement devrait s'amortir sur plusieurs projets.

BTD/GL/P&F

12

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Avantages et inconvénients (2)

- Le développement d'applications devrait progressivement intégrer plusieurs frameworks (IHM, Communication,…). Malheureusement des frameworks ont été principalement conçu dans l'optique d'évolution interne que dans celle d'intégration avec d'autres frameworks.
- Evolution d'applications peut conduire à l'évolution du framework.
- Maintenance du framework : fonctionnelle, non-fonctionnelle, généralisation, est une activité coûteuse.

BTD/GL/P&F 13

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Positionnement (1)

- Un **framework** est une architecture réutilisable de l'ensemble ou d'une partie d'un système représenté par un ensemble de classes abstraites et la façon dont leurs instances interagissent.
- Un **framework** est un squelette d'une application qui peut être adapté par le développeur de l'application.
- Un **pattern** décrit un problème à résoudre, une solution et le contexte dans lequel il travaille.
- Les **patterns** ont pour but de décrire des solutions récurrentes.
- Un **framework** contient en général plusieurs **patterns**. Les patterns sont plus abstraits que les frameworks.

BTD/GL/P&F 14

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Positionnement (2)

- Des patterns de conception sont des micro-éléments architecturaux d'un framework.
- Les frameworks sont au centre des techniques de réutilisation. Ils sont plus abstraits et plus flexibles que des composants, mais plus concrets et plus faciles à réutiliser que les patterns de conception.
- Les frameworks réutilisent à la fois la conception et le code.
- Les patterns sont illustrés par des programmes, mais les frameworks sont les programmes.
-

BTD/GL/P&F

15

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Design Patterns

- Chaque pattern décrit un problème qui apparaît très souvent dans notre environnement et qui constitue un corps de solution à ce problème de façon que cette solution puisse être utilisée des millions de fois sans faire deux fois la même chose (Christopher Alexander 77).

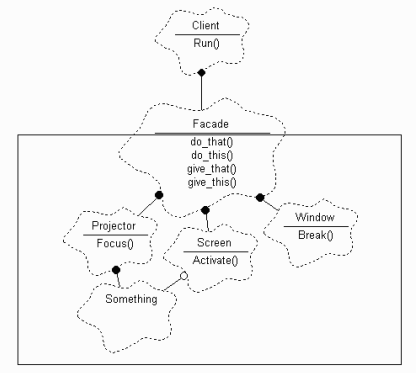
BTD/GL/P&F

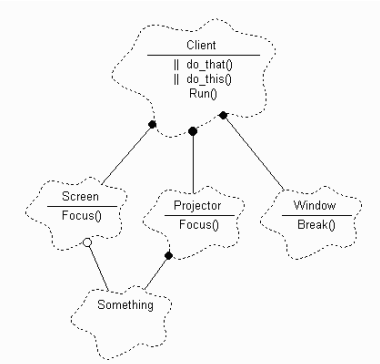
16

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exemple de pattern : Façade





BTD/GL/P&F

17

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Quatre éléments décrivent un pattern

- **Le nom** : identifie le problème de conception, la solution et les conséquences en quelques mots. Ce nom augmente immédiatement le vocabulaire de conception. Ce vocabulaire permet de communiquer et augmenter le niveau d'abstraction.
- **Le problème** : décrit quand on peut appliquer le pattern. Il présente le problème dans son contexte. Il peut préciser les spécificités du problème et des conditions d'application.
- **La solution** : décrit les éléments qui constituent la réponse, leurs relations, responsabilités et collaborations. On ne décrit pas une implémentation particulière, car un pattern peut être considéré comme un schéma qui peut être appliqué dans différentes situations. En réalité, un pattern constitue une description abstraite d'un problème de conception et indique comment une structuration générale de classes et d'objets le résout.
- **Les conséquences** : indiquent les résultats et les principes d'application du pattern. Ces conséquences aident à comprendre et à évaluer le pattern.

BTD/GL/P&F

18

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Format de description d'un pattern (1)

- **Nom et classification** : le nom prend place dans le vocabulaire de conception, la classification place le pattern dans l'ensemble.
- **Objectif** : décrit brièvement les quoi, pourquoi et limitations
- **Egalement connu sous** : donne d'autres noms pour ce pattern
- **Motivation** : décrit un scénario qui illustre le problème et comment une configuration de classes et d'objets résout ce problème. Ce scénario va aider dans des descriptions plus abstraites de certains patterns.
- **Utilisation** : donne des situations dans lesquelles le pattern peut être appliqué. Quels sont les exemples de mauvaise utilisation du pattern et comment reconnaître ses situations.

BTD/GL/P&F

19

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Format de description d'un pattern (2)

- **Structure :** donne une représentation graphique des classes du pattern en utilisant la notation basée sur UML et les diagrammes d'interaction pour exprimer la séquence d'appels et de collaborations entre objets.
- **Participants :** donne la liste des classes, objets intervenant dans la pattern et leur rôles.
- **Collaborations :** indique comment les participants collaborent pour remplir leurs rôles.
- **Conséquences :** indique comment le pattern remplit ses objectifs, quels sont les tendances et les résultats d'utilisation du pattern et quels éléments de la structure laisse-t-il évoluer librement.

BTD/GL/P&F

20

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Format de description d'un pattern (3)

- **Implémentation :** donne des principes et des astuces pour obtenir une bonne implémentation.
- **Code de principe :** les fragments de codes indiquent comment on peut implémenter le pattern.
- **Utilisations connues :** décrit les exemples d'utilisation du pattern dans des systèmes utilisés.
- **Patterns connexes :** indique des patterns semblables, leurs différences et compatibilités éventuelles.

BTD/GL/P&F 21

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Description

- **Un Design pattern c'est :**
  - la description d'un problème rémanent et d'une solution classique.
  - une solution au problème pouvant être réutilisée sans être toutefois identique dans ses emplois.
  - Un design pattern décrit une partie de la solution avec les relations avec les autres parties du système et la technique d'architecture logicielle.
  - Mais ce n'est pas une brique, car un pattern dépend de son environnement :
    - ✓ une règle, car un pattern ne s'applique pas mécaniquement
    - ✓ une méthode, car un pattern ne dirige pas la solution à prendre : c'est **la solution**

BTD/GL/P&F 22

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Avantages et désavantages

- **Avantages**
  - un vocabulaire commun
  - capitalisation de l'expérience
  - niveau d'abstraction élevée qui permet des architectures de haut niveau
  - réduire la complexité
  - guide/catalogue de solutions
- **Inconvénients**
  - effort de synthèse : reconnaître, abstraire
  - apprentissage, expérience
  - patterns se dissolvent en étant utilisés
  - nombre de patterns lequel convient ?
  - différents niveaux de patterns s'appuyant sur d'autres patterns

BTD/GL/P&F

23

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Vocabulaire

- **nom** : réunifie l'idée vocabulaire commun
- **problème** : quand appliquer la forme le contexte
- **solution** : éléments de la solution, relations, etc. pas de manière précise mais suggestive
- **conséquences** : résultats et compromis d'utilisation

BTD/GL/P&F

24

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Interactions entre formes et langage

– Influence du langage sur le pattern
  - ✓ langage implante les formes de bas niveau
  - ✓ certaines formes utilisent des concepts spécifiques à certains langages donc non indépendance
  - ✓ certaines formes conduisent à des implémentations compliquées

Influence pattern langage
  - ✓ capitalise la réflexion de programmation

– Application lors de la conception :
  - ✓ trouver les bons objets
  - ✓ bien choisir granularité des objets
  - ✓ spécifier interface des objets
  - ✓ spécifier implantation des objets
  - ✓ concevoir pour l'évolution

BTD/GL/P&F

25

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Les différents patterns

– ***Creational patterns (formes de création)***

– ***Structural patterns (formes de structure)***

– ***Behavioural patterns (formes de comportement)***

BTD/GL/P&F

26

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Creational patterns (formes de création)

- **Abstraire le processus d'instanciation**
  - rendre indépendante la façon dont les objets sont créés utilisés implémentés
  - encapsuler la connaissance de la classe concrète qui instancie
  - cacher qui crée, ce qui crée et comment
- **Principes**
  - **abstract factory** : on passe un paramètre à la création qui défini ce que l'on va créer
  - **builder** : on passe en paramètre un objet qui sait construire l'objet à partir d'une description
  - **factory method** : la classe sollicitée appelle des méthodes abstraites, il suffit de sous classer
  - **prototype** : des prototypes variés existent qui sont copiés et utilisées
  - **singleton** : unique instance

BTD/GL/P&F

27

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Structural patterns (formes de structure)

- **Comment les objets sont complémentaires**
  - les patterns sont complémentaires les uns les autres
- **Principes**
  - **adapter** : rendre un objet conforme à un autre
  - **bridge** : pour lier une abstraction à une implémentation
  - **composite** : basé sur des objets primitifs et composants
  - **decorator** : ajoute des services à un objet
  - **facade** : cache une structure complexe
  - **flyweigth** : petits objets destinés à être partagés
  - **proxi** : un objet en cache un autre

BTD/GL/P&F

28

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Behavioural patterns (formes de comportement)

- **Des algorithmes**
  - des comportements entre objets
  - des formes de communication entre objets
- **Principes**
  - **Chain of responsability**
  - **Command**
  - **Interpreter**
  - **Iterator**
  - **Mediator**
  - **Memento**
  - **Observer**
  - **State**
  - **Strategy**
  - **template method**
  - **visitor**

BTD/GL/P&F

29

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Exemple : Le pattern Strategy

- *Le problème*
  - Lors de manipulations de tableaux, on a besoin de trouver une valeur de pivot.
  - De meilleurs résultats peuvent être obtenus en utilisant différents algorithmes de recherche du pivot pour différentes données.
  - On peut alors utiliser le pattern strategy.
- *But*
  - Définir différents algorithmes, les encapsuler et les rendre interchangeables.
  - Ce pattern résout les différents problèmes :
    - ✓ Comment modifier les règles de sélection dun pivot sans modifier l'algorithme principal (quick sort) ?
    - ✓ Faire en sorte que les algorithmes soient compatibles avec l'interface sans tenir compte de l'implémentation.

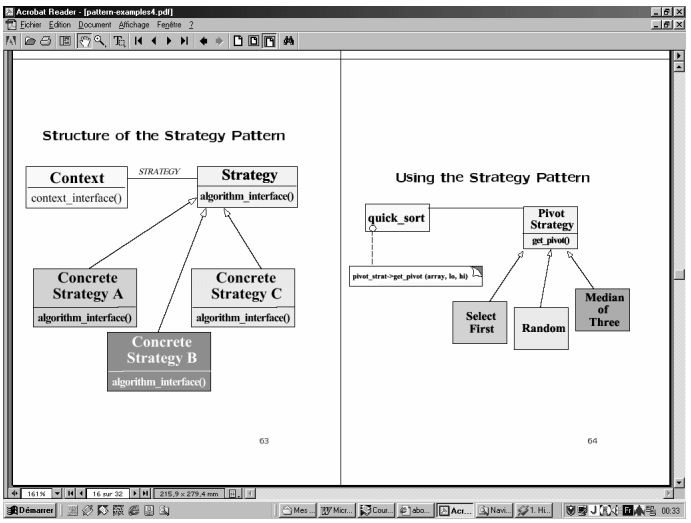
BTD/GL/P&F

30

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Structure



BTD/GL/P&F

31

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Implémentation

```
ARRAY est le type spécifique de données
template <class ARRAY>
void sort (ARRAY &array)
{
Pivot<ARRAY> *pivot_strat =
Pivot<ARRAY>::make_pivot(Options::instance()->pivot_strat ());
quick_sort (array, pivot_strat);
}
template <class ARRAY, class PIVOT_STRAT>
quick_sort (ARRAY &array, PIVOT_STRAT *pivot_strat)
{
for (;;)
{
 ARRAY::TYPE pivot; // typename ARRAY::TYPE pivot...
 pivot = pivot_strat->get_pivot (array, lo, hi);
// Partition array[lo, hi] relative to pivot...
}
}
```

BTD/GL/P&F

32