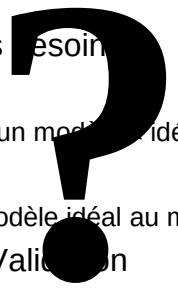
Conception de Logiciel *avec* les Design Patterns et UML (Unified Modeling Language)

Pr. Jean-Marc Jézéquel
IRISA - Univ. Rennes I

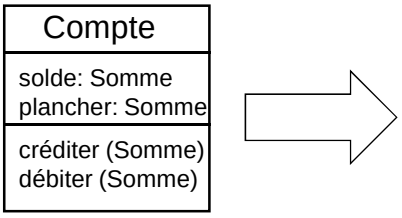
Campus de Beaulieu
F-35042 Rennes Cedex
Tel : +33 299 847 192 Fax : +33 299 842 532
e-mail : jezequel@irisa.fr
http://www.irisa.fr/prive/jezequel

Processus de développement avec UML

- Approche itérative, incrémentale, dirigée par les cas d'utilisation
  - Expression des besoins
  - Analyse
    - » Elaboration d 'un modèle idéal »
  - Conception
    - » passage du modèle idéal au monde réel
  - Réalisation et Validation



Générateur de code *built-in*

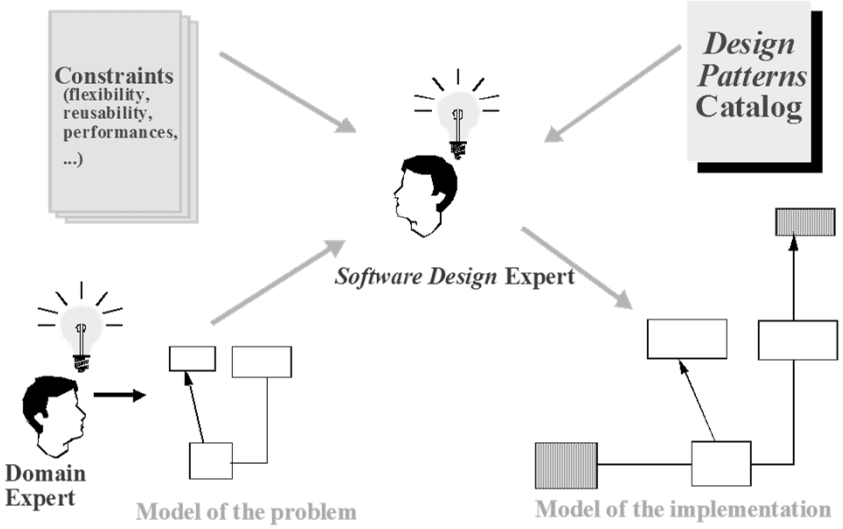



```
class Compte {
private int solde;
private int plancher;
public int getSolde(){
    return solde;
  }
…
public void crediter(int montant) {
  }
public void crediter(int montant) {
  }
}
```

Seulement un point de départ, car dans la vrai vie le code qu'on veut est plus complexe => intégration de préoccupation non-fonctionnelles

3

La conception dans le processus de développement avec UML



4

Le rôle du Designer

- Le designer est celui qui doit simplifier, donner une personnalité forte et invisible à ce qu 'il crée, élaguer, épurer, désencombrer, créer les produits adaptés (selon Jasper Morrison)

- There are two ways of constructing a software design (C. A. R. Hoare):
  - One way is to make it so simple that there are obviously no deficiencies
  - and the other way is to make it so complicated that there are no obvious deficiencies.
  - The first way is far more difficult to achieve…

5

Processus de développement avec UML

- Inspiré de OMT, Catalysis, Rapshody…

- Conception Systémique (Architecture)
  - choix stratégiques de réalisation du système
  - beaucoup d'ingéniérie, peu d'informatique
  - « patrons » d'architecture

- Conception Objet (détaillée)
  - préparation de la projection vers langage programmation
    - » aspects dynamiques, fonctionnels, relations...
  - « patrons » de conception détaillée (GoF)

6

Conception Systémique (Architecture)

- Organisation en sous-systèmes
- Choix de l'implantation du mode de contrôle du logiciel
- Conception de la concurrence
- Allocation des sous-systèmes aux tâches et processeurs
- Choix des modes de stockages des données
- Gestion de l'accès aux ressources globales
- Prévisions des conditions aux limites
- Choix des priorités pour les contraintes non-fonctionnelles

7

Organisation et accès aux données

- Base de données : avantages
  - Prévues pour gérer de grandes quantités de données
  - crash recovery, partage, distribution, intégrité
  - interface standardisée (SQL)
- Base de données : inconvénients
  - surcoût en performances
  - difficiles à manipuler si traitements complexes
  - interface médiocre avec les langages de programmation
- Modèle d 'architecture « 3 tiers »

8

Génération du shéma de base de données

- Si BD relationelle
  - Class => Table
  - Associations, attributs => colonnes
    - » Gestion des clés pour les associations
  - Gestion de l'héritage
- Si BD « moderne » e.g. BigTable, NoSql
  - Idem tables de hachage contenant tous les objets
- Pour certaines applications, tout en mémoire avec sérialization à la demande + journaling
  - Pas de BD!

9

Conception Objet (détaillée)

- Projection des aspects dynamiques
  - Exemples d'implantation de StateCharts
    - » Cas le plus simple, Réification d'événements, Réification d'état, Réification d'état et d'événement
- Conception des algorithmes
- Optimisations
- Ajustement de l'héritage
- Conception des relations
  - cas simples
  - Réification d'une relation
- Principes de conception objet
- Révision de la hiérarchie de classes

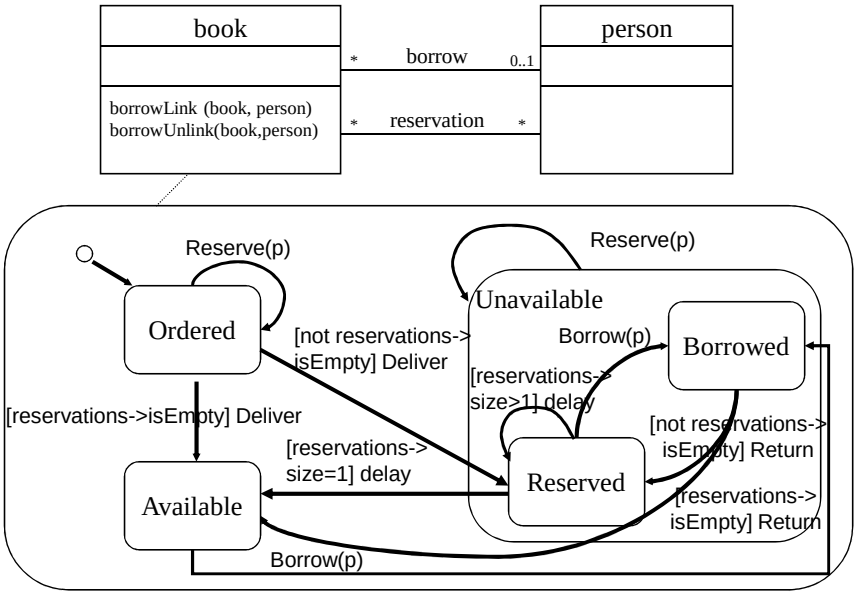
10

Projection des aspects dynamiques

- Présence d'une méthode par opération possible dans chaque classe
  - diagramme de séquences => méthodes et constructeurs
  - statecharts => m = f (state, event)
- Solution simple : pour chaque événement *e* (signal, condition, expiration de délai) du diagramme d'état
  - Créer une méthode process_*e* (paramètres optionnels)
  - avec un traitement par cas selon l'état de l'objet
- Utilisation possible des *design patterns*
  - COMMAND : réification des événements
  - STATE : réification des états
  - ou les deux => double dispatch

11

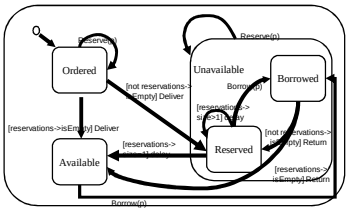
Exemples d'implantation de StateChart



12

Cas le plus simple

- Coder l'état sur un type énuméré
  - booléen, entier…
  - Introduire méthode d'accès à l'état (idem OCL)
- Pour chaque événement *e* :
  - Créer une méthode :
    - process_*e* (paramètres) avec traitement par cas selon l'état de l'objet



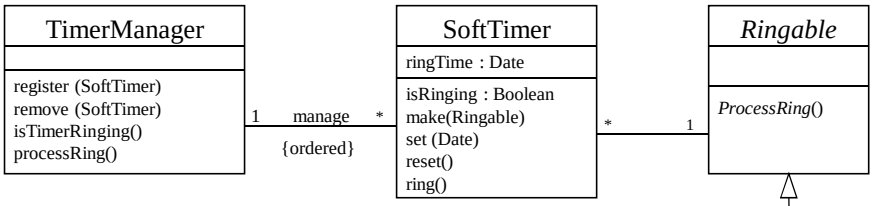

```
if Ordered() or Unavailable() then
   addReservation(this,p)
if Reserved() then
   if reservations.size = 1 then
      state := available
   end -- else stay in Reserved()
   removeReservation(reservations.first)
end -- else ignore
```

book
state : Integer
+Ordered() : Boolean +Available() : Boolean +Unavailable() : Boolean +Reserved() : Boolean +Borrowed() : Boolean +ProcessDeliver() +ProcessBorrow(p:person) +ProcessReturn() +ProcessReserve(p:person) +ProcessTimeout()

13

Gestion des délais sur les transitions

- Pattern *SoftTimer*

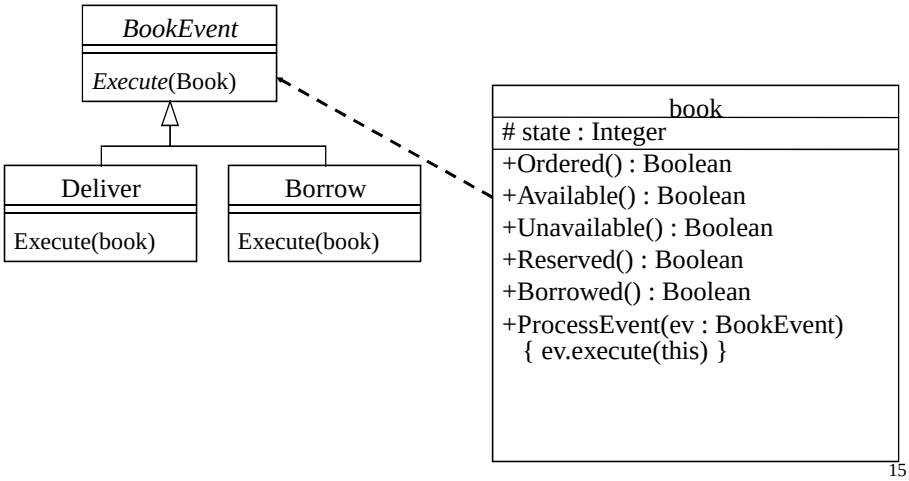


  - Sur chaque transition entrante
    - » myTimer.set(AwakingDate)
  - Sur chaque transition sortante
    - » myTimer.reset()
  - *TimerManager* : regarde périodiquement *t.isRinging* en tête de sa file, et si oui appelle *t.ring()*

14

Réification d'événements

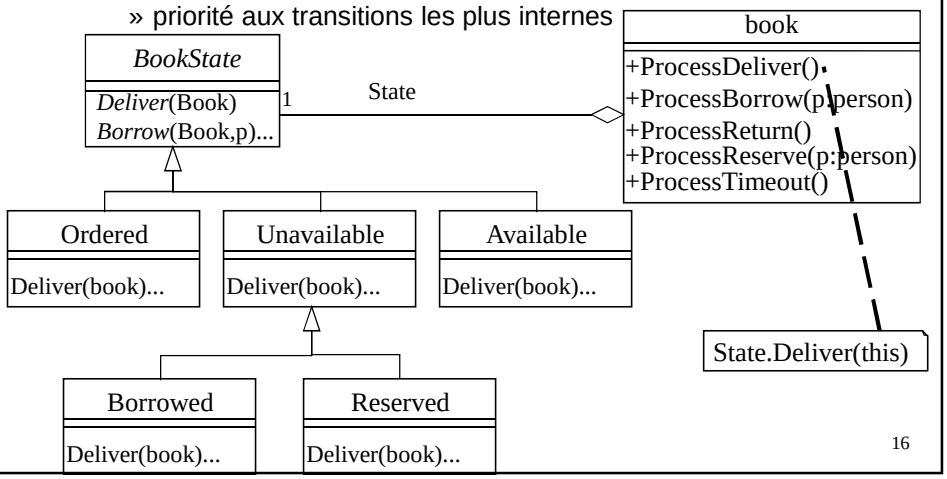
- Pattern « command »
  - une classe par événement, possibilité d'héritage



15

Réification d'état

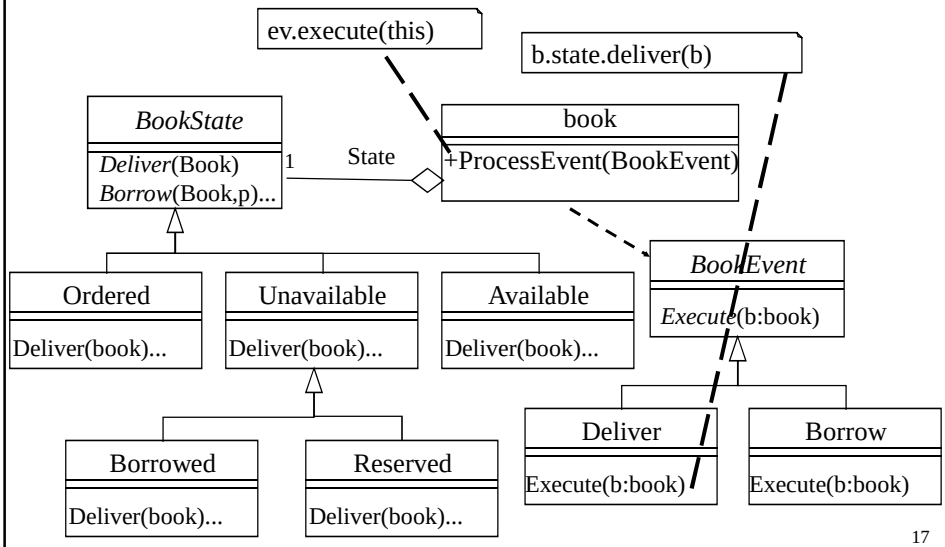
- Pattern « state »
  - une sous-classe par sous-état
  - hiérarchie d'états correspond à la hiérarchie de classes
    - » priorité aux transitions les plus internes



16

Réification d'état *et* d'événement

- Patterns *State* et *Command* simultanément



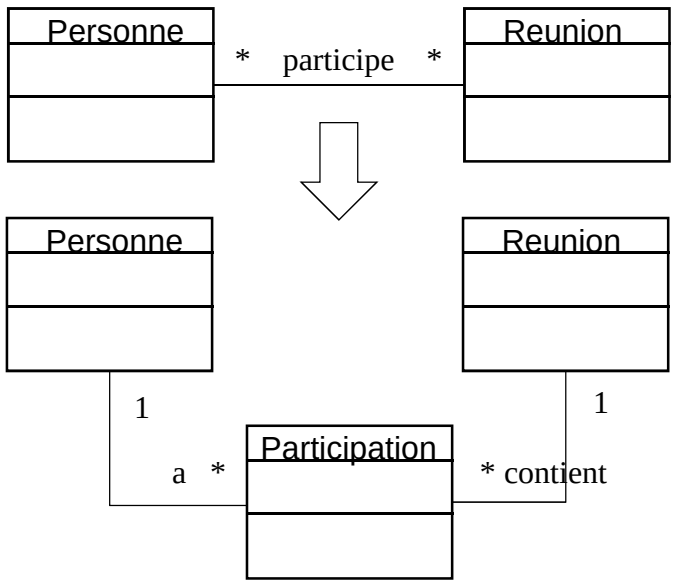
17

Conception des associations

- Analyse des sens de traversée des relations
- A sens unique, selon la multiplicité :
  - vers 1 => attribut (référence)
  - vers n => Liste ou Map
- Bi-directionnels, selon la fréquence :
  - un sens peu fréquent => attribut (référence) + recherche dans l'autre sens
  - si mise à jour peu fréquentes => attributs dans les 2 sens, mais structure complexe à mettre à jour
  - sinon, à l'aide d'un objet matérialisant cette relation
    - » Exemple : relation être-membre d'un groupe de multicast

21

Reification d'une association



22

Principes de conception Object

- Single Responsibility Principle (SRP)
  - Une classe par concept, et un concept par classe
- Open/Closed Principle (OCP)
  - Ouvert aux extensions par héritage, mais stable vs client
- Liskov Substitution Principle (LSP)
  - a.k.a. Design by Contract : une sous-classe doit honorer les contrats de ses super-classes
- Dependency Inversion Principle (DIP)
  - Le concret doit dépendre de l'abstrait (interfaces)
- Interface Segregation Principle (ISP)
  - Un programme ne doit pas dépendre de méthodes qu'il n'utilise pas

23

Révision de la hiérarchie de classes

- Essayer de simplifier le modèle
  - Einstein *« as simple as possible, but no simpler »*
  - St Exupéry *« un bon design n'est pas un design auquel on ne peut rien ajouter, mais un design auquel on ne peut rien enlever »*
- Revoir la découpe en paquetages
  - Classes groupées afin que les paquetages soient le plus faiblement couplés possible (cibles des tests unitaires)
- Documenter toutes les décisions de conception

24