

Software Engineering

Lecture 4

Modeling & OO design

David Baelde

`baelde@lsv.fr`

MPRI

2019

Agenda

- ▶ Modélisation
- ▶ UML
- ▶ Conception OO
- ▶ Patrons de conception

Modélisation

Conception de **modèles abstraits** du système.

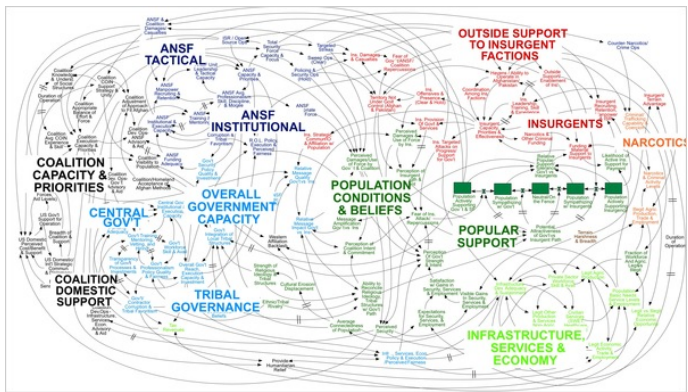
Différents aspects

- ▶ **Vues**: contexte, interaction, structure, comportement
 - ▶ **Niveaux**: architecture, composants et interface, code
 - ▶ **Formalisme**: différents objectifs
 - ▶ Convaincre le client
 - ▶ Organiser le développement
 - ▶ Guider le développement
 - ▶ Générer des tests appropriés
 - ▶ Comprendre et faire évoluer le code
- ↪ une spec. formelle, quelques diagrammes, un teaser...

UML

Unified Modeling Language

- ▶ Notation **graphique** semi-formelle
- ▶ **Standard** ISO/IEC 19501:2005 (UML 1.4.2)
- ▶ Principalement **orienté objet**
- ▶ 14 types de diagrammes



When we understand that slide, we'll have won the war.
General McChrystal



We Have Met the Enemy and He Is PowerPoint,
NY Times, April 2010.

Prendre les modèles au sérieux

Utilisations formelles...

- ▶ Correct by design, e.g., affinements successifs en méthode B
- ▶ Model-checking: spec \Rightarrow modèle (\rightsquigarrow code)
- ▶ Fragment xUML doté d'une sémantique formelle
+ specs exécutables en Object Constraint Language (OCL)

...ou semi-formelles

- ▶ Model-driven engineering: générer l'essentiel du code
- ▶ Génération de code à partir d'un diagramme de classes
- ▶ Modèle graphique d'une GUI (glade)
- ▶ Automate, DSL (e.g., SQL-like, parser)

Dans la “vraie vie” ?

Les diagrammes sont utiles: ébauches, vues d'ensemble. . .

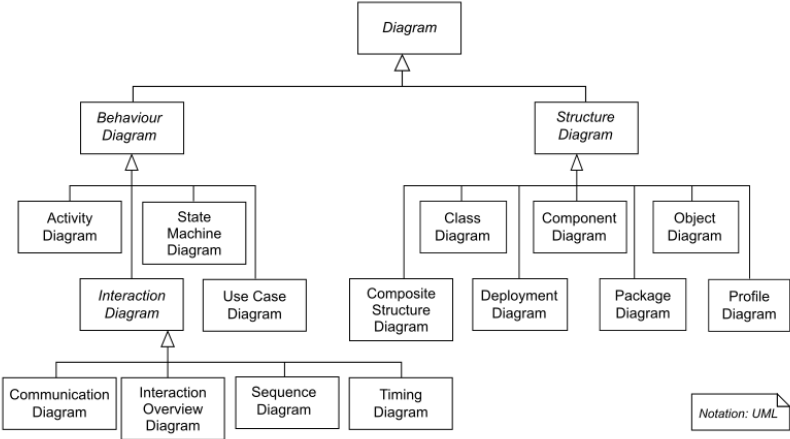
Tant qu'à faire, autant utiliser le standard UML, quitte à prendre des libertés avec.

Le **code** est rarement dérivé de modèles, mais les modèles peuvent être extraits du code.

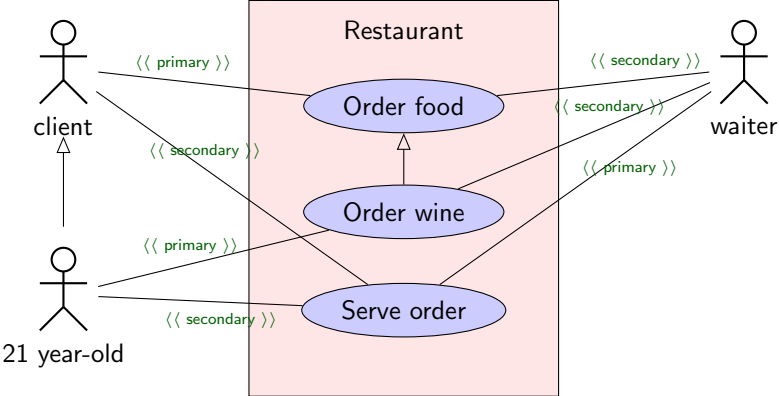
(Typiquement en même temps que la doc.)

UML diagrams

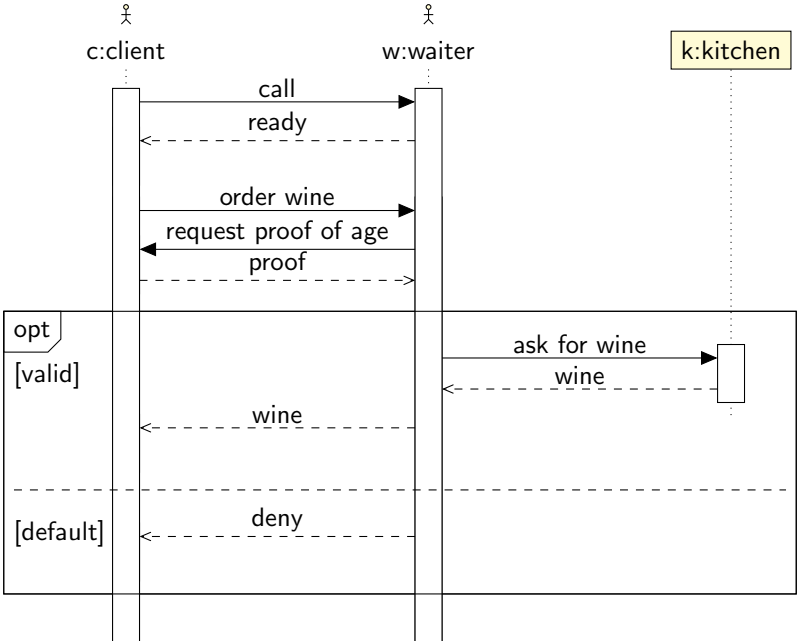
UML diagrams



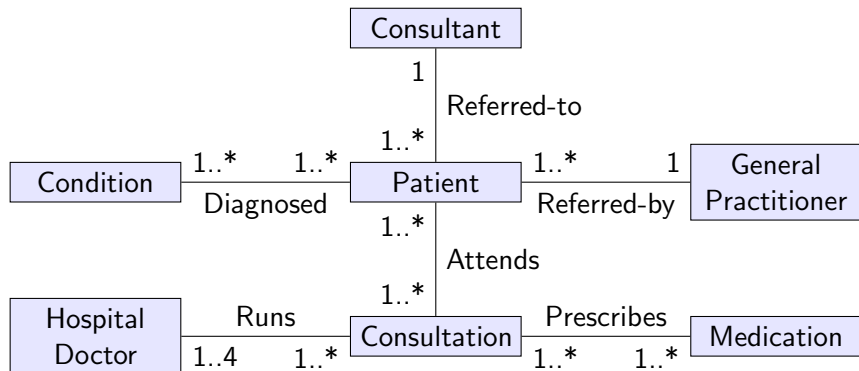
Behavior: use case



Behavior: sequence



Structure: classes



Exemple ne détaillant pas les **classes**, concentré sur les **associations**.
Similaire aux modèles de données utilisés en BDD.

Classes, en détail et en pratique

Démo avec `dia`:

- ▶ Attributs et opérations, visibilité et portée
- ▶ Généralisation
- ▶ Classes et opérations abstraites
- ▶ Interfaces et réalisation
- ▶ Aggrégation et composition
- ▶ *Limitations et déviation de la norme...*

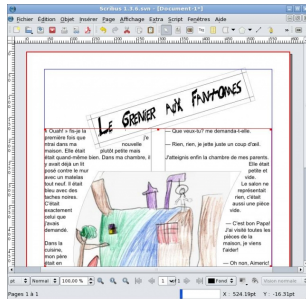
Utilisation de `dia2code` et `doxygen` pour les conversions entre diagrammes et code.

Patrons de conception

Quelques problèmes de conception

Logiciel de mise en page

- ▶ Représentation du document
- ▶ Formattage
- ▶ Analyses du texte
- ▶ Commandes
- ▶ Multiples environnements graphiques



Nous cherchons des solutions facilitant la maintenance et l'évolution du code. À travers ces solutions nous rencontrerons plusieurs *design patterns* à la portée plus générale.



Design Patterns: Elements of Reusable Object-Oriented Software, E. Gamma, R. Johnson, R. Helm, J. Vlissides, Pearson Education, 1994.

Représentation du document

Objectif

- ▶ Morceaux de texte, dessin, images, sur des pages
- ▶ Organisation en blocs, flottante, etc.
- ▶ Opérations: afficher, éditer, formater, analyser

Solution générale

Composition récursive, un glyphe composite est aussi un glyphe

Code et diagrammes: ML, UML, C++

```
glyph*.ml, Composite.dia, glyph_safe.{h,cpp}
```

Questions

- ▶ Possibilité d'ajouter un glyphe?
- ▶ Facilité d'ajouter un glyphe?
- ▶ Opérations composites: sûreté vs. transparence?

Analyse

Divers algos analysent et modifient la structure du texte:
hyphénisation, compter les mots, correction orthographique, etc.

Problèmes

- ▶ Notions de parcours communes à plusieurs algos
- ▶ Comment traiter différemment différents glyphes?

Itérateur

Isoler la fonctionnalité de parcours:

- ▶ `void first(TraversalType t)` pour initialiser
- ▶ `void next()` pour avancer
- ▶ `bool isDone()` pour savoir si on a fini
- ▶ `Glyph current()` pour l'élément courant

Où intégrer cette fonctionnalité?

Itérateur

Isoler la fonctionnalité de parcours:

- ▶ `void first(TraversalType t)` pour initialiser
- ▶ `void next()` pour avancer
- ▶ `bool isDone()` pour savoir si on a fini
- ▶ `Glyph current()` pour l'élément courant

Où intégrer cette fonctionnalité?

- ▶ Ajouter à `Glyph` les opérations d'itérateur

Itérateur

Isoler la fonctionnalité de parcours:

- ▶ `void first(TraversalType t)` pour initialiser
- ▶ `void next()` pour avancer
- ▶ `bool isDone()` pour savoir si on a fini
- ▶ `Glyph current()` pour l'élément courant

Où intégrer cette fonctionnalité?

- ▶ Ajouter à `Glyph` les opérations d'itérateur
 ~> difficile à étendre, une seule traversée à la fois
- ▶ Interface itérateur séparée
 + différentes implémentations initialisées sur un glyphe

Visiteur

L'hyphénation ne va s'intéresser qu'aux mots dans un parcours

Comment traiter différemment différents glyphs?

Solutions

- ▶ Implémenter la partie de l'analyse liée à la classe `C` en (re)définissant la méthode `analyse` de cette classe

Visiteur

L'hyphénation ne va s'intéresser qu'aux mots dans un parcours
le spellcheck va aussi aller voir les sous-titres d'images, etc.

Comment traiter différemment différents glyphes?

Solutions

- ▶ Implémenter la partie de l'analyse liée à la classe `C`
en (re)définissant la méthode `analyse` de cette classe
- ▶ Encapsuler l'analyse dans un *Visitor*

↪ `visitor.cpp`

Stratégies

Ajout de mots à une ligne \rightsquigarrow nouvelle ligne

Différents algos (vitesse vs. efficacité, règles typographiques)

Diagrammes pour deux solutions: `Formatted[12].dia`

\rightsquigarrow patrons *Strategy* et *Factory*

Commandes

Le menu de l'application donne accès à diverses fonctionnalités:
undo, sauver, synthèse vocale, diffuser sur un réseau social, etc.

Solution 1

On code la définition des entrées du menu à la main.

- ▶ Couplage entre interface et fonctionnalités
- ▶ Pas facile à maintenir (évolutions, versions différentes)

Solution 2

Design pattern "Commande"

- ▶ Possibilité de plugins

Emballage

Objectifs

- ▶ Afficher une page avec un cadre autour
- ▶ Afficher une (partie de) page avec barre de défilement

Solutions

- ▶ Par héritage, e.g., dériver `PageWithBorder` de `Page`

Emballage

Objectifs

- ▶ Afficher une page avec un cadre autour
- ▶ Afficher une (partie de) page avec barre de défilement
- ▶ Idem pour d'autres glyphes que la page entière

Solutions

- ▶ Par héritage, e.g., dériver `PageWithBorder` de `Page`

Emballage

Objectifs

- ▶ Afficher une page avec un cadre autour
- ▶ Afficher une (partie de) page avec barre de défilement
- ▶ Idem pour d'autres glyphes que la page entière
- ▶ Faire les deux en même temps

Solutions

- ▶ Par héritage, e.g., dériver `PageWithBorder` de `Page`
- ▶ Par composition: glyphes composites `Border` et `Scrollbar`
 \rightsquigarrow patron *Decorator*

C'est tout pour aujourd'hui

Pour conclure

- ▶ Il y a toujours plus d'une façon de faire
- ▶ Apprendre à réfléchir à l'évolution du code
- ▶ Comprendre les forces et faiblesses d'un style/paradigme
- ▶ Ne pas hésiter à enrichir son diagramme de classes
- ▶ Penser à la composition, ne pas tout faire par héritage