

Mise à Niveau UML

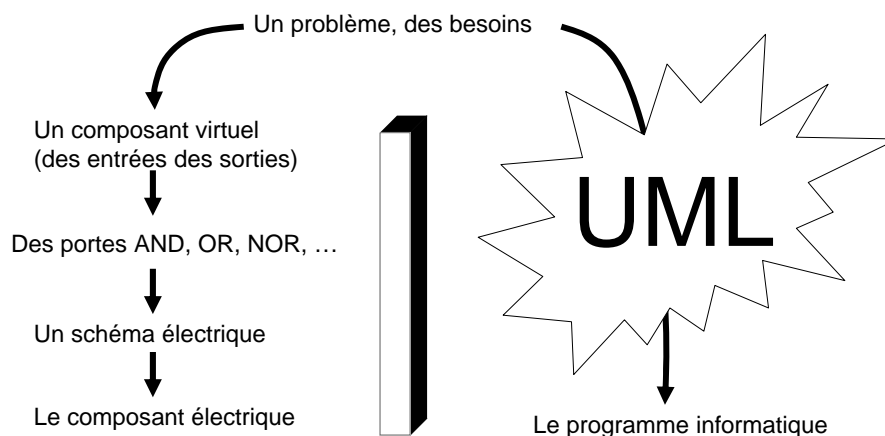
(Unified Modeling Language)

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>

1

L'Electricien et l'Informaticien



2

Plan

- Introduction
 - Du code aux modèles
 - Pourquoi UML ?
 - L'historique
 - Etat actuel
 - UML pour l'utilisateur
 - Dans la pratique : modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Base
 - Avancée
 - Aspects statiques du système
 - Description des données et de leurs relations
 - Base
 - Avancée
 - OCL
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Vision implantation
 - Quoi de neuf pour UML 2.x
- Conclusion

3

Plan

- Introduction
 - Du code aux modèles
 - Pourquoi UML ?
 - L'historique
 - Etat actuel
 - UML pour l'utilisateur
 - Dans la pratique : modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Aspects statiques du système
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement
 - Quoi de neuf pour UML 2.x
- Conclusion

4

1 - Introduction

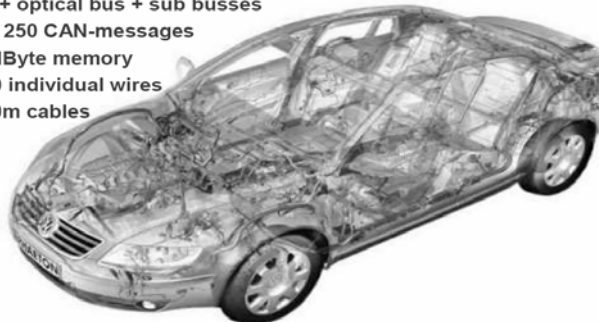
L'ingénierie du logiciel aujourd'hui

5

Automotive industry: Volkswagen

Phaeton

- ◆ 61 networked ECUs
- ◆ 3 bus systems + optical bus + sub busses
- ◆ 2500 signals in 250 CAN-messages
- ◆ more than 50 MByte memory
- ◆ more than 2000 individual wires
- ◆ more than 3800m cables



6

Google

- 300 000 serveurs
 - répartis dans une vingtaine de datacenters.
- Une phénoménale capacité de calcul qui permet à Google d'assurer le fonctionnement d'un grand nombre de services
 - Google Earth ...
- répondre à plus d'1 milliard de requêtes par jour,

7

Nokia

- 1,1 milliard de téléphones portables en circulation
 - 100 millions de plus par an
- Des milliers de versions de logiciels
- Time-to-market ~ 3 mois

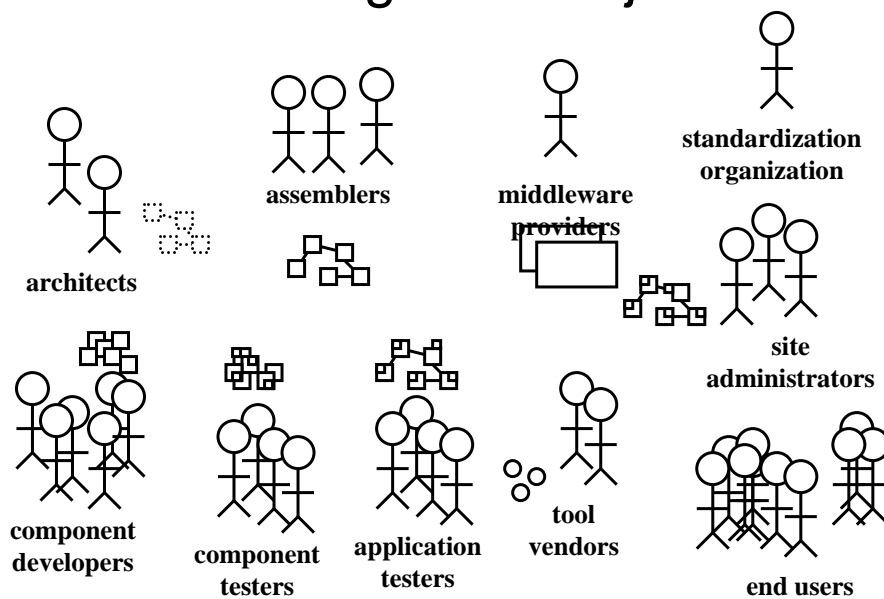


Des logiciels complexes...

- Logiciels de grande taille
 - des millions de lignes de code
 - des équipes nombreuses
 - durée de vie importante
 - des lignes de produits
 - plateformes technologiques complexes
 - évolution continue
- Logiciels complexes
- Logiciels critiques
- ...

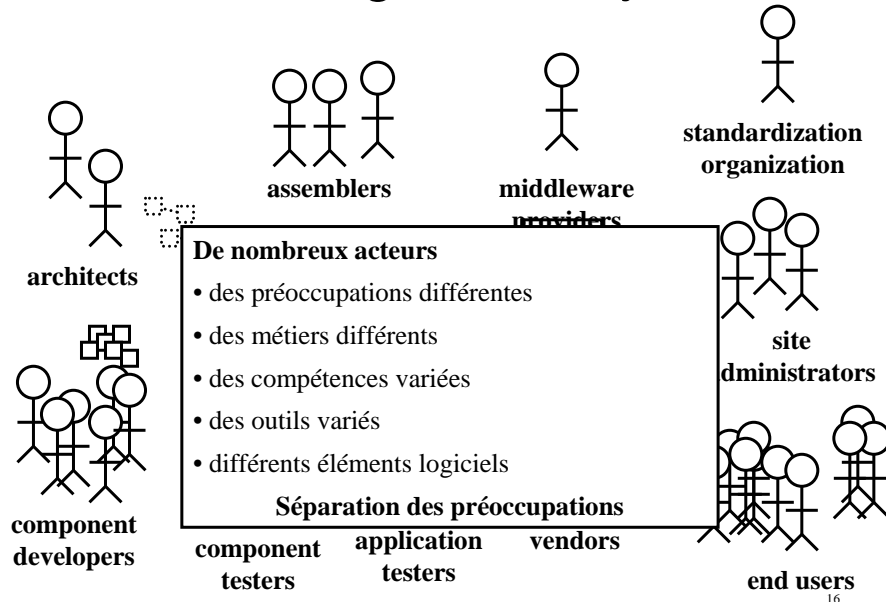
9

L'industrie logicielle aujourd'hui

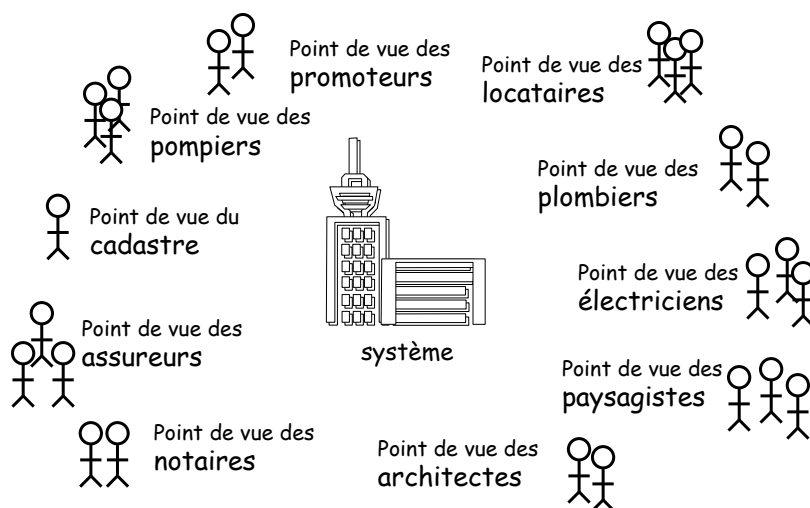


15

L'industrie logicielle aujourd'hui

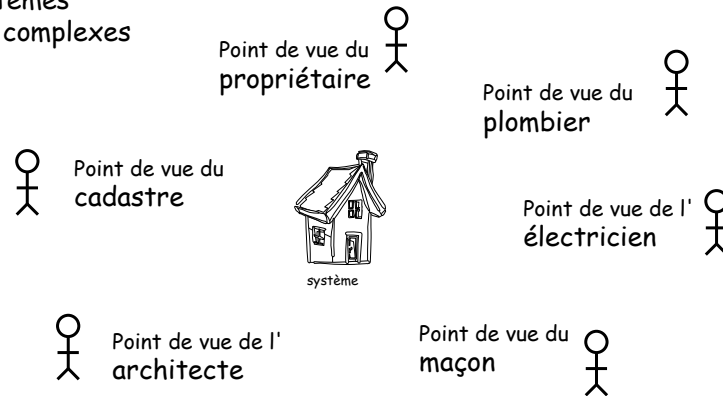


Séparations des préoccupations



Séparations des préoccupations

Utile même pour
des systèmes
"moins" complexes



18

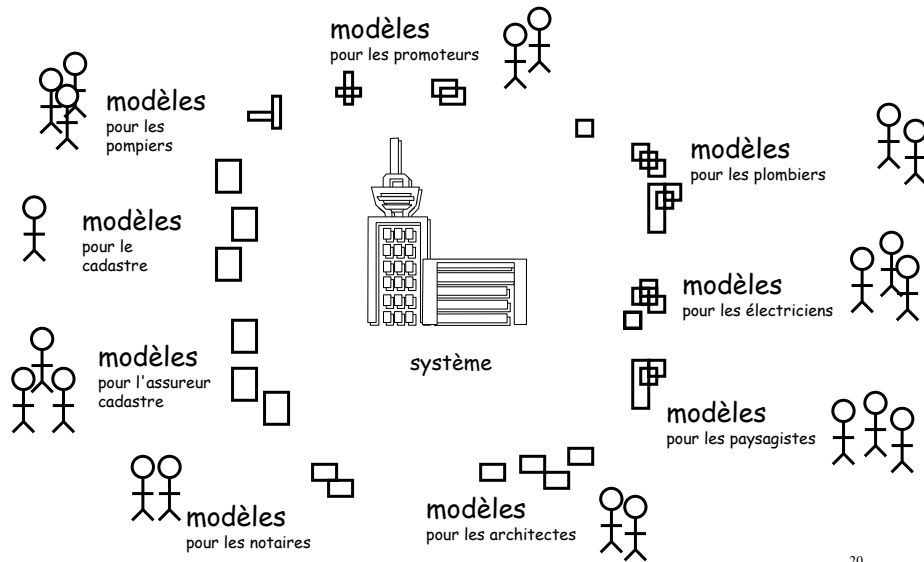
Problématique

- Complexité croissante des logiciels
- Séparations des préoccupations
- Séparations des métiers
- Multiplicité des besoins
- Multiplicité des plateformes
- Évolution permanente

**Logiciel = Code ?
Est-ce la solution ?**

19

multiples modèles d'un même système



20

Ingénierie Dirigée par le Code
ou
Ingénierie Dirigée par les Modèles ?

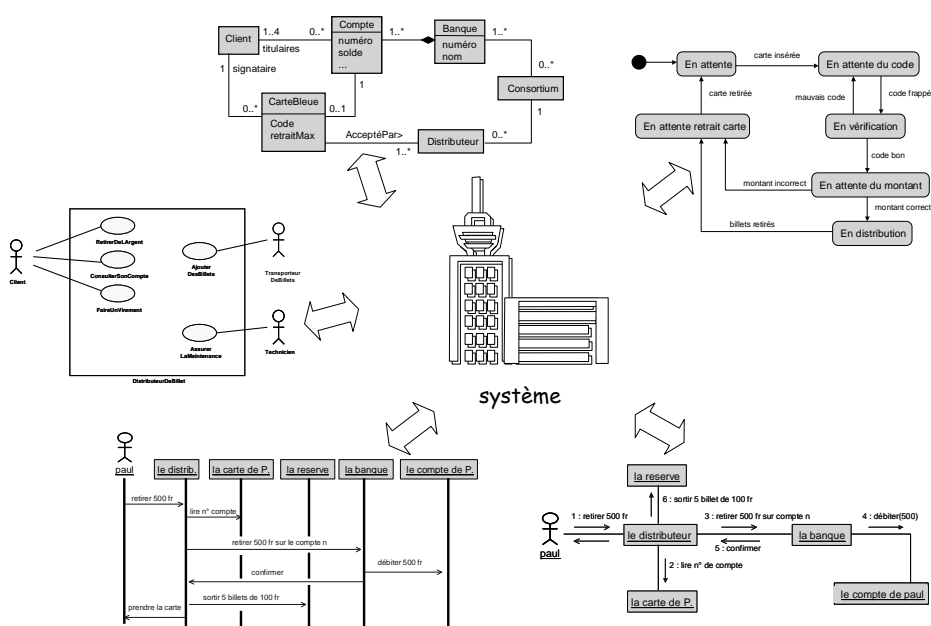
Telle est la question...

21

MDO

- Ce cours montre comment à partir de la notation UML il est possible de définir un certain nombre de modèles
 - plus ou moins abstraits
 - plus ou moins élaborés
 - mais décrivant tous une facette complémentaire du système à réaliser selon un point de vue ou un métier différent.
- L'accent est mis sur les règles de cohérence intra et inter-modèles, ainsi que sur la différence qui peut être faite entre
 - modèles conceptuels, de spécification et d'implémentation
 - modèles métiers et modèles techniques
 - modèles produits et modèles de processus.

22



23

Synthèse

- Des Modèles plutôt que du Code
 - Un modèle est la simplification/abstraction d'un aspect de la réalité
 - Nous construisons donc des modèles afin de mieux comprendre les systèmes que nous développons
 - Nous modélisons des systèmes complexes parce que nous sommes incapables de les comprendre dans leur totalité
 - Le code ne permet pas de simplifier/abstraire la réalité

34

2 - Pourquoi UML ?

35

Des Méthodes de modélisation

- L'apparition du paradigme objet à permis la naissance de plusieurs méthodes de modélisation
 - OMT, OOSE, Booch, Fusion, ...
- Chacune de ces méthodes fournies une notation graphique et des règles pour élaborer les modèles
- Certaines méthodes sont outillées

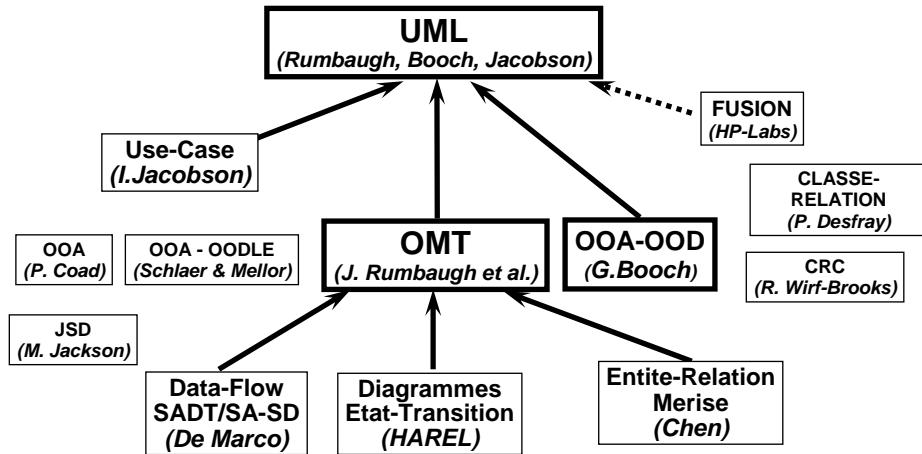
37

Trop de Méthodes

- Entre 89 et 94 : le nombre de méthodes orientées objet est passé de 10 à plus de 50
- Toutes les méthodes avaient pourtant d'énormes points communs (objets, méthode, paramètres, ...)
- Au milieu des années 90, G. Booch, I. Jacobson et J. Rumbaugh ont chacun commencé à adopter les idées des autres. Les 3 auteurs ont souhaité créer un langage de modélisation unifié

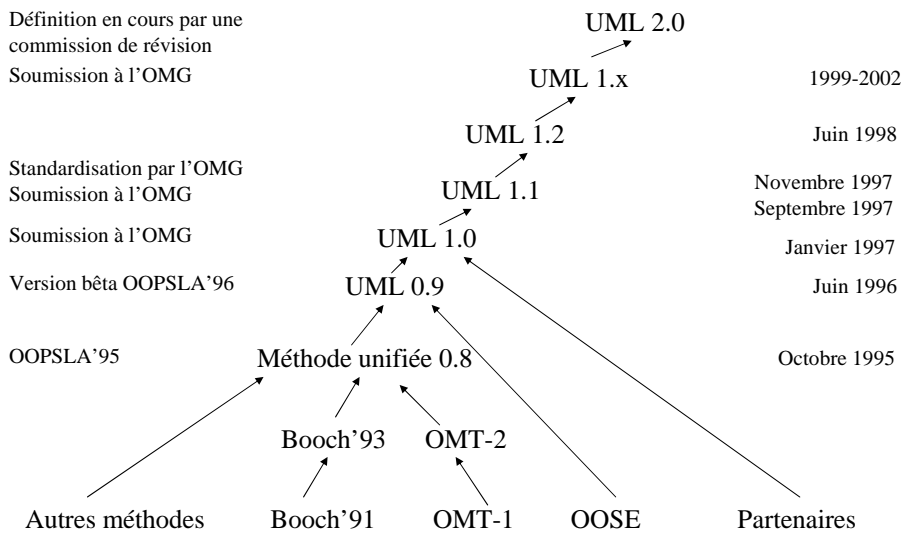
38

Généalogie de UML



39

Historique



40

Aujourd'hui

- UML est le langage de modélisation orienté objet le plus connu et le plus utilisé au monde
- UML s'applique à plusieurs domaines
 - OO, RT, Deployment, Requirement, ...
- UML n'est pas une méthode
 - RUP
- Peu d'utilisateurs connaissent le standard, ils ont une vision outillée d'UML (Vision Utilisateur)
 - 5% forte compréhension, 45% faible compréhension, 50% aucune compréhension
- UML est fortement critiqué car pas assez formel
- Le marché UML est important et s'accroît
 - MDA et MDD, UML2.1, IBM a racheté Rational !!!

41

3 – UML pour l'utilisateur

42

UML Pourquoi

- Réfléchir
- Définir la structure « gros grain »
- Documenter
- Guider le développement
- Développer, Tester, Auditer



43

La consécration des méthodes fondées sur la modélisation

- L'approche par modélisation facilite
 - Communication (et sa précision)
 - avec donneur d'ordre
 - entre différentes phases de développement et de maintenance
 - Capacité de vérification
 - Cohérence
 - Complétude
 - Continuité entre les différentes phases du cycle de vie
 - cf. Jackson et JSD
 - N.B.: continuité \Leftrightarrow isomorphique ou plongement/projection

44

Un peu de Méthodologie...

- Une méthode de développement de logiciels, c'est :
 - Une notation
 - La syntaxe --- graphique dans le cas de UML
 - Un méta-modèle
 - La sémantique --- paramétrable dans UML (*stéréotypes*)
 - Un processus
 - Détails dépendants du domaine d'activité :
 - Informatique de gestion
 - Systèmes réactifs temps-réels
 - *Shrink-wrap* software (PC)

45

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


46

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement

47

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - *Cas d'utilisation*
 - Aspects statiques du système
 - Description des données et de leurs relations
 - OCL
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement
- 

48

Use case

Concepts de base

49

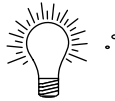
Expression des besoins et OOAD

- Sujet longtemps négligé (e.g. OMT)
- Question de l'expression des besoins pourtant fondamentale
 - Et souvent pas si facile (*cible mouvante*)
 - cf. *syndrome de la balançoire*
- Object-Oriented Software Engineering (Ivar Jacobson et al.)
 - Principal apport : la technique des acteurs et des cas d'utilisation
 - Cette technique est intégrée a UML

50

Quatre objectifs

☆ Se comprendre



⌚ Représenter le système



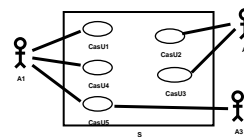
⌚ Exprimer le service rendu

⌚ Décrire la manière dont le système est perçu

51

Modèle des cas d'utilisation

- Buts :
 - modéliser le point de vue des **utilisateurs**
 - définir les limites précises du **système**
- Notation très simple, compréhensible par tous, y compris le client
- Permet de structurer :
 - les besoins (cahier des charges)
 - le reste du développement
- Modèle communicationnel



52

Modèle des cas d'utilisation

- Moyens :

- Les acteurs UML
- Les *use-cases* UML
- Utilisation d'un dictionnaire du domaine

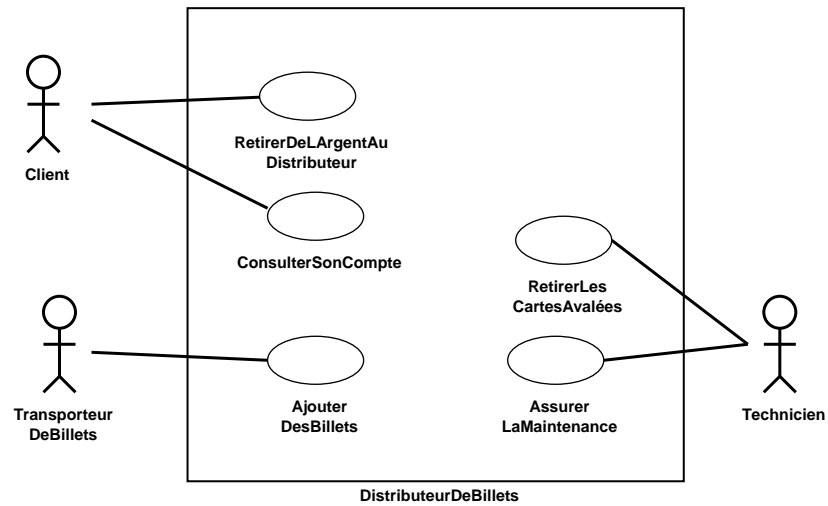
53

Exemple de cas d'utilisation



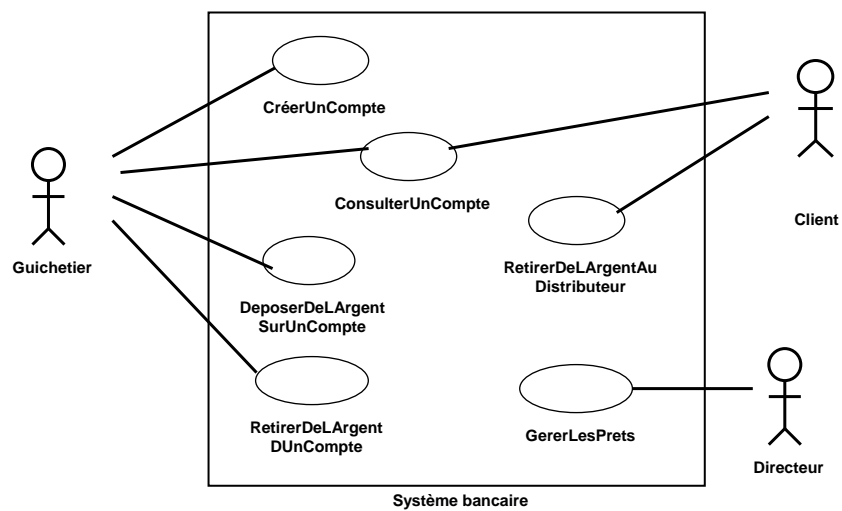
56

Diagramme de cas d'utilisation



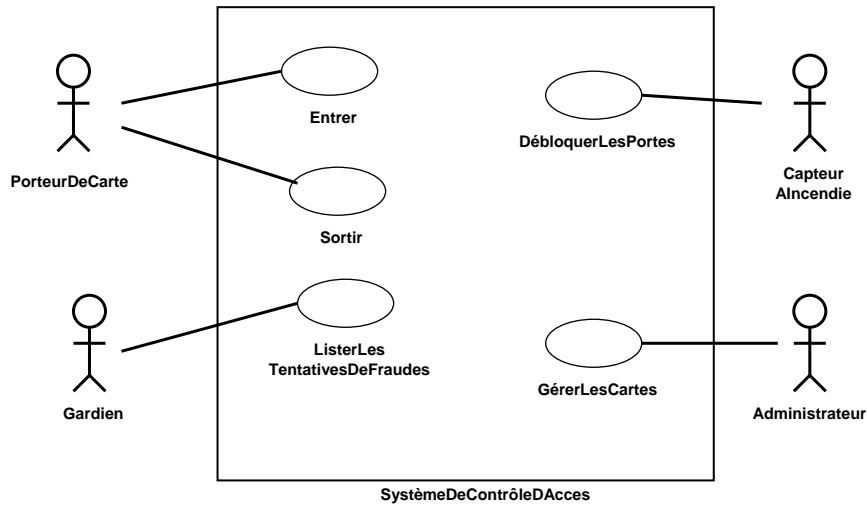
57

Diagramme de cas d'utilisation



58

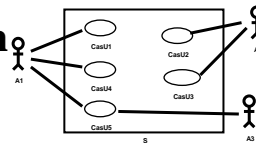
Modèle de cas d'utilisation



59

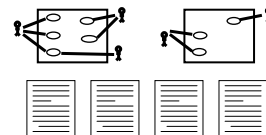
Modèle des cas d'utilisation

● Diagramme de cas d'utilisation



■ Modèle de cas d'utilisation

- ◆ descriptions textuelles,
- ◆ diagrammes de cas d'utilisation
- ◆ diagrammes de séquences
- ◆ Dictionnaire de données
- ◆ ...



60

Modèle "communicationnel"

- Modèle **informel** centré **utilisateur**

- Avant tout sous forme textuelle

- Diagramme utilisé

- pour les réunion de "brainstorming"
- pour simplifier la communication
- pour structurer les documents
- pour structurer le développement

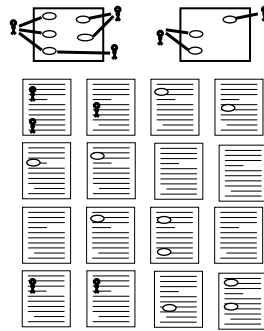


Diagramme = plan du document

61

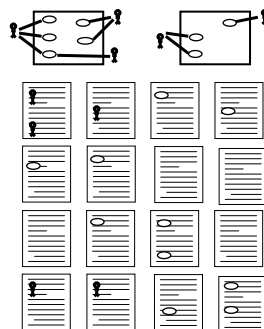
Langage peu formalisé

- Modèle de cas d'utilisation **peu standardisé** par UML

- Différents styles

- Différentes interprétations

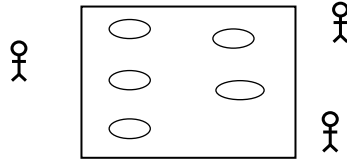
- Modèle construit par raffinements successifs et **consensus grandissant**



Peu de formalisme,
beaucoup de bon sens et de **communication**

62

Elements de base



Acteurs



Cas d'utilisation



Systeme



63



CasDUtilisationX

Cas d'utilisation (CU)

- Cas d'utilisation (CU)

- une manière d'utiliser le système
- une suite d'interactions entre un acteur et le système

- Correspond à une fonction du système **visible par l'acteur**
- Permet à un acteur d'atteindre **un but**
- Doit être **utile en soi**
- Regroupe un **ensemble de scénarii** correspondant à un même **but**



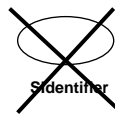
Entrer



~~EnregistrerEntrée~~



RetirerDeLArgentAu
Distributeur



~~SIdentifier~~

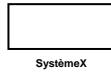


~~EntrerPendant
LesHeuresDOuverture~~



~~TapierSonCode~~

64

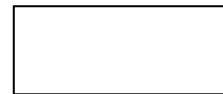


Le système

- Le système est
 - modélisé par un ensemble de cas d'utilisation
 - vu comme une **boîte noire**
- Le système contient :
 - les cas d'utilisation,
 - mais pas les acteurs.
- Un modèle de cas d'utilisation permet de définir :
 - les fonctions essentielles du système,
 - les **limites du système**,
 - le système par rapport à son environnement,
 - **délimiter le cadre du projet !**



DistributeurDeBillets



SystemeDeControleDAcces



SystemeDeSécuritéIncendie

65



Client

Acteurs

Un Acteur =

- élément **externe** qui *interagit* avec le système
- (prend des décisions, des initiatives. Il est "actif".)
- **rôle** qu'un "utilisateur" joue par rapport au système



PorteurDeCarte



Gardien



Administrateur



Capteur Alncendie

66



Cient

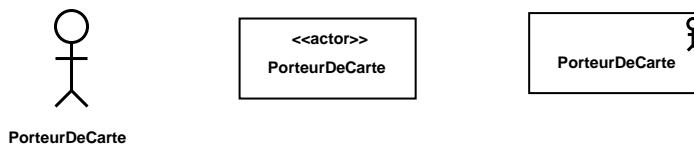
Acteurs vs. utilisateurs

Ne pas confondre la notion d'Acteur et de personne utilisant le système

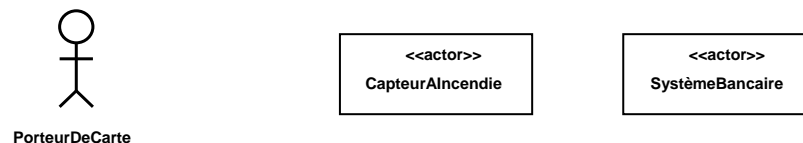
- Une même personne peut jouer plusieurs rôles
ex: Maurice est directeur mais peut jouer le rôle de guichetier
- Plusieurs personnes peuvent jouer un même rôle
ex: Paul et Pierre sont deux clients
- Un rôle par rapport au système plutôt que position dans l'organisation
ex: PorteurDeCarte plutôt qu'Enseignant
- Un acteur n'est **pas forcément** un être **humain**
ex: un distributeur de billet peut être vu comme un acteur⁶⁷

Différents notations possibles

- Notations alternatives pour les acteurs



- Note de style :
utiliser plutôt le stéréotype <<actor>> pour les acteurs non humains



68



Différents types d'acteurs

Liste pour ne pas oublier d'acteurs :

- **Utilisateurs principaux**
ex: client, guichetier
- **Utilisateurs secondaires**
ex: contrôleur, directeur, ingénieur système, administrateur...
- **Périphériques externes**
ex: un capteur, une horloge externe, ...
- **Systèmes externes**
ex: système bancaires

69

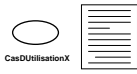


Description préliminaire des cas d'utilisation

- Pour chaque cas d'utilisation
 - choisir un **identificateur représentatif**
 - notes de style :
 - choisir une forme verbale décrivant une action
 - l'acteur est généralement le sujet
 - identification concise mais précise
 - éviter les connecteurs (et, ou, puis, ...)
 - terme provenant autant que possible du métier
 - utiliser par exemple le style MajMin
 - terme générique comme "Gérer" en cas de besoin

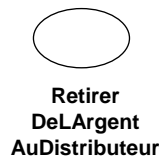
Gérer = Créer, Supprimer, Ajouter, Modifier, ...
Exemple: GérerLesDroits

74



Description préliminaire des cas d'utilisation

- Pour chaque cas d'utilisation
 - ◆ choisir un identificateur représentatif
 - ◆ donner une **description textuelle simple**
 - ◆ la fonction réalisée doit être comprise de tous
 - ◆ préciser ce que fait le système, ce que fait l'acteur
 - ◆ pas trop de détails, se concentrer sur le **scénario "normal"**



Retirer
DeLArgent
AuDistributeur

Lorsqu'un *client* a besoin de liquide il peut en utilisant un distributeur retirer de l'argent de son compte. Pour cela :

- le *client* insère sa carte bancaire dans le distributeur
- le *système* demande le code pour l'identifier
- le *client* choisit le montant du retrait
- le *système* vérifie qu'il y a suffisamment d'argent
- si c'est le cas, le *système* distribue les billets et débite le compte du client
- le *client* prend les billets et retire sa carte

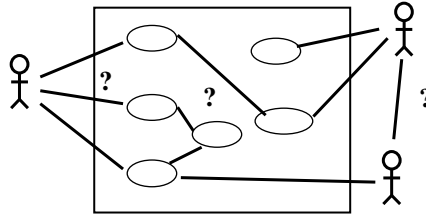
75

Attention

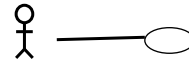
"A common sign of a novice (or academic) use case modeler is a preoccupation with use case diagrams and use case relationships, rather than writing text. ... Use case diagrams and use case relationships are secondary in use case work. Use cases are text documents. Doing use case work means to write text."
[Applying UML and Patterns, Craig Larman]

76

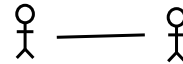
Relations entre éléments de base



- Relations acteurs <-> cas d'utilisation ?



- Relations acteurs <-> acteurs ?

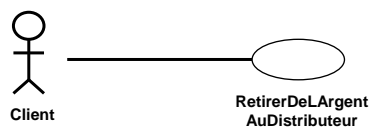


- Relations cas d'utilisation <-> cas d'utilisation ?

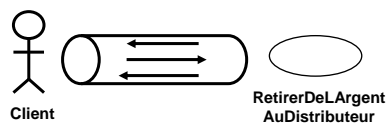


77

Relation acteur - cas d'utilisation



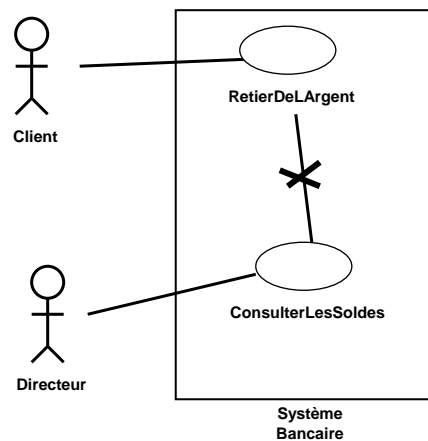
- Point de vue besoin: Représente la **possibilité d'atteindre un but**
- Point de vue système: Représente un **canal de communication**
- Echange de messages, potentiellement dans les deux sens
- Protocole particulier concernant le cas d'utilisation considéré



78

Relation cas d'utilisation - cas d'utilisation

- **Communications internes non modélisées.**
- UML se concentre sur la description du système et de ses interactions avec l'extérieur
- Formalisme bien trop pauvre pour décrire l'intérieur du système. Utiliser les autres modèles UML pour cela.
- Autres relations possibles entre CU (slides suivants)



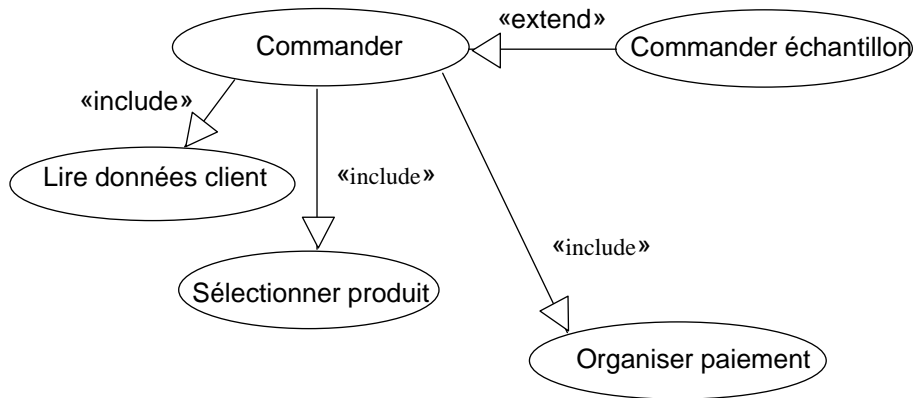
79

Relations sur les *use-cases*

- Utilisation (*«include»* / *« uses »*)
 - Utilisation d'autres use-cases pour en préciser la définition
- Extension (*«extend»* / *« extends »*)
 - Un use-case étendu est une spécialisation du use-case père

80

Relations sur les *use-cases* : notation



81

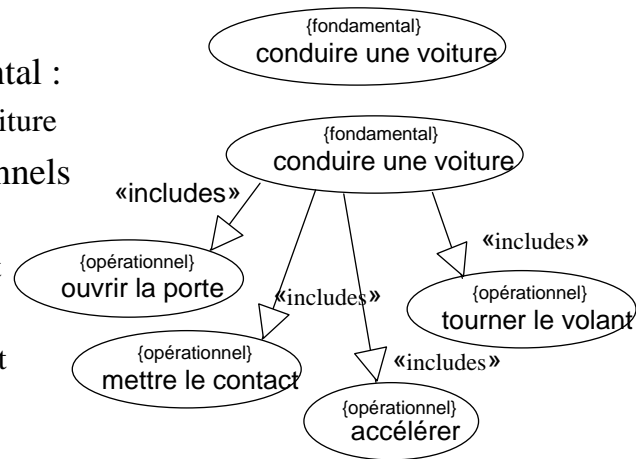
Exprimer le service rendu

- **Besoins fondamentaux : manières d'utiliser le système**
 - Représentation globale par cas d'utilisation
 - \forall taille du système, seulement de 3 à 10 Use Cases
- **Besoins opérationnels : interactions avec le système**
 - Représentation détaillée par raffinement des cas d'utilisation
 - Début de décomposition fonctionnelle : ne pas aller trop loin

82

Besoins fondamentaux et opérationnels

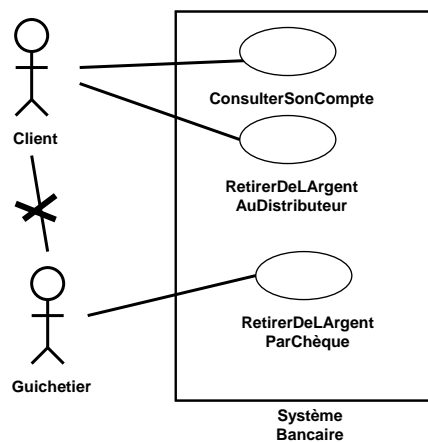
- Besoin fondamental :
 - Conduire une voiture
- Besoins opérationnels
 - Ouvrir la porte
 - Mettre le contact
 - Accélérer
 - Tourner le volant
 - ...



83

Relation de communication acteur-acteur

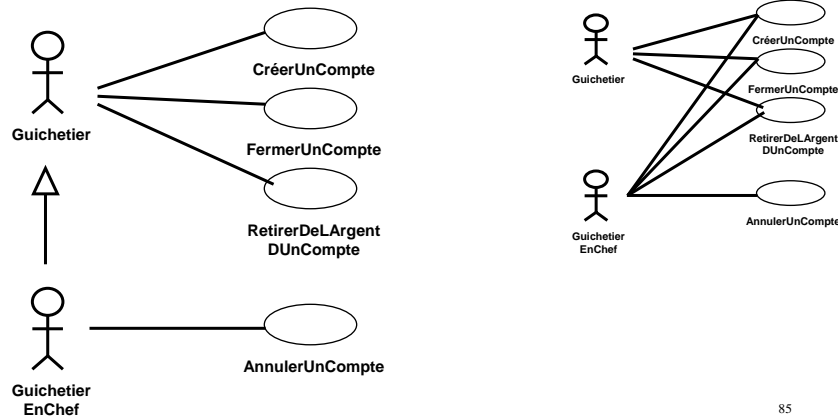
- **Communications externes non modélisée**
- UML se concentre sur la description du système et de ses interactions avec l'extérieur



84

Relation acteur - acteur : généralisation

- La seule relation entre acteurs est la relation de généralisation

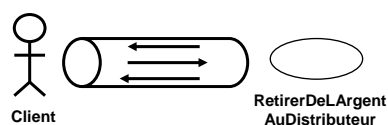


85

Retour sur la relation de communication

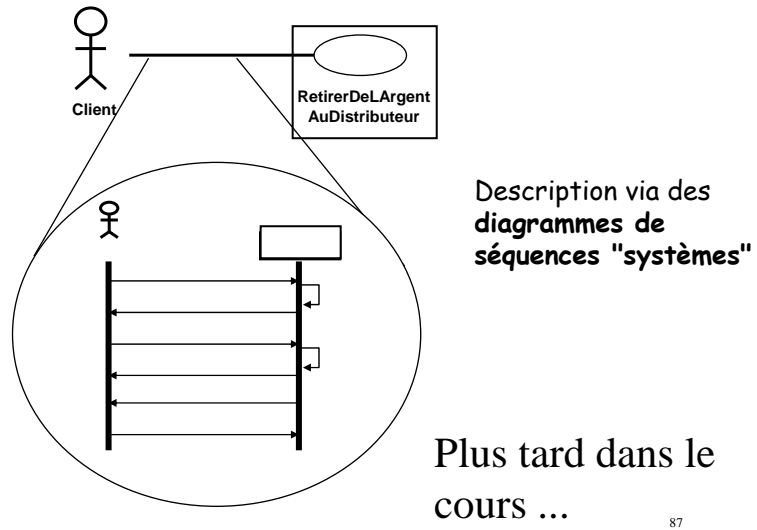


- Association acteur/CU vu comme un canal de communication
- Décrit le comportement du système vu de l'extérieur
- Echange de messages



86

Description de l'interaction

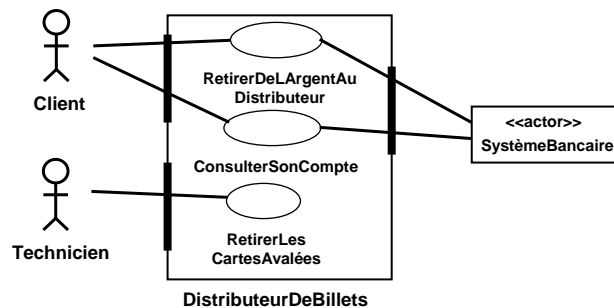


87

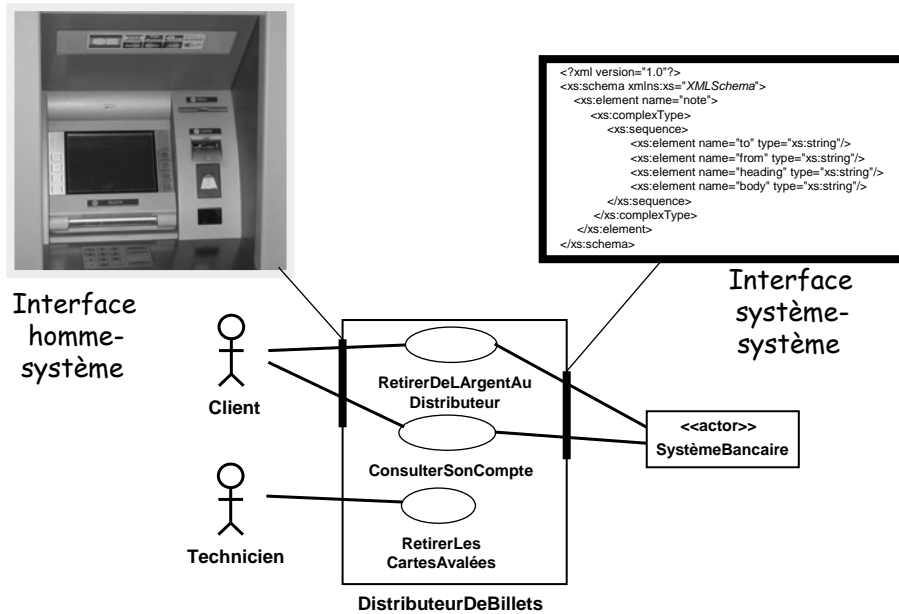
Limites du système et interfaces

Limites du système \Rightarrow interface de communication

- acteur humain \Rightarrow interface homme - système
- acteur logiciel \Rightarrow interface logicielle (e.g. API)

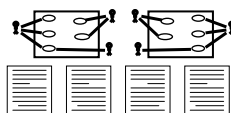


88



Description (par la suite) dans les documents de **spécification externes**

89



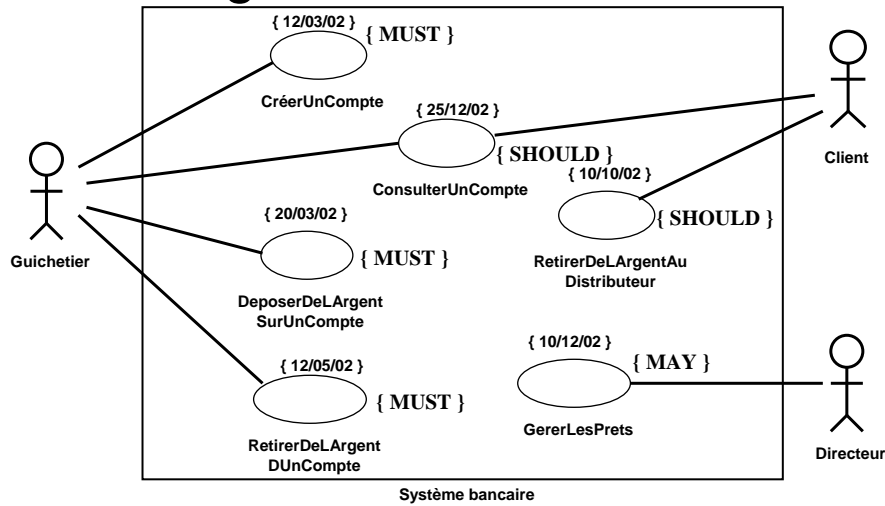
Modèle préliminaire de cas d'utilisation

- Equivalent à définir une **table des matières**
- Permet d'avoir une vision globale du système
- Quelques transparents

- Décorer le diagramme
 - pour préciser des priorités : MUST, SHOULD, MAY
 - pour préciser des risques
 - pour préciser des critères qualités : utilisabilité
 - pour

90

Exemple de diagramme de CU décoré



91

Le Processus Unifié

- (1) Définir le modèle de cas d'utilisation
 - (1.1) Trouver les acteurs
 - (1.2) Décrire brièvement chaque acteur
 - (1.3) Trouver les cas d'utilisation
 - (1.4) Décrire brièvement chaque cas d'utilisation
 - (1.5) Décrire le modèle comme un tout

- (2) Définir des priorités et les risques entre CU

92

Conclusion

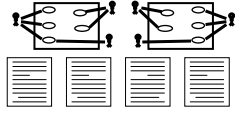
127

ATTENTION

*"Congratulations: Use Cases Have Been
Written, and Are Imperfect"*

[Applying UML and Patterns, Craig
Larman]

128



Modèle préliminaire des cas d'utilisation

- Équivalent à définir une **table des matières** et des résumés pour chaque chapitre
- Pas de règles strictes
- Effectuer les meilleurs regroupement possibles
- Rester simple !
- Structuration possible en termes de paquetages
- Culture d'entreprise

Stabilisation du modèle par **consensus**

grandissant

Pour en savoir un plus...



Chapter

6

USE-CASE MODEL: WRITING
REQUIREMENTS IN CONTEXT

Chapitre gratuit téléchargeable à
http://www.craiglarman.com/book_applying_2nd/Applying_2nd.htm

Pour un template "standard" de description de cas d'utilisation

<http://alistair.cockburn.us/usecases/uctempla.htm>

Pour en savoir encore plus ...



Des livres spécialisés

131

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - ➔ ● *Description des données et de leurs relations*
 - OCL
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement

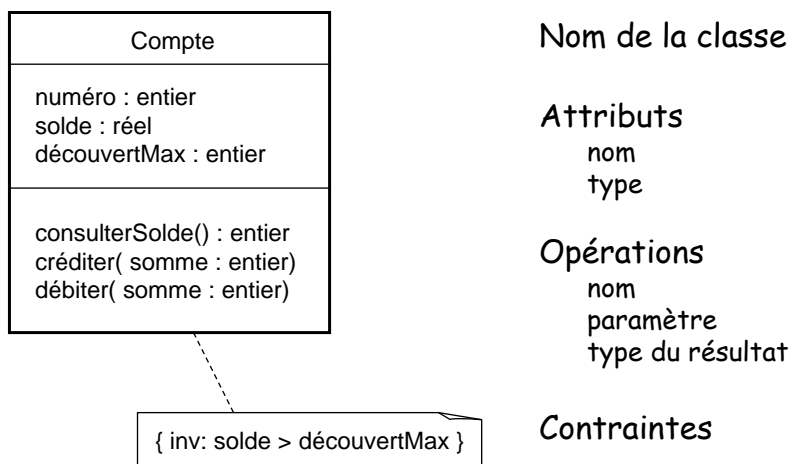
134

Classes et Objets

Concepts de base

135

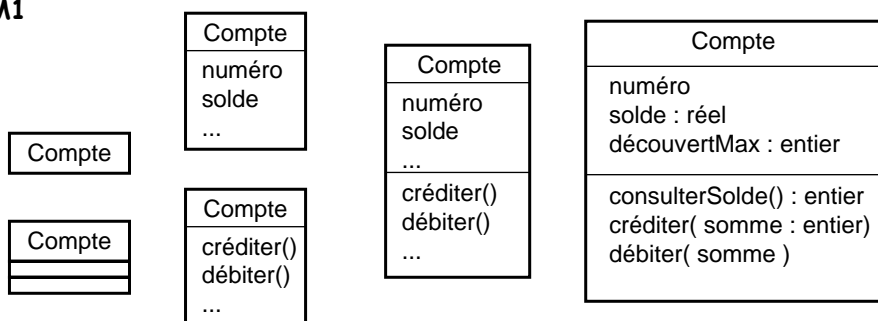
Notation pour les classes



137

Notations simplifiées pour les classes

M1



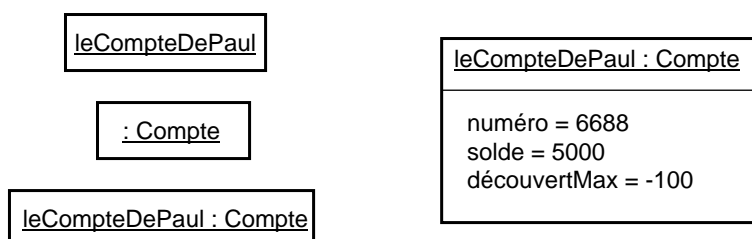
Note de style :

- les noms de classes commencent par une majuscule
- les noms d'attributs et de méthodes commencent par une minuscule

138

Notations pour les objets

M0



Convention :

- les noms d'objets commencent par une minuscule et sont soulignés

139

Classe vs. Objets

Une **classe** spécifie la structure et le comportement d'un ensemble d'objets de même nature

- La structure d'une classe est constante

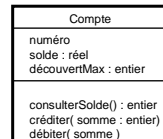


Diagramme de classes

M1

M0

- Des **objets** peuvent être ajoutés ou détruits pendant l'exécution
- La valeur des attributs des objets peut changer

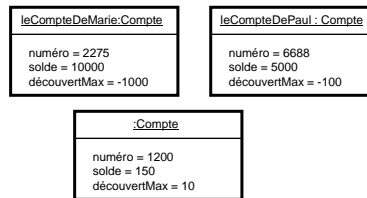
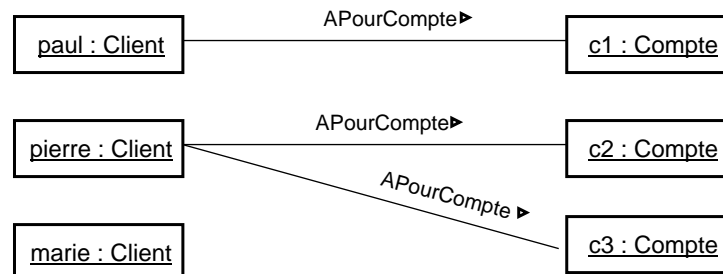


Diagramme d'objets

140

Liens (entre objets)

Un **lien** indique une connexion entre deux objets



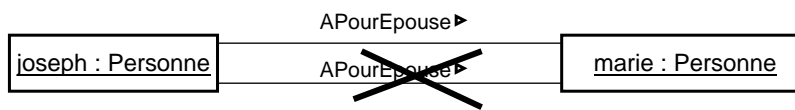
Note de style :

- les noms des liens sont des formes verbales et commencent par une majuscule
- indique le sens de la lecture (ex: « paul APourCompte c1 »)

141

Contrainte sur les liens

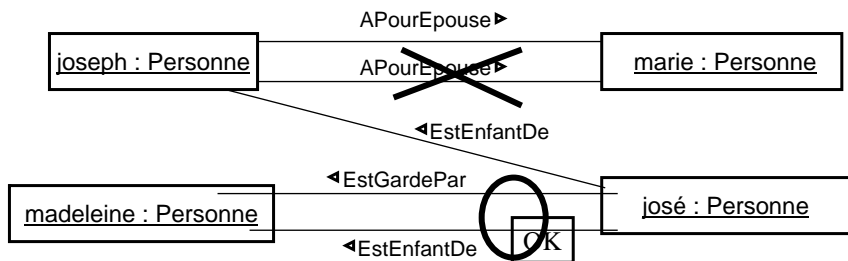
- Au maximum un lien d'un type donné entre deux objets donnés*



142

Contrainte sur les liens

- Au maximum un lien d'un type donné entre deux objets donnés*

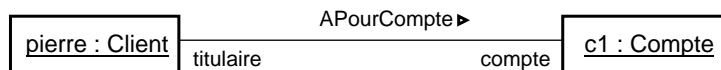


- Contrainte importante pour comprendre les "classes associatives"
- (*) Contrainte pouvant être relâchée via {nonunique} en UML 2.0
- ... voir plus loin les concepts avancés

143

Rôles

Chacun des deux objets joue un **rôle** différent dans le lien



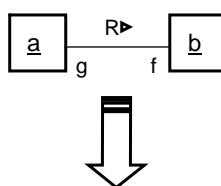
Note de style :

- choisir un groupe nominal pour désigner un rôle
- si un nom de rôle est omis, le nom de la classe fait office de nom

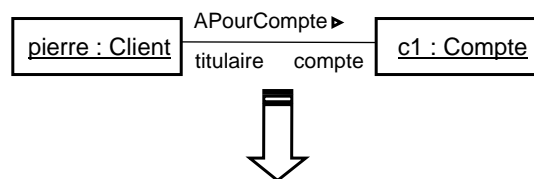
144

3 noms pour 1 concept

utilisations différentes selon le contexte



- a R b
- b "joue le rôle de" f "pour" a
- a "joue le rôle de" g "pour" b

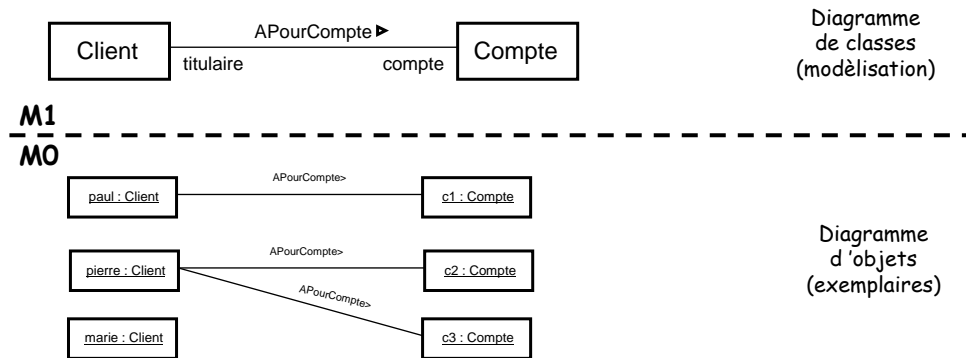


- pierre a pour compte c1
- c1 joue le rôle de compte pour pierre
- pierre joue le rôle de titulaire pour c1

145

Associations (entre classes)

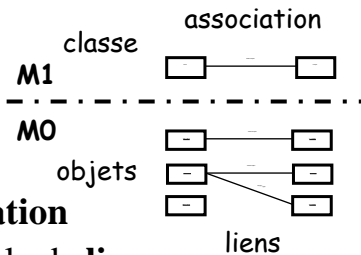
Une **association** décrit un ensemble de liens de même "sémantique"



146

Association vs. Liens

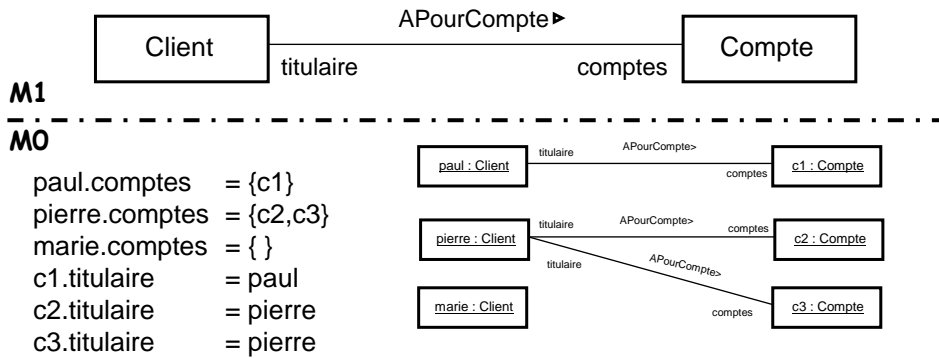
- Un **lien** lie deux **objets**
 - Une **association** lie deux **classes**
 - Un **lien** est une instance d'**association**
 - Une **association** décrit un ensemble de **liens**
- Des **liens** peuvent être ajoutés ou détruits pendant l'exécution, (ce n'est pas le cas des associations)



Le terme "relation" ne fait pas partie du vocabulaire UML

147

Utiliser les rôles pour «naviguer»

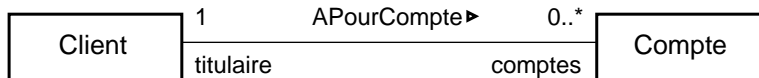


Nommer en priorité les rôles

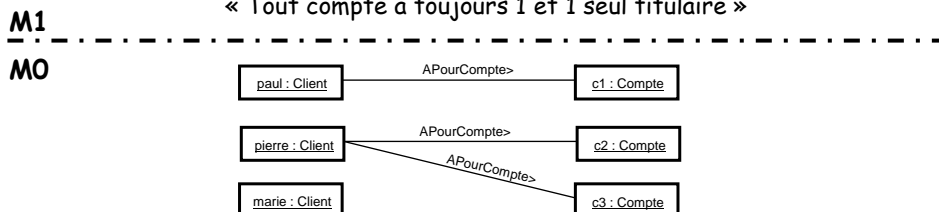
149

Cardinalités d'une association

- Précise combien d'objets peuvent être liés à un seul objet source
- Cardinalité minimale et cardinalité maximale ($C_{min}..C_{max}$)
- Doivent être des constantes

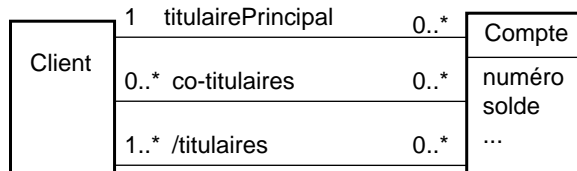


« Tout client a toujours 0 ou plusieurs comptes »
 « Tout compte a toujours 1 et 1 seul titulaire »



150

Contraintes entre associations



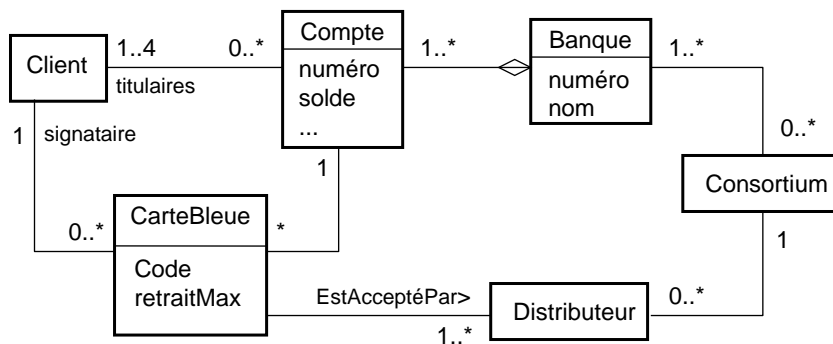
Les cardinalités sont loins d'être suffisantes pour exprimer toutes les contraintes...

... décrire les contraintes en langue naturelle (ou en OCL)

- (1) Un client ne peut pas être à la fois titulaire principal et co-titulaire d'un même compte.
- (2) Les titulaires d'un compte sont le titulaire principal et les co-titulaires le cas échéant

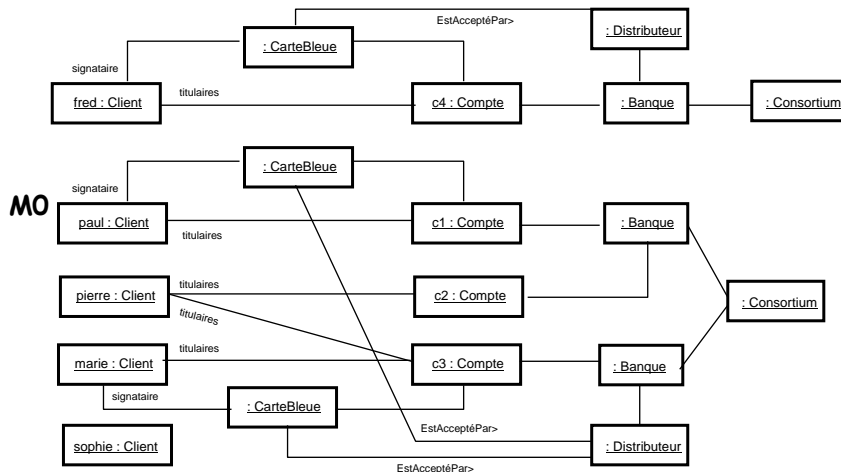
154

Diagramme de classes



155

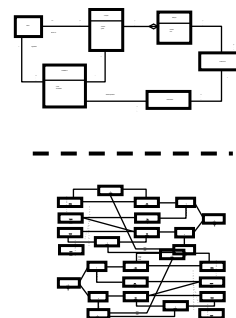
Diagrammes d'objets



156

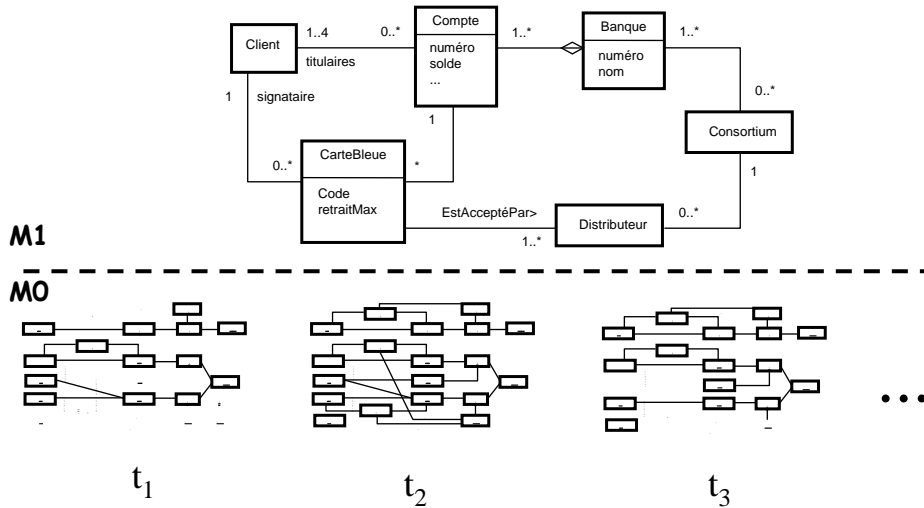
Diagrammes de classes vs. d'objets

- Un diagramme de classes
 - définit l'ensemble de tous les états possibles
 - les contraintes doivent toujours être vérifiées
- Un diagramme d'objets
 - décrit un état possible à un instant t, un cas particulier
 - doit être conforme au modèle de classes
- Les diagrammes d'objets peuvent être utilisés pour
 - expliquer un diagramme de classe (donner un exemple)
 - valider un diagramme de classe (le "tester")



157

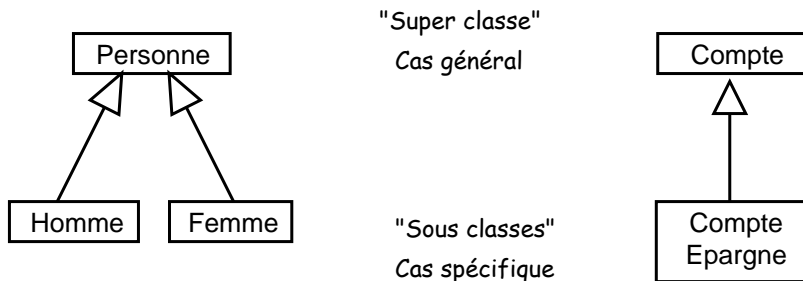
Diagrammes de classes vs. d'objets



158

Généralisation / Spécialisation

Une classe peut être la généralisation d'une ou plusieurs autres classes. Ces classes sont alors des spécialisations de cette classe.



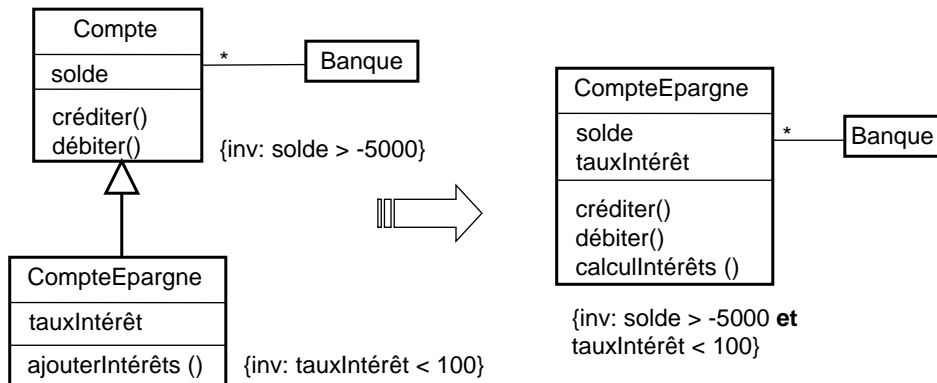
Deux points de vue lié (en UML) :

- relation d'héritage
- relation de sous-typage

159

Relation d'héritage

Les sous-classes « *héritent* » des propriétés des super-classes (attributs, méthodes, associations, contraintes)

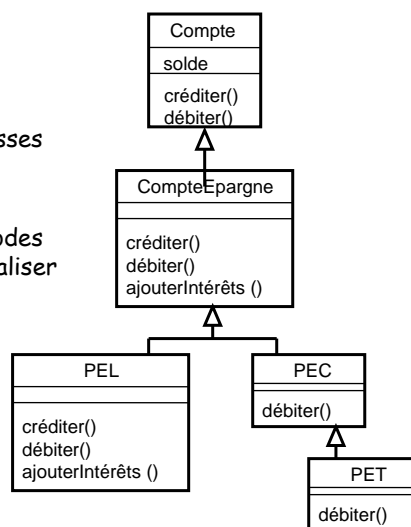


160

Relation d'héritage et redéfinitions

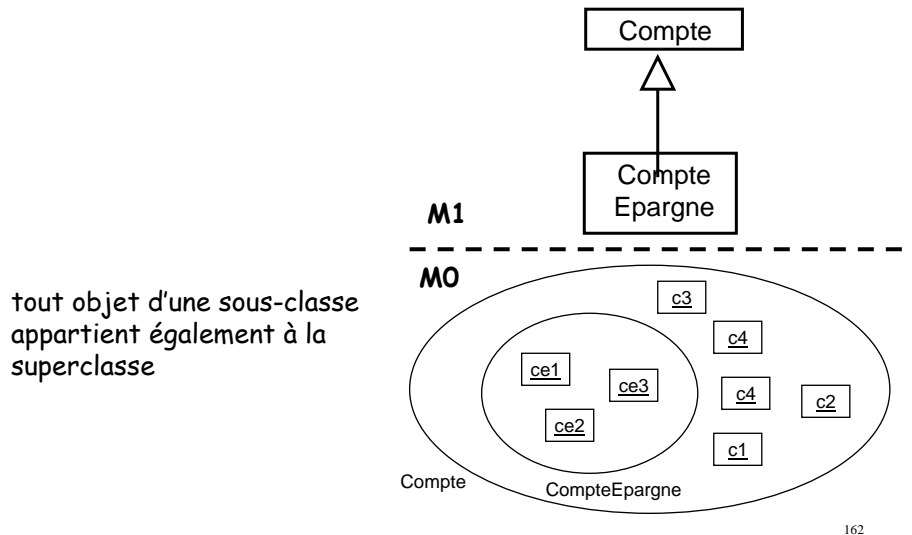
Une opération peut être "redéfinie" dans les sous-classes

Permet d'associer des méthodes spécifiques à chaque pour réaliser une même opération

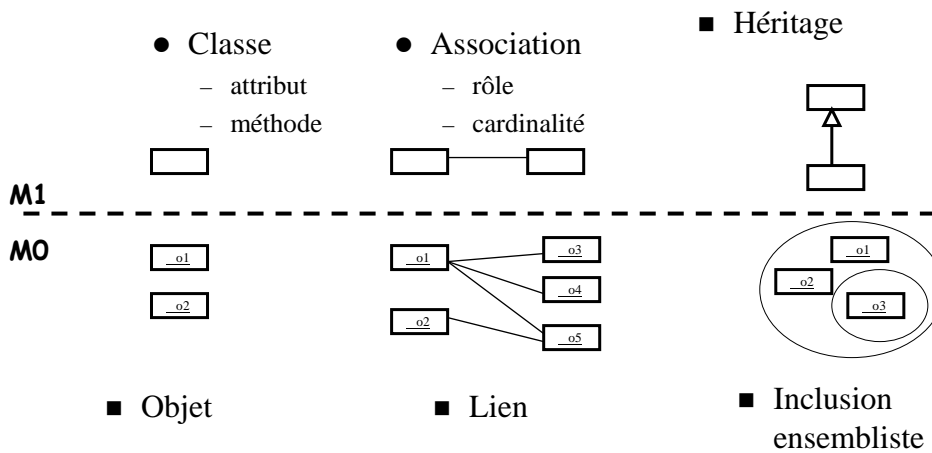


161

Relation de sous typage, Vision ensembliste



Synthèse des concepts de base



Classes et Objets

Utilisation avancée

164

UML ? ... une question de style et
de contexte

S'adapter ...

- au niveau d'abstraction
- au domaine d'application
- aux outils utilisés
- aux savants et ignorants
- ingénierie vs. retro-ingénierie

165

+ # - Visibilité des éléments

- Restreindre l'accès aux éléments d'un modèle
 - Contrôler et éviter les dépendances entre classes et paquetages
- | | | |
|---|---------|--|
| + | public | visible |
| # | protégé | visible dans la classe et ses sous-classes |
| - | privé | visible dans la classe uniquement |
| ~ | package | visible dans la package uniquement |
- Utile lors de la conception et de l'implémentation, pas avant !
 - N'a pas de sens dans un modèle conceptuel (abstrait)
 - N'utiliser que lorsque nécessaire
 - La sémantique exacte dépend du langage de programmation !

166

Déclaration d'attributs

[/] [*visibilité*] *nom* [: *type*] [*card ordre*] [= *valeur-initiale*] [{ *props...* }]

exemples:

```
age
+age
/age
- solde : Integer = 0
# age : Integer [0..1]
# numsecu : Integer {frozen}
# motsClés : String [*] {addOnly}
nbPersonne : Integer
```

- Adapter le niveau de détail au niveau d'abstraction

167

Déclaration d'opérations

[/] [*visibilité*] *nom* [(*params*)] [: *type*] [{ *props...* }]
params := [*in* / *out* / *inout*] *nom* [: *type*] [=*defaut*] [{
props... }]

```
/getAge()  
+ getAge() : Integer  
- updateAge( in date : Date ) : Boolean  
# getName() : String [0..1]  
+getAge() : Integer {isQuery}  
+addProject() : { concurrency = sequential }  
+addProject() : { concurrency = concurrent  
}  
+main( in args : String [*] {ordered} )
```

- Adapter le niveau de détail au niveau d'abstraction

168

Attributs dérivés

- Attributs dont la valeur peut être déduite d'autres éléments du modèle
 - e.g. *âge* si l'on connaît la date de naissance
 - notation : /age
- En termes d'analyse, indique seulement une **contrainte** entre valeurs et non une indication de ce qui doit être calculé et ce qui doit être mémorisé

169

Représentation des invariants

- Des contraintes peuvent être ajoutées aux éléments du modèle UML
 - notation entre { }
- Invariants = Propriétés vraies pour l'ensemble des instances de la classe
 - dans un état stable, chaque instance doit vérifier les invariants de sa classe
 - exprimés à l'aide d'OCL
 - Object Constraint Language
 - e.g. {solde >= plancher}

Compte {solde>=plancher}
solde: Somme plancher: Somme
créditer (Somme) débiter (Somme)

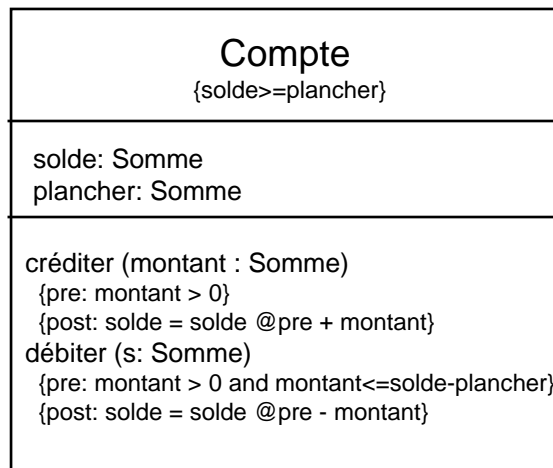
170

Représentation des pré/post conditions

- Préconditions
 - {«precondition» *expression booléenne* OCL}
 - Abrégé en: {pre: *expression booléenne* OCL}
- Postconditions
 - {«postcondition» *expression booléenne* OCL}
 - Abrégé en: {post: *expression booléenne* OCL}
 - Operateur « valeur précédente » (idem *old Eiffel*):
 - *expression OCL* @pre

171

Etre abstrait et précis avec UML

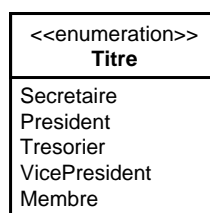
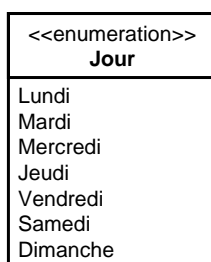


Analyse précise ou “analyse par contrat”

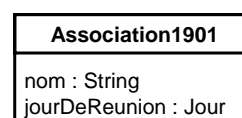
172

Enumération

● Définition



■ Utilisation

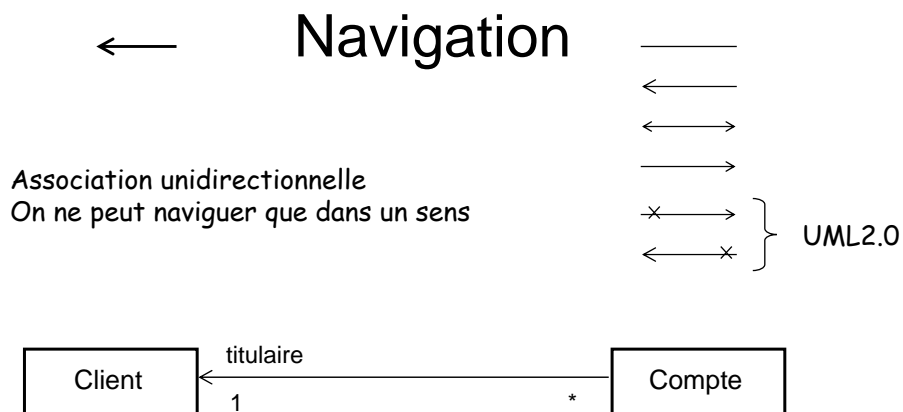


173

Raffinement du concept d'association

- Association uni/bi directionnelle
- Composition
- Aggrégation
- {frozen}, {addonly}, {ordered}, {nonunique}
- Classes associatives
- Associations qualifiées

174

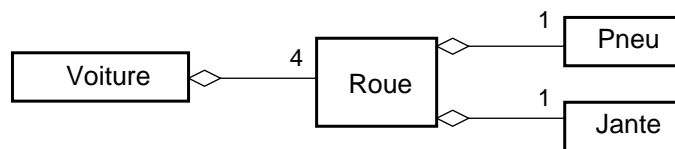


A priori, que dans les diagrammes de spécifications et d'implémentation
En cas de doute, ne pas mettre de flèche !!!

175

◇ Composition

Notion intuitive de "composants" et de "composites"



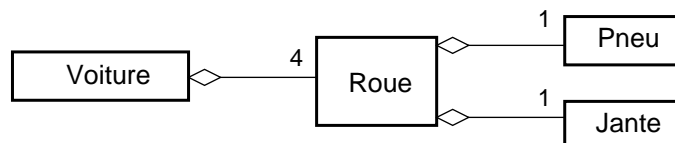
composition =
cas particulier d'association
+ contraintes décrivant la notion de "composant"...

176

◇ Composition

Contraintes liées à la composition :

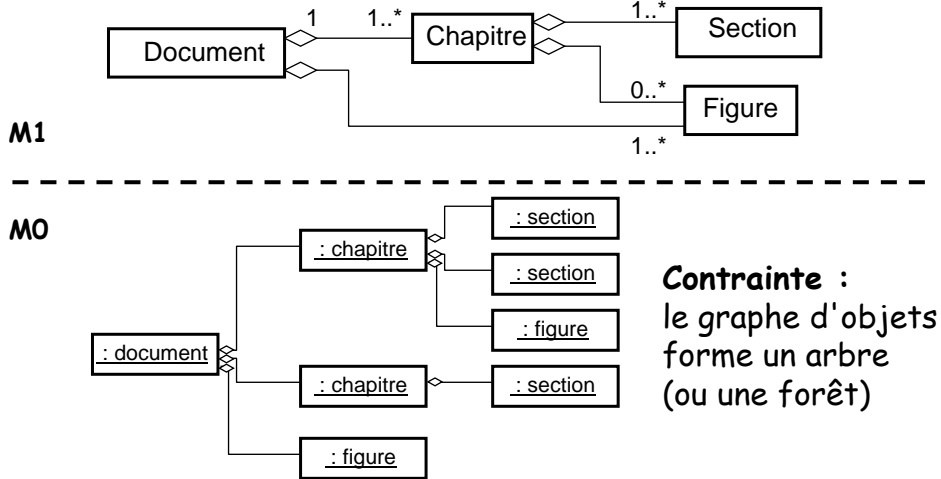
1. Un objet *composant* ne peut être que dans 1 seul objet *composite*
2. Un objet *composant* n'existe pas sans son objet *composite*
3. Si un objet composite est détruit, ses composants aussi



Dépend de la situation modélisée !
(Ex: vente de voitures vs. casse)

177

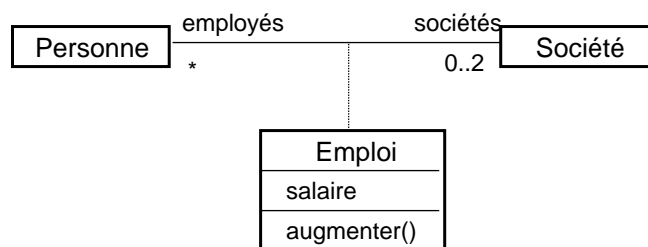
Composition



179

Classes association

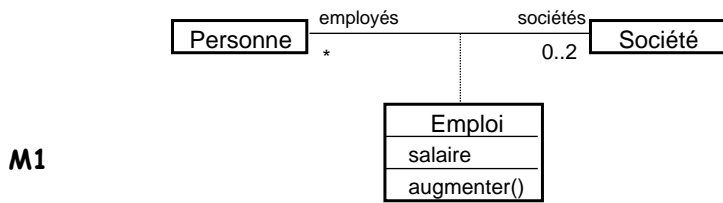
Pour associer des attributs et/ou des méthodes aux associations
=> **classes associations**



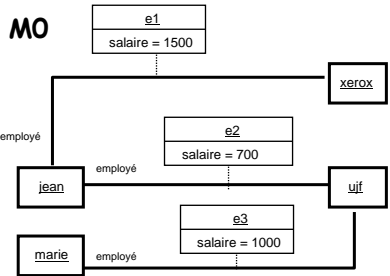
Le nom de la classe correspond au nom de l'association
(problème: il faut choisir entre forme nominale et forme verbale)

186

Classes association



M1



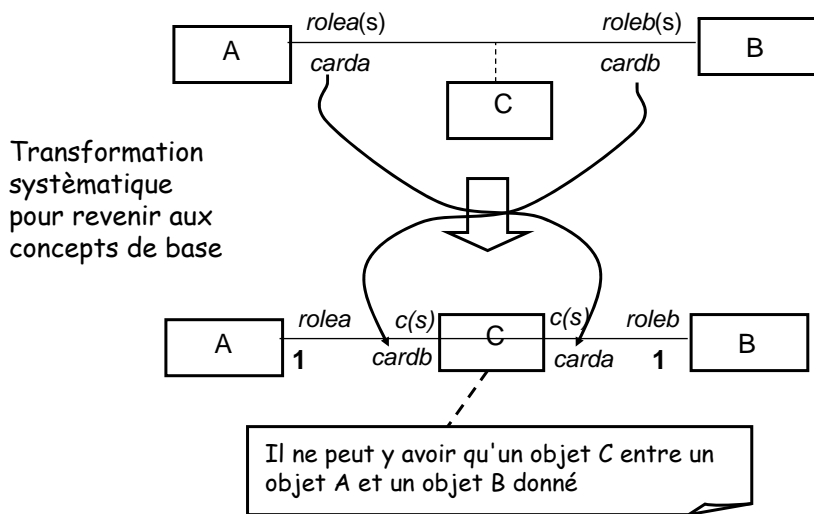
Le salaire est une information correspondant

- ni à une personne,
- ni à une société,

mais à un emploi (un couple personne-société).

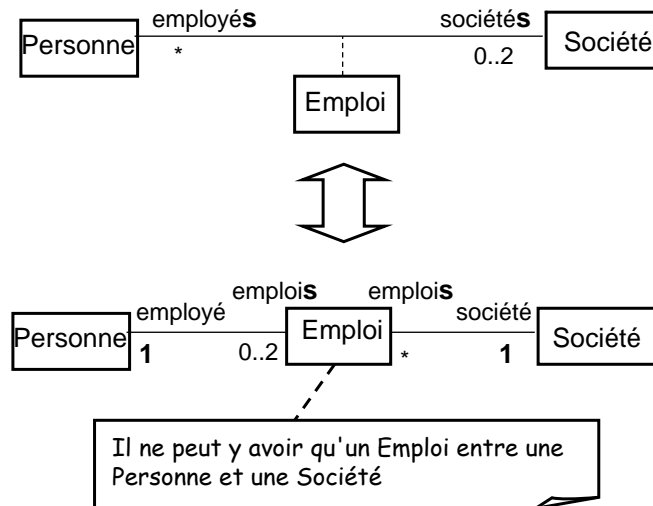
187

Classes association : traduction



190

Classes association : traduction



191

Associations qualifiées

Un **qualifieur** est un attribut (ou un ensemble d'attributs) dont la valeur sert à déterminer l'ensemble des instances associées à une instance via une association.



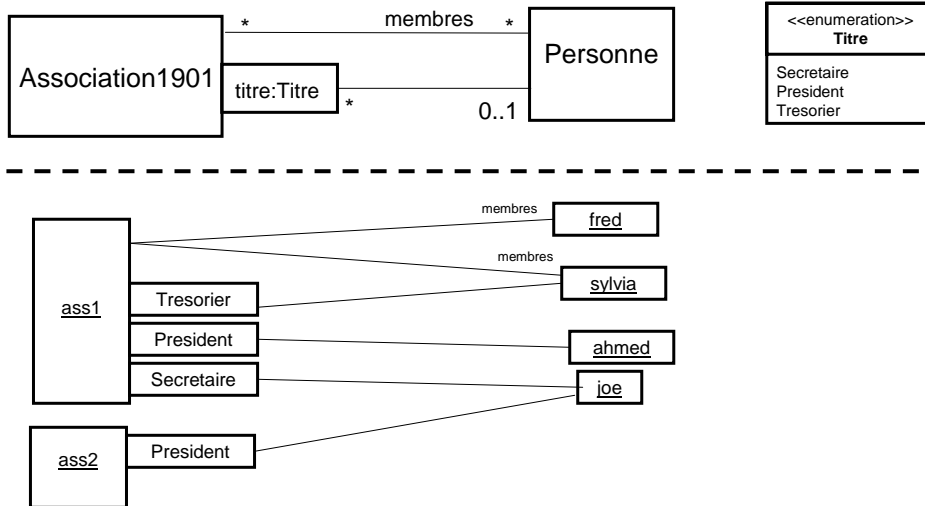
"Pour un répertoire, à un nom donné on associe qu'un fichier (ou 0 s'il existe aucun fichier de ce nom dans ce répertoire)."

Correspond à la notion intuitive d'index absente ci-dessous



202

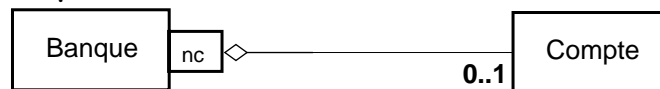
Exemple



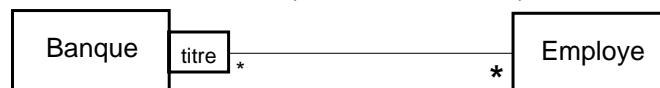
203

Cardinalité des Associations Qualifiées

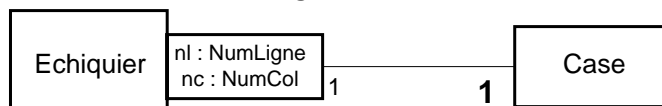
Cas classique: cardinalité 0..1



Cas plus rare: cardinalité * (pas de contrainte particulière exprimée)



Cas plus rare: cardinalité 1 (généralement c'est une erreur)



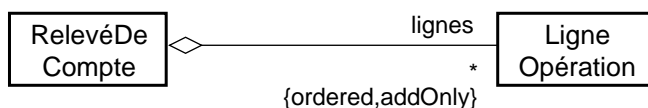
0 comme cardinalité minimale, sauf si le domaine de l'attribut qualifieur est fini et toutes les valeurs ont une image.

204

Contraintes prédéfinies sur les associations

Par exemple :

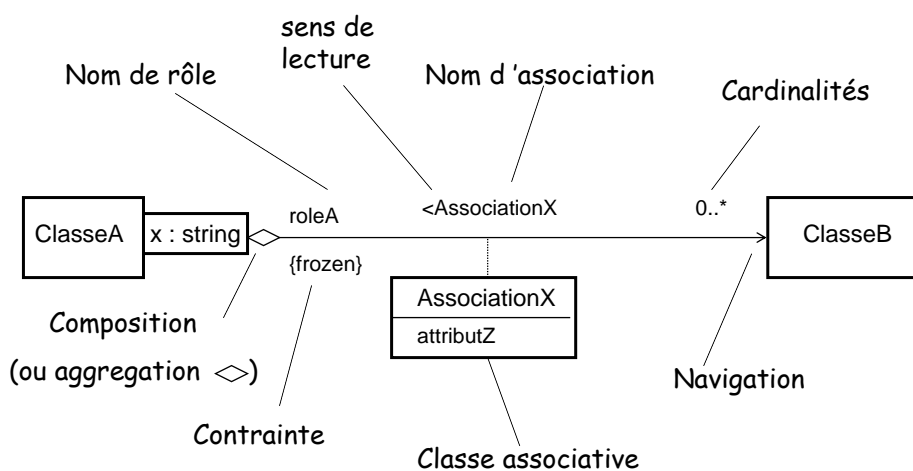
- { ordered } : les éléments de la collection sont ordonnés
- { nonUnique } : répétitions possibles (UML2.0)
- { frozen } : fixé lors de la création de l'objet, ne peut pas changer
- { addOnly } : impossible de supprimer un élément



Il est possible de définir de nouvelles contraintes

208

Synthèse sur les associations



215

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - OCL
 - ➔ ● *Structuration en paquetages*
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement

287

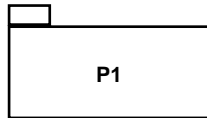
Notion de package

- Élément structurant les classes
 - Modularisation à l'échelle supérieure
 - Un package partitionne l'application :
 - Il référence ou se compose des classes de l'application
 - Il référence ou se compose d'autres packages
 - Un package régleme la visibilité des classes et des packages qu'il référence ou le compose
 - Les packages sont liés entre eux par des liens d'utilisation, de composition et de généralisation
 - Un package est la représentation informatique du contexte de définition d'une classe

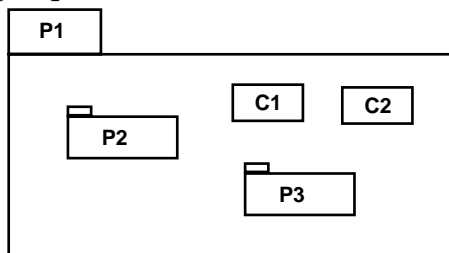
288

Représentation d'un package

- Vue graphique externe



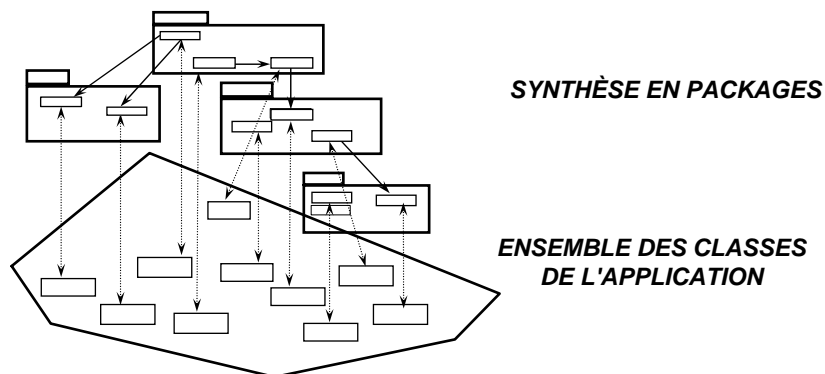
- Vue graphique externe et interne



289

Partitionnement d'une application

- Définition de vues partielles d'une application

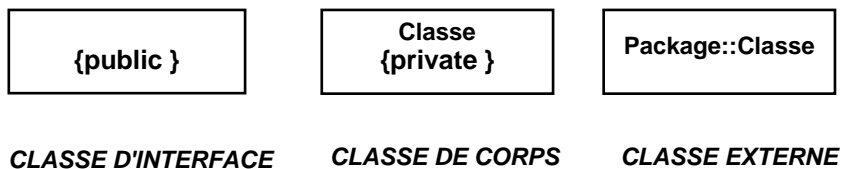


N.B.: une classe appartient à un et un seul package

290

Visibilité dans un package

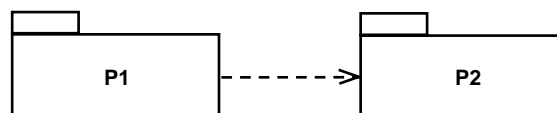
- Réglementation de la visibilité des classes
 - **Classes de visibilité publique :**
 - classes utilisables par des classes d'autres packages
 - **Classes de visibilité privée :**
 - classes utilisables seulement au sein d'un package
- Représentation graphique



291

Utilisation entre packages

- Définition
 - Il y a utilisation entre packages si des classes du package utilisateur accèdent à des classes du package utilisé
 - Pour qu'une classe d'un package p1 puisse utiliser une classe d'un package p2, il doit y avoir au préalable une déclaration **explicite** de l'utilisation du package p2 par le package p1
- Représentation graphique

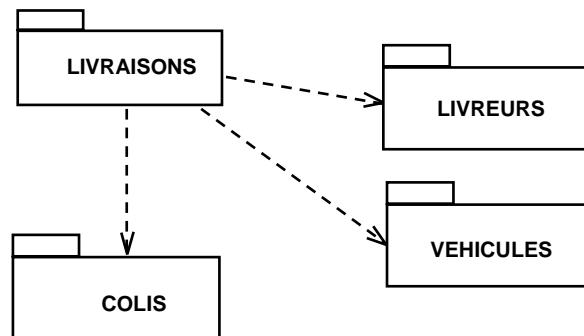


Vue externe du package P1

292

Utilisation entre packages

- Exemple (vue externe du package livraisons)



293

Stéréotypes de package

- Stéréotype « global »

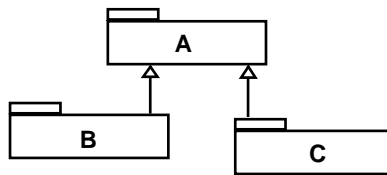


- Tous les packages du systèmes ont une dépendance avec ce système
- Ne pas abuser de ce stéréotypes ;)

294

Héritage entre packages

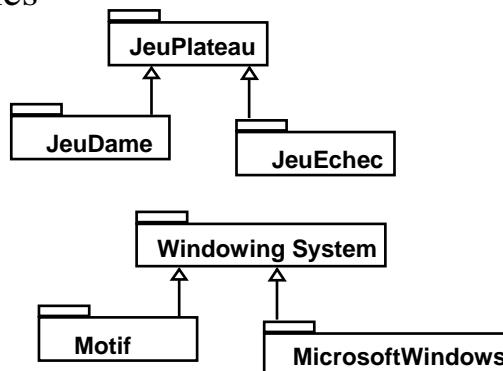
- Package spécifique B doit être conforme à l'interface du package A
- Intérêt : substitution



295

Héritage entre packages

- Exemples



296

Utilité des packages

- Réponses au besoin
 - Contexte de définition d'une classe
 - Unité de structuration
 - Unité d'encapsulation
 - Unité d'intégration
 - Unité de réutilisation
 - Unité de configuration
 - Unité de production

297

Structuration par packages (vs) décomposition hiérarchique

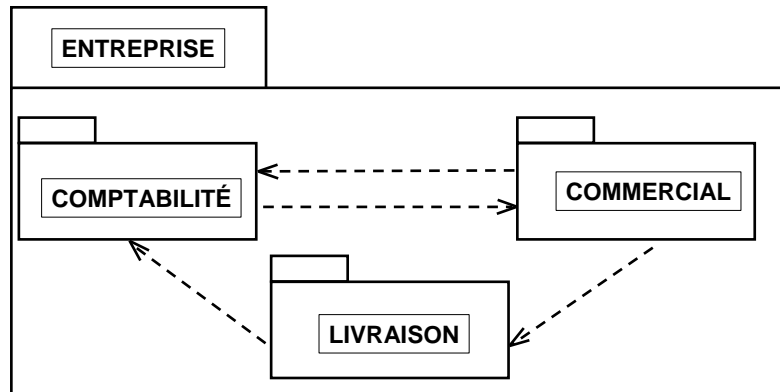
- ◆ **Pour les grands systèmes, il est nécessaire de disposer d'une unité de structuration :**
 - ❖ À un niveau supérieur,
 - ❖ Plus souple que :
 - ❖ La composition de classe
 - ❖ Le référencement de packages

**=> domaines de structuration :
*Packages décomposables en packages***

298

Exemple : Package entreprise

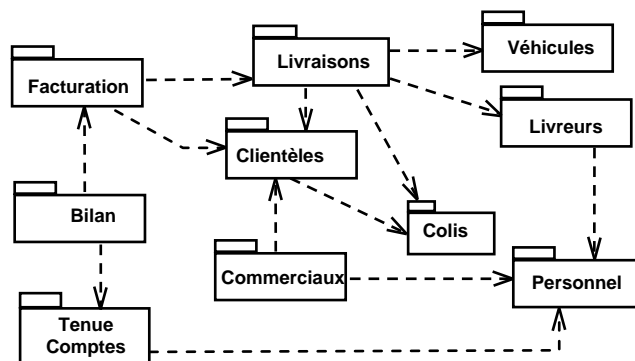
- Exemple de composition



299

Exemple : Package entreprise

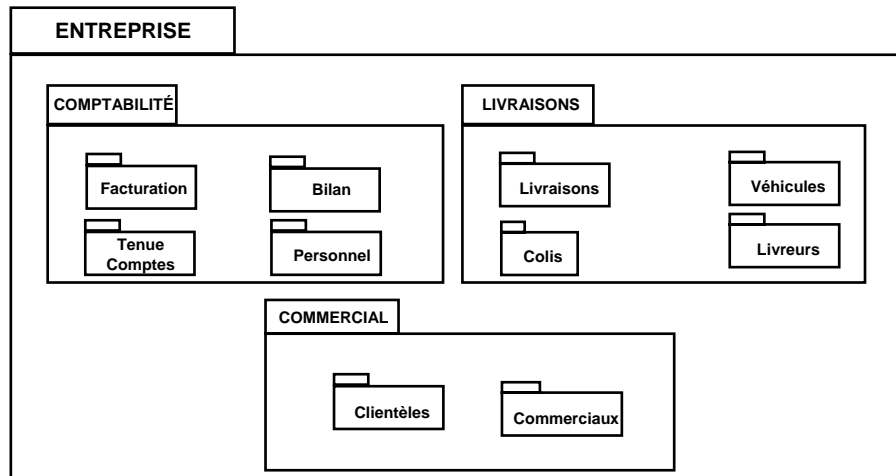
- Ensemble des packages terminaux de l'application



300

Exemple : Package entreprise

- Composition des packages en sous-packages



301

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - OCL
 - Structuration en paquetages
 - ➔ – *Aspects dynamiques du système (comportemental)*
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement

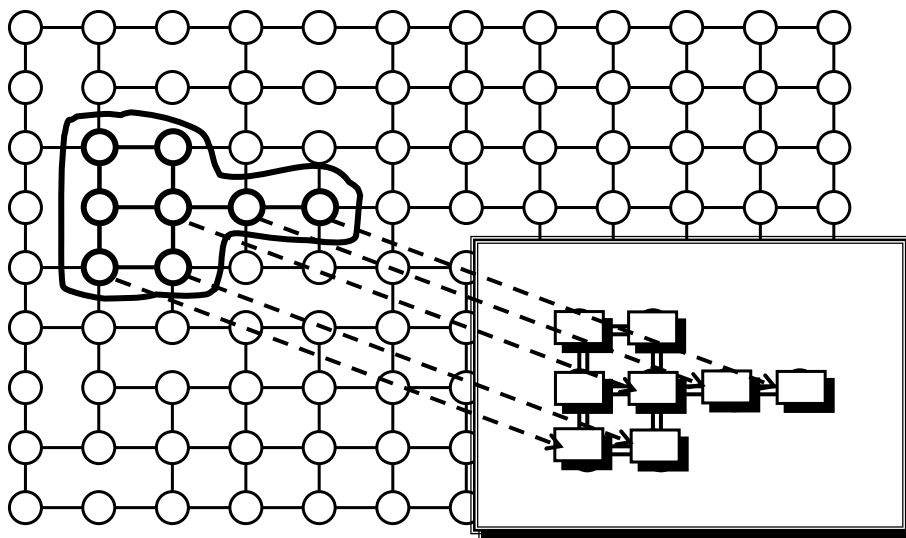
302

Aspects dynamiques du système

- Jusqu'ici, système décrit *statiquement*:
 - Décrivent les messages (méthodes ou opérations) que les instances des classes peuvent recevoir mais ne décrivent pas l'émission de ces messages
 - Ne montrent pas le lien entre ces échanges de messages et les processus généraux que l'application doit réaliser
- Il faut maintenant décrire comment le système évolue dans le temps
- On se focalise d'abord sur les *collaborations* entre objets.
Rappel :
 - objets : simples
 - gestion complexité : par collaborations entre objets simples

303

Collaborations (au niveau instances)



Selon T. Reenskaug...

304

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - OCL
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - *Diagramme de séquences (scénarios)*
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement



305

Diagrammes de séquences (scénarios)

- Représentation des interactions entre acteurs et objets
- Vision temporelle d'une interaction
 - Chaque objet est symbolisé par une barre verticale
 - Le temps s'écoule de haut en bas, de sorte que la numérotation des messages est optionnelle.
 - Diagramme dual du diagramme de collaboration
- Souvent utilisé pour représenter une instance de cas d'utilisation
- De manière plus générale, représentation temporelle d'une interaction
 - Bien adapté pour de longues séquences
 - Ne visualise pas les liens
 - Complémentaire du diagramme de collaboration

306

Diagrammes de séquences (scénarios)

- Dérivés des scénarios de OMT :
 - Montrent des exemples de coopération entre objets dans la réalisation de processus de l'application
 - Illustrent la dynamique d'enchaînement des traitements à travers les messages échangés entre objets
 - le temps est représenté comme une dimension explicite
 - en général de haut en bas
- Les éléments constitutifs d'un scénario sont :
 - Un ensemble d'objets (et/ou d'acteurs)
 - Un message initiateur du scénario
 - La chronologie des messages échangés subséquentement
 - Les contraintes de temps (aspects temps réel)

307

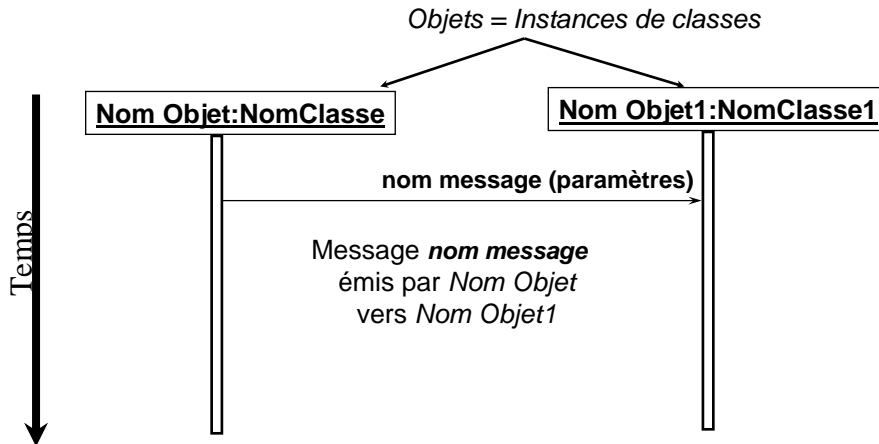
Messages et Stimuli

- Un Stimulus est une communication entre deux instances, dans le but de déclencher une action.
- Un Message est une spécification de Stimulus, qui définit les rôles de l'émetteur et du récepteur.

308

Syntaxe graphique

- Objets et messages



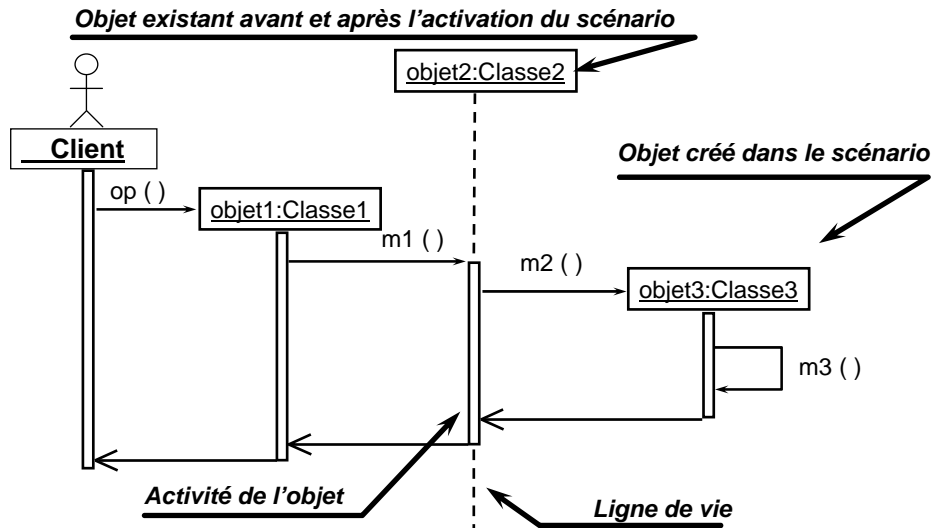
309

Ligne de vie et activation

- La «ligne de vie» représente l'existence de l'objet à un instant particulier
 - Commence avec la création de l'objet
 - Se termine avec la destruction de l'objet
- L'activation est la période durant laquelle l'objet exécute une action lui-même ou via une autre procédure

310

Notation



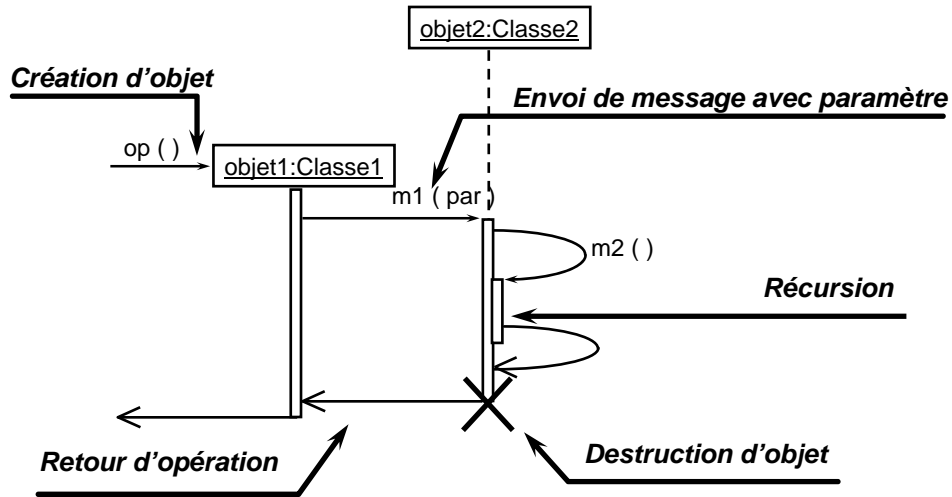
311

Messages

- Communication entre objets
 - Des paramètres
 - Un retour
- Cas particuliers
 - Les messages entraînant la construction d'un objet
 - La récursion
 - Les destructions d'objets

312

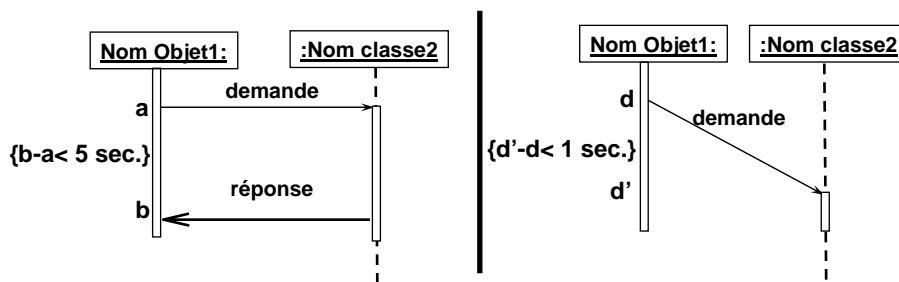
Notations



313

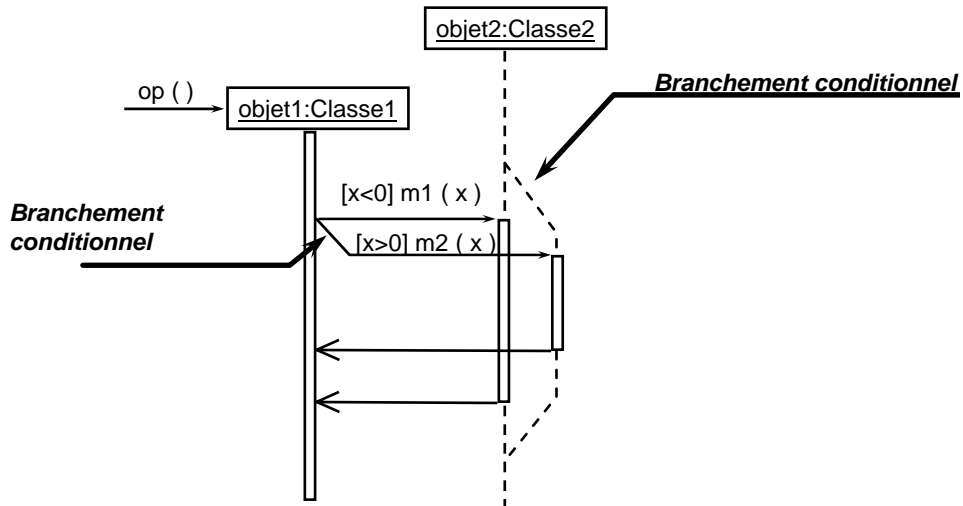
Aspects asynchrones et temps réel

- Lecture du scénario et chronologie
 - Un scénario se lit de haut en bas dans le sens chronologique d'échange des messages.
 - Des contraintes temporelles peuvent être ajoutées au scénario



314

Représentation de conditionnelles



315

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - **Diagramme de collaborations (entre objets)**
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement



316

Diagrammes de collaboration

- Les scénarios et diagrammes de collaboration:
 - Montrent des exemples de coopération des objets dans la réalisation de processus de l'application
- Les scénarios :
 - Illustrent la dynamique d'enchaînement des traitements d'une application en introduisant la dimension temporelle
- Les diagrammes de collaboration
 - Dimension temporelle représentée par numéros de séquence : définition d'un ordre partiel sur les opérations
 - Représentation des objets et de leurs relations
 - Utilisent les attributs et opérations

317

Diagrammes de collaboration

- Représentation d'une collaboration entre rôles
 - Booch : Société d'objets collaborant (UML 1.5 : de rôles communicants)
- Représentation spatiale d'une interaction
 - Mise en avant de la structure
 - Représentation des structures complexes (récursives par exemple)
- Pas d'axe temporel
 - Diagramme dual du diagramme de séquence
- Des rôles ou des objets dans une situation donnée
- Des liens relient les objets qui se connaissent
- Les messages échangés par les objets sont représentés le long de ces liens
- L'ordre d'envoi des messages est matérialisé par un numéro de séquence

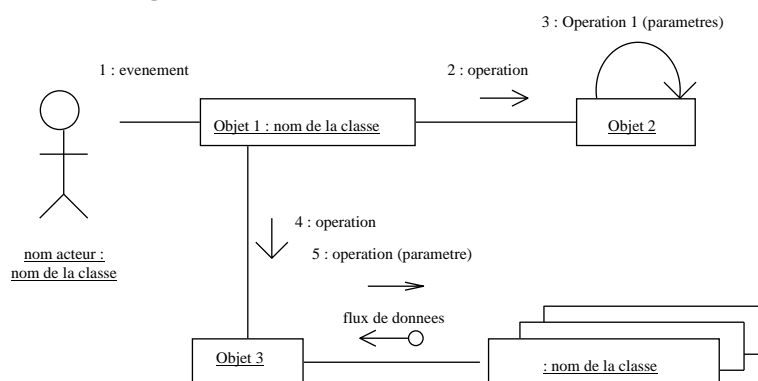
318

Diagramme de collaboration

- Représentation de collaborations de rôles
 - Description de la réalisation d'un Classifier ou d'une opération
 - Ensemble de rôles (ClassifierRoles + AssociationRoles)
 - / ClassifierRoleName : ClassifierName
- Représentation de collaborations d'instances
 - Ensemble d'objets (Objets + Liens)
 - Conforme à un diagramme de collaboration de rôles
- Représentation d'interactions
 - Ensembles d'objets + Stimuli (Instances de Messages)

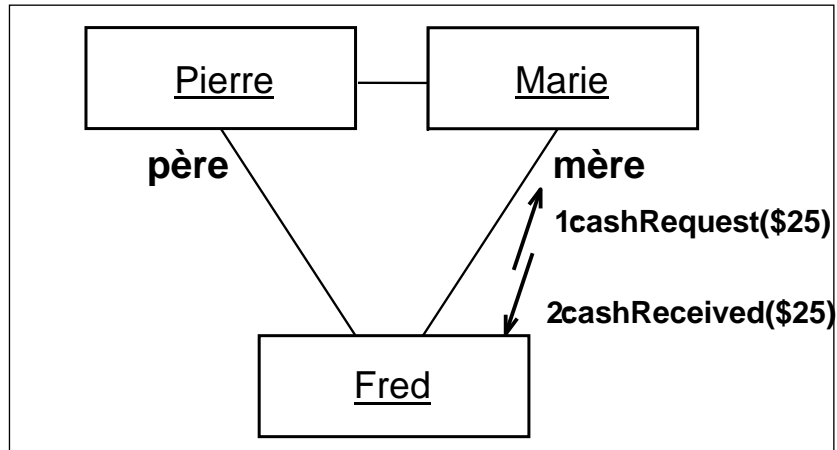
319

Diagramme de collaboration



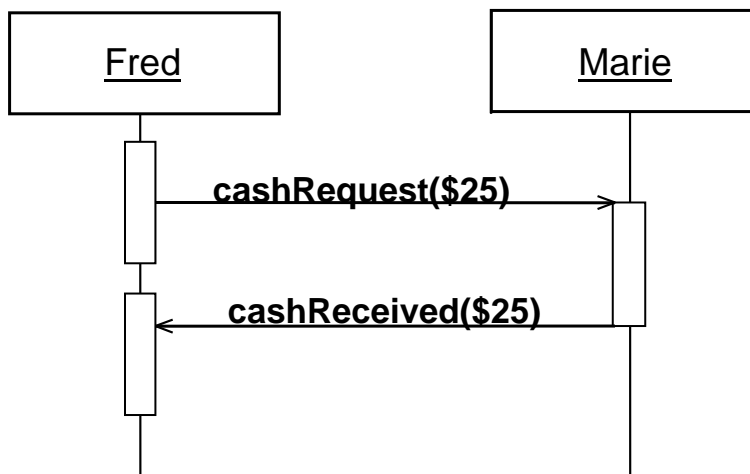
320

Représentation d'une collaboration (niveau instance)



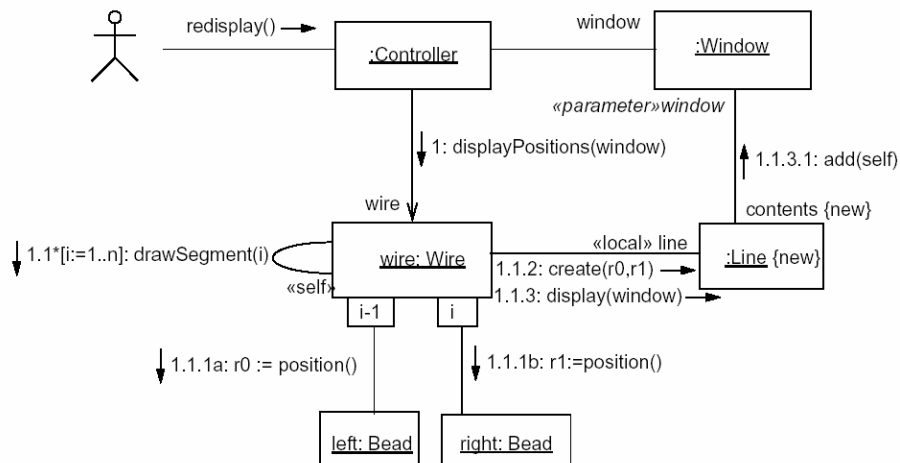
321

Equivalent au diagramme de séquence:



322

Diagramme de collaboration (niveau instances)



323

Le concept de messages

- L'unité de communication entre rôles
- Concept très général pouvant être mis en œuvre suivant de nombreuses variantes (les stimuli)
- Regroupe les flots de contrôle et les flots de données

324

Utilisation des diagrammes de collaboration

- Ils peuvent être attachés à :
 - Une classe
 - Une opération
 - Un use-case
- Ils s'appliquent
 - En spécification
 - En conception (illustration de *design patterns*)

326

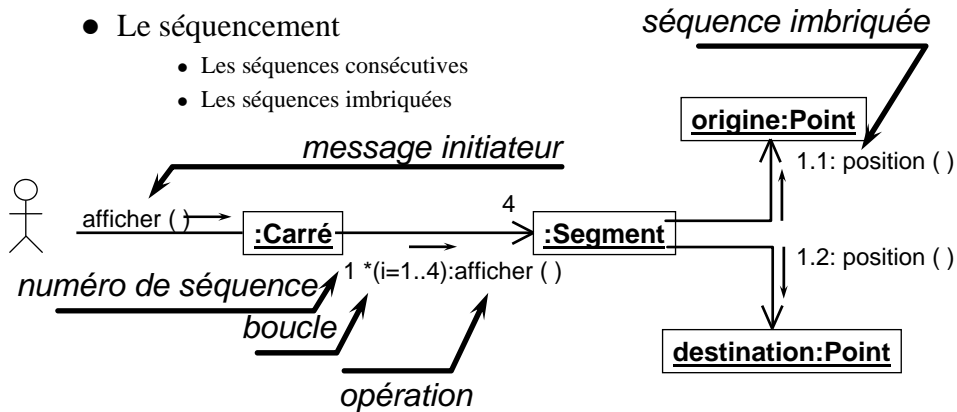
Eléments constitutifs

- Un contexte contenant les éléments mis en jeu durant l'opération :
 - Un acteur
 - Un ensemble d'objets, d'attributs et de paramètres
 - Des relations entre ces objets
- Des interactions
 - Des messages
 - Un message initiateur du diagramme provenant d'un
 - Acteur de l'application,
 - Objet de l'application.
 - Les numéros de séquence des messages échangés entre les objets de cet ensemble suite au message initiateur

327

Syntaxe graphique

- Les messages
 - Opérations
 - Réception d'événements
- Le séquençage
 - Les séquences consécutives
 - Les séquences imbriquées



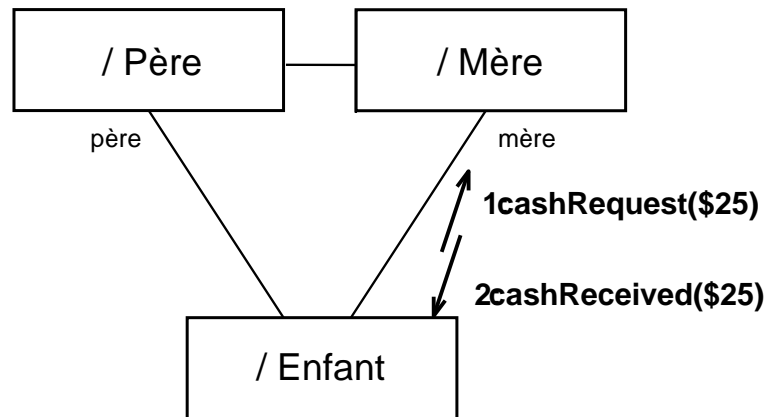
328

Questions auxquelles répondent les collaborations

- Quel est l'objectif ?
- Quels sont les objets ?
- Quelles sont leurs responsabilités ?
- Comment sont-ils interconnectés ?
- Comment interagissent-ils ?

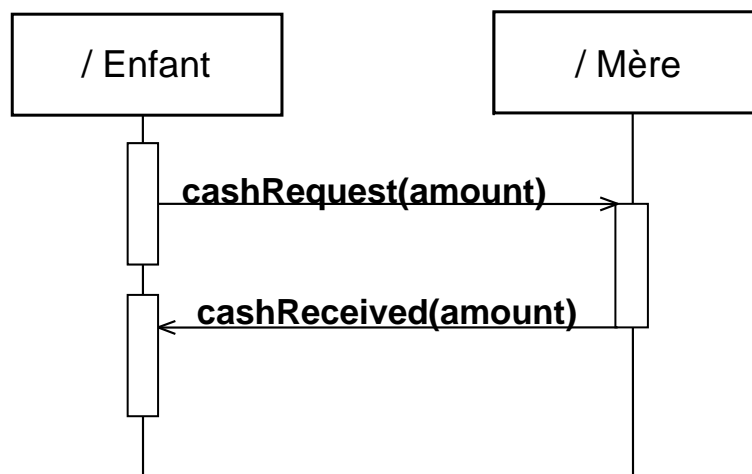
329

Collaboration de Niveau Spécification *un exemple simple*



330

Diagramme de séquence de niveau spécification



331

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - *Diagramme d'états-transitions (Harel)*
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement



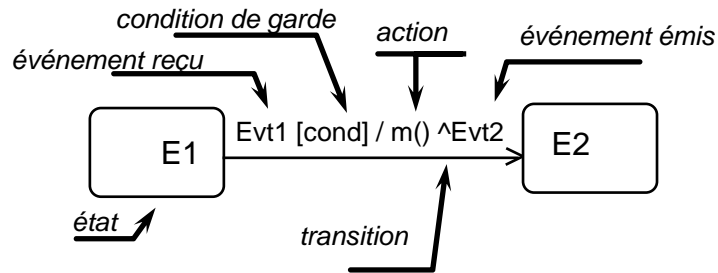
332

Les diagrammes d'états

- Attachés à une classe
 - Généralisation des scénarios
 - Description systématique des réactions d'un objet aux changements de son environnement
- Décrivent les séquences d'états d'un objet ou d'une opération :
 - En réponse aux «stimulis» reçus
 - En utilisant ses propres actions (transitions déclenchées)
- Réseau d'états et de transitions
 - Automates étendus
 - Essentiellement *Diagrammes de Harel (idem OMT)*

333

Syntaxe graphique: diagramme d'états



Syntaxe :

ÉvénementReçu (param : type, ...) [condition de garde] / Action ^ÉvénementsEmis

334

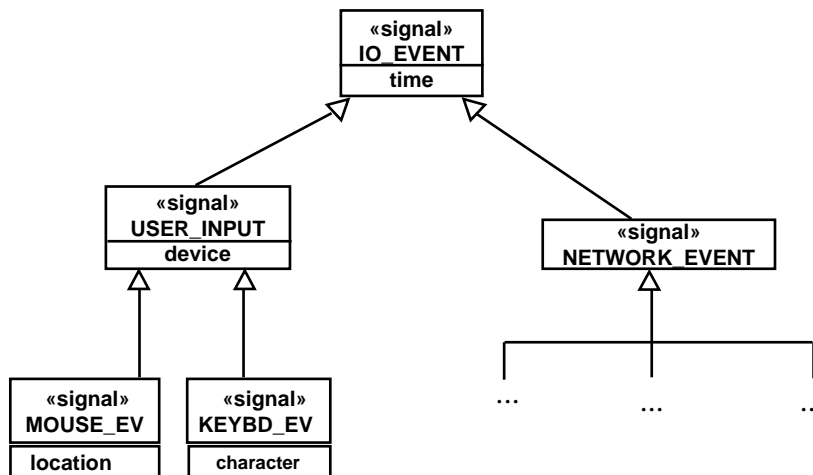
Notion d'événements

- Stimulis auxquels réagissent les objets
 - Occurrence déclenchant une transition d'état
- Abstraction d'une information instantanée échangée entre des objets et des acteurs
 - Un événement est instantané
 - Un événement correspond à une communication unidirectionnelle
 - Un objet peut réagir à certains événements lorsqu'il est dans certains états.
 - Un événement appartient à une *classe d'événements* (classe stéréotypée «signal»).

335

Les événements

- Les événements sont considérés comme des objets



336

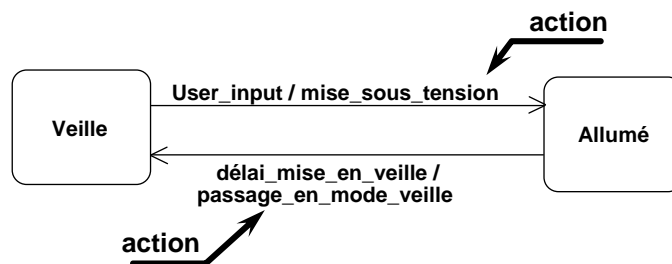
Typologie d'événements

- Réalisation d'une condition arbitraire
 - transcrit par une condition de garde sur la transition
- Réception d'un signal issu d'un autre objet
 - transcrit en un événement déclenchant sur la transition
- Réception d'un appel d'opération par un objet
 - transcrit comme un événement déclenchant sur la transition
- Période de temps écoulée
 - transcrit comme une expression du temps sur la transition

337

Notion d'action

- Action : opération *instantanée* (conceptuellement) et *atomique* (ne peut être interrompue)
- Déclenchée par un événement
 - Traitement associé à la transition
 - Ou à l'entrée dans un état ou à la sortie de cet état



338

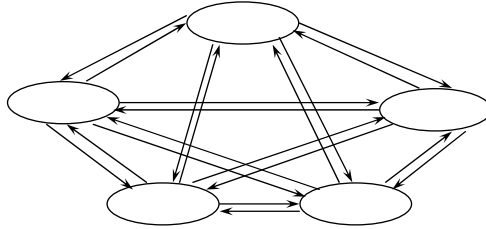
Notion d'états

- Etat : situation stable d'un objet parmi un ensemble de situations pré-définies
 - conditionne la réponse de l'objet à des événements
 - programmation réactive / « temps réel »
 - Intervalle entre 2 événements, il a une durée
- Peut avoir des variables internes
 - attributs de la classe supportant ce diagramme d'états

339

Structuration en sous-états

- Problème d'un diagramme d'états plats
 - Pouvoir d'expression réduit, inutilisable pour de grands problèmes
 - Explosion combinatoire des transitions.

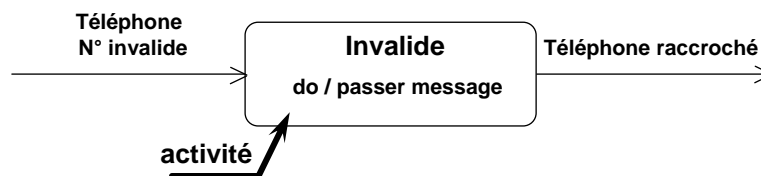


- Structuration à l'aide de super/sous états (+ hiérarchies d'événements)
 - représentés par imbrication graphique

340

Notion d'activité dans un état

- Activité : opération se déroulant continuellement tant qu'on est dans l'état associé
 - *do/ action*

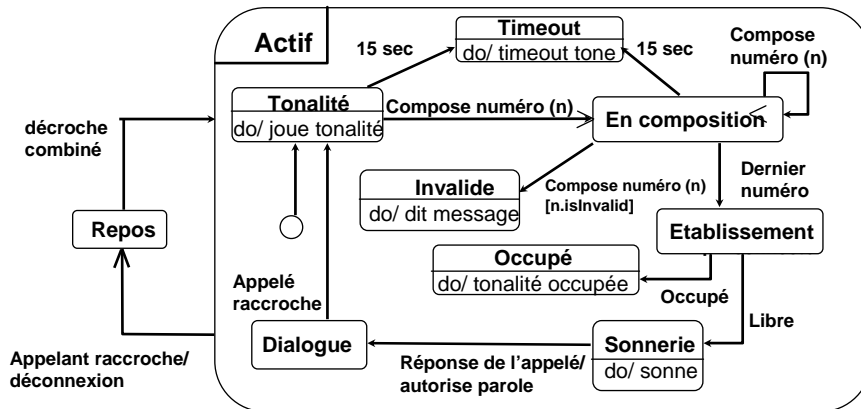


- Une activité peut être interrompue par un événement.

341

Exemple de diagramme d'états

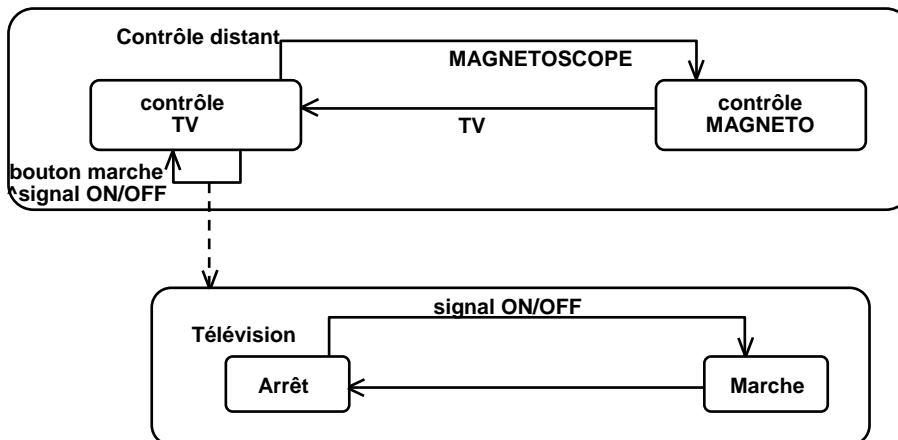
- un téléphone :



342

Émission d'événements

- Automate d'états d'une télécommande double :
 - TV + MAGNETOSCOPE



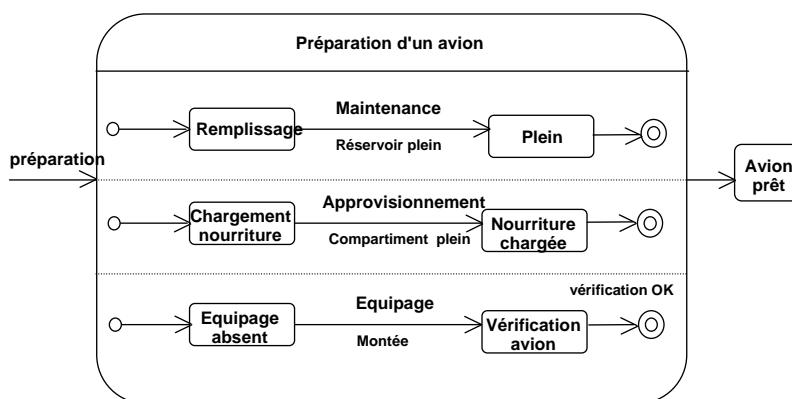
343

Diagrammes d'états concurrents

- Utilisation de sous-états concurrents pour ne pas à avoir à expliciter le produit cartésien d'automates
 - si 2 ou plus aspects de l'état d'un objet sont indépendants
 - Activités parallèles
- Sous-états concurrents séparés par pointillés
 - « swim lanes »

344

Exemple de concurrence



345

Etat-transition (résumé)

- Format :
 - événement (arguments) [conditions] / action ^événements provoqués
- Déclenchement :
 - par un événement (peut être nul).
 - Peut avoir des arguments.
 - Conditionné par des expressions booléennes sur l'objet courant, l'événement, ou d'autre objets.
- Tir de la transition :
 - Exécute certaines actions instantanément.
 - Provoque d'autres événements ; globaux ou vers des objets cibles.

346

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - *Diagramme d'activités*
 - Vision implantation
 - Diagramme de composants et de déploiement



347

Les diagrammes d'activité

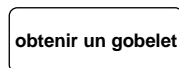
- Traitements effectués par une opération
 - Description d'un flot de contrôle procédural
 - Réseau d'actions et de transitions : automate dégénéré
 - La transition s'effectue lorsque l'opération est terminée
 - Pas de déclenchement par événement asynchrone
 - Sinon utilisation diagrammes d'états classiques
- Attachés à
 - une classe,
 - une opération,
 - ou un *use-case* (*workflow*)

348

Etat-action et décision

- Etat-action = raccourci pour un état où il y a :
 - une action interne
 - au moins une transition sortante
 - production d'un événement implicite : action accomplie
 - Pas de production/réaction à des événements explicites
- Modélisation d'une étape dans l'exécution d'un algorithme

- Notation :



- Décision = branchement sur plusieurs transitions

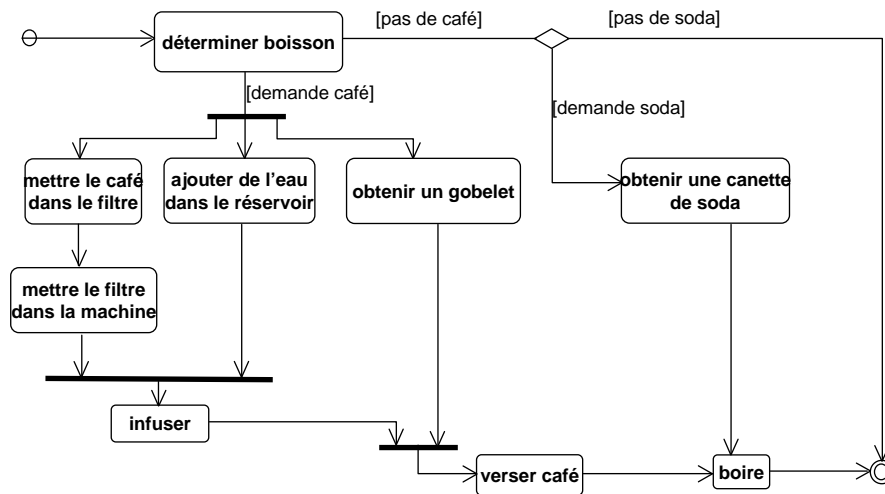
- Notation :

```
graph LR; I[ ] -- "[coût >= 50]" --> D{ }; D -- "[coût < 50]" --> O[ ]; D --> B[ ]
```

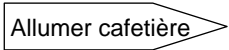
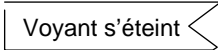
349

Exemple de diagramme d'activité

opération PréparerBoisson de la classe Personne



Stéréotypes optionnels

- Emission de signal  Allumer cafetière
- Réception de signal  Voyant s'éteint
- On obtient une syntaxe graphique proche de SDL
 - langage de description de spécifications
 - populaire dans le monde télécom

351

Liens modèles statiques/dynamiques

- Le modèle dynamique définit des séquences de transformation pour les objets
 - Diagramme d'état généralisant pour chaque classe ayant un comportement réactif aux événements les scénarios et collaborations de leurs instances
 - Les variables d'état sont des attributs de l'objet courant
 - Les conditions de déclenchement et les paramètres des actions exploitent les variables d'état et les objets accessibles
 - Diagrammes d'activités associés aux opérations/transitions/méthodes
- Les modèles dynamiques d'une classe sont transmis par héritage aux sous-classes

352

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - – *Vision implantation*
 - *Diagramme de composants et de déploiement*

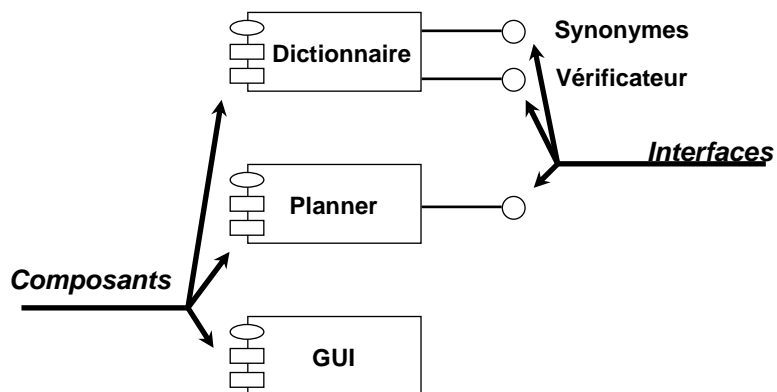
353

Diagrammes d'implantation

- Diagrammes de composants
 - Dépendances entre composants logiciels
 - code source
 - binaires, DLL
 - exécutables
- Diagrammes de déploiement
 - Configuration des composants
 - Localisation sur les nœuds d'un réseau physique

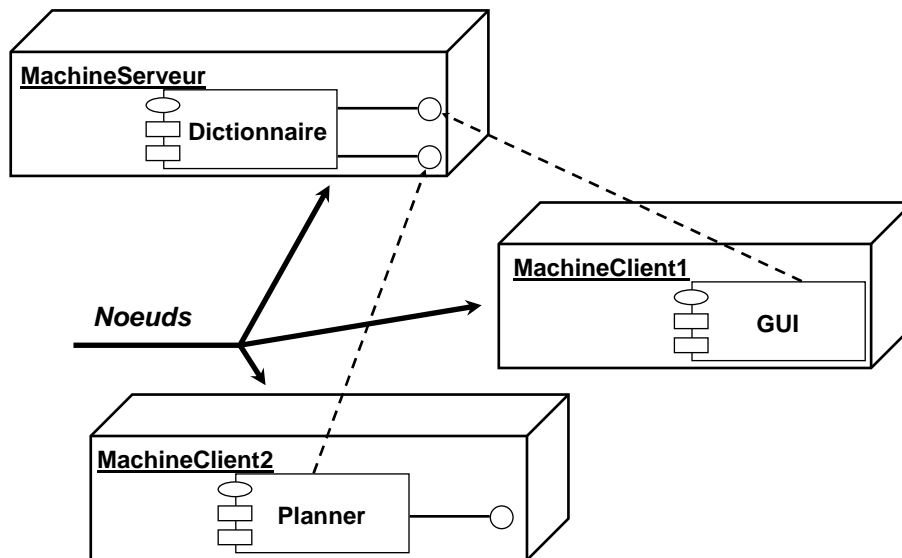
354

Exemples de composants



355

Exemple de déploiement



356

Modélisation UML

- Modélisation selon 4 points de vue principaux :
 - Vision utilisateur du système
 - Cas d'utilisation
 - Aspects statiques du système
 - Description des données et de leurs relations
 - Structuration en paquetages
 - Aspects dynamiques du système (comportemental)
 - Diagramme de séquences (scénarios)
 - Diagramme de collaborations (entre objets)
 - Diagramme d'états-transitions (Harel)
 - Diagramme d'activités
 - Vision implantation
 - Diagramme de composants et de déploiement

357