Bases de la modélisation objet

avec UML (Unified Modeling Language)

Pr. Jean-Marc Jézéquel
IRISA - Univ. Rennes

e-mail : jezequel@irisa.fr
http://people.irisa.fr/Jean-Marc.Jezequel
BlueSky, X @jmjezequel

1

1 - Introduction

L'ingénierie du logiciel aujourd'hui

2

Université de Rennes

Software is eating the world (Marc Andreessen)

- Le logiciel forme le tissu de notre société
  - Tel l'oxygène qu'on respire, force motrice essentielle, bien qu'invisible, du monde actuel
  - pratiquement aucun aspect de la société qune soit pas facilité ou médiatisé par le logiciel
- *Toute (grande) entreprise est désormais une entreprise de logiciels (S. Nadela)*
  - Le logiciel est une technologie clé de transformation et d'habilitation qui impacte de nombreux domaines de l'économie moderne
- Les logiciels stimulent le progrès scientifique dans de nombreux domaines
  - IA, la science des données, la médecine, l'ingénierie, etc.
- Les logiciels sont essentiels pour nos infrastructures critiques
  - énergie, les télécommunications, l'aérospatiale, l'automobile, la finance, la santé
- Le volume total de logiciels dans le monde croît à un rythme exponentiel
  - Double tous les 3,5 ans

3

Université de Rennes

slido

Please download and install the Slido app on all computers you use

Join at slido.com
#4818904

ⓘ Start presenting to display the joining instructions on this slide.

slido

Please download and install the Slido app on all computers you use

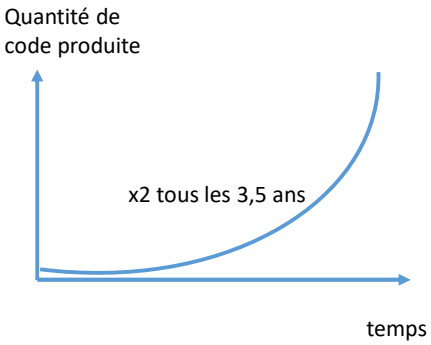
Classer ces logiciels par ordre croissant de taille

ⓘ Start presenting to display the poll results on this slide.

Université de Rennes

Taille de quelques logiciels (lignes de code)

- MS-DOS: 4 k
- WhatsApp: 30 k
- Telegram: 50 k
- Zoom: 60 k
- TikTok: 80 k
- Space Shuttle: 400 k
- Minecraft: 500 k
- Instagram: 1 M
- US Military Drone: 3.5 M
- YouTube: 5.4 M
- World of Warcraft: 5.5 M
- Boeing 787: 6.5 M
- Google Chrome: 6.7 M
- Twitter: 10 M
- Android: 12 M
- iOS: 12 M
- Mozilla Firefox: 21 M
- Windows XP: 45 M
- Large Hadron Collider: 50 M
- Ubuntu: 50 M
- Facebook: 62 M
- MacOS X: 84 M
- Tesla: 100 M
- Google: 2 milliards



Ce n'est pas demain que les (bons) développeurs seront au chômage!

Même avec Copilot/ChatGPT…

6

Université de Rennes

Google

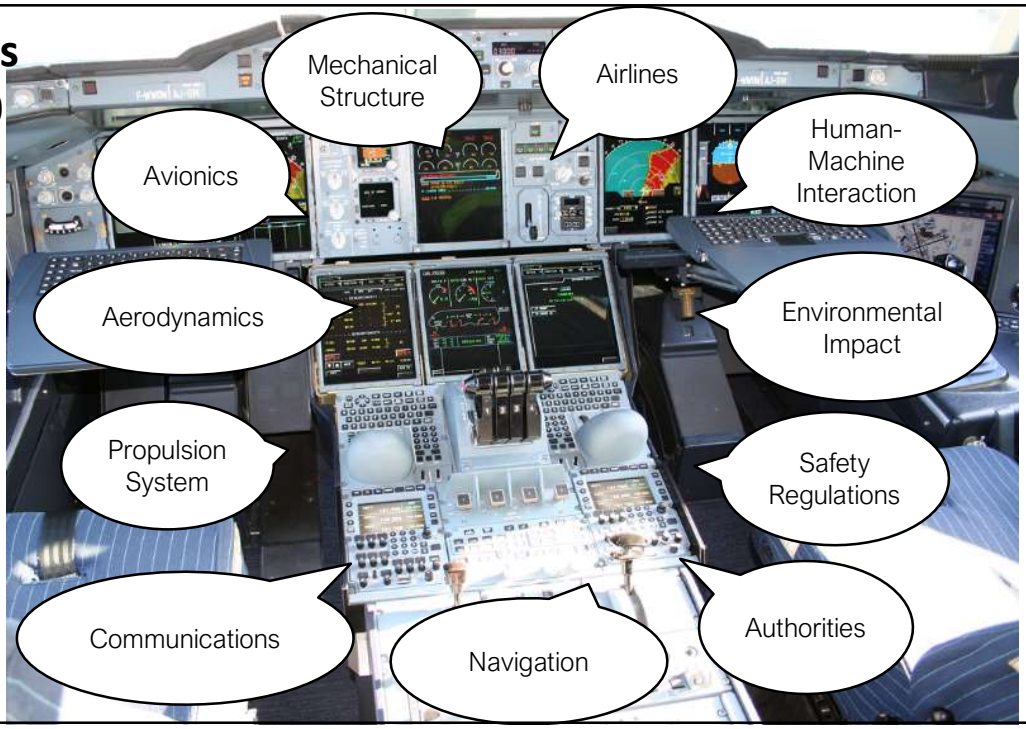
- 500 000 serveurs
  - répartis dans une trentaine de datacenters.
- Une phénoménale capacité de calcul qui permet à Google d'assurer le fonctionnement d'un grand nombre de services
  - Google Earth …
- répondre à plus d'1 milliard de requêtes par jour,
  - chacune interrogeant 10 milliards de pages Web
  - en moins d'un cinquième de seconde

The Google codebase includes approximately one billion files and has a history of approximately 35 million commits spanning Google's entire 18-year existence. The repository contains 86TB[a] of data, including approximately two billion lines of code in nine million unique source files.

7

Université de Rennes

Airbus A 380



Université de Rennes

Automotive industry: Volvo

- 100,000,000 LOC
- 30,000,000 function calls
- 10,000,000 conditional statements
- 3,000,000 functions
- 100.000 functional requirements
- 7,000 signals among
- 120 ECUs

- **LOC growing by one order of magnitude every ten years since 2000!**

10

Université de Rennes

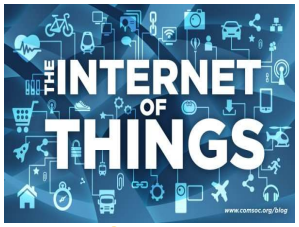
Samsung

- 1,5 milliard de smartphones en circulation
  - 300 millions de plus par an
- Des milliers de versions de logiciels
- Time-to-market ~ 3 mois

11

Université de Rennes

From Zero Power Computing …



…to Systems of Systems

- Global control of a massively distributed system totaling about 10^9 LoC

Université de Rennes

Programming-in-the-Large : Gérer la complexité due à la taille

Crossing a Stream

13

Université de Rennes

slido

Please download and install the Slido app on all computers you use

Qu'est-ce qui rend à votre avis ces logiciels si complexes?

ⓘ Start presenting to display the poll results on this slide.

Université de Rennes

Des logiciels souvent complexes...

- Sources de complexité
  - Règles « métiers » sophistiquées et changeantes
    - lignes de produits
    - évolution continue et durée de vie importante
  - Plateformes technologiques complexes
    - enjeux de fiabilité, sécurité, efficacité, utilisabilité, etc.
- => Logiciels de grande taille
  - des millions de lignes de code
  - des équipes nombreuses pour les élaborer et les faire évoluer
- Besoin d'ingénierie : **Génie logiciel**
  - Aspects métiers ← **BMO**
  - Aspects techniques
  - Aspects humains

16

Université de Rennes

La modélisation pour gérer la complexité

- Comme dans toutes les autres sciences
  - **Un modèle est l'abstraction d'un aspect de la réalité pour un objectif donné**
    - Ou pour un logiciel d'un aspect du business pour en automatiser une partie
  - Par ex. en chimie 2H + O -> H2O
- Les modèles en informatique
  - 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
    - Il **est** la réalité (artificiel vs. naturel)

17

Université de Rennes

Objectifs généraux du cours

- Acquérir les bases de la modélisation objet : comment il est possible de définir un certain nombre de modèles orientés objets
  - 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…
    - … Afin de préparer une conception et réalisation dans un langage à objets
  - Approche dite *Domain-Driven Design* (DDD)
- Etre capable d'affronter la complexité d'un projet informatique
  - CAVEAT: les projets menés dans la formation ne seront jamais très complexes
    - Application de méthodes avancées sur des choses trop simples pour en voir l'intérêt
    - Il faudra donc faire un effort d'imagination pour transposer…

Université de Rennes

Bases de la modélisation Objet

- Ce cours introduit les notions de base de la modélisation de logiciel…
  - objets, classes, modularité, contrats, associations, généralisation/spécialisation, cas d'utilisations, diagrammes de séquence, automates…),
  - Ces notions sont illustrées dans le langage UML
    - *Mais ceci n'est pas un cours d'UML!*
- … et déroule sur une étude de cas comment les mettre en œuvre

- Objectif : être capables, à partir d'un cahier des charges, de construire de manière systématique un **modèle d'analyse** bien formé exprimé en UML
  - afin de préparer une conception/implantation dans un langage comme Java/C#/C++
  - Approche dite *Domain-Driven Design* (DDD)

- Modalités du CC
  - CC : Quizz à l'issue de chacune des vidéos : total comptant pour 10% de la note de BMO
  - CC : TP/mini-projet noté : 40% de la note de CC de BMO
  - CT portant sur le cours : 50% de la note de BMO
  - Rattrapage possible en juin de la note de CT

Université de Rennes

Cours en video sur Moodle/youtube

- Découpés en épisodes de 10-12 minutes chaque
  - https://www.youtube.com/playlist?list=PLQIe-0fSFn6CGJXyC5Bf6ECQq4f0-l2BG
- Partie 1, épisodes 1 à 5
  - Quizz associés doivent être faits au plus tard le dimanche 12/1
- Partie 2, épisodes 6 à 13
  - Quizz associés doivent être faits au plus tard le dimanche 19/1
- Partie 3, épisodes 14 à 21
  - Quizz associés doivent être faits au plus tard le dimanche 2/02
- Les notes des quizz seront relevées le lendemain de la deadline
  - NB: vous pouvez refaire autant de fois que vous voulez chaque quizz
    - Pas pourvous sanctionner, mais vous permettre de faire le point vous-même

Université de Rennes

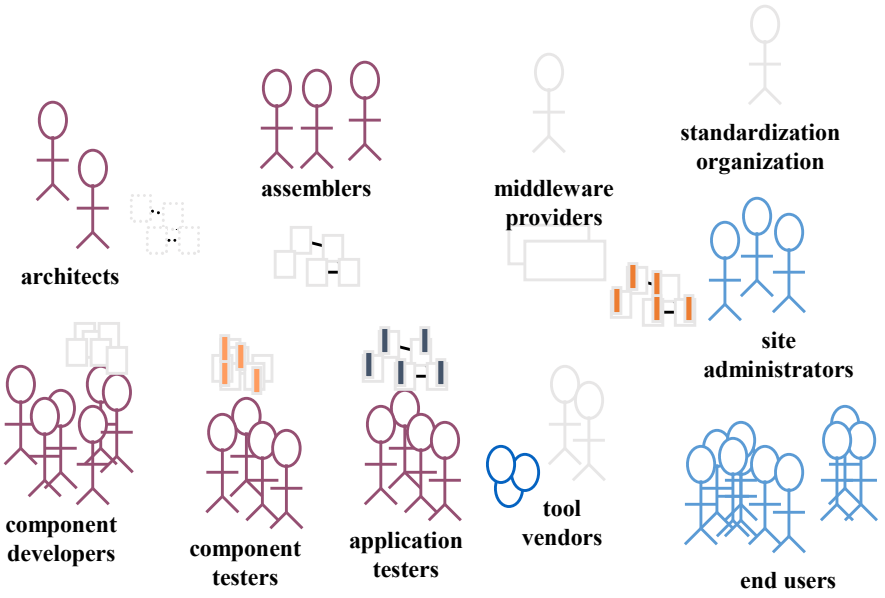
2 - Pourquoi UML ?

Un langage de modélisation pour le logiciel orienté objet

21

Université de Rennes

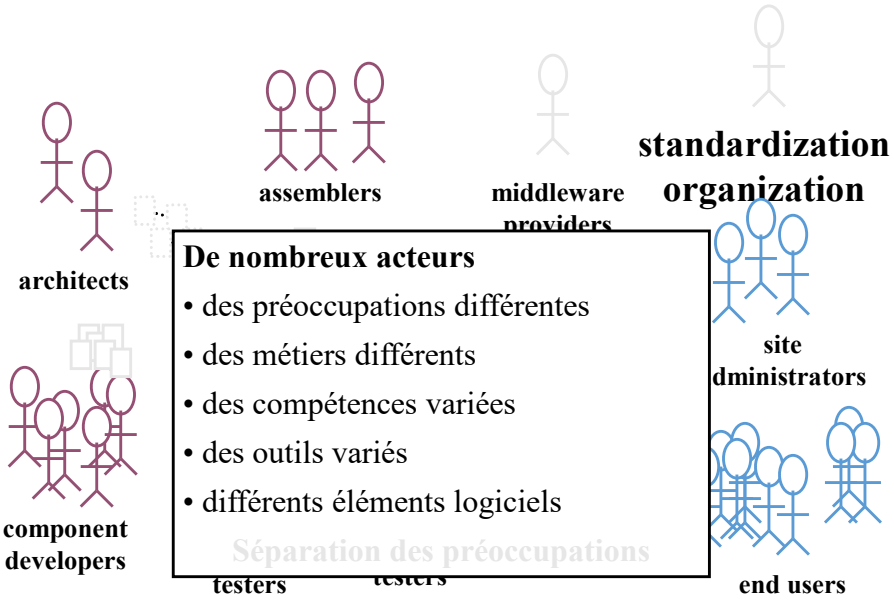
L'industrie logicielle aujourd'hui



22

Université de Rennes

L'industrie logicielle aujourd'hui



23

Université de Rennes

Problématique

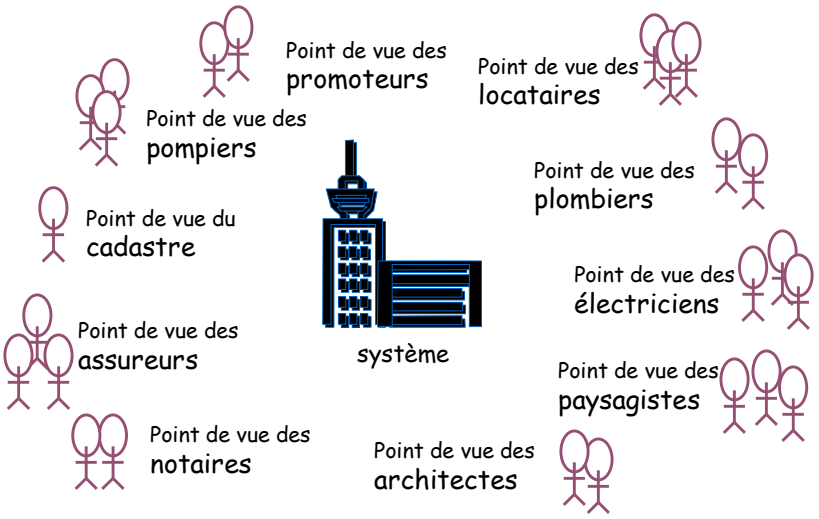
- Complexité croissante des logiciels
- Séparations des préoccupations
- Séparations des métiers
- Multiplicité des besoins
- Multiplicité des plateformes
- Évolution permanente

Logiciel >> Code

24

Université de Rennes

Séparations des préoccupations

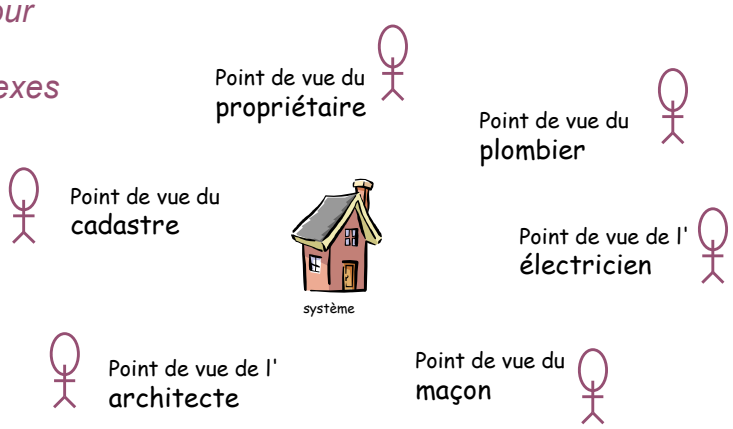


25

Université de Rennes

Séparations des préoccupations

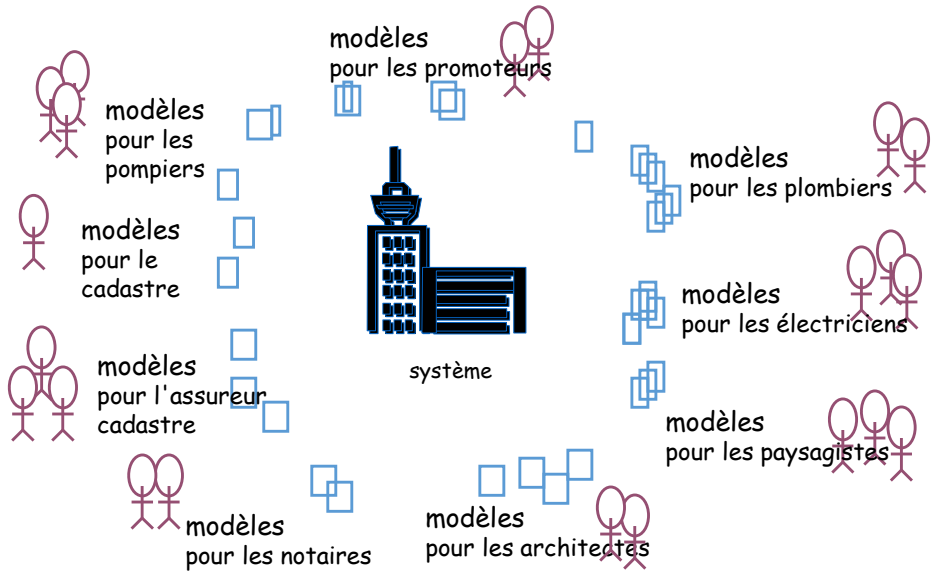
Utile même pour des systèmes "moins" complexes



26

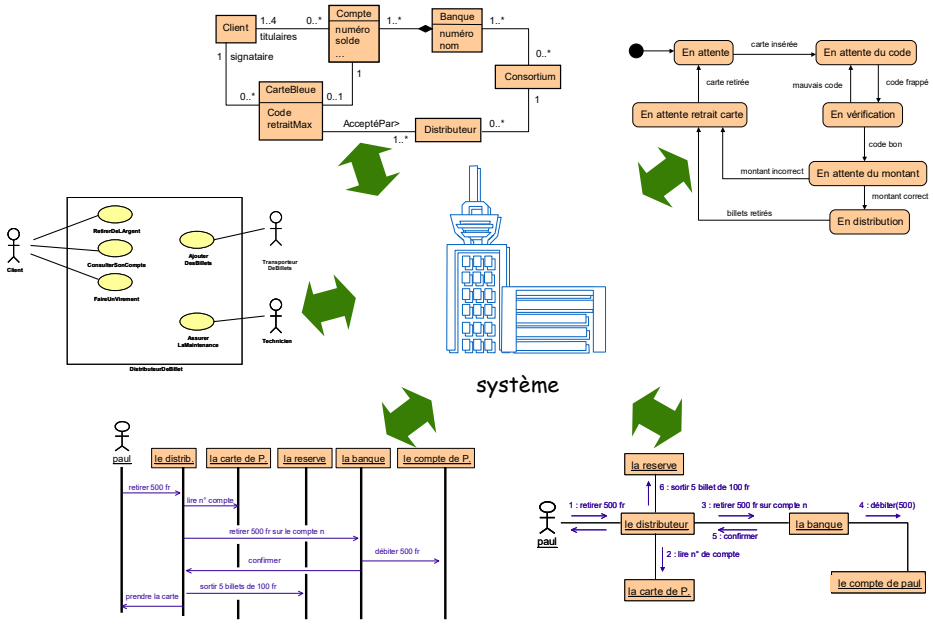
Université de Rennes

Multiples modèles d'un même système



27

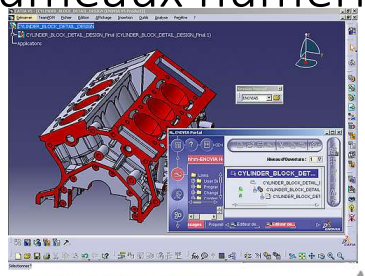
Université de Rennes

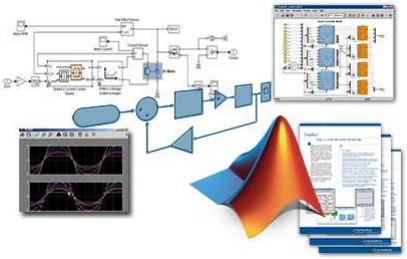


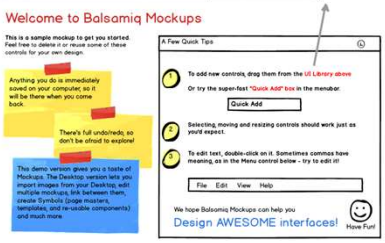
28

Université de Rennes

Langages de modélisation multiples
Model = Digital Twins (Jumeaux numériques)





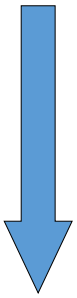


29

Université de Rennes

UML : Langage de modélisation pour le logiciel

- Réfléchir en séparant les préoccupations
  - En 4 points de vue principaux
- Définir la structure « gros grain »
- Documenter
- Guider le développement
- Développer, Tester, Auditer



30

Université de Rennes

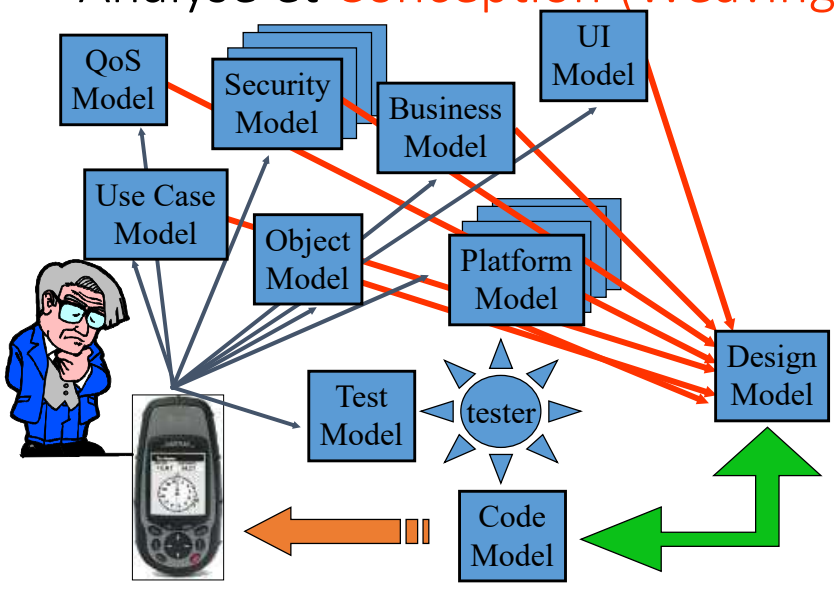
Modélisation UML : séparation des préoccupations

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

31

Université de Rennes

Analyse et Conception (Weaving)



Université de Rennes

Cours en video sur Moodle/youtube

- Découpés en épisodes de 10-12 minutes chaque
  - https://www.youtube.com/playlist?list=PLQIe-0fSFn6CGJXyC5Bf6ECQq4f0-l2BG
- Partie 1, épisodes 1 à 5
  - Quizz associés doivent être faits au plus tard le dimanche 12/1
- Partie 2, épisodes 6 à 13
  - Quizz associés doivent être faits au plus tard le dimanche 19/1
- Partie 3, épisodes 14 à 21
  - Quizz associés doivent être faits au plus tard le dimanche 2/02
- Les notes des quizz seront relevées le lendemain de la deadline
  - NB: vous pouvez refaire autant de fois que vous voulez chaque quizz
    - Pas pourvous sanctionner, mais vous permettre de faire le point vous-même

slido

Please download and install the Slido app on all computers you use

Modalités de l'évaluation

ⓘ Start presenting to display the poll results on this slide.

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

42

Université de Rennes

"Objet" (Définition)

- Formellement : la fermeture transitive d'une fonction
- Concrètement : encapsulation d'un état avec un ensemble d'opérations travaillant sur cet état
  - abstraction d'une entité du monde réel
  - existence temporelle :
    - création, évolution, destruction
  - identité propre à chaque objet
  - peut être vu comme une machine
    - ayant une mémoire privée et une unité de traitement,
    - et rendant un ensemble de services

43

Université de Rennes

“Classe”
Définition en temps que type

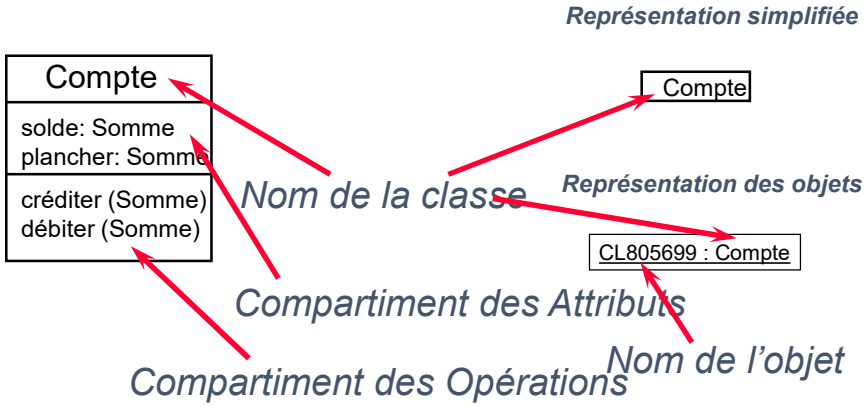
- Implantation d’un type de donné abstrait
- Description des propriétés et des comportements communs à un ensemble d’objets
  - un objet est une instance d’une classe (eq. variable vs. type)
- Chaque classe a un nom, et un corps qui défini :
  - les attributs possédés par ses instances
  - les opérations définies sur ses instances,
    - et permettant d’accéder, de manipuler, et de modifier leurs attributs
  - une classe peut elle même être un objet (*Smalltalk*)

44

Université de Rennes

Notations UML pour classes et objets

- Représentation d’une classe



45

Université de Rennes

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

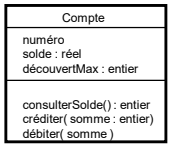


Diagramme de classes

M1

M0

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

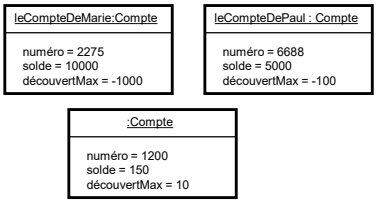


Diagramme d 'objets

46

Université de Rennes

"Classe" Définition en temps que module

- Modularité = Gestion complexité des systèmes
- Concept présent depuis longtemps en informatique
  - subroutines, unité de compilation (le fichier en C)
- Notion de module intégrée aux langages
  - Modula-2 (module), Ada83 (package), puis lang. à objets
- Favorise :
  - masquage d'information (abstraction)
  - encapsulation (facilite les modifications à portée locale)
  - dissociation interface/implantation=> composant réutilisable

47

Université de Rennes

+ # - Visibilité des éléments

- Restreindre l'accés aux éléments d'un modèle
- Contrôler et éviter les dépendances entre classes et paquetages

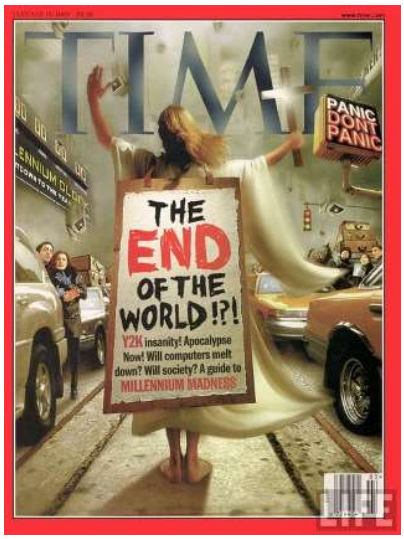
+	public	visible de tous
#	protégé	visible dans la classe et ses sous-classes
-	privé	visible dans la classe uniquement
~	package	visible dans la package uniquement

- 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 !

49

Université de Rennes

Contre exemple de modularité : le bug de l'an 2000



Université de Rennes

Encapsulation/masquage d'information

date
year: String[2]
equal (date) isLess(date) delta(date)

- Un choix de format de stockage pour l'année
  - 3 opérations principales (et leurs dérivées) + print
- Correction bug an 2000?
  - modification du format de stockage
- Sans encapsulation/masquage d'information:
  - Cout de la modification proportionnel à la taille du programme
    - Car il faut passer sur toutes les lignes pour reporter l'éventuelle modification
- Avec encapsulation/masquage d'information:
  - Cout de la modification indépendant de la taille du programme
    - Le reste du code est protégé de mes choix

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, relations, héritage, contrats
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

52

Université de Rennes

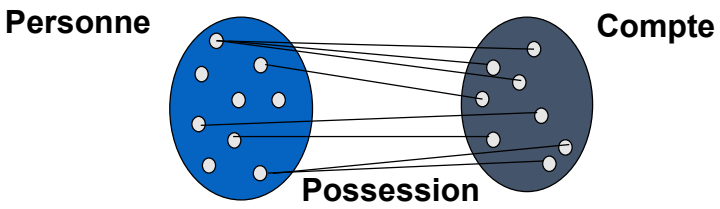
Relations

- Deux points de vue :
  - Une relation met en correspondance des éléments d'ensembles
  - Une relation permet la description d'un concept à l'aide d'autres concepts
- Une contrainte :
  - Une relation est un lien stable entre deux objets

53

Université de Rennes

Vue ensembliste d'une relation : Graphe de la relation



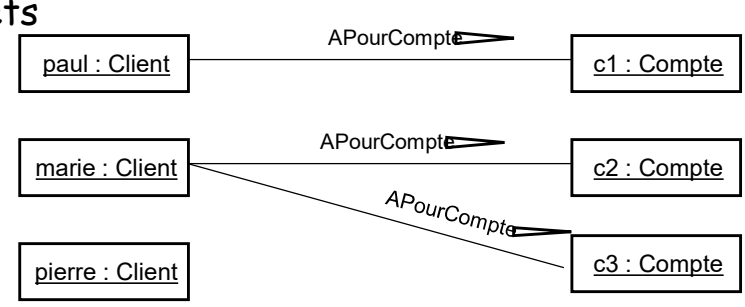
Une association met en correspondance des éléments d'ensembles

54

Université de Rennes

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 »)

55

Université de Rennes

Associations (entre classes)

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

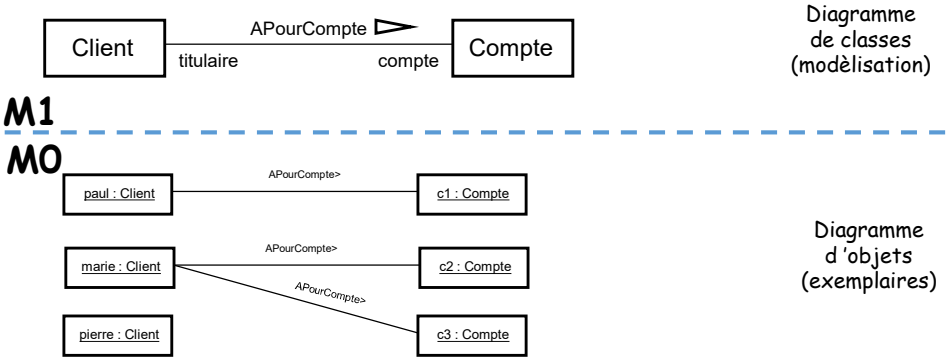


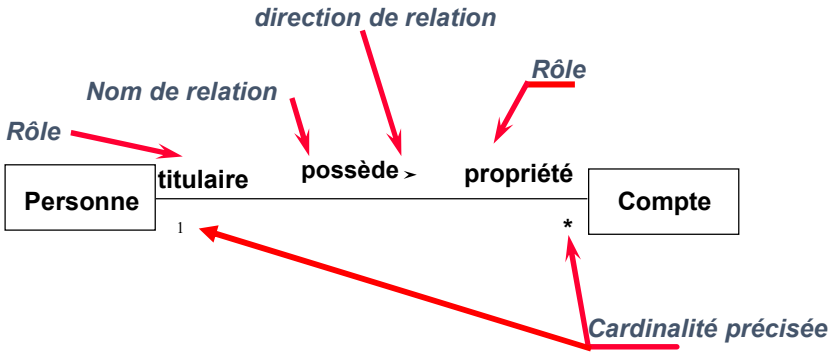
Diagramme de classes (modèlisation)

Diagramme d 'objets (exemplaires)

56

Université de Rennes

Représentation des associations : direction, rôle, cardinalité

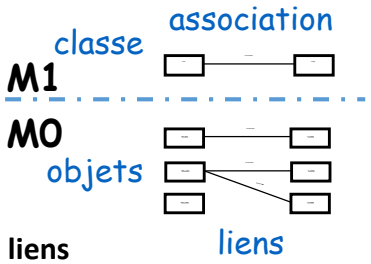


57

Université de Rennes

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**

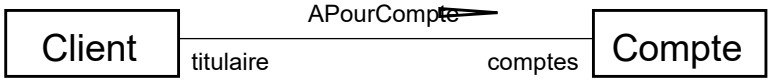
- Des **liens** peuvent être ajoutés ou détruits pendant l 'exécution, (ce n 'est pas le cas des associations)

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

58

Université de Rennes

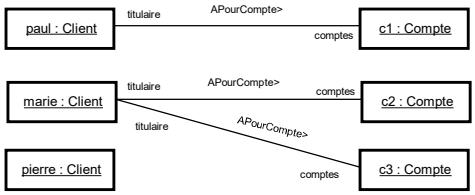
Utiliser les rôles pour «naviguer»



M1

M0

paul.comptes= {c1}
marie.comptes= {c2,c3}
pierre.comptes= { }
c1.titulaire = paul
c2.titulaire = marie
c3.titulaire = marie



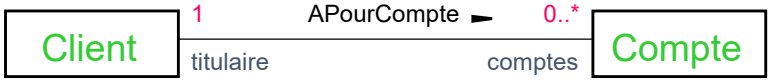
Nommer en priorité les rôles

59

Université de Rennes

Cardinalités d'une association

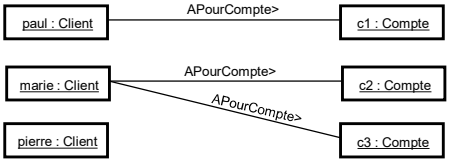
- Précise combien d'objets peuvent être liés à un seul objet source
- Cardinalité minimale et cardinalité maximale ($C_{min}..C_{max}$)
- Doivent être des constantes



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

M1

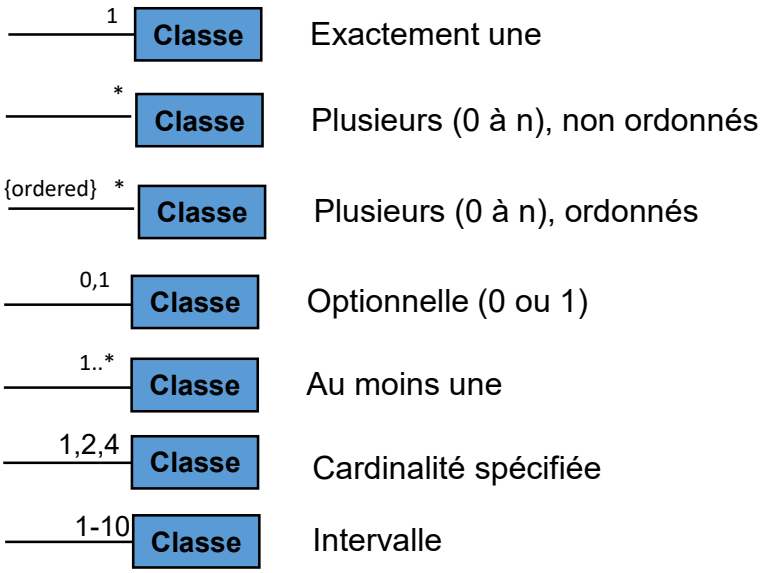
M0



60

Université de Rennes

Cardinalité d'une association

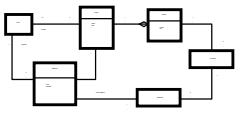


61

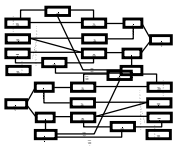
Université de Rennes

Diagrammes de classes vs. d'objets

- Un diagramme de classes
  - défini 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



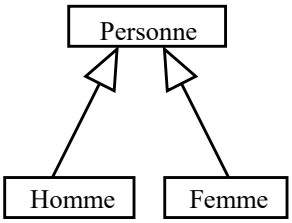
- Les diagrammes d'objets peuvent être utilisés pour
  - expliquer un diagramme de classe (donner un exemple)
  - valider un diagramme de classe (le "tester")

62

Université de Rennes

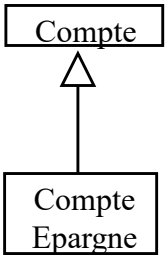
Héritage et Polymorphisme : Généralisation / Spécialisation

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



"Super classe"

Cas général



"Sous classes"

Cas spécifique

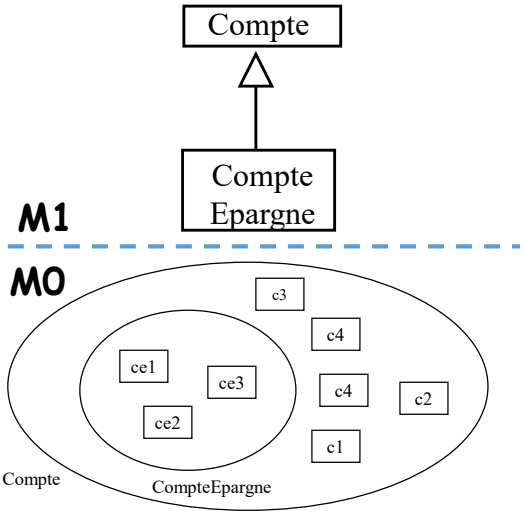
On dit que CompteEpargne **hérite** *de Compte*

63

Université de Rennes

Relation de sous typage, Vision ensembliste

tout objet d'une sous-classe appartient également à la superclasse



64

Université de Rennes

Deux points de vue complémentaires

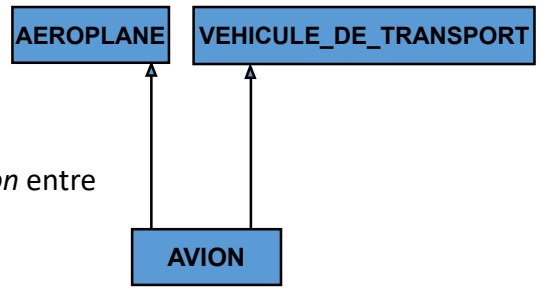
- Mécanisme d'extension de module : héritage
  - ajout de fonctionnalités dans la sous-classe
  - "customisation" et combinaison de composants logiciels
  - réutilisation de code
- Mécanisme de classification : sous-typage
  - relation X est-une-sorte-de Y (est substituable à)
  - organisation des systèmes complexes (cf. Linnaeus)
  - réutilisation d'interface

65

Université de Rennes

Héritage multiple

- Héritage multiple, exemple :
  - Un avion est *à la fois*
    - Un Aéroplane
    - Un Véhicule de transport
  - L'ensemble des avions est donc *l'intersection* entre
    - l'ensemble des aéroplanes et
    - l'ensemble des véhicules de transport



- Correspond en Java à l'implantation de plusieurs interfaces

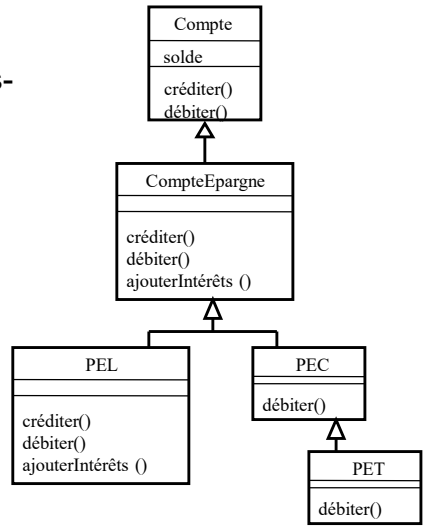
66

Université de Rennes

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

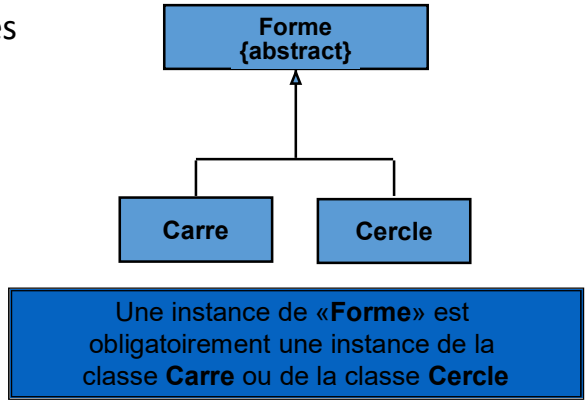


67

Université de Rennes

Classes abstraites

- Capturent des comportements communs
- Ne peuvent donc pas être instanciées
- Servent à structurer un système
- Peuvent avoir des opérations dont l'implantation est absente
  - pure virtual en C++
  - abstract en Java
- Classes sans instances immédiates

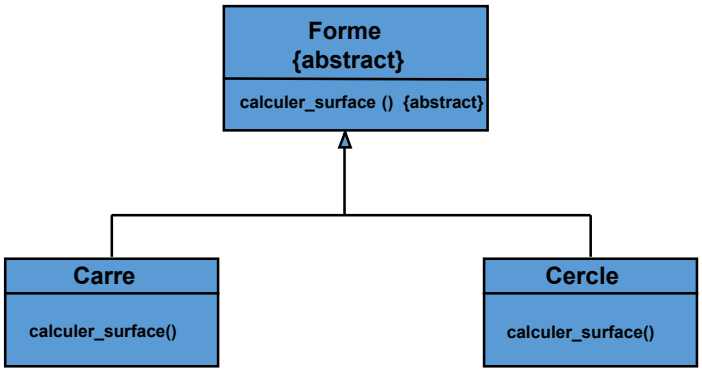


68

Université de Rennes

Représentation des opérations abstraites

- Opération sans corps d'une classe abstraite



69

Université de Rennes

Polymorphisme et liaison dynamique

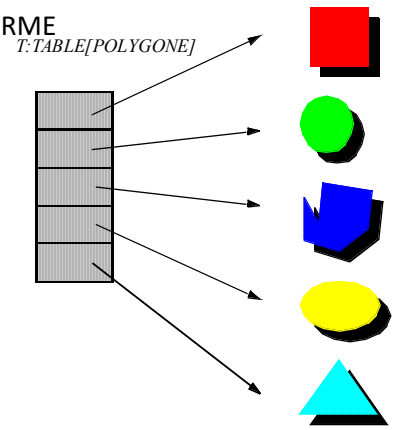
- Polymorphisme : possibilité de changer de forme
  - f : FORME; c: CERCLE; k: CARRE;
  - f := c; f := k
- Liaison dynamique : l'effet de l'appel d'une opération d'un objet dépend de sa forme effective à l'exécution
  - f.imprimer; -- *différent selon que f est CERCLE ou CARRE*
- Espace de nommage réduit et uniforme
  - Mise en facteur des parties communes
  - Possibilité de "conteneurs" hétérogènes

70

Université de Rennes

Polymorphisme : exemple

- T contient des FORMEs
  - en fait des instances de sous-classes de FORME
- Programmes de type :
  - T[1] <- carré
    T[2] <- cercle
    ...
    Pour tout i
      T[i].imprimer
- Si ajouts ultérieurs:
  - e.g. triangle
- Pas de modifications globales
  - case-less programming



71

Université de Rennes

Héritage et typage

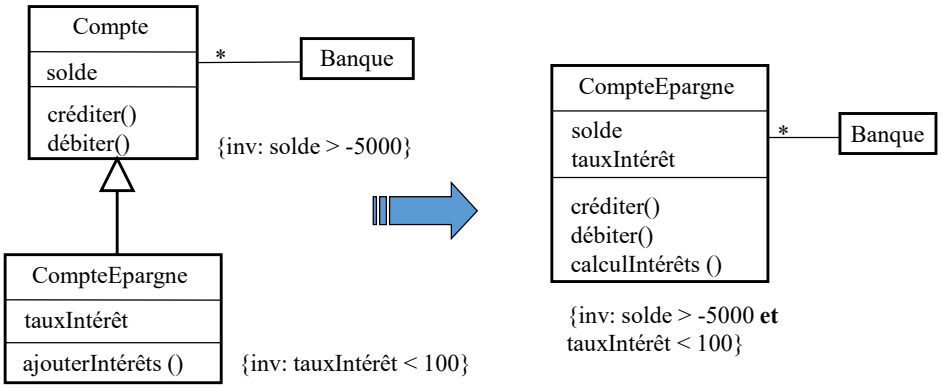
- typage statique :
  - x.f légal <=> vérification avant l'exécution que tout objet potentiellement désigné par x dispose d'une méthode f
- liaison dynamique :
  - la *bonne* interprétation de f est choisie
- langages à objets :
  - Python, Smalltalk : typage dynamique, liaison dynamique
  - Java, C# : typage statique, liaison dynamique
  - C++ : typage statique, liaison dynamique pour les fonctions "virtuelles"
  - UML : au choix (!)

72

Université de Rennes

Relation d'héritage en UML

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

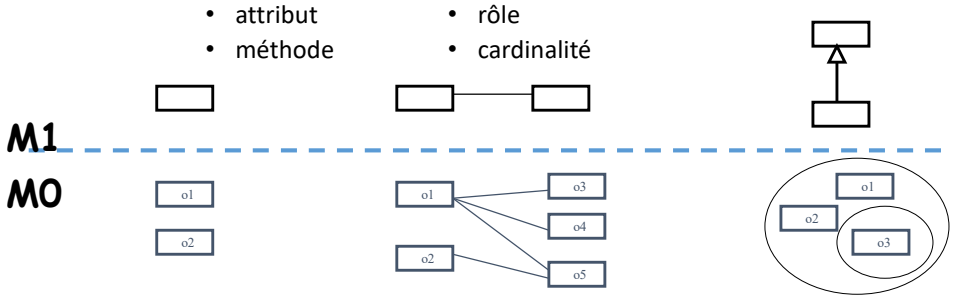


73

Université de Rennes

Synthèse des concepts de base

- Classe
  - attribut
  - méthode
- Association
  - rôle
  - cardinalité
- Héritage



- Objet
- Lien
- Inclusion ensembliste

74

Université de Rennes

Classes et Objets

Utilisation avancée

Université de Rennes

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

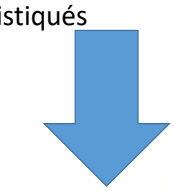
S'adapter ...

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

Université de Rennes

Principe de l'approche objet

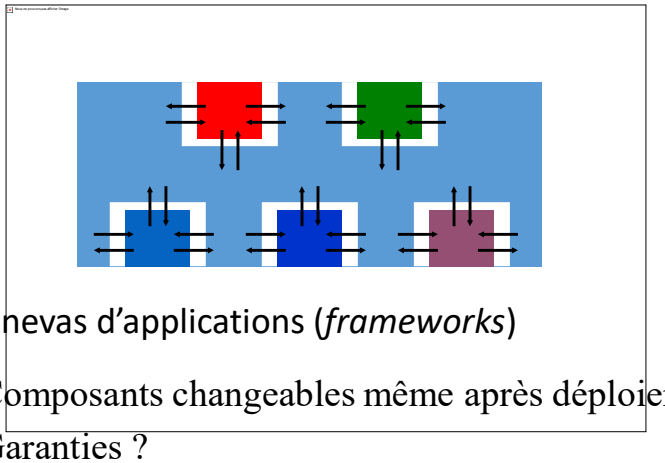
- Structurer les systèmes autour des objets
  - Plutôt qu'autour des fonctions
  - Composition d'éléments simples pour obtenir des systèmes sophistiqués
- L'approche par modélisation du domaine facilite
  - Communication (et V&V)
    - avec donneur d'ordre (Validation)
    - entre les activités de dvp et de maintenance (Vérification)
  - Continuité entre les différentes phases du cycle de vie
    - cf. Jackson et JSD
- Obtenir des systèmes modulaires et maintenables
  - Notion de *lignes de produits*
  - Assemblage de briques de base vs. dvp ad-hoc
  - Produire des *canevas d 'application* vs. un programme



77

Université de Rennes

Approche OO : Modélisation et Composants



- Canevas d'applications (*frameworks*)
  - Composants changeables même après déploiement
  - Garanties ?
    Fonctionnelles , synchronisation, performances, QdS

78

Université de Rennes

De la difficulté de la validation intra-composant

```
Acquérir une valeur positive n
Tant que n > 1 faire
        si n est pair
        alors n := n / 2
        sinon n := 3n+1
Sonner alarme;
```

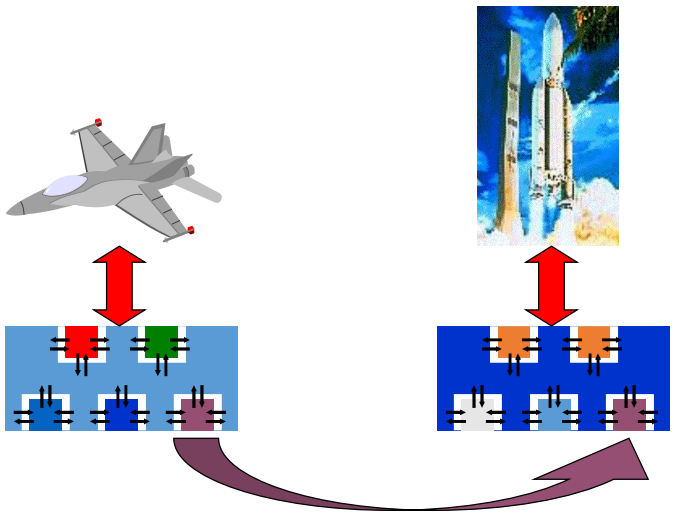
- Prouver que l'alarme est sonnée pour tout n?
  - Revient à prouver que la suite $n_{i+1} = n_i$ pair ? $n_i/2$: $3n_i+1$ converge
  - C'est la conjecture de Syracuse…
- Indécidabilité de certaines propriétés
  - problème de l'arrêt de la machine de Turing…

- **Recours au test**
  - ici, si machine 32 bits, 2^31 = 10^10 cas de tests
  - **5 lignes de code => *10 milliards de tests* !**

84

Université de Rennes

Validité inter-composants : Peut-on (ré)-utiliser un composant?

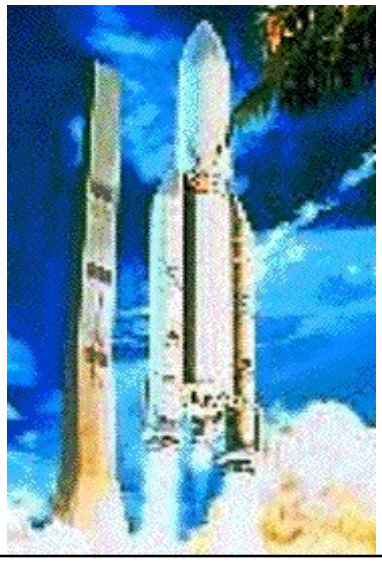


85

Université de Rennes

Ariane 501 Vol de qualification Kourou, ELA3 -- 4 Juin 1996,12:34 UT

- H0 -> H0+37s : nominal
- Dans SRI 2:
  - BH (Bias Horizontal) > 2^15
  - convert_double_to_int(BH) fails!
  - exception SRI -> crash SRI2 & 1
- OBC disoriented
  - Angle attaque > 20°,
  - charges aérodynamiques élevées
  - Séparation des boosters



86

Université de Rennes

Ariane 501 : Vol de qualification

Kourou, ELA3 -- 4 Juin 1996,12:34 UT

- H0 + 39s: auto-destruction (coût: 500M€)

87

Université de Rennes

Pourquoi ? (cf. *IEEE Comp. 01/97*)

- Pas une erreur de programmation
  - Non-protection de la conversion = décision de conception ~1980
- Pas une erreur de conception
  - Décision justifiée vs. trajectoire Ariane 4 et contraintes TR
- Problème au niveau du test d'intégration
  - Comme toujours, aurait pu être détecté. Mais gigantesque espace de test vs. ressources limitées
  - En plus, SRI inutile à cette étape du vol!

88

Université de Rennes

Pourquoi? (cf. *IEEE Computer 01/97*)

- Réutilisation dans Ariane 5 d'un composant de Ariane 4 ayant une contrainte « cachée » !
  - Restriction du domaine de définition
    - Précondition : abs(BH) < 32768.0
  - Valide pour Ariane 4, mais plus pour Ariane 5

89

Université de Rennes

Spécification = contrat *entre un composant et ses clients*

- Dans la vie réelle, différents types de contrats
  - *Du « Contrat social » de Jean-Jacques Rousseau au "cash & carry"*
- De même, plusieurs types de contrats dans un monde réparti

90

Université de Rennes

Quatre niveaux de contrats logiciels

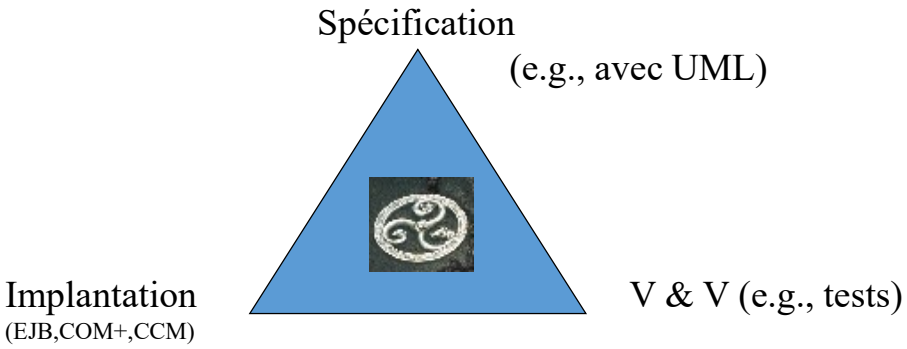
- Elémentaire (syntaxique)
  - le programme compile…
- Comportemental (fonctionnel)
  - pré et post conditions
- Synchronisations
  - e.g. *path expressions*, etc. [McHale]
- Qualité de service (quantitative)
  - Négociation dynamique possible

Cf. IEEE Computer July 1999

91

Université de Rennes

Composant de confiance?



Confiance = cohérence entre ces 3 aspects

92

Université de Rennes

Représentation des contrats avec OCL

- Typage par signature des méthodes insuffisant
  - besoin de pouvoir exprimer des restrictions
    - valeurs d 'entrées *et* de sorties
  - besoin de préciser la sémantique
    - ce que fait une méthode (le quoi) sans entrer dans le détail comment
      - Préserve le masquage d'information : Indépendance vis-à-vis de l'implantation
- Inspirée par la notion de Type Abstrait de Données:
  Spécification = Signature +
  - Préconditions (conditions sous lesquelles une méthode peut être appelée)
  - Postconditions (propriétés garanties par une méthode)
  - Invariants de classe (vrai à l'entrée et à la sortie des méthodes)

93

Université de Rennes

OCL : Object Constraint Language

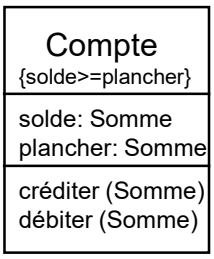
- Langage de description de contraintes de UML
  - des contraintes restreignant les domaines de valeurs peuvent être ajoutées aux éléments du modèle UML
- Contrainte = expression booléenne (sans effet de bord) portant sur
  - opérations usuelles sur types de base (Boolean, Integer...)
  - attributs d'instances et de classes
  - opérations de « query » (fonctions sans effet de bord)
  - associations du modèle UML
  - états des StateCharts associés

94

Université de Rennes

Représentation des contraintes OCL

- Directement dans le modèle
  - notation entre **{ } accrochée à un élément de modèle**



- Dans un document séparé, en précisant le **contexte**

context Compte inv:
solde >= plancher

  - Invariants = Propriétés vraies pour l'ensemble des instances de la classe
    - dans un état stable, chaque instance doit vérifier les invariants de sa classe

95

Université de Rennes

Précondition:
ce qui doit être respecté par le client

- Spécification des conditions nécessaires pour qu'un client soit autorisé à appeler une méthode
  - exemple: montant > 0
- Notation en UML
  - {«precondition» *OCL boolean expression*}
  - Abbreviation: {pre: *OCL boolean expression*}

96

Université de Rennes

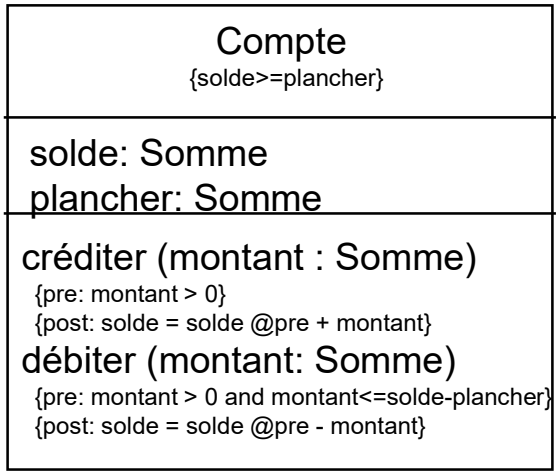
Postcondition:
Ce qui doit être assuré par l'implantation

- Spécification de ce qui sera vrai à la complétion d'un appel valide à une méthode
  - exemple: solde = solde @pre + montant
- Notation en UML
  - {«postcondition» *OCL boolean expression*}
  - Abbreviation: {post: *OCL boolean expression*}
  - Opérator pour accéder à la valeur « d 'avant » (idem *old* Eiffel):
    - *OCL expression* @pre

97

Université de Rennes

Etre abstrait et précis avec UML



Analyse précise ou "analyse par contrat"

98

Université de Rennes

Conception par Contrat

- Le contraire de la *Programmation Défensive*
  - Essayer de tester tout ce qui pourrait poser problème
    - code lourd et complexe à maintenir et tester
    - quoi faire lorsqu'on détecte un problème?
- Assertions des contrats :
  - jouent un rôle crucial dans la séparation nette des responsabilités dans un système modulaire
  - contrat entre l'appelant d'une routine (le client) et l'implantation de la routine (le contractant) :

Pourvu que le client appelle la routine dans des conditions où l'invariant de classe du contractant et la précondition de la routine sont respectés, alors le contractant promet que lorsque la routine terminera, le travail spécifié dans la postcondition sera effectué, et l'invariant de classe sera respecté.

99

Université de Rennes

Intérêt pratique des contrats

- Specification, documentation
  - *Not a software fault tolerance gadget*
  - *Might help system fault tolerance...*
- Help V&V
  - When assertions are monitored
  - Never doing debugging again
- Help allocate responsibilities during integration
  - No longer have to find a scapegoat ;-)

100

Université de Rennes

Contract Violations: Preconditions

- The client broke the contract.
  - The provider does not have to fulfill its part of the contract.
  - If contracts are monitored, an exception should be raised
    - making it easy to identify the exact origin of the fault.

```
Unhandled exception: Routine failure. Exiting program.
Exception history:
================================================================================
Object Routine
Type of exception          Description          Line
================================================================================
#<BANK_ACCOUNT5f0c0>                            BANK_ACCOUNT:deposit
precondition violated      positive_amount      63
--------------------------------------------------------------------------------
#<USER 5f000>                                   USER:test
Routine failure                                 90
--------------------------------------------------------------------------------
#<DRIVER 5f010>                                 DRIVER:make
Routine failure                                 18
--------------------------------------------------------------------------------
```

101

Université de Rennes

Contract violations: Postconditions

- The implementation of a method did not comply with its promise: This is a bug

```
Unhandled exception: Routine failure. Exiting program.
Exception history:
================================================================================
Object Routine
Type of exception          Description          Line
================================================================================
#<BANK_ACCOUNT5f0c0>                            BANK_ACCOUNT:deposit
postcondition violated     deposited            70
--------------------------------------------------------------------------------
#<USER 5f000>                                   USER:test
Routine failure                                 90
--------------------------------------------------------------------------------
#<DRIVER 5f010>                                 DRIVER:make
Routine failure                                 18
--------------------------------------------------------------------------------
```

- Again, easy to trace...(between lines 63-70)

Université de Rennes

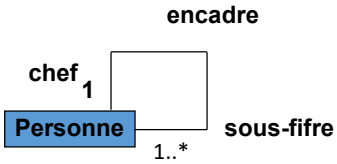
Raffinements du concept d'association

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

Université de Rennes

Cas particuliers d'association

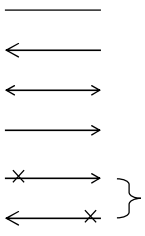
- Relations réflexives



Une association réflexive lie des objets de même classe

Université de Rennes

Navigation ←



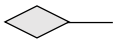
Association unidirectionnelle
On ne peut naviguer que dans un sens

UML 2.0

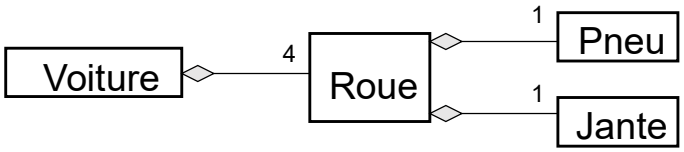
A priori, que dans les diagrammes de spécifications et d'implémentation
En cas de doute, ne pas mettre de flêche !!!

Université de Rennes

Composition



Notion intuitive de "composants" et de "composites"



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

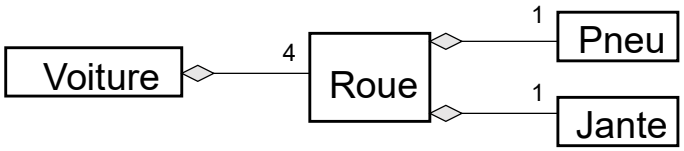
109

Université de Rennes

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

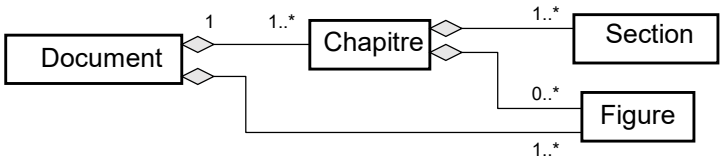


Dépend de la situation modélisée !
(Ex: vente de voitures vs. casse)

110

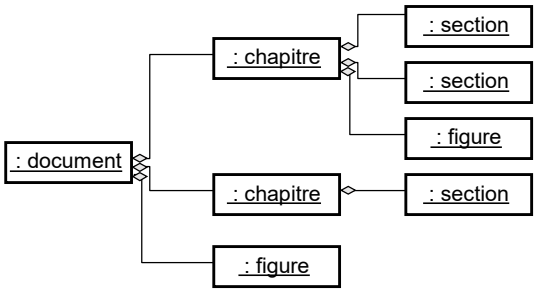
Université de Rennes

Composition



M1

M0



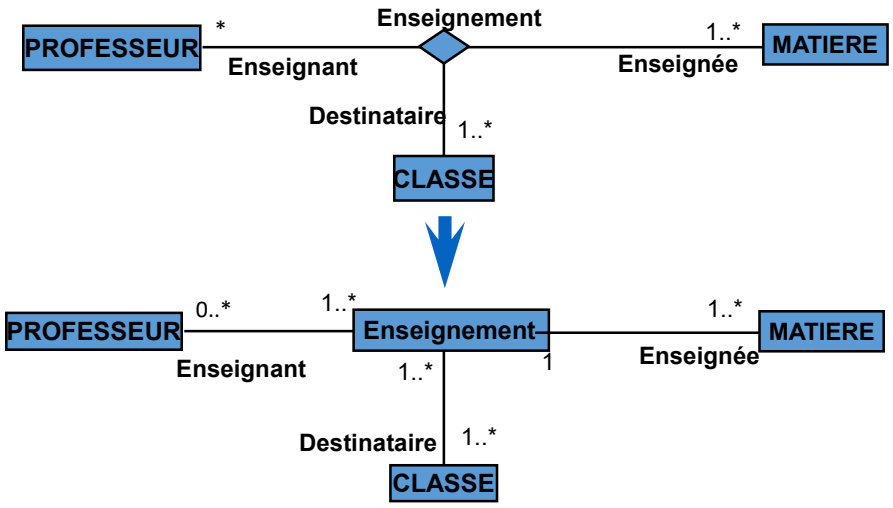
Contrainte :
le graphe d'objets forme un arbre (ou une forêt)

111

Université de Rennes

Associations n-aires

- Associations entre plus de 2 classes *(à éviter si possible)*

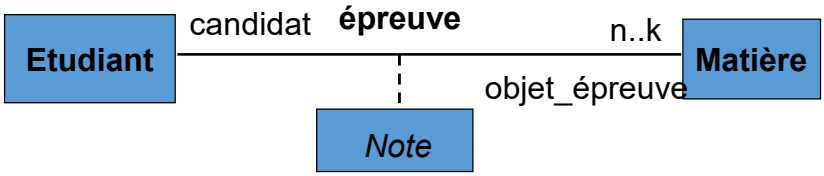


113

Université de Rennes

Associations attribuées

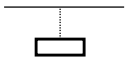
- L'attribut porte sur le lien



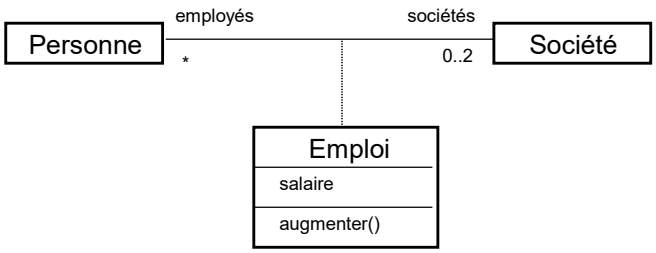
114

Université de Rennes

Classes association



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

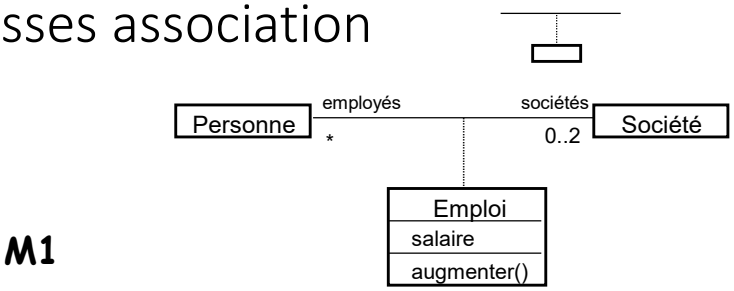


Le nom de la classe correspond au nom de l'association
(problème: il faut choisir entre forme nominale et forme verbale)

115

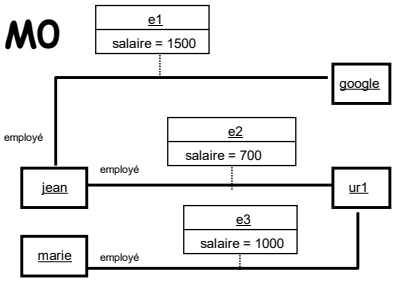
Université de Rennes

Classes association



M1

M0

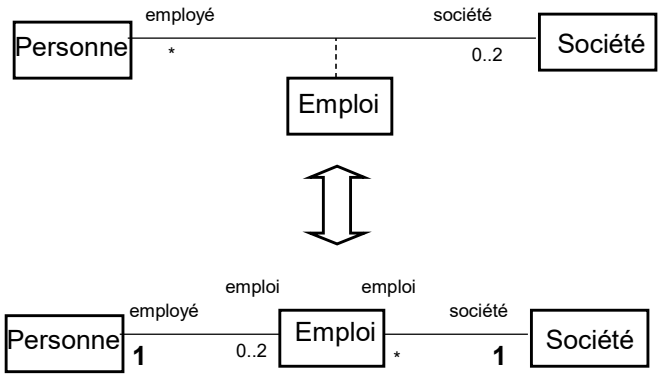


Le salaire est une information correspondant
- ni à une personne,
- ni à une société,

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

Université de Rennes

Classes association : traduction



Université de Rennes

Qualifieurs d'associations

- Un qualifieur est un attribut spécial qui permet, dans le cas d'une relation 1-vers-plusieurs ou plusieurs-vers-plusieurs, de réduire la cardinalité. Il peut être vu comme une clé qui permet de distinguer de façon unique un objet parmi plusieurs.
- Exemple

Association non qualifiée

Association qualifiée

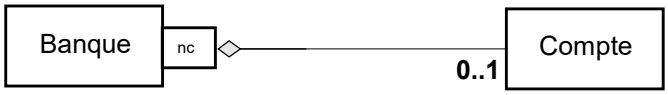
Un répertoire + un nom de fichier identifient de façon unique un fichier

118

Université de Rennes

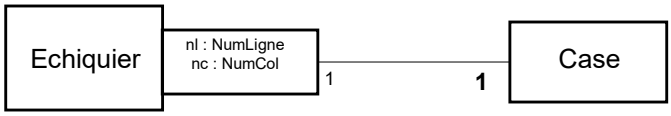
Cardinalité des Associations Qualifiées

Cas classique: cardinalité 0..1



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

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

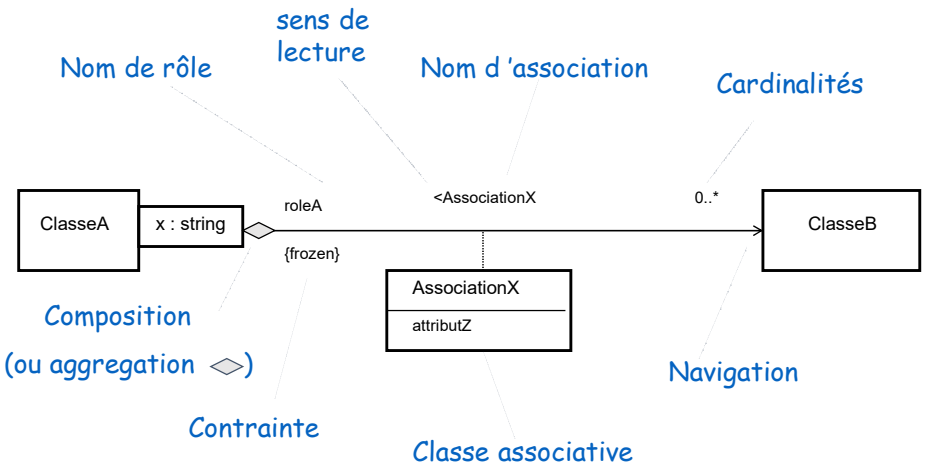


0 comme cardinalité minimale, sauf si le domaine de l'attribut qualifieur est fini et toutes les valeurs ont une image.

119

Université de Rennes

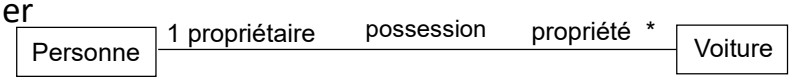
Synthèse sur les associations



Université de Rennes

Contraintes OCL navigant les associations

- Chaque association est un chemin de navigation
- Le contexte d 'une expression OCL est le point de départ (la classe de départ)
- Les noms de rôles sont utilisés pour identifier quelle relation on veut naviguer



Context Voiture inv:
self.propriétaire.age >= 18

Université de Rennes

Navigation des relations 0..*

- Par navigation on n 'obtient plus un scalaire, mais une *collection* d 'objets
- OCL défini 3 sous-types de collections
  - **Set** : obtenu par navigation d'une relation 0..*
    - *Context Personne inv: propriété* retourne un Set[Voiture]
    - chaque élément est présent au plus une fois
  - **Bag :** si plus d'un pas de navigation
    - un élément peut être présent plus d 'une fois
  - **Sequence** : navigation d'une relation {ordered}
    - c 'est un Bag ordonné
- Nombreuses opérations prédéfinies sur les types *collection.*

Syntaxe :
Collection->opération

122

Université de Rennes

Opérations de base sur collections

- *isEmpty*
  - vrai si la collection n'a pas d'éléments

Context Personne inv:
age<18 implies propriété->isEmpty

- *notEmpty*
  - vrai si la collection a au moins un élément
- *size*
  - nombre d'éléments dans la collection
- *count (elem*)
  - nombre d'occurrences de *elem* dans la collection

123

Université de Rennes

Opération *select*

- Syntaxes possibles
  - collection->select(elem:T | expr)
  - collection->select(elem | expr)
  - collection->select(expr)
- Sélectionne le sous-ensemble de *collection* pour lequel la propriété *expr* est vraie
- e.g.

```
context Personne inv:
propriété->select(v: Voiture | v.kilometrage<100000)->notEmpty
```

- ou en raccourci :

```
context Personne inv:
propriété->select(kilometrage<100000)->notEmpty
```

125

Université de Rennes

Opération *forAll*

- Syntaxes possibles
  - collection->forall(elem:T | expr)
  - collection->forall(elem | expr)
  - collection->forall(expr)
- Vrai si *expr* est vrai pour chaque élément de *collection*
- e.g.

```
context Personne inv:
propriété->forall(v: Voiture | v.kilometrage<100000)
```

- ou en raccourci :

```
context Personne inv:
propriété->forall(kilometrage<100000)
```

126

Université de Rennes

Autres opérations OCL

- exists (expr)
  - Vrai si *expr* est vrai pour au moins un élément de la collection
- includes(elem), excludes(elem)
  - vrai si *elem* est présent (resp. absent) dans la collection
- includesAll(coll)
  - vrai si tous les éléments de *coll* sont dans la collection
- union (coll), intersection (coll)
  - opérations classiques ensemblistes
- asSet, asBag, asSequence
  - conversions de type

127

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

128

Université de Rennes

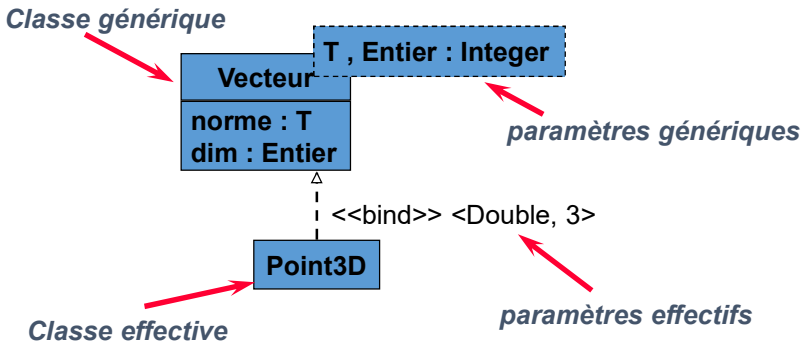
La généricité (Classes paramétrées)

- Indispensable pour les classes “conteneurs”
  - En Pascal, liste d’entiers, de réels, etc.
  - en C (C++) liste de (void *)
- Solution en Ada : donner un nom formel au type des éléments du conteneur (Liste[T])
  - Solution similaire en Eiffel (idem Ada) ou C++ (templates)
  - N’existe pas en Java = l’un de ses 3 points faibles principaux
  - Pas ce problème en Smalltalk (typage dynamique)
- Classes génériques vs. classes effectives

129

Université de Rennes

Représentation de la généricité



130

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

131

Université de Rennes

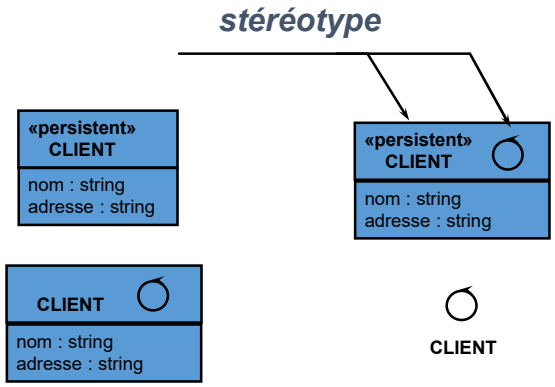
Les stéréotypes

- Nouveaux éléments de modélisation instanciant
  - Des classes du méta modèle UML (pour les stéréotypes de base UML)
  - Des extensions de classes du méta modèle UML (pour les stéréotypes définis par l'utilisateur)
- Peuvent être attachés aux éléments de modélisations et aux diagrammes :
  - Classes, objets, opérations, attributs, généralisations, relations, acteurs, uses-cases, événements, diagrammes de collaboration ...

132

Université de Rennes

Notations pour les stéréotypes

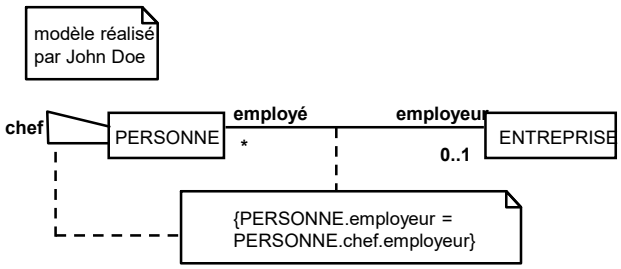


133

Université de Rennes

Les notes

- Compléments de modélisation
  - Attachés à un élément du modèle ou libre dans un diagramme
  - Exprimés sous forme textuelle
  - Elles peuvent être typées par des stéréotypes



134

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

135

Université de Rennes

Notion de package

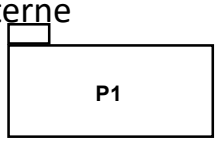
- Élément structurant les classes
  - Modularisation à l'échelle supérieure
  - Un package partitionne l'application :
    - Il référence ou se compose des classes de l'application
    - Il référence ou se compose d'autres packages
  - Un package réglemente la visibilité des classes et des packages qu'il 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

136

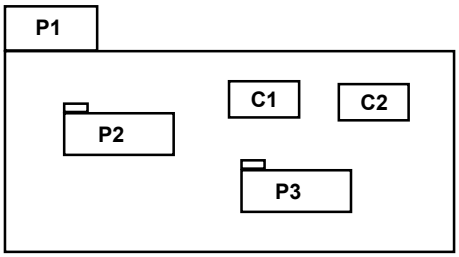
Université de Rennes

Représentation d'un package

- Vue graphique externe



- Vue graphique externe et interne

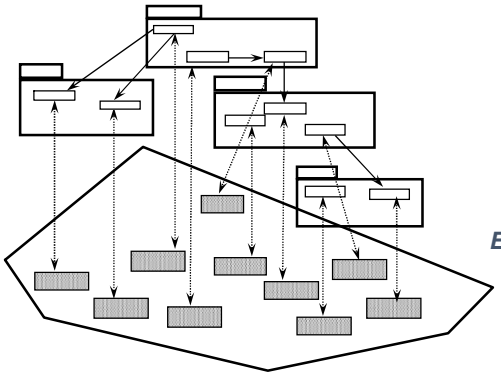


137

Université de Rennes

Partitionnement d'une application

- Définition de vues partielles d'une application



SYNTHÈSE EN PACKAGES

ENSEMBLE DES CLASSES DE L'APPLICATION

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

138

Université de Rennes

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

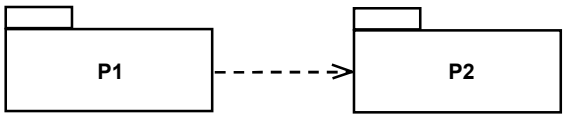
CLASSE D'INTERFACE *CLASSE DE CORPS* *CLASSE EXTERNE*

139

Université de Rennes

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



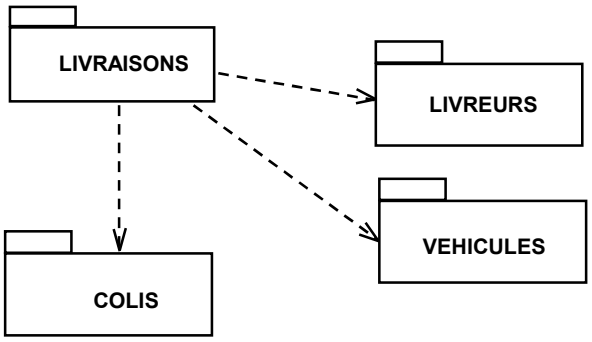
Vue externe du package P1

140

Université de Rennes

Utilisation entre packages

- Exemple (vue externe du package livraisons)

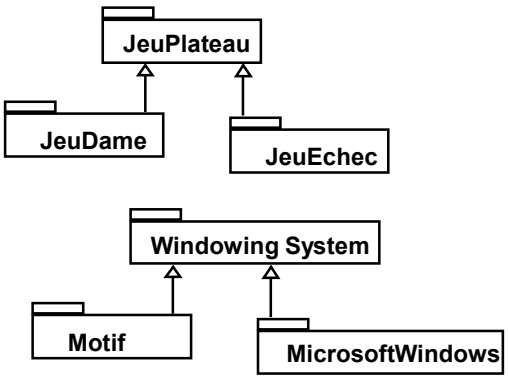


141

Université de Rennes

Héritage entre packages

- Exemples



142

Université de Rennes

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

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

Université de Rennes

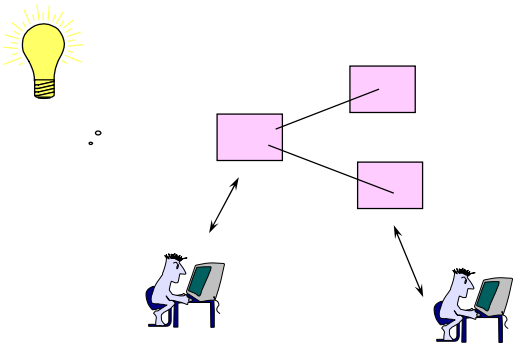
Expression des besoins et OOAD

- Sujet longtemps négligé (e.g. OMT)
  - dérive de certains projets vers le « conceptuel »
- 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

149

Université de Rennes

Quatre objectifs



- Se comprendre
- Représenter le système
- Exprimer le service rendu
- Décrire la manière dont le système est perçu

150

Université de Rennes

Les moyens

- On imagine le système « fini »
- On montre comment on interagit avec lui
  - Les acteurs UML
  - Les *use-cases* UML

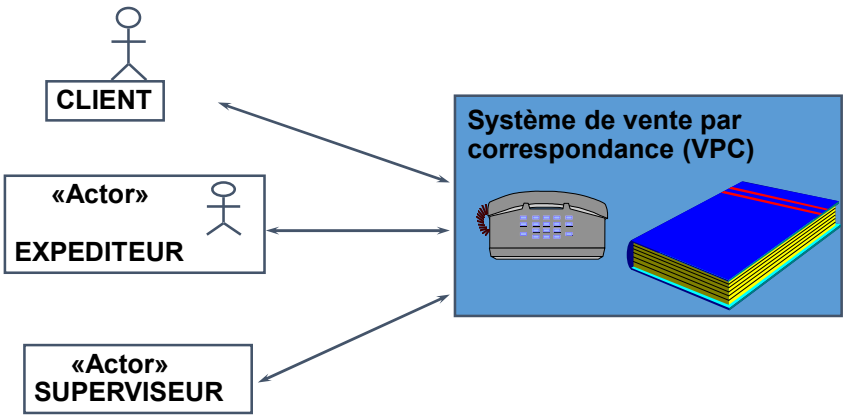
Université de Rennes

Acteurs

- Entité externe au système et amenée à interagir avec lui
  - Un acteur «joue un rôle» vis-a-vis du système
- Un acteur est une classe
- Un acteur peut représenter un être humain, un autre système, ...
- L'identification des acteurs permet de délimiter le système

Université de Rennes

Acteurs : notations



155

Université de Rennes

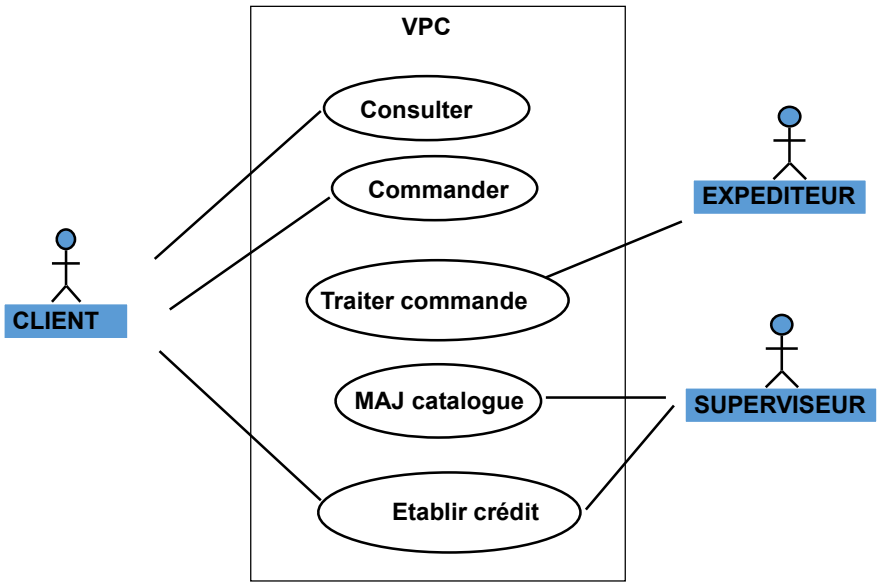
Les cas d'utilisation (use-cases)

- Un cas d'utilisation est une manière particulière d'utiliser le système
  - séquence d'interactions entre le système et un ou plusieurs acteurs
  - Ils s'expriment par des diagrammes de séquences
- La compilation des cas d'utilisation décrit de manière informelle le service rendu par le système
  - fournissent une expression "fonctionnelle" du besoin
  - peuvent piloter la progression d 'un cycle en spirale
- Les cas d'utilisation sont nommes en utilisant la terminologie décrite dans le dictionnaire

156

Université de Rennes

Cas d'utilisation : exemple et notation



157

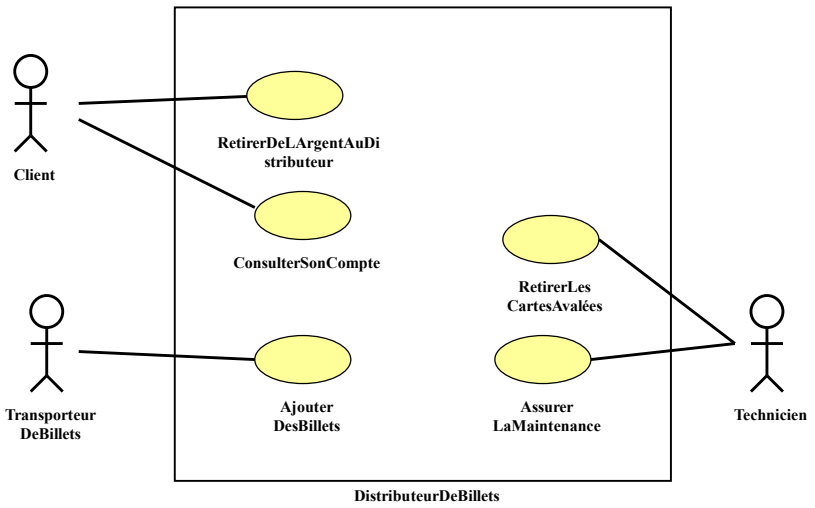
Université de Rennes

Exemple de cas d'utilisation

158

Université de Rennes

Diagramme de cas d ’utilisation



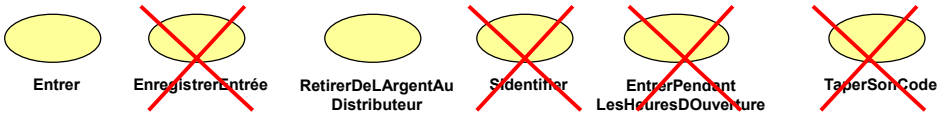
159

Université de Rennes

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**



160

Université de Rennes

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

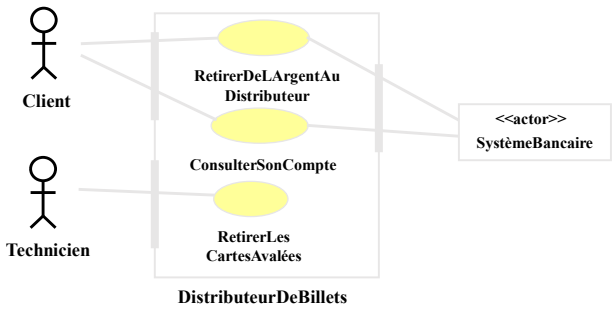
Lorsqu'un *client* a besoin de liquide il peut en utilisant un distributeur retirer de l'argent de son compte. Pour cela :
- le *client* insère sa carte bancaire dans le distributeur
- le *système* demande le code pour l 'identifier
- le *client* choisit le montant du retrait
- le *système* vérifie qu '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

161

Université de Rennes

Limites du système et interfaces

Limites du système → interface de communication
- acteur humain → interface homme – système
- acteur logiciel → interface logicielle (e.g. API)



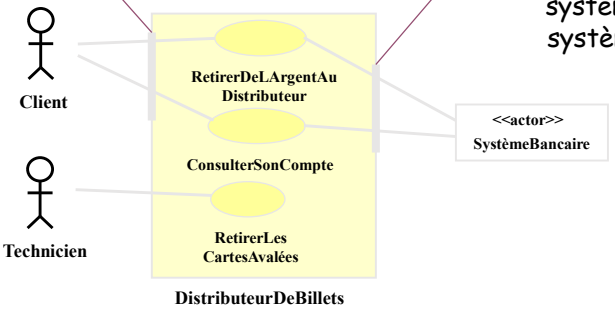
162

Université de Rennes

Interface homme-système

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Interface système-système



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

163

Université de Rennes

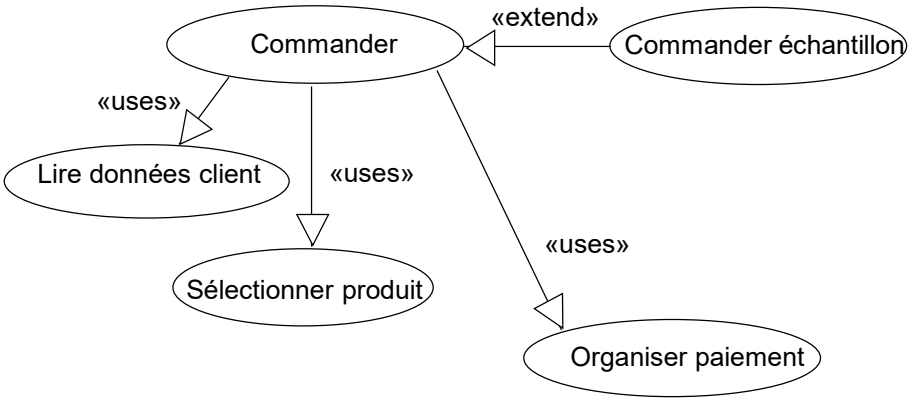
Relations sur les *use-cases*

- Communication
  - Lien entre le use case et l'acteur.
- Utilisation *(«uses»)*
  - Utilisation d'autres use-cases pour en préciser la définition
- Extension *(«extends»)*
  - Un use-case étendu est une spécialisation du use-case père

164

Université de Rennes

Relations sur les *use-cases* : notation



165

Université de Rennes

Exprimer le service rendu

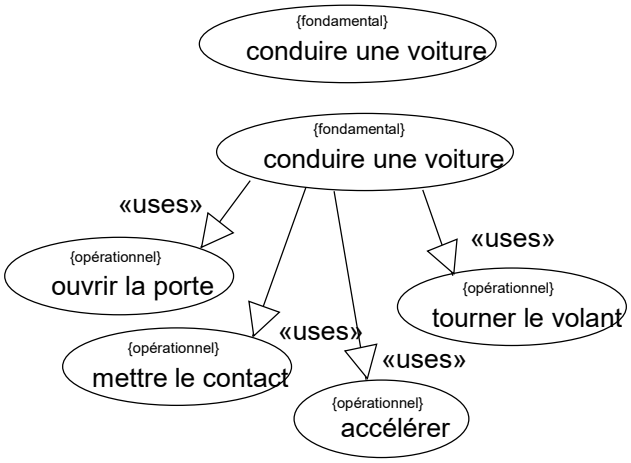
- Besoins fondamentaux : manières d'utiliser le système
  - Représentation globale par cas d'utilisation
  - ∀ taille du système, seulement de 3 à 10 Use Cases
- Besoins opérationnels : interactions avec le système
  - Représentation détaillée par raffinement des cas d'utilisation
  - Début de décomposition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

166

Université de Rennes

Besoins fondamentaux et opérationnels

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

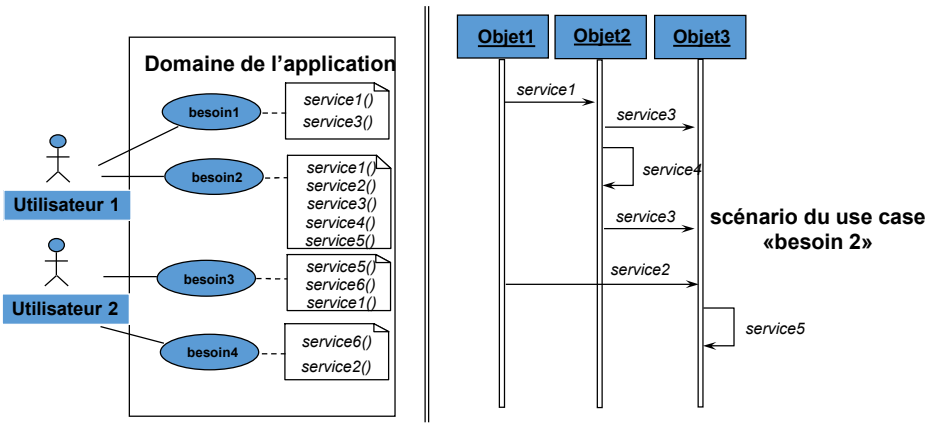


167

Université de Rennes

Utiles pour l'établissement de scénarios

- Modélisation d'exemples issus des use-cases



168

Université de Rennes

ATTENTION

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

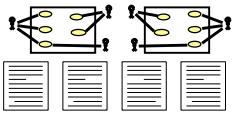
[Applying UML and Patterns, Craig Larman]

"*A big danger of use cases is that people make them too complicated and get stuck. Usually you'll get less hurt by doing too little than by doing too much*".

[UML Distilled, Martin Fowler]

169

Université de Rennes



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**

170

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

171

Université de Rennes

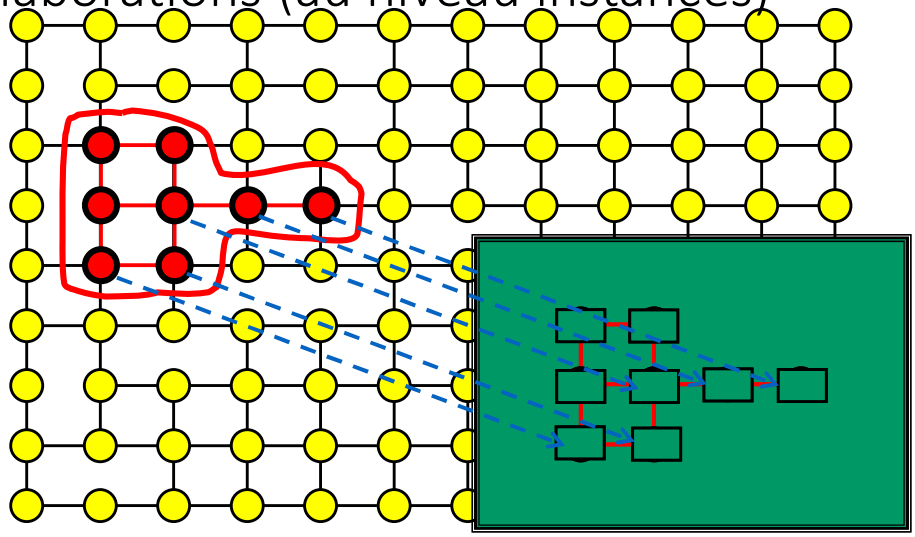
Aspects dynamiques du système

- 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

172

Université de Rennes

Collaborations (au niveau instances)



173

Selon T. Reenskaug...

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

174

Université de Rennes

Diagrammes de séquences (scénarios)

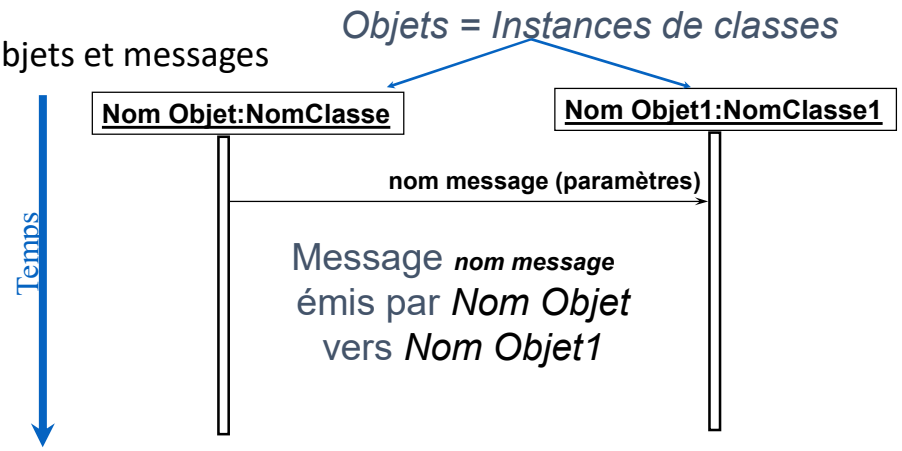
- Dérivés des scénarios de OMT :
  - Montrent des exemples de coopération entre objets dans la réalisation de processus de l'application
  - Illustrent la dynamique d'enchaînement des traitements à travers les messages échangés entre objets
  - le temps est représenté comme une dimension explicite
    - 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 subséquement
  - Les contraintes de temps (aspects temps réel)

175

Université de Rennes

Syntaxe graphique

- Objets et messages



176

Université de Rennes

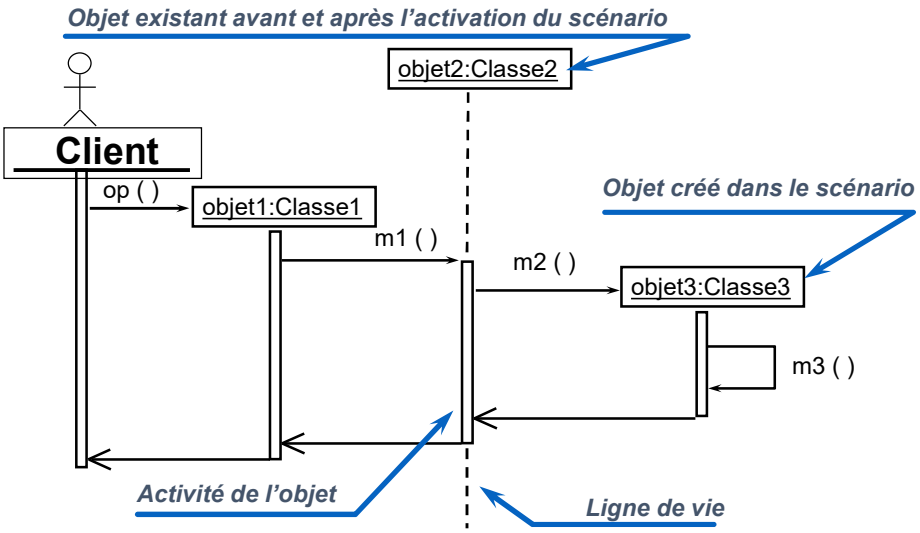
Ligne de vie et activation

- 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

177

Université de Rennes

Notation



178

Université de Rennes

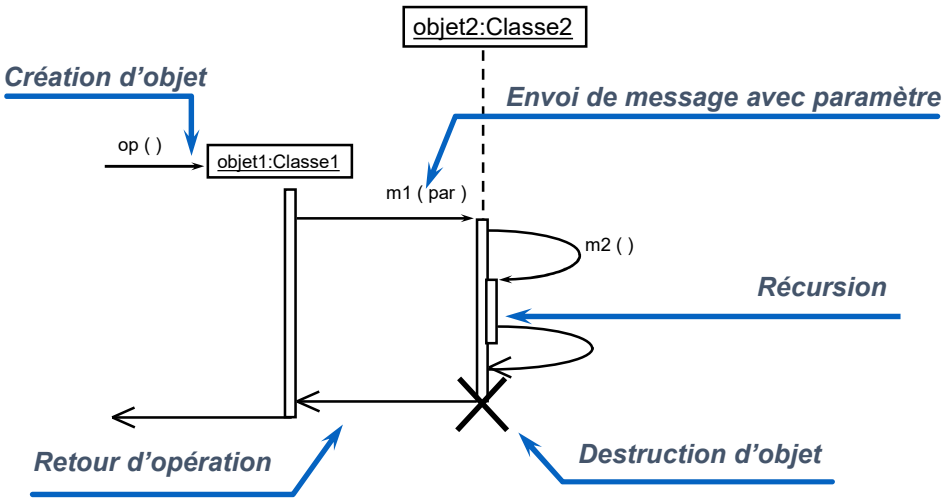
Messages

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

179

Université de Rennes

Notations

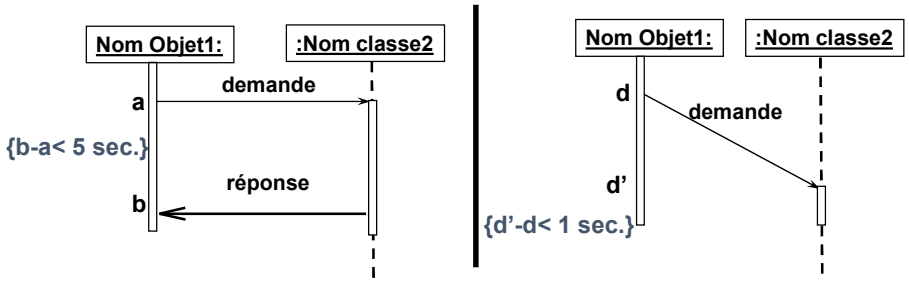


180

Université de Rennes

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



181

Université de Rennes

Fragments combinés (UML2)

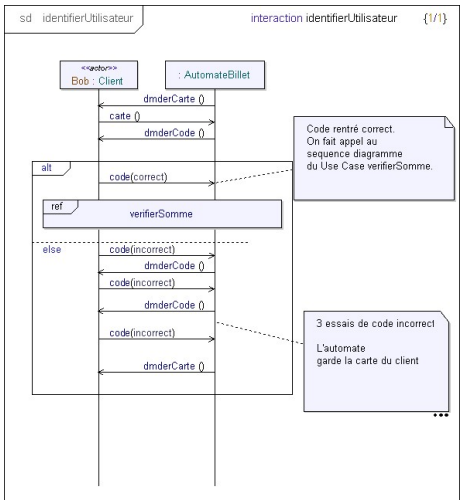
- Les fragments combinés permettent de décrire des diagrammes de séquence de manière compacte.
- Pour combiner des fragments de séquences, il existe une dizaine d'opérateurs définis dans la notation UML 2.0.
  - **Alternative (alt)**
  - **Option (opt)**
  - **Boucle (loop)**
  - **Parallèle (par)**
  - « Break », pour indiquer que le reste du scénario ne sera pas couvert
  - « Neg », pour les scénarios d'abus ou invalides
  - « Critical », pour désigner une section critique
  - « Ignore » et « Consider », pour filtrer les message
  - « Assert », pour indiquer une assertion dans un scénario
  - Séquençage faible ou stricte

183

Université de Rennes

Fragments combinés: alternative

- Exemple: soit l'utilisateur entre un code correct et dans ce cas le diagramme de séquence relatif à la vérification du code est appelé, soit l'utilisateur entre un code erroné trois fois et sa carte est gardée.
- On remarque ici une référence (**ref**) vers un diagramme défini ailleurs.



184

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

185

Université de Rennes

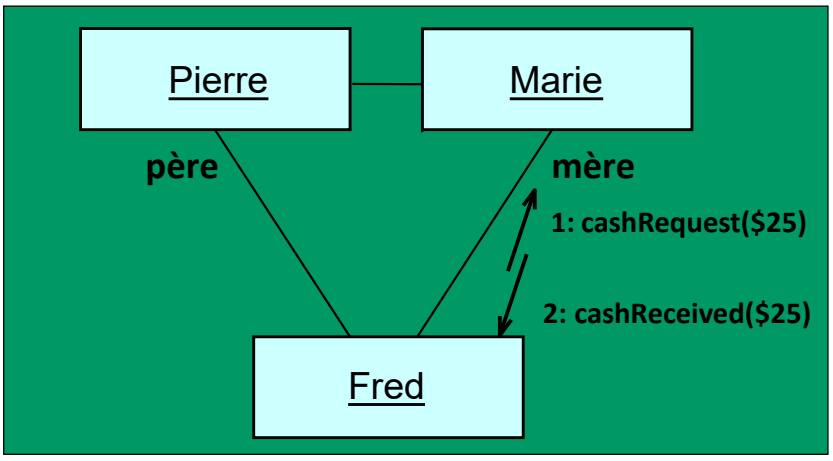
Diagrammes de collaboration

- 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

186

Université de Rennes

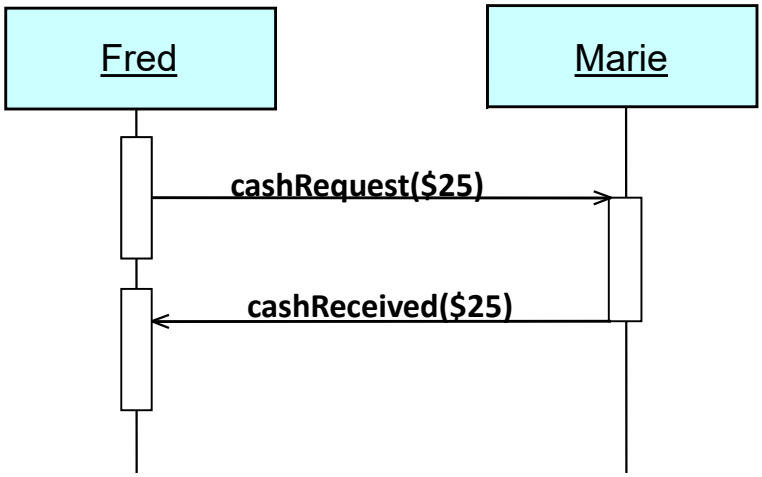
Représentation d'une collaboration (niveau instance)



187

Université de Rennes

Equivalent au diagramme de séquence:



188

Université de Rennes

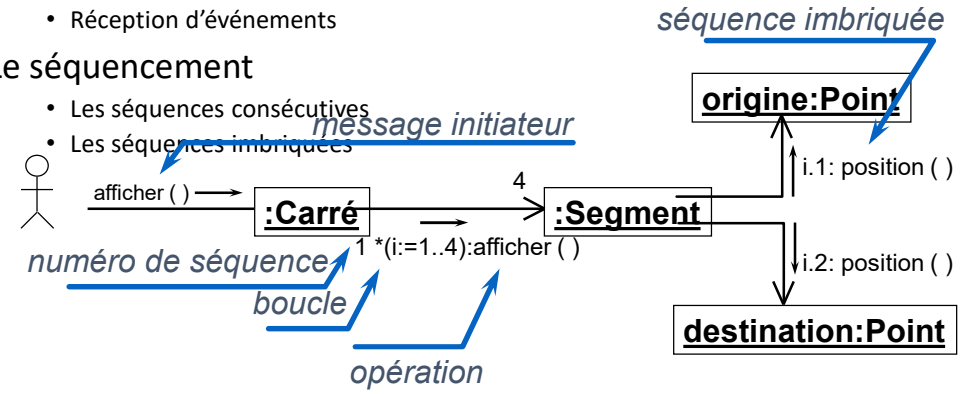
Eléments constitutifs

- 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

190

Syntaxe graphique

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



191

Questions auxquelles répondent les collaborations

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

192

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

193

Université de Rennes

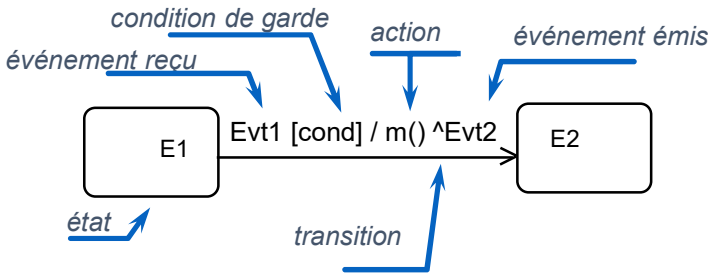
Les diagrammes d'états

- 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)*

194

Université de Rennes

Syntaxe graphique: diagramme d'états



Syntaxe :

EvénementReçu (param : type, ...) [condition de garde] / Action ^EvénementsEmis

Université de Rennes

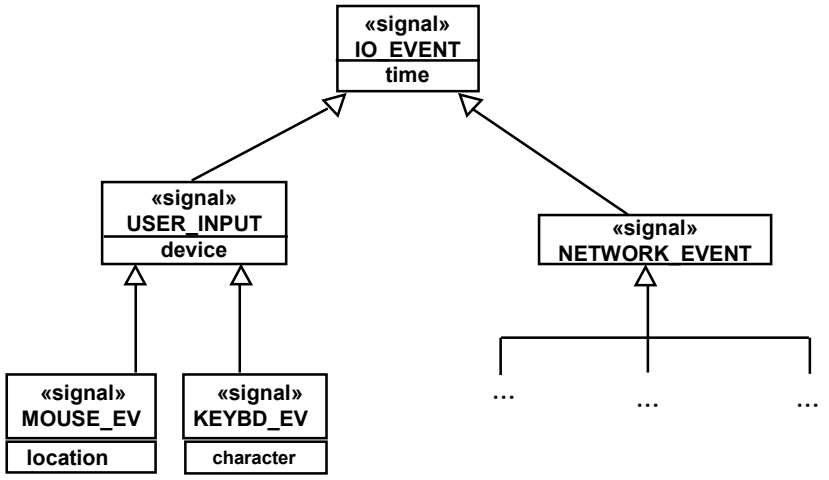
Notion d'événements

- 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»).

Université de Rennes

Les événements

- Les événements sont considérés comme des objets



197

Université de Rennes

Typologie d'événements

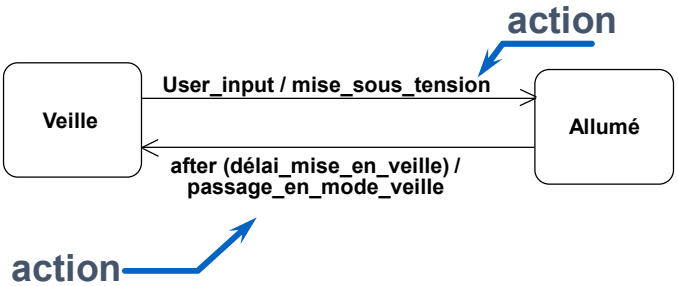
- Réalisation d'une condition arbitraire
  - transcrit par une condition de garde sur la transition
- Réception d'un signal issu d'un autre objet
  - transcrit en un événement déclenchant sur la transition
- Réception d'un appel d'opération par un objet
  - transcrit comme un événement déclenchant sur la transition
- Période de temps écoulée
  - transcrit comme une expression du temps sur la transition
  - E.g. *after(5s)*

198

Université de Rennes

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



199

Université de Rennes

Notion d’états

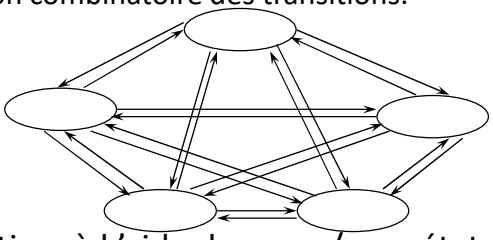
- Etat : situation stable d’un objet parmi un ensemble de situations pré-définies
  - conditionne la réponse de l’objet à des événements
    - programmation réactive / « temps réel »
  - Intervalle entre 2 événements, il a une durée
  - Abstraction de l’état concret d’un objet
    - Par ex. Compte avec solde codé sur un entier -> 2^32 états concrets
      - Mais pour l’application de banque, 2 états important : Débiteur ou Créditeur
- Peut avoir des variables internes
  - attributs de la classe supportant ce diagramme d’états

200

Université de Rennes

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

201

Université de Rennes

Notion d'activité dans un état

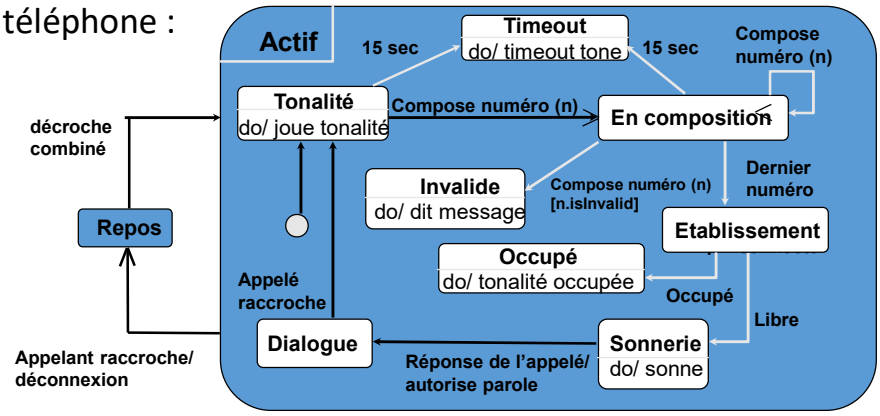
- Activité : opération se déroulant continuement tant qu'on est dans l'état associé
  - do/ *action*
- Une activité peut être interrompue par un événement.

202

Université de Rennes

Exemple de diagramme d'états

- un téléphone :

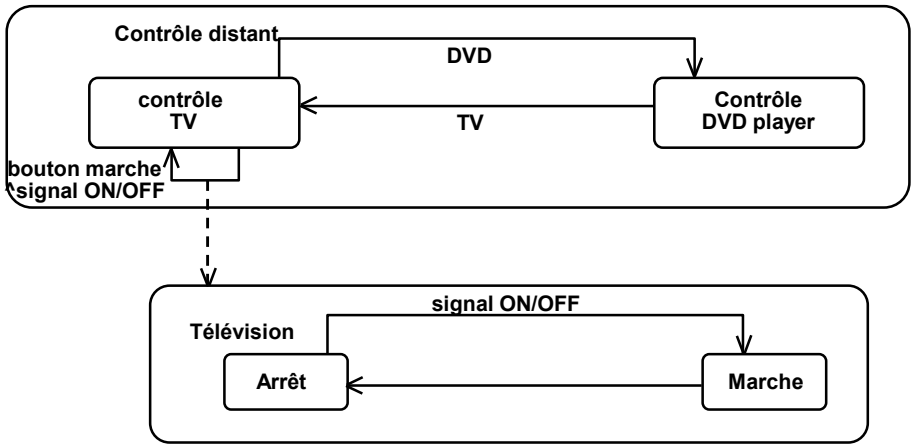


203

Université de Rennes

Émission d'événements

- Automate d'états d'une télécommande double :
  - TV + DVD Player



204

Université de Rennes

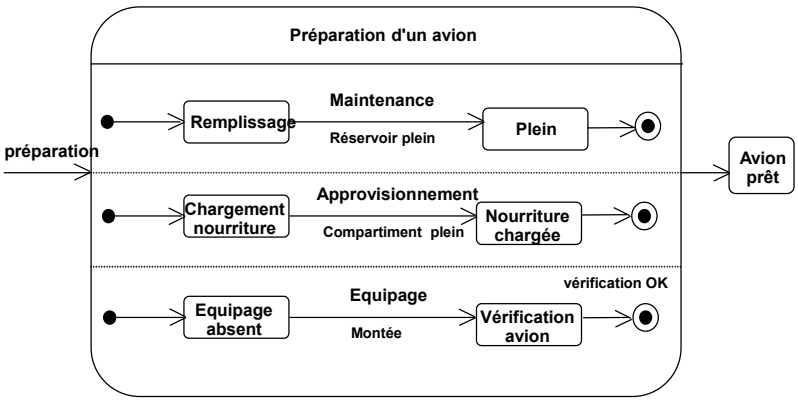
Diagrammes d'états concurrents

- 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* »

205

Université de Rennes

Exemple de concurrence

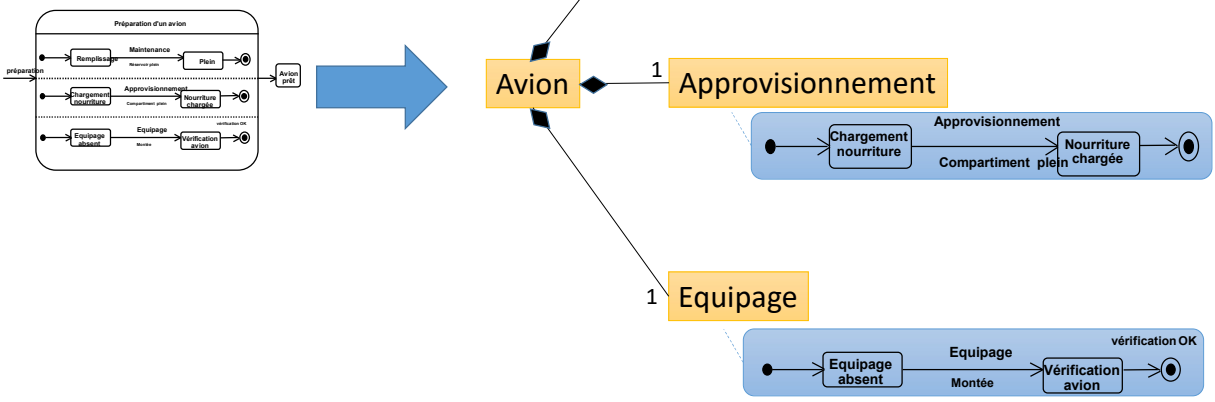


206

Université de Rennes

Privilégier la composition de points de vues

Diagramme de classes
+ 3 machines à état simples



Université de Rennes

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.

208

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

209

Université de Rennes

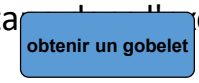
Les diagrammes d'activité

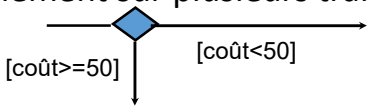
- Traitements effectués par une opération
  - Description d'un flot de contrôle procédural
    - Réseau d'actions et de transitions : automate dégénéré
    - La transition s'effectue lorsque l'opération est terminée
  - Pas de déclenchement par événement asynchrone
    - Sinon utilisation diagrammes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

210

Université de Rennes

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 éta[illegible]écution d'un algorithme
  - Notation : 

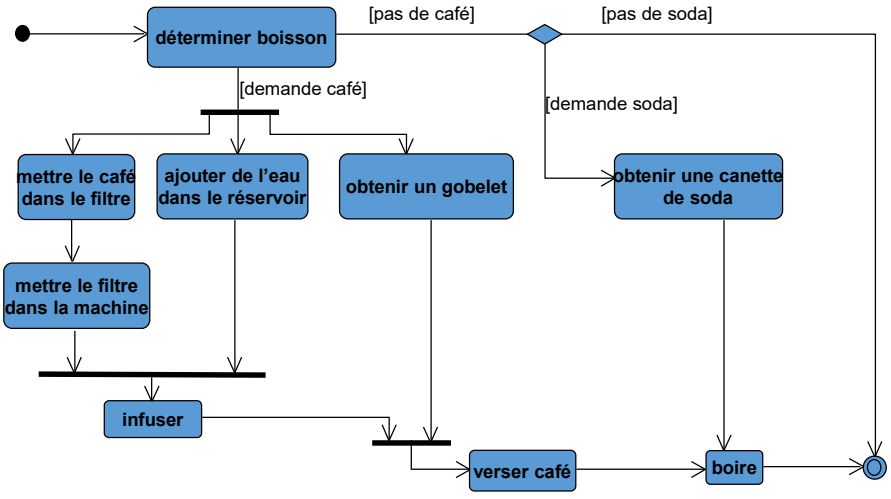
- Décision = branchement sur plusieurs transitions
  - Notation : 

211

Université de Rennes

Exemple de diagramme d'activité

opération PréparerBoisson de la classe Personne



212

Université de Rennes

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

214

Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

215

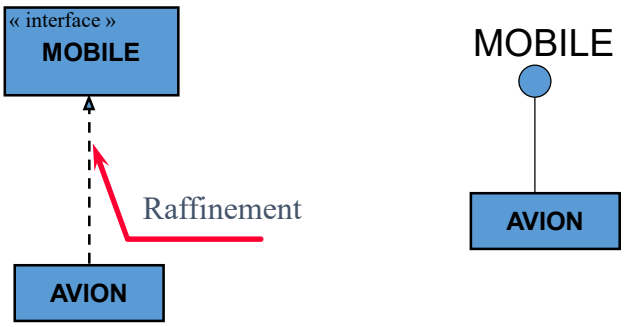
Université de Rennes

Le diagramme d'architecture ou diagramme de structure composite

- Description de la structure interne d'un objet complexe (composants) lors de son exécution (au *run-time* - décrire l'exécution du programme)
- Description de ses points d'interaction avec le reste du système
- Inspiré par l'électronique

Université de Rennes

Interfaces et « lollipop »



Université de Rennes

Port

- Un **port** permet de spécifier les *points d'interactions* d'un objet
  - entre l'objet et son environnement
  - entre l'objet et sa décomposition interne.

- Un port indique les **interfaces** proposées par l'objet (entrées) ainsi que les interfaces requises (sorties)
  - Les ports sont reliés au moyen de **connecteurs** (lignes) sur lesquels transitent des messages
- Un **port comportemental** (*Behaviour port*) est directement associé à la machine à états de la classe qui porte ce port.
  - Tous les messages envoyés sur ce port sont consommés par la machine à états de la classe.

Université de Rennes

Port

- Les ports apparaissent essentiellement dans les diagrammes de **classes** et d'**architecture**.
- Un port peut être placé sur :
  - une classe
  - un part
  - sur le pourtour (frame) d'un diagramme d'architecture

Université de Rennes

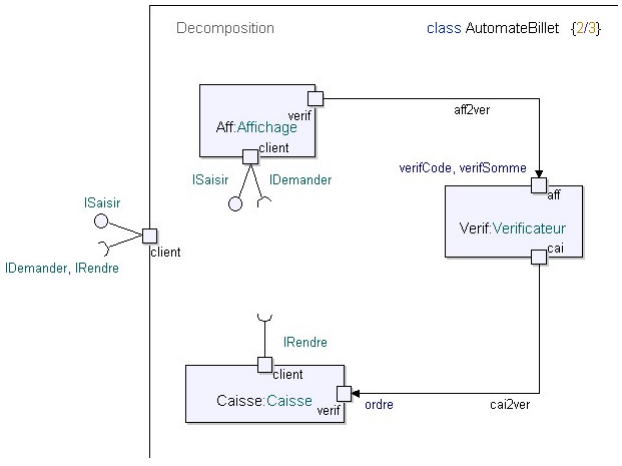
Parts

- Désigne une ou plusieurs instances qui compose(nt) une classe
- Un part est représenté dans un diagramme d'**architecture** (nouveau diagramme proposé par UML2.0) aussi appelé **diagramme structure composite**

- Un part représente une ou plusieurs instances d'une classe grâce à des contraintes de multiplicité

Université de Rennes

Exemple de diagramme d'architecture



Université de Rennes

Modélisation UML

- Modélisation selon 4 points de vue principaux :
  - Aspects statiques du système *(le QUI?)*
    - Description des objets et de leurs relations
      - Modularité, contrats, relations, généricité, héritage
    - Structuration en paquetages
  - Vision utilisateur du système *(le QUOI?)*
    - Cas d'utilisation
  - Aspects dynamiques du système *(le QUAND?)*
    - Diagramme de séquences (scénarios)
    - Diagramme de collaborations (entre objets)
    - Diagramme d'états-transitions (Harel)
    - Diagramme d'activités
  - Vision implantation *(le OU?)*
    - Diagramme de composants et de déploiement

222

Université de Rennes

Processus de développement avec UML

223