Ingénierie des besoins logiciels

Standish Group, CHAOS Report

CHAOS FACTORS OF SUCCESS

FACTORS OF SUCCESS	POINTS	INVESTMENT
Executive Sponsorship	15	15%
Emotional Maturity	15	15%
User Involvement	15	15%
Optimization	15	15%
Skilled Resources	10	10%
Standard Architecture	8	8%
Agile Process	7	7%
Modest Execution	6	6%
Project Management Expertise	5	5%
Clear Business Objectives	4	4%

Why ?

- Lack of user input
- Unclear objectives
- Incomplete requirements and specifications
- Changing requirements and specifications
- Lack of planning

Analyse des besoins

Description et spécification du système logiciel à développer

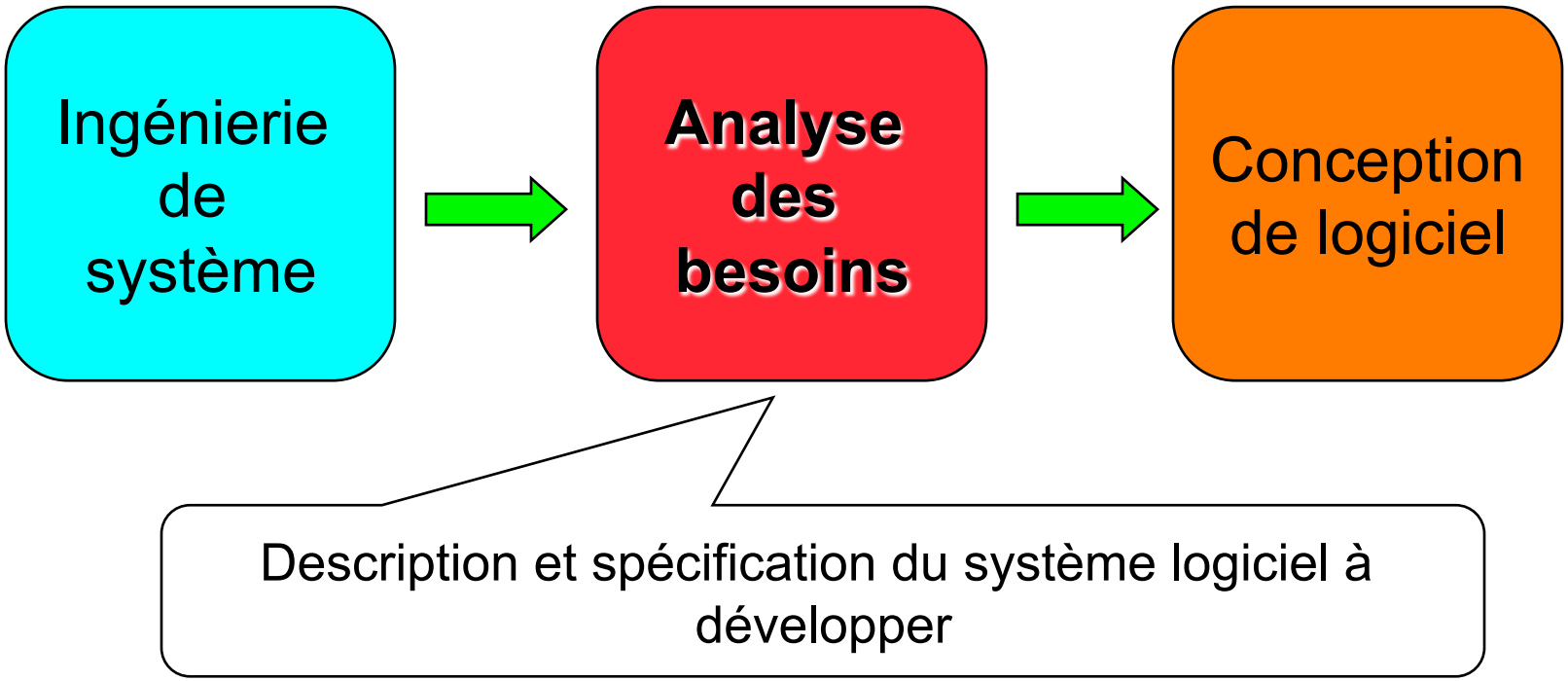
Analyse orientée objet = Object-Oriented Analysis (OOA)

Object-Oriented Analysis and Design = OOAD

Objectifs

- Le rôle et la définition de l'analyse des besoins
- La modélisation
- Le prototypage
- Analyse orientée objet

Rôle de l'analyse des besoins



Rôle de l'analyse des besoins

- L'une des étapes les plus importantes
  - étape déterminante pour la suite
  - aspects contractuels

- L'une des étapes les plus difficiles
  - différents intervenants :
    le client, les utilisateurs, les développeurs…
  - le problème n'est pas encore formalisé

Les problèmes potentiels

Le client

- Une vague idée des requis
- Change souvent des idées

L'équipe de développement

Commençons on trouvera les détails plus tard

Conduit à des erreurs de spécification et de développement

Beaucoup de systèmes informatiques ne sont jamais utilisés car ils ne correspondent pas aux besoins du client !

Analyse de domaine

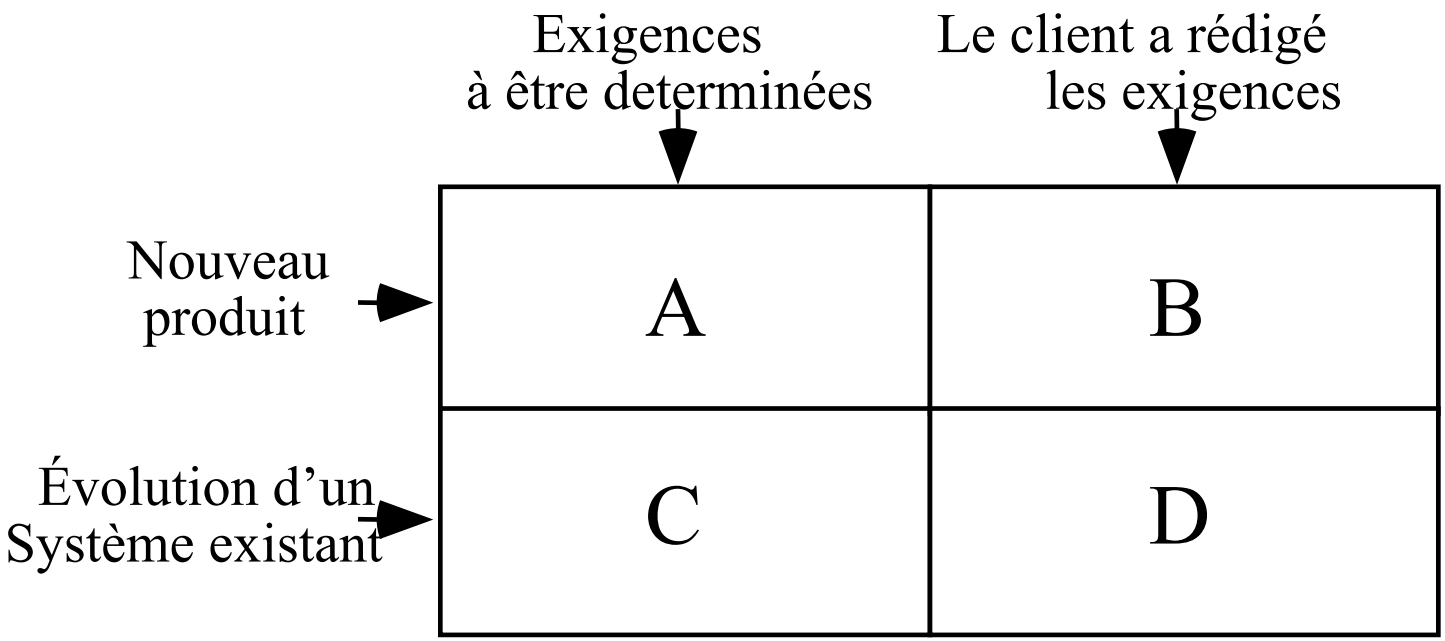
- Il s'agit du processus par lequel un ingénieur logiciel peut apprendre à connaître le domaine de l'application:
  - Le *domaine* correspond au champs d'application dans lequel le logiciel sera utilisé.
  - Un expert de ce domaine est une personne ayant une connaissance approfondie de ce domaine
  - Cette étude a pour but ultime de mieux comprendre le problème à résoudre

- Bénéfices découlant de l'analyse de domaine:
  - Accélère le développement
  - Permet le développement d'un meilleur système
  - Permet d'anticiper les possibles extensions

Document d'analyse de domaine

A. Introduction
B. Glossaire
C. Connaissances générales
D. Les clients et utilisateurs
E. L'environnement
F. Tâches et procédures en usage
G. Logiciel concurrent
H. Similarités avec d'autres domaines

Le point de départ

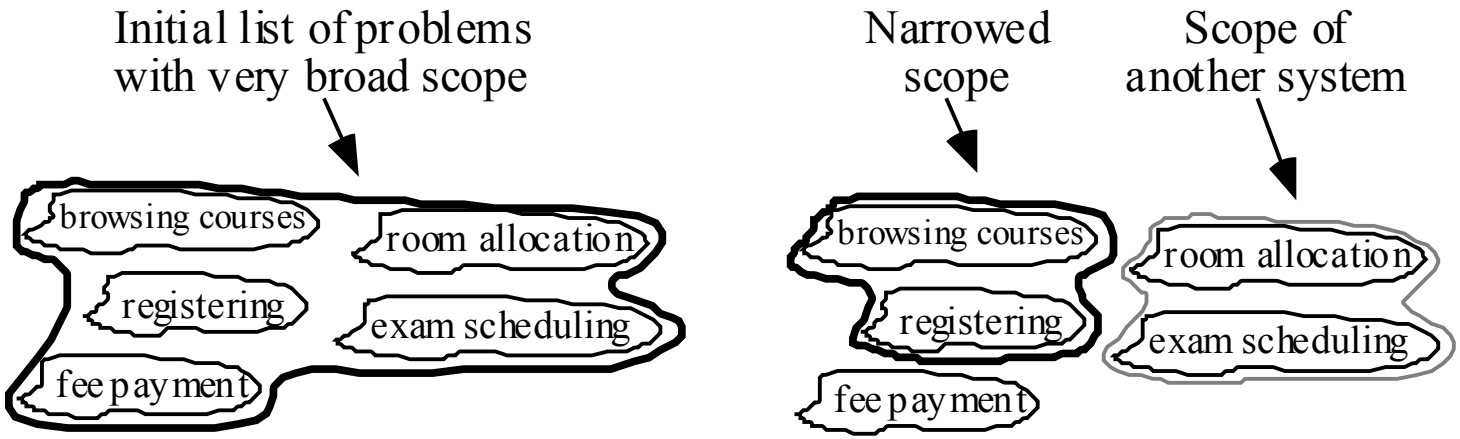


Définition du problème & de sa portée

- Un problème peut être exprimé comme:
  - Une difficulté auquel se voit confronté le client ou les utilisateurs
  - Ou une opportunité qui produira un certain bénéfice tel que une augmentation de productivité ou de ventes.
- La résolution du problème mène au développement d'un logiciel
- Un bon énoncé de problème doit être clair et concis

Définir la portée

- Cerner la portée en définissant de façon plus précise le problème à résoudre
  - Lister toutes les choses que le système devrait avoir à faire
    - Exclure le plus de choses possibles pour réduire la complexité du problème
    - Établir des buts plus généraux si le problème devient trop simple
- Exemple: un système automatisé d'inscription universitaire



Qu'est-ce qu'une exigence?

- Tout énoncé à propos du système à concevoir avec lequel tous les joueurs impliqués sont en accord et qui doit devenir vrai afin de résoudre adéquatement le problème du client.
  - Clair et concis
  - Concerne le système
  - Toutes les parties prenantes ont confirmé sa validité
  - Aide à résoudre le problème

- Un ensemble d'exigences est un *cahier des charges* (*requirements document*).

Définition des exigences

- Restituer les informations sous forme synthétique
- Ce qui n'est pas écrit n'existe pas !
- Préciser les besoins, pas la solution
- Préciser ce que doit faire le système
  - et aussi ce qu'il n 'est pas sensé faire.
  - mais surtout pas comment il doit le faire.
- Etablir des priorités
- Arriver à un consensus avec le client

MoSCoW technique:

- Must have
- Should have
- Could have
- Won't have or Would like

Types des exigences

- Besoins fonctionnels
  - à quoi sert le système
  - ce que doit faire le système, les fonctions utiles
- Besoins non fonctionnels
  - performance, sûreté, confidentialité, portabilité, etc.
  - chercher des critères mesurables

Exigences fonctionnelles

- Quelles doivent être les *entrées* du système
- Quelles *sorties* le système doit-il produire
- Quelle sont les *données* qui devront être stockées pour usage par d'autres systèmes
- Quels sont les *calculs* à effectuer
- La *mise en marche* et la *synchronisation* de tous ces éléments

Exigences non-fonctionnelles

- *Doivent être vérifiables*
- Trois catégories
  1. Catégorie liée à l'usage, l'efficacité, la maintenance et la réutilisation
     - Temps de réponse
     - Rendement, débit
     - Utilisation des ressources
     - Fiabilité
     - Disponibilité
     - Restauration après pannes
     - Facilité de maintenance et d'amélioration
     - Facilité de réutilisation

Exigences non-fonctionnelles

2. Catégorie liée à l'environnement et à la technologie
   - Plate-forme
   - Technologie à utiliser

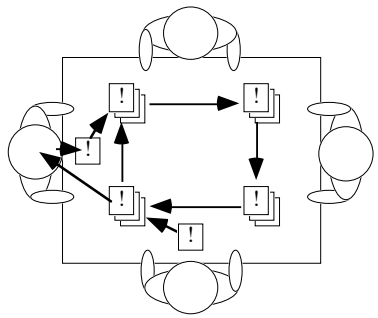
3. Catégorie liée à la planification et à la méthode de développement du projet
   - Processus de développement à utiliser
   - Coût et date de livraison
     - (ces derniers éléments se retrouvent souvent dans le contrats signé par les parties)

Techniques pour capturer des exigences

- Observation
  - Lire tous les documents disponibles et discuter avec les utilisateurs
  - Observer des utilisateurs potentiel à leur travail
    - leur demander en quoi consiste leur travail, ce qu'il sont en train de faire
  - Filmer et enregistrer des sessions de travail
- Entrevues
  - Mener une série d'entrevues
    - Demander des détails spécifiques
    - Demander à chacun sa vision du problème et de sa solution
    - Demander à chacun si ils ont des idées à proposer
    - Demander à chacun de proposer des sources d'information intéressantes
    - Demander leur de faire des dessins et diagrammes

Techniques pour capturer des exigences

- Séance de réflexion (Brainstorming)
  - Recourir à un modérateur expérimenté
  - Disposer l'assistance autour d'une table
  - Lancer une question de départ
  - Demander à chacun des participants d'écrire une réponse et de la passer à son voisin



- ***Développement conjoint d'applications (JAD)*** **est une technique reposant sur des séances intensives de réflexion**

Techniques pour capturer des exigences

- Prototypage
  - Sa forme la plus simple: *un prototype sur papier*.
    - Un ensemble de schémas montrant le système en action
  - Le plus commun: une maquette de l'interface usager
    - Écrit dans un langage permettant la conception rapide d"interface
    - N'effectue aucun calcul, aucune réelle interaction
    - Peut concerner un aspect particulier du système

Techniques pour capturer des exigences

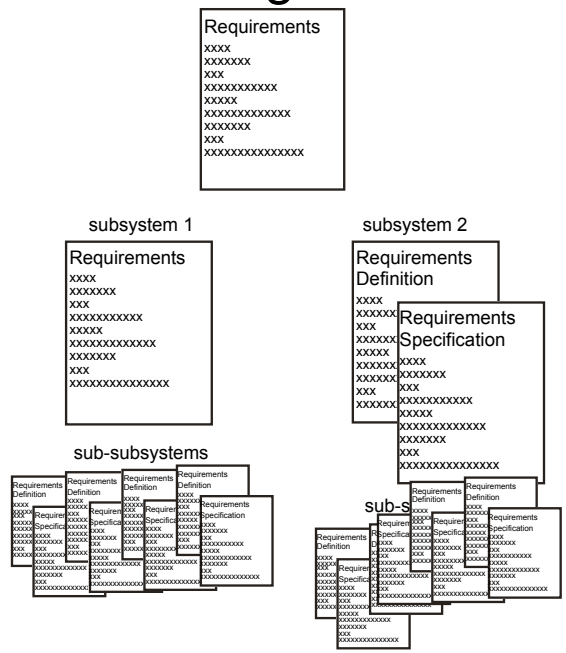
- Analyse de cas informelle
  - Déterminer les différents types d'utilisateur du système (acteurs)
  - Déterminer les tâches que chacun de ces acteurs doivent accomplir avec le système
- Diagramme de cas d'utilisation

Types de document pour les exigences

Les deux extrêmes:

1. Un résumé informel des exigences utilisant le plus souvent quelques diagrammes accompagnés d'un court texte
   On parle alors de la *définition* des exigences
2. Une liste détaillée de spécifications décrivant le système en détail
   - On parle alors de la *spécification* des exigences

Dans le cas d'un système de grande taille, plusieurs tels documents existent et sont organisés de façon hiérarchique



Niveau de détail requis

- Le niveau de détail requis dépend de plusieurs facteurs:
  - La taille du système
  - Le fait que ce système doit s'interfacer avec d'autres système
  - Le public cible
  - L'étape où en est rendu le projet
  - Le niveau d'expérience et de connaissance du domaine et de la technologie
  - Le coût encouru par l'inclusion d'exigences erronées ou ambiguës

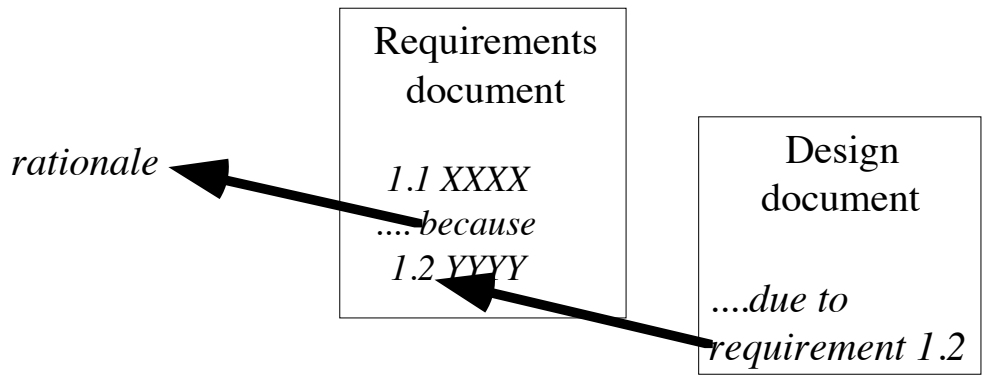
Révision des exigences

- **Chacune des exigences**
  - Doit avoir un *bénéfice* qui l'emporte sur les coûts qu'elle engendre
  - Doit être *importante* dans la solution du problème
  - Doit être exprimée d'une façon *claire, concise et consistante*
  - Doit être *non-ambiguë*
  - Doit être en *concordance* avec les standards et pratiques
  - Doit mener à un système de *qualité*
  - Doit être *réaliste* considérant les ressources disponibles
  - Doit être *vérifiable*
  - Doit être uniquement *identifiable*
  - Ne doit *pas sur-contraindre* le système

Cahier des charges...

- Ce document doit être:
  - Suffisamment complet
  - Bien organisé
  - clair
  - Accepté par les différentes parties

- Traçabilité:



Cahier des charges

A. Problème
B. Information de base
C. Environnement et modèles
D. Exigences fonctionnelles
E. Exigences non-fonctionnelles

Faire face au changements

- Les exigences changent parce que:
  - Les procédures évoluent
  - La technologie change
  - Le problème est mieux maîtrisé

- L'analyse des exigences ne s'arrête jamais
  - Toujours continuer à interagir avec le client et les utilisateurs
  - Tout changement doit être avantageux
    - De petits changements cosmétiques sont souvent faciles et peu coûteux
    - Des changement plus fondamentaux doivent être entrepris avec grande précaution
      - Effectuer des changements dans un système en cours de développement peut mener à une mauvaise conception et à des retard de livraison
  - Certains changements sont en fait des ajouts au système
    - Éviter de rendre le système plus gros,
    - il doit simplement devenir meilleur

Difficultés et risques liés au domaine et aux exigences

- Mauvaise compréhension du domaine ou du problème
  - *Une bonne analyse de domaine et le prototypage sont la clé*
- Les exigences changent rapidement
  - *Effectuer un développement incrémental, rendre le système flexible, procéder à des révision régulières*
- Vouloir trop en faire
  - *Bien délimiter le problème, bien planifier le temps de travail*
- Certaines exigences peuvent être difficiles à concilier
  - *Bien discuter des alternatives, faire des prototypes comparatifs*
- Certaines exigences sont difficiles à saisir et à exprimer
  - *Subdiviser une exigence complexe en plus petite exigences, éviter les ambiguïtés, faire des prototypes*

Quelques conseils

- Les besoins doivent être compris par tous
  - utiliser la langue naturelle
  - utiliser des schémas si nécessaire
    - tout schéma doit être commenté
    - tout schéma doit toujours comporter un légende
- Structurer le document
  - suivre un plan standard si disponible
  - numéroter les sections, références, index, …

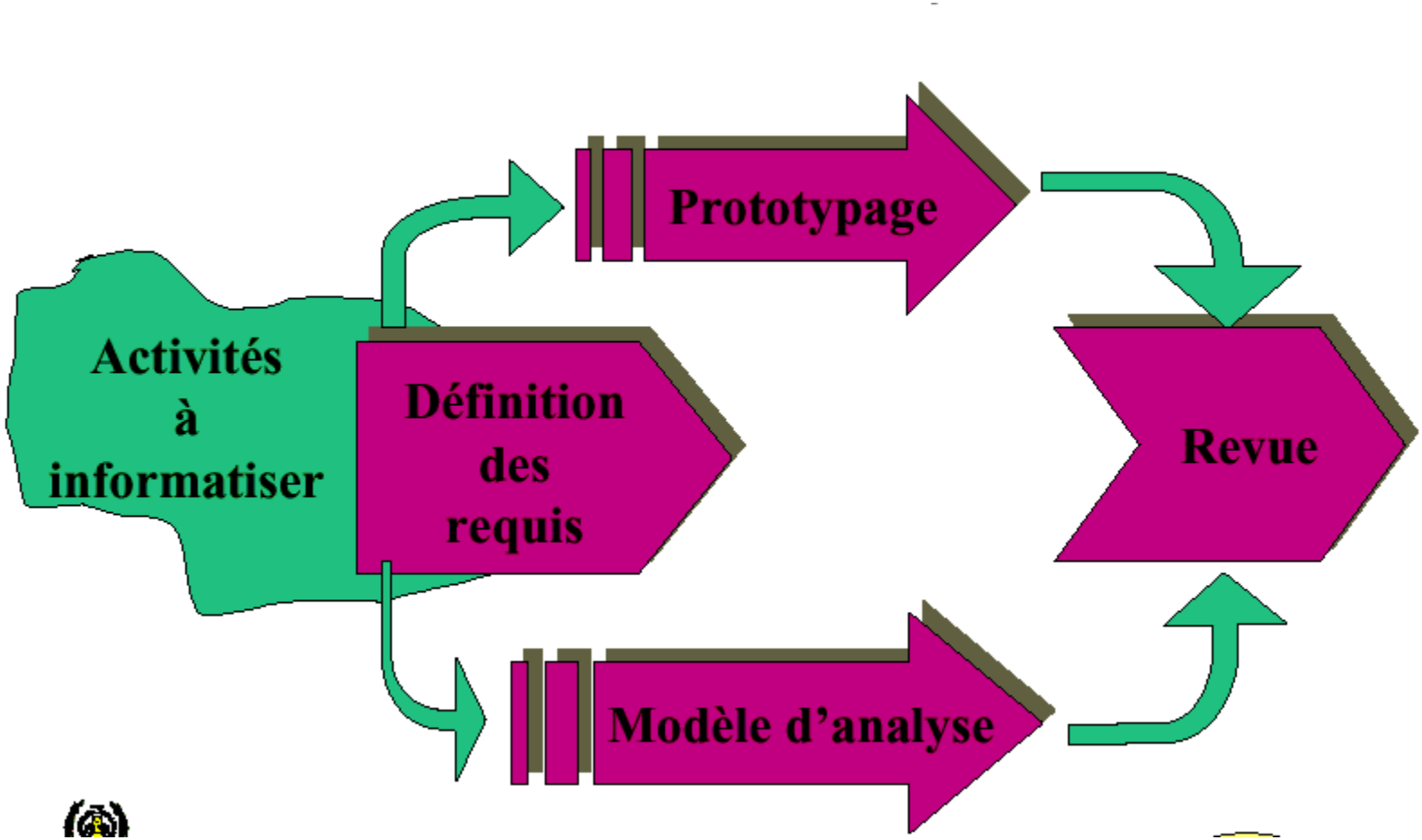
Spécification des exigences

- Interface entre le client et les développeurs
  - doit être compris par les deux
  - défini dans le détail ce qui doit être réalisé
  - doit être précis car contractuel
- Utilisation de techniques semi-formelles
  - e.g. diagrammes de cas d'utilisation UML
  - e.g. diagrammes de classes UML

Analyse des exigences

- Travailler avec le client pour définir des exigences globales au niveau du produit
- Bâtir un modèle pour l'analyse:
  + Identifier des données
  + Définir les fonctions
  + Représenter le comportement
- Prototyper les aspects incertains
- Développer une spécification qui guidera le design
- Réaliser des revues techniques formelles

Processus d'analyse



Prototypage rapide

Définition

C'est un modèle opérationnel, fonctionnement équivalent à un sous-ensemble du produit

But:

- Donner rapidement au client une compréhension du produit
- Produit réalisé pour le changement

Les situations typiques

- Les délais trop courts
- Besoins mal définis
- Satisfaction des usagers
- Mise à l'essai de nouvelles idées
- Applications simples
- Applications peu fréquentes

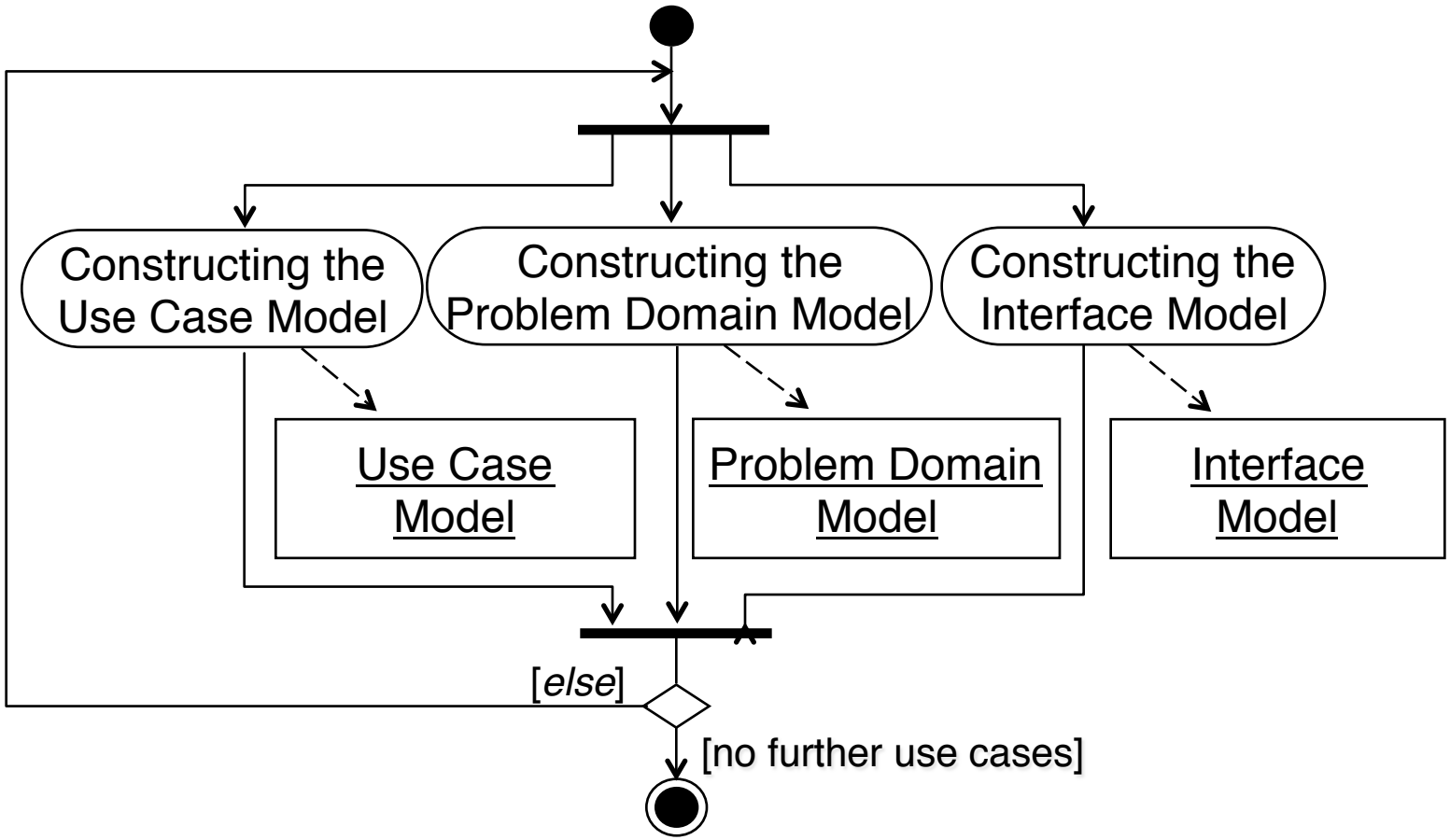
Utilisation des protypes

- Évaluation des risques
- Simulateurs
- Modèles en spirale
- Contraint de temps d'exécution
- Évaluation des interfaces
- Consensus sur les spécifications

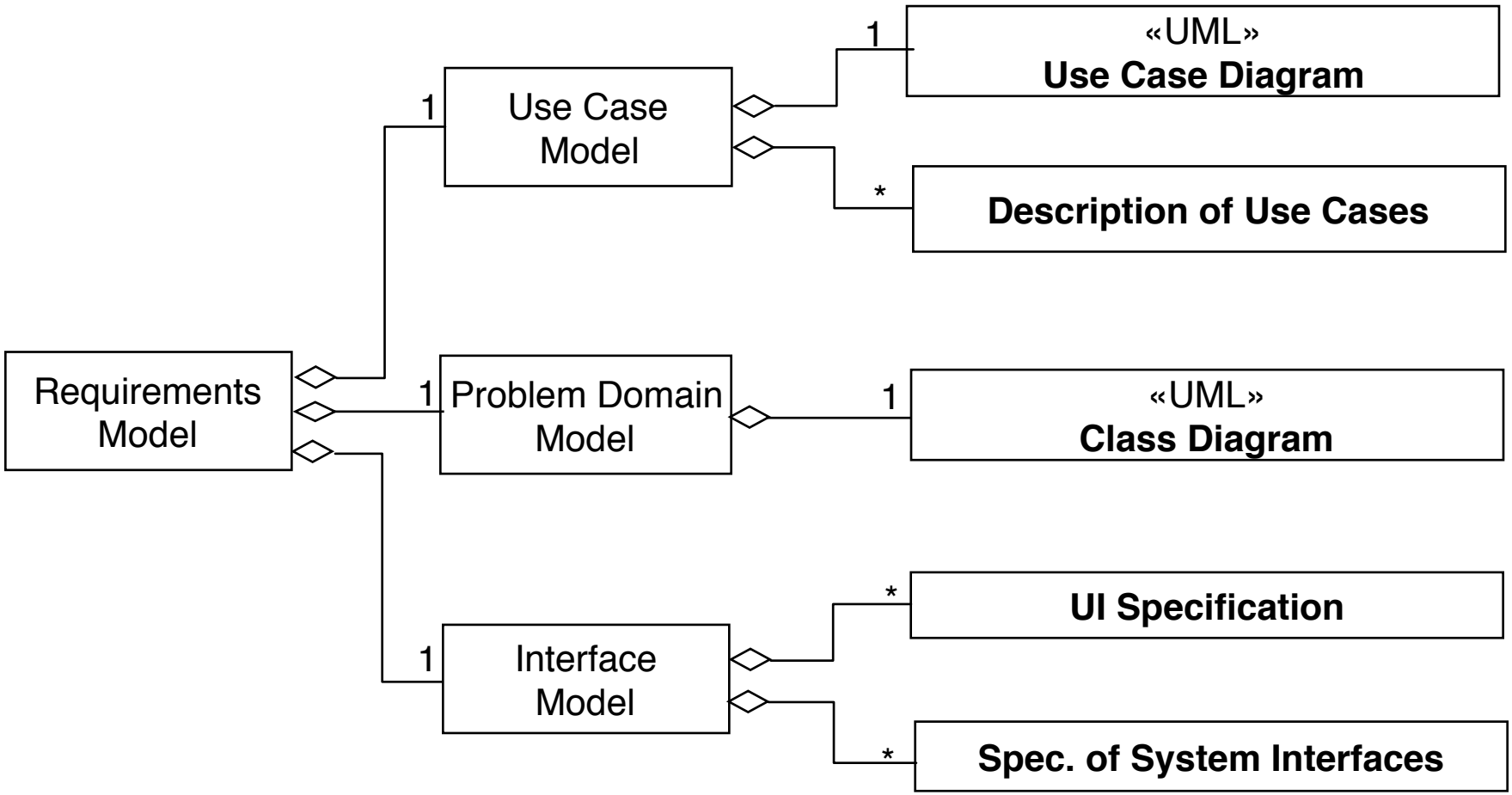
Désavantages des protypes

- Facilite (trop) de changement
- Pas de qualité opérationnelle
- Tendance à le conserver
- Contraint de temps d'exécution
- Peut remplacer les spécifications mais jamais le design

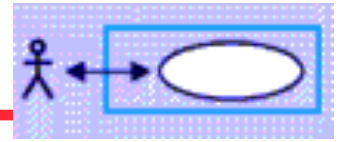
Modèle des exigences (processus)



Modèle des exigences (résultats)



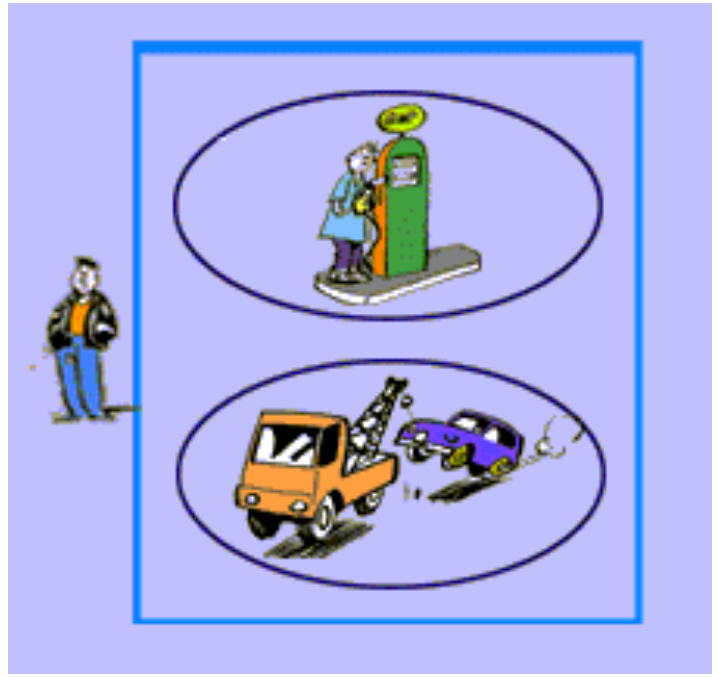
Modèle de cas d'utilisation



Cet outil permet de spécifier les services que le système doit fournir au domaine.

Un cas d'utilisation se caractérise par:

- Son domaine applicatif, dans lequel sont définis les objets et les règles de gestion associées à sa réalisation.
- Les acteurs, c'est à dire les interlocuteurs du système dans le domaine: utilisateurs, équipements, autre systèmes.
- L'événement déclencheur, élémentaire ou combiné, qui doit être défini indépendamment du système.



Développer des cas d'utilisation

- Un *cas d'utilisation* est une séquence typique d'actions qu'entreprend un utilisateur afin d'accomplir une tâche donnée
  - L'objectif d'une *analyse de cas* est de modéliser le système
    … du point de vue de la façon par laquelle l'utilisateur interagit avec le système
    … afin d'accomplir ses objectifs
  - Un *modèle des cas d'utilisation* consiste en
    - un ensemble de cas d'utilisation
    - une description ou un diagramme expliquant comment ils sont inter-reliés

Les cas d'utilisations

- En général, un d'utilisation doit couvrir l'ensemble des étapes à suivre dans l'accomplissement d'une tâche donnée
- Le cas d'utilisation doit décrire l'interaction de l'utilisateur avec le système
  - et non les opérations que doit réaliser le système.
- Le cas d'utilisation devrait être écrit, dans la mesure du possible, d'une façon indépendante de toute interface utilisateur
- Un cas d'utilisation ne doit inclure que les interactions avec le système

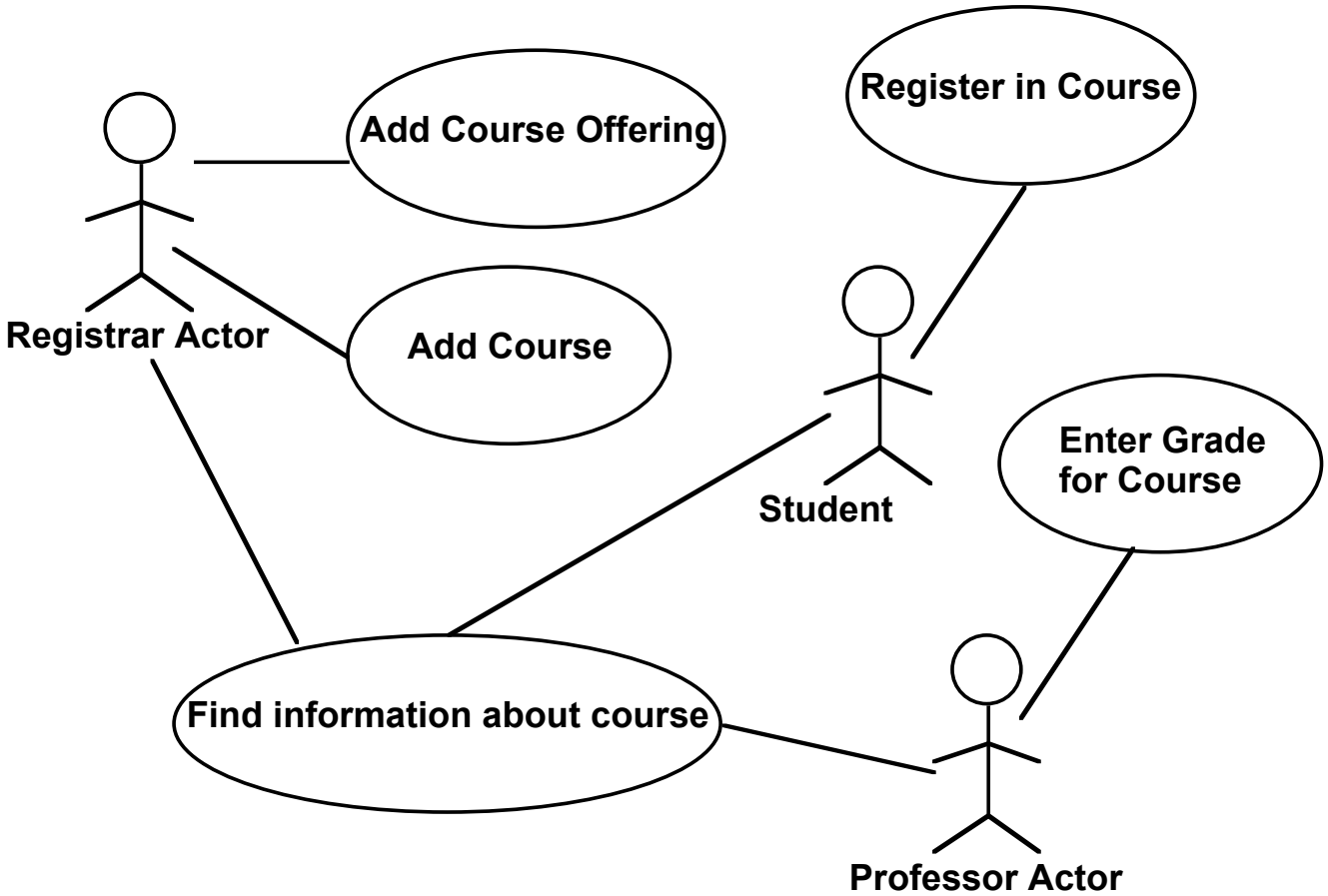
Scénarios

- Un scénario est une *instance* d'un cas d'utlisation
  - Il exprime une occurrence *spécifique* d'un cas d'utilisation
    - un acteur spécifique ...
    - à un moment spécifique ...
    - avec des données spécifiques

Comment décrire un cas d'utilisation?

- A. Nom: **Une appellations courte et descriptive**
- B. Acteurs: **Énumérer les acteurs impliqués**
- C. Buts: **Expliquer ce que les acteurs veulent accomplir**
- D. Préconditions: **Décrire l'état du système avant l'exécution de ce cas d'utilisation**
- E. Description: **Une courte description informelle**
- F. Référence: **Autre cas d'utilisation apparentés**
- G. Déroulement: **Décrire chacune des étapes, sur 2 colonnes**
- H. Postconditions: **Décrire l'état du système après l'exécution de ce cas d'utilisation**

Exemple


Add Course Offering
Register in Course
Registrar Actor
Add Course
Student
Enter Grade
for Course
Find information about course
Professor Actor

Extensions

- Utilisés afin de spécifier des interactions *optionnelles* tenant compte des cas *exceptionnels*
- En créant ainsi un cas d'utilisation d'extension, la description principale peut demeurer simple
- Un cas-type d'extension doit aussi énumérer toutes les étapes mentionnées dans le cas d'utilisation principal
  - incluant le traitement de la situation exceptionnelle

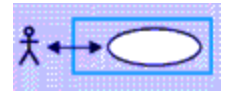
Généralisations

- Similaire aux super-classes dans les diagrammes de classes
- Un cas-type généralisé représente plusieurs cas d'utilisation similaires
- La spécialisation d'un cas d'utilisation fournit les détails spécifiques relatifs à un de ces cas d'utilisations similaires

Inclusions

- Sert à décrire les points communs contenus dans plusieurs cas d'utilisation différents
- Sont donc inclus dans d'autres cas d'utilisations
  - Même des cas d'utilisation très différents peuvent partager un certain nombre d'actions identiques
  - Permet d'éviter la répétition de ces détails dans chacun des cas d'utilisations
- Représente l'exécution d'une tâche de plus bas niveau pour accomplir un but aussi de plus bas niveau

Modèle de cas d'utilisation



Étude de cas

Capture des besoins fonctionnels

Référence: UML par la pratique

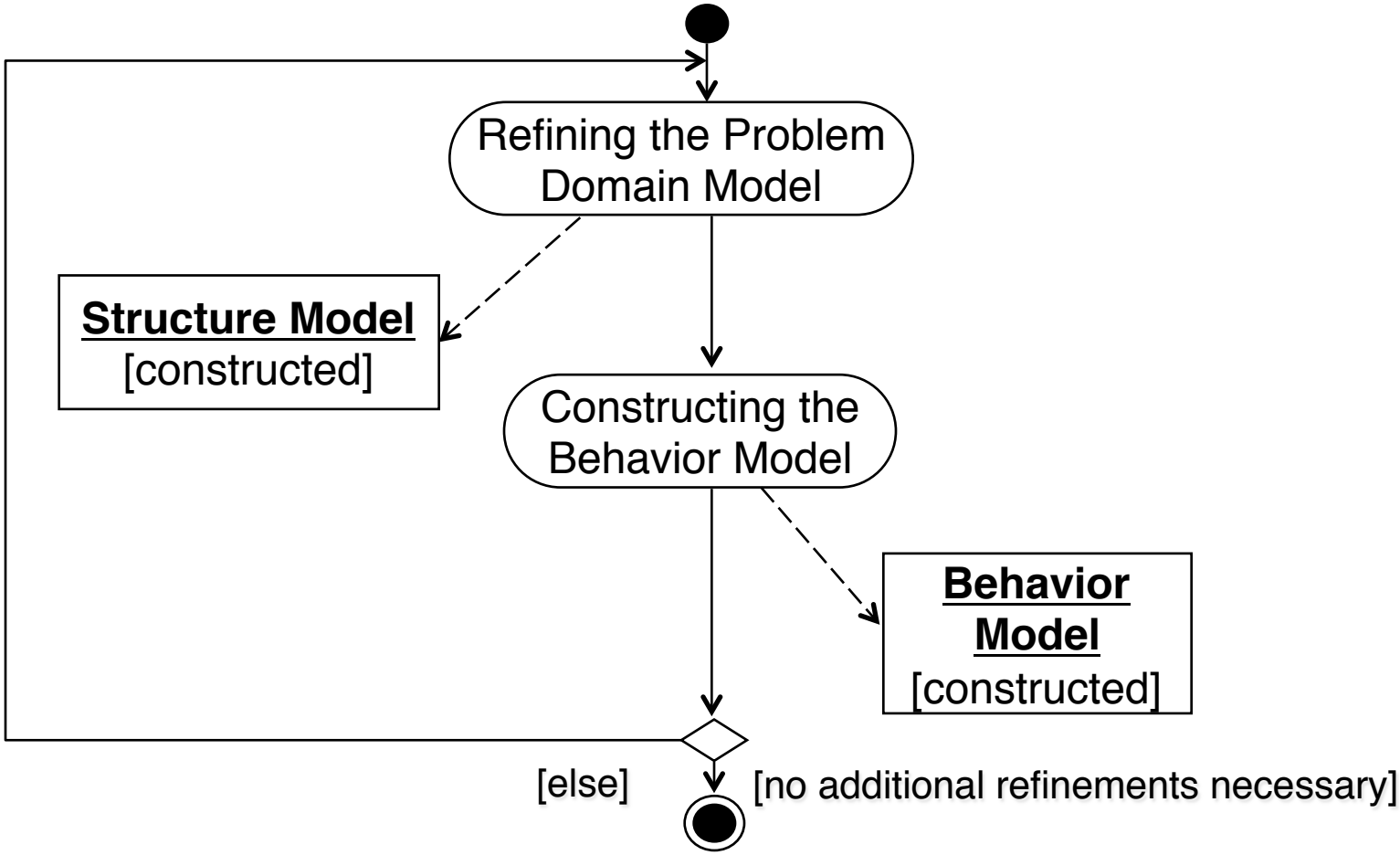
De Pascal Roques

Modélisation statique

Vues statiques du système

- diagrammes de cas d'utilisation
- diagrammes d'objets
- diagrammes de classes
- diagrammes de composants
- diagrammes de déploiement

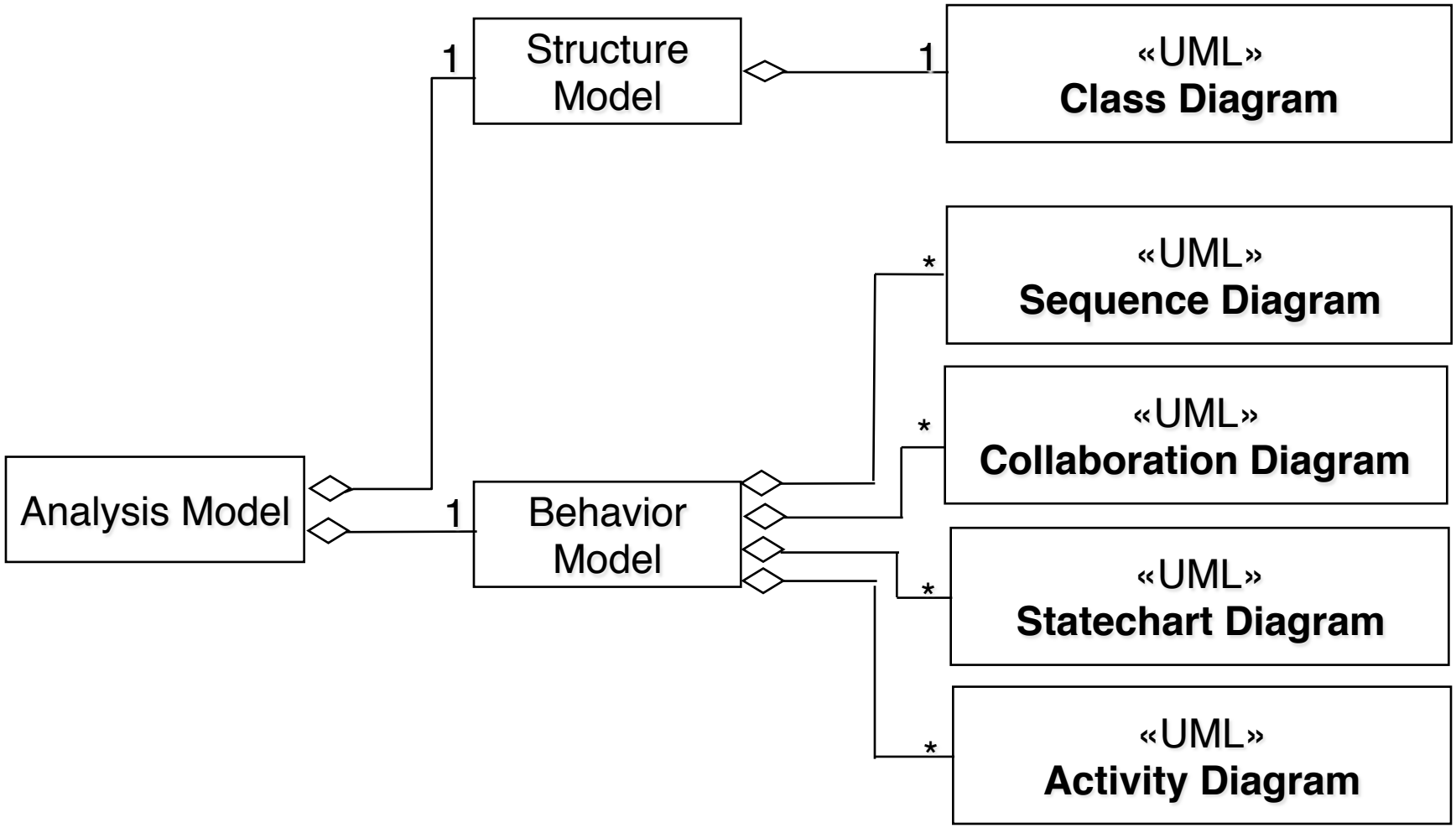
Modèle d'analyse



Modèle d'analyse

- Modéliser le domaine d'information
  + Définir des données et représenter leurs relations
- Modéliser le domaine de la fonctionnalité
  + Identifier les fonctions qui transforment les données
- Modéliser le comportement
  + Indiquer les différent états du système
  + Spécifier les événements qui engendrent les changement d'état du système
- Partitionner le modèle
  + Raffiner chacun des modèles

Analyse orientée objet



Analyse orientée objet

Définition de l'activité

Analyse Orientée Objet - le processus de création d'un modèle du domaine de la solution en identifiant les classes et les objets qui constituent le vocabulaire du domaine de la solution

Analyse orientée objet

Définition et terminologies

Objet

Entité réel dont les traitements (méthodes) et les attributs sont modélisés pour une application réelle

Héritage

Propriété permettant la description générique des objets

Classe

Regroupement d'objets qui ont exactement les mêmes propriétés, attributs et traitements

Méta-Classe

Classe dont les instantiations sont d'autres classes

Analyse orientée objet

Définition et terminologies

Hiérarchie

Relation entre les classes. Les objets aux niveaux inférieurs héritent des données et traitements des objets aux niveaux supérieurs

Message

Unité de communication entre deux objets

Polymorphisme

Le polymorphisme permet avec un même message d'obtenir différentes interprétations selon les destinataires.

Type de données abstrait

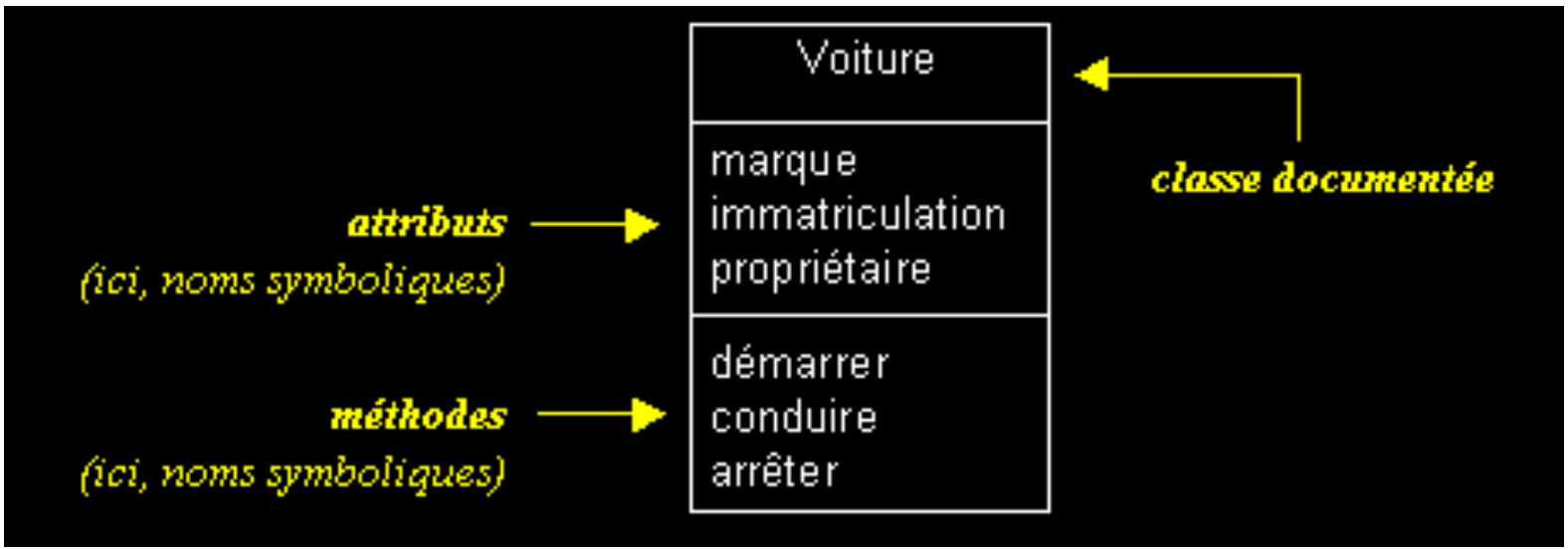
Nom utilisé dans certain langages de programmation pour désigner un nouveau type de donné qui est défini par l'usager pour encapsuler la définition des données de l'objet et ses traitements

Modélisation statique

Les rôles des classes et des objets

Durant l'analyse, l'analyste doit:

- Identifier les classes et les objets qui forment le vocabulaire du domaine de la solution. Ces classes et ces objets sont appelés les abstractions du domaine

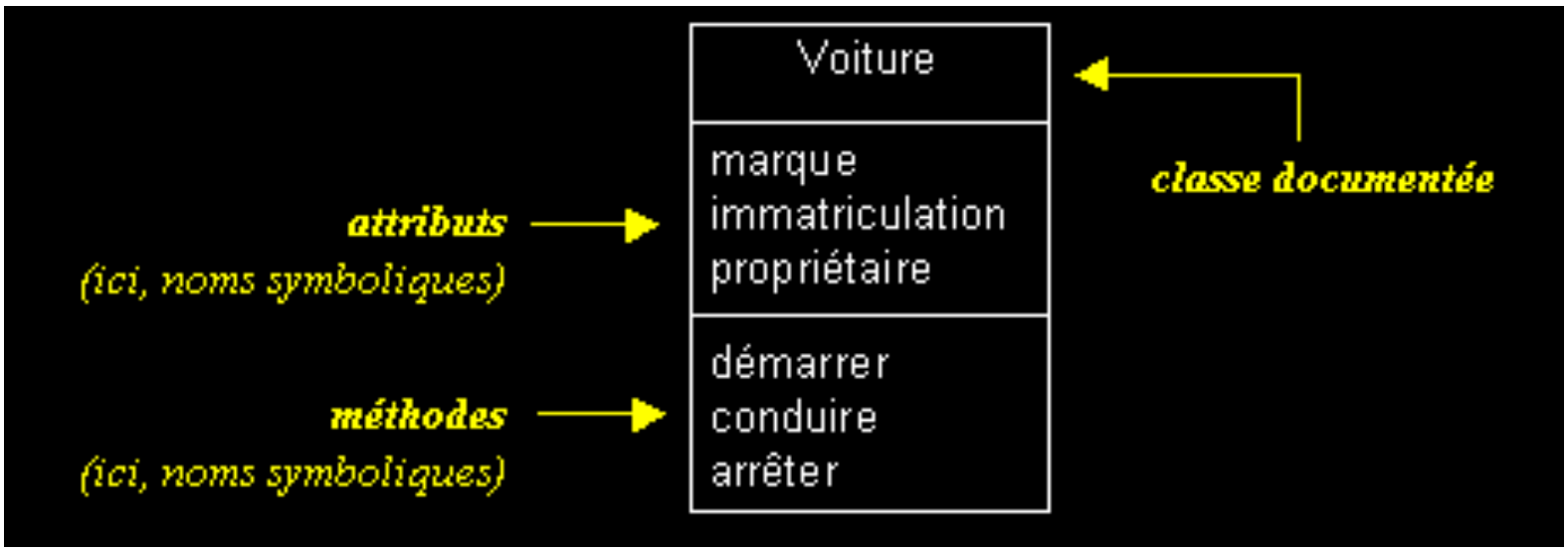


Modélisation statique

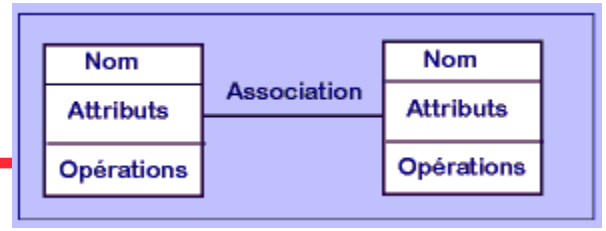
Les rôles des classes et des objets

Durant l'analyse, l'analyste doit:

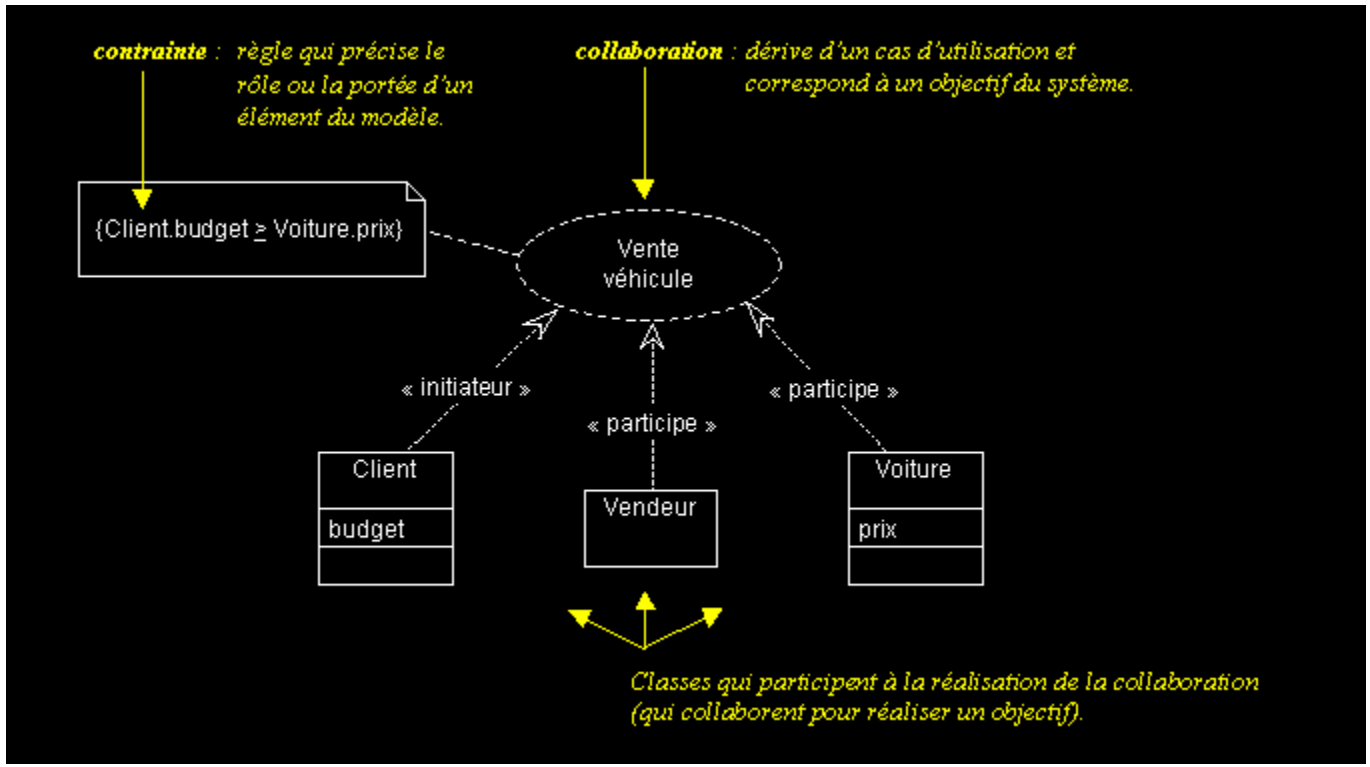
- Identifier les classes et les objets qui forment le vocabulaire du domaine de la solution. Ces classes et ces objets sont appelés les abstractions du domaine



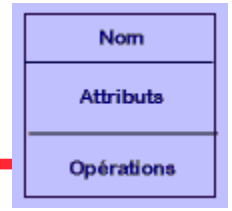
Modélisation statique



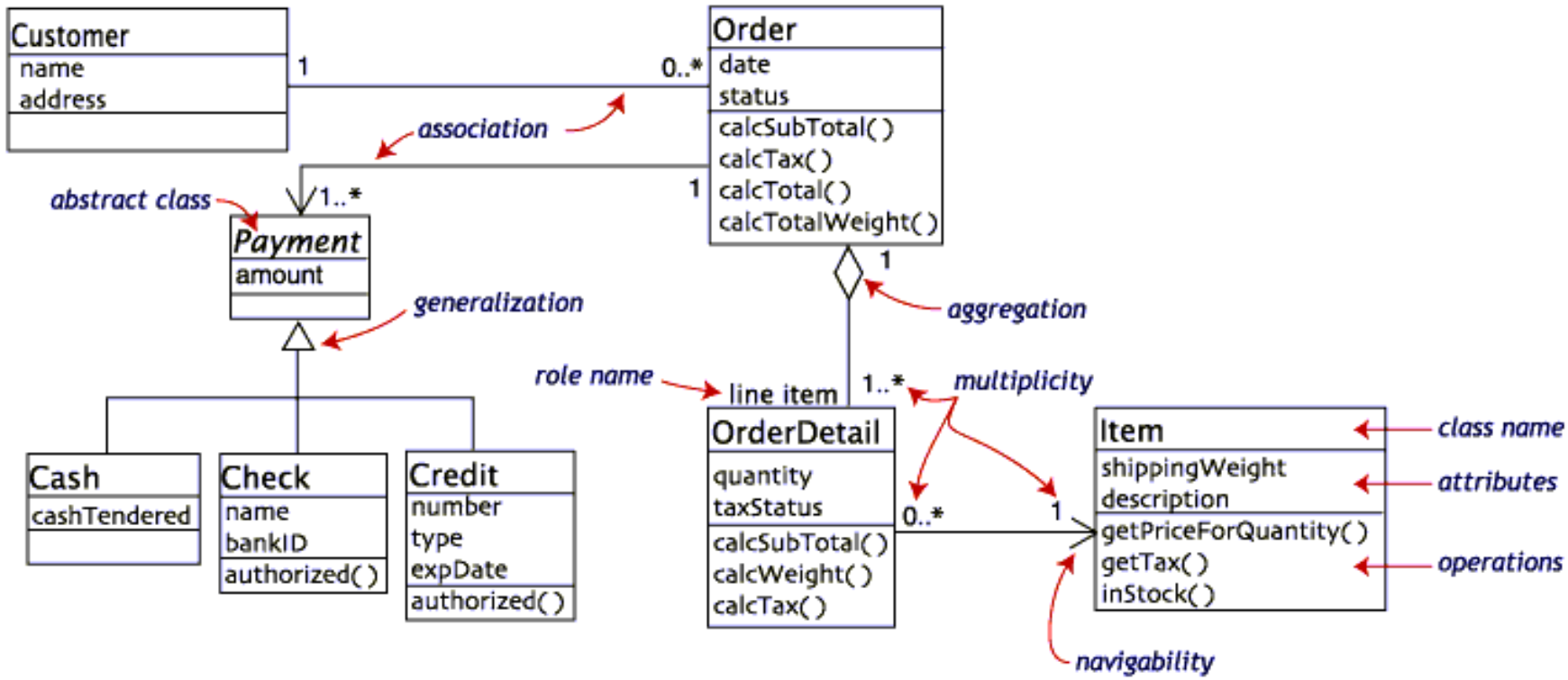
- Développer les structures qui permettent aux ensembles d'objets de décrire les comportements satisfaisant les besoins du domaine. On appelle ces structures les mécanismes de l'implémentation



Les diagrammes de classe



Exemple



The class diagram above models a customer order from a retail catalog. The central class is the **Order**. Associated with it are the **Customer** making the purchase and the **Payment**. A **Payment** is one of three kinds: **Cash**, **Check**, or **Credit**. The order contains **OrderDetails** (line items), each with its associated **Item**.

Séquence d'activités suggérée

- Identifier un premier ensemble de classes candidates
- Ajouter les associations et attributs
- Trouver les généralisations
- Lister les responsabilités principales de chacune des classes
- Décider quelles seront les opérations
- Effectuer quelques itérations jusqu'à satisfaction:
  - Ajouter ou retirer des classes, associations, attributs, généralisations, responsabilités ou opérations
  - Identifier les interfaces
  - Appliquer les patrons de conception appropriés (Chapitre 6)

- ***Il ne faut être ni trop rigide ni trop désorganisé***

Identifier les classes *métiers*

- Lors du développement du modèle du domaine (métier), les classes sont *découvertes*
- Lorsque le reste du système est élaboré, de nouvelles classes sont *inventées*
  - Ce sont les classes requises pour obtenir un système fonctionnel
  - L'invention de classes peut se produire à n'importe quel stage du développement
- La réutilisation doit toujours demeurer une priorité
  - Les cadres d'application
  - Les possibles extensions du système
  - Les systèmes similaires

Technique pour découvrir classes métiers

- Examiner un document écrit telle qu'une description des exigences
- Extraire les *noms* et en faire des classes candidates
- Éliminer les noms qui:
  - sont redondants
  - représentent des instances
  - sont vagues ou trop généraux
  - sont inutiles dans le contexte de l'application
- Porter attention aux classes qui, dans le domaine, représentent des catégories d'utilisateurs.

Identifier les associations et les attributs

- Débuter avec les classes considérées centrales dans le système.
- Déterminer les données que chacune de ces classes devrait contenir et les relations avec les autres classes.
- Ajouter graduellement les autres classes.
- Éviter d'ajouter trop d'attributs ou d'associations à une classe
  - Un système manipulant peu d'information est toujours plus facile à maîtriser

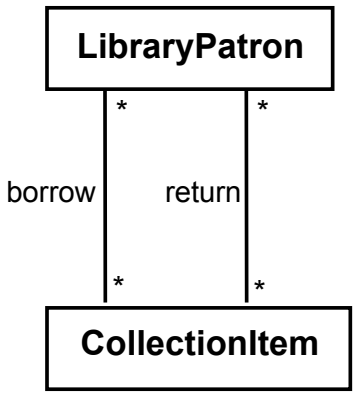
Quelques trucs permettant d'identifier des associations

- Une association devrait exister si une classe
  - *possède*
  - *contrôle*
  - *est connecté à*
  - *est relié à*
  - *est une partie de*
  - *est fait de parties de*
  - *est membre de*
  - *a comme membres*

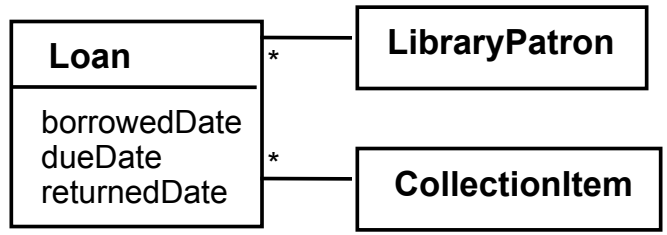
  une autre classe dans le modèle
- Spécifier ensuite la multiplicité à chaque extrémité de l'association
- Étiqueter clairement cette association

Actions versus associations

- Une erreur fréquente est de représenter une *action* comme si elle était une association



Erreur, ici les associations sont en fait des actions



Préférable: L'opération d'emprunt créé un prêt

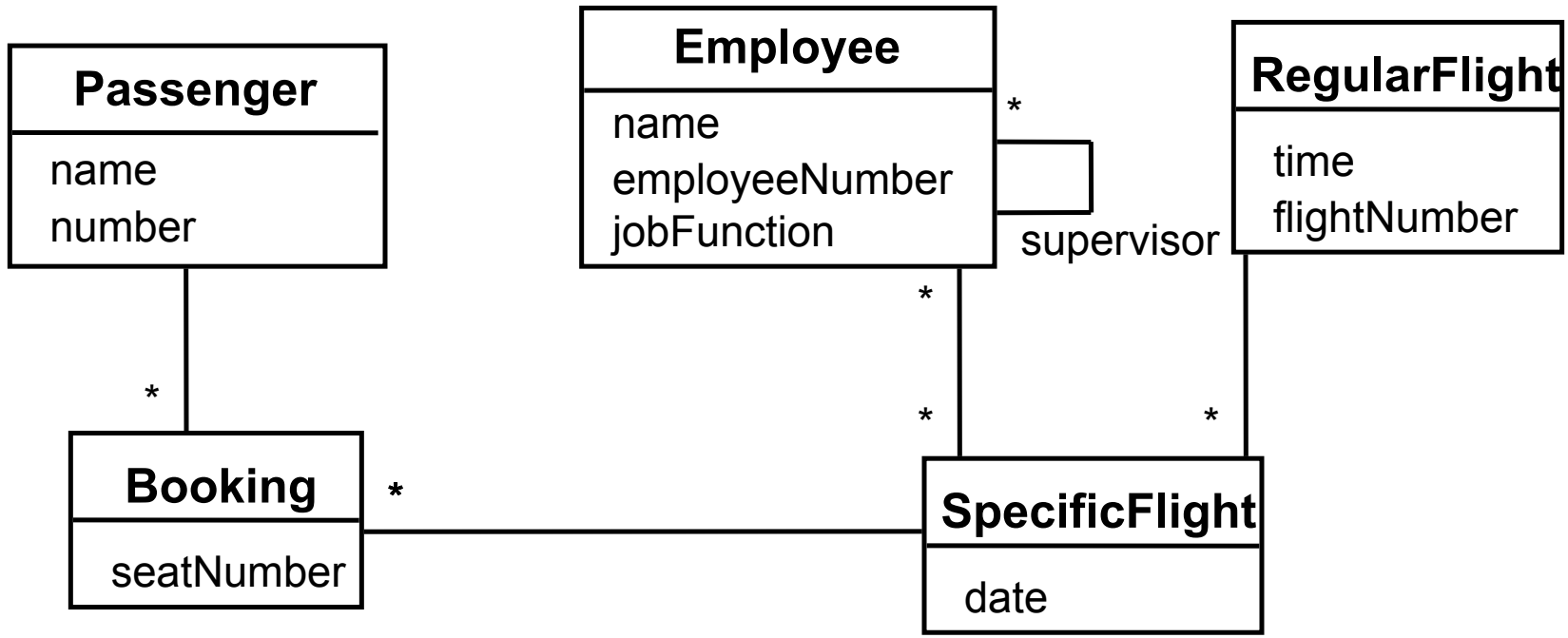
Identifier les attributs

- Déterminer l'information qui doit être maintenu dans chacune des classes
- Plusieurs noms ayant été rejetés comme classes deviennent alors des attributs
- Un attribut contient en général une valeur simple
  - E.g. chaîne de caractères, nombre

Quelques trucs pour identifier des attributs

- Il ne doit pas y avoir duplication des attributs
- Si un sous-ensemble des attributs forme un groupe cohérent, alors une nouvelle classe devrait peut être introduite

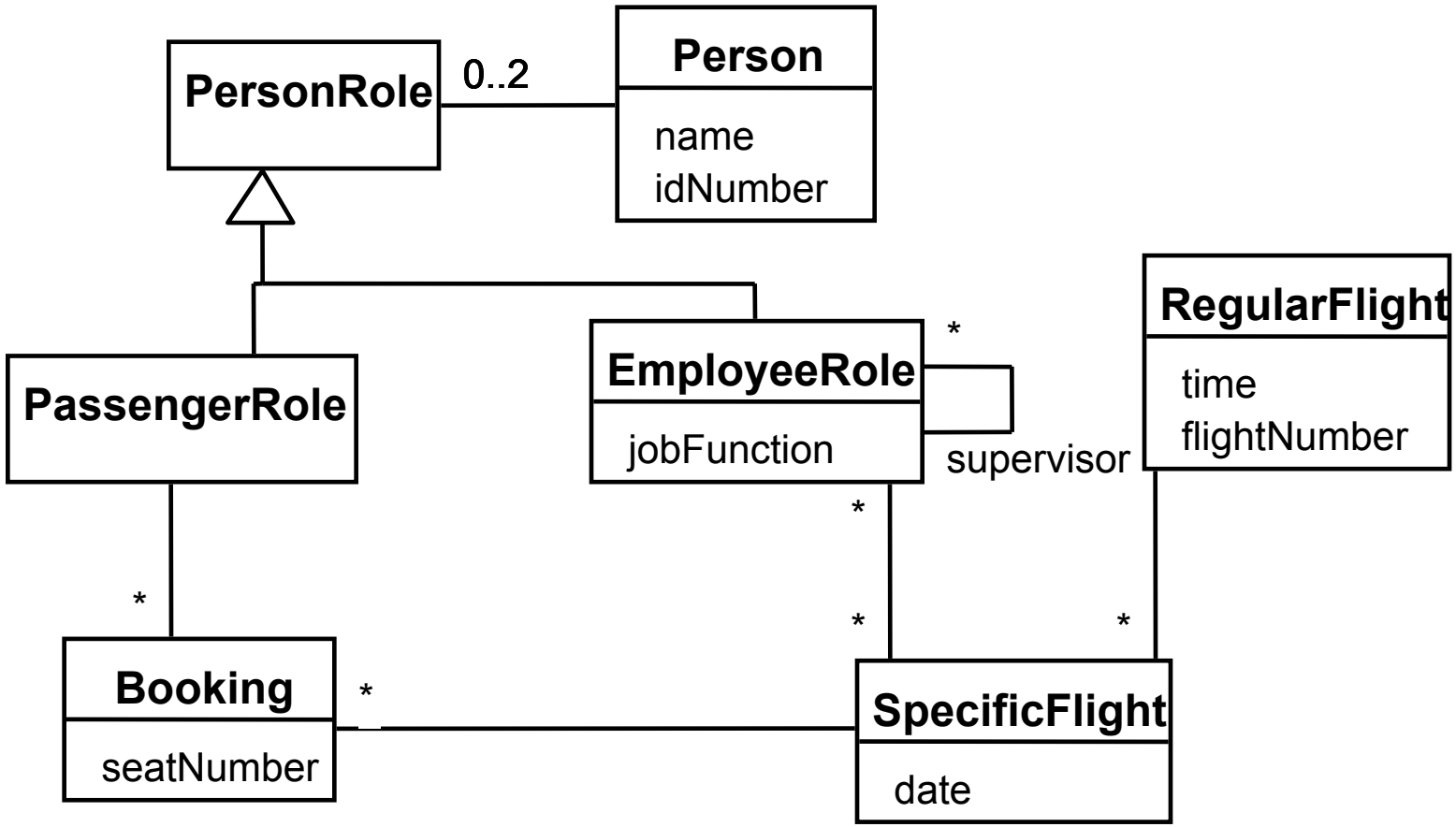
Un exemple


Passenger
name
number
Employee
name
employeeNumber
jobFunction
supervisor
RegularFlight
time
flightNumber
Booking
seatNumber
SpecificFlight
date
*
*
*
*
*
*
*

Identifier les généralisations et les interfaces

- Il existe deux façon d'identifier les généralisations:
  - De bas en haut
    - Grouper ensemble les classes similaires
  - De haut en bas
    - Spécialiser, au besoin, les classes plus générales
- Créer une *interface*, plutôt qu'une super-classe si
  - deux classes partageant certaines opérations sont considérées dissimilaires
  - une classe à généraliser possède déjà sa propre super-classe
  - Différentes implémentations de la même classe pourraient être disponibles

Un exemple (généralisation)



Allouer des responsabilités aux classes

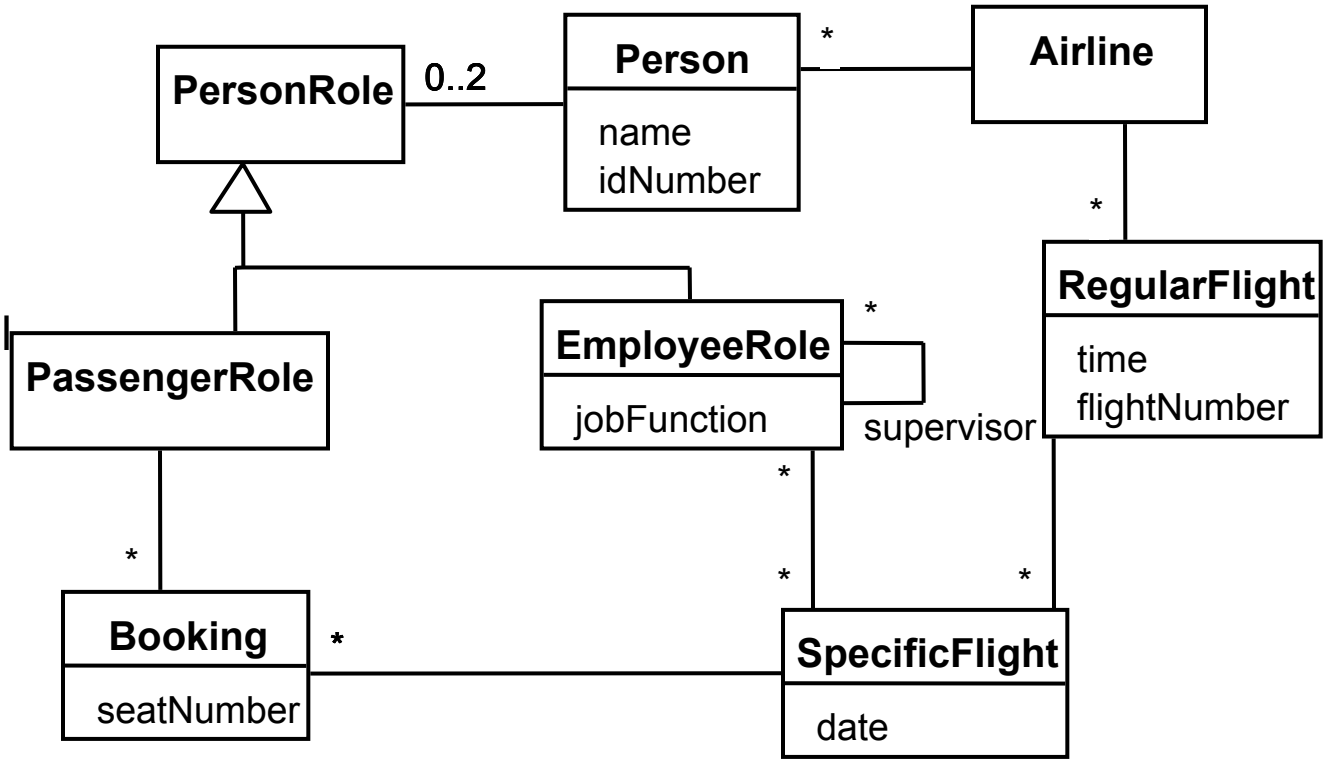
- Une *responsabilité* est quelque chose que le système doit faire.
  - Chacune des exigences fonctionnelles doivent être attribuées à l'une des classes
    - Toutes les responsabilités d'une classe doivent être reliées entre elles.
    - Si une classe a trop de responsabilités, elle devrait être scindée en plusieurs classes
    - Si une classe a aucune responsabilité, alors celle-ci est probablement inutile
    - Lorsqu'une responsabilité ne peut être attribués à aucune classe, alors c'est qu'une nouvelle classe devrait être introduite
  - Afin de déterminer les responsabilités
    - Effectuer une analyse de cas
    - Recherche des verbes et des noms décrivant des *actions*

Catégories de responsabilités

- Fixer et obtenir la valeur d'un attribut
- Créer et initialiser de nouvelle instances
- Sauvegarder et récupérer de l'information persistante
- Détruire des instances
- Ajouter et détruire des liens
- Copier, convertir, transformer, transmettre, afficher
- Calculer des résultats numériques
- Naviguer et rechercher
- Tout autre tâche…

Un exemple (responsabilités)

- Créer un nouveau vol régulier
- Rechercher un vol spécifique
- Modifier les attribut d'un vol
- Créer un vol spécifique
- Traiter la réservation d'un passager
- Annuler une réservation



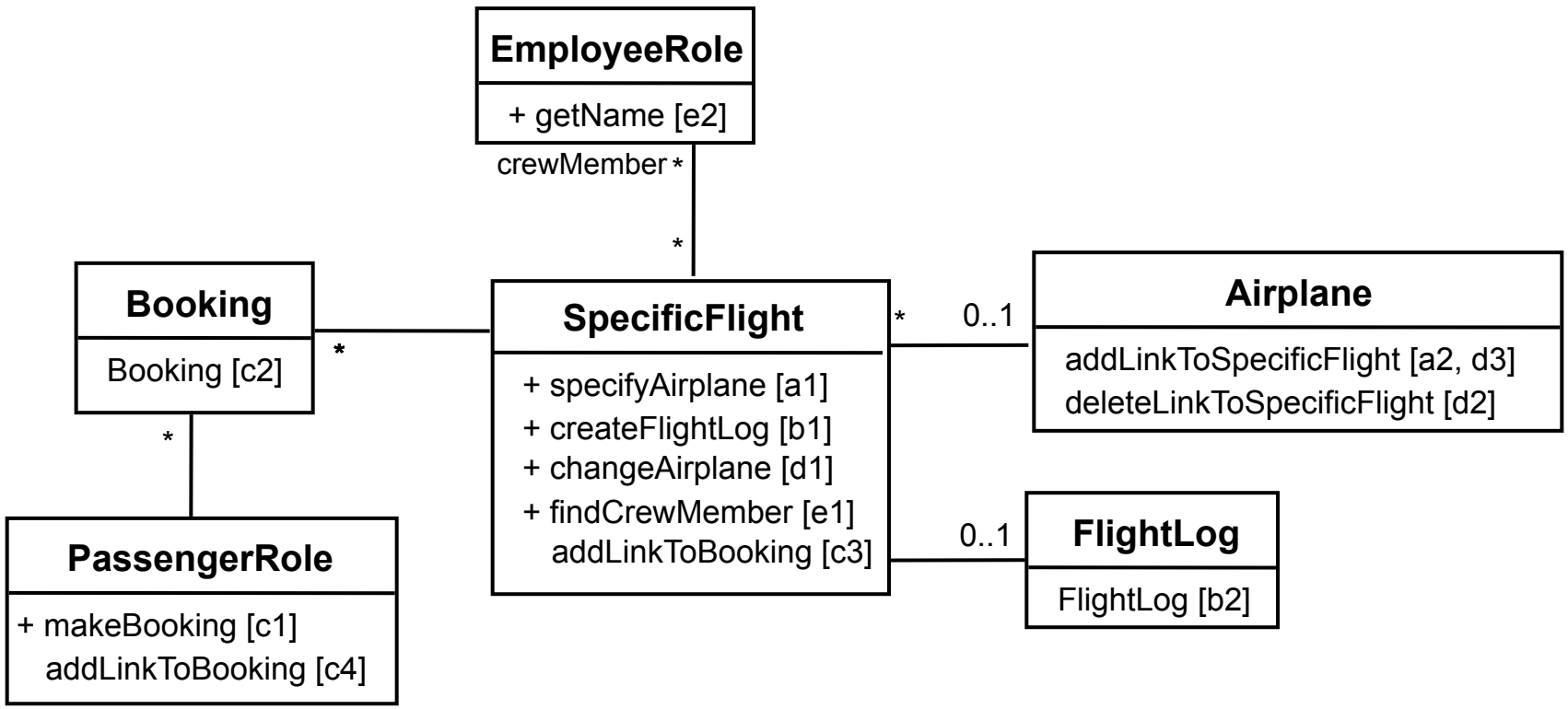
Prototypage d'un diagramme de classes sur papier

- A mesure que les classes sont identifiée, écrire leur nom sur une petit carton
- Lorsqu'un attribut ou une responsabilité est identifiée, celle-ci est ajouter au carton de la classe correspondante
  - Si les responsabilités proposées ne peuvent entrer sur un seul carton:
    - C'est sans doutes qu'il faut scinder la classe en deux.
- Disposer les cartons sur un tableau afin de créer le diagramme de classes.
- Lier les cartons par des traits afin de représenter les associations et les généralisations.

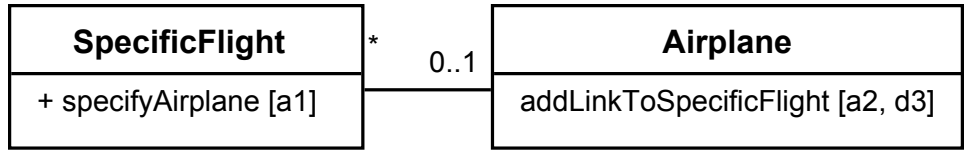
Identifier les opérations

- Les opérations servent à réaliser les responsabilités
  - Il peut y avoir plusieurs opérations pour une responsabilité
  - Les opérations de base réalisant ces responsabilités seront déclarées `public`
  - Une méthode qui collabore à la réalisation d'une responsabilité sera, dans la mesure du possible, déclarée `private`

Un exemple

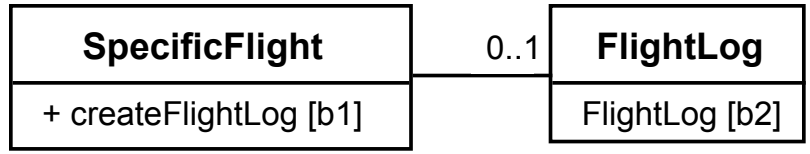

EmployeeRole
+ getName [e2]
crewMember
*
*
Booking
Booking [c2]
*
*
SpecificFlight
+ specifyAirplane [a1]
+ createFlightLog [b1]
+ changeAirplane [d1]
+ findCrewMember [e1]
addLinkToBooking [c3]
*
0..1
Airplane
addLinkToSpecificFlight [a2, d3]
deleteLinkToSpecificFlight [d2]
0..1
FlightLog
FlightLog [b2]
PassengerRole
+ makeBooking [c1]
addLinkToBooking [c4]

Collaboration ‘a’



- *Créer un lien bi-directionl entre deux objets existants*;
- **e.g. ajouter une lien entre une instance de `SpecificFlight` et une instance de `Airplane`.**
- **1. (publique) L’instance de `SpecificFlight`**
  - **Créé un lien unidirectionnel vers l’instance de `Airplane`**
  - **Puis appelle l’opération 2.**
- **2. (non-publique) L’instance de `Airplane`**
  - **Créé un lien unidirectionnel vers l’instance de `SpecificFlight`**

Collaboration ‘b’



- *Créer un objet et le lier à un objet existant*
- **e.g. créer un Fl`ightLog`, et le lier à un `SpecificFlight`.**

- **1. (publique) L’instance de `SpecificFlight`**
  - **Appelle le constructeur de `FlightLog` (opération 2)**
  - **Puis créé un lien unidirectionnel vers la nouvelle instance de `FlightLog`.**
- **2. (non-publique) Le constructeur de `FlightLog`**
  - **Créé un lien unidirectionnel vers l’instance de `SpecificFlight`.**

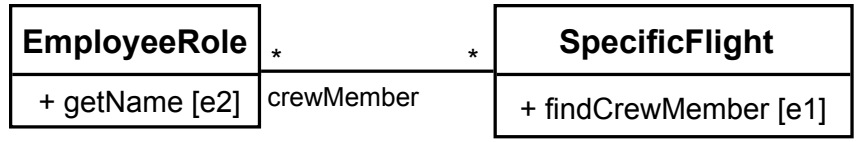
Collaboration ‘c’

- *Créer une association à partir de deux objets existants*
- **e.g. créer une instance de `Booking`, liant un `SpecificFlight` à un `PassengerRole`.**
- **1. (publique) L’instance de `PassengerRole`**
  - Appelle le constructeur de **`Booking`** (opération 2).
- **2. (non-publique) Le constructeur de `Booking` doit, entre autres,**
  - Créé un lien unidirectionnel vers une instance de **`PassengerRole`**
  - Créé un lien unidirectionnel vers une instance de **`SpecificFlight`**
  - Appelle les opérations 3 et 4.
- **3. (non-publique) L’instance de `SpecificFlight`**
  - Créé un lien unidirectionnel vers une instance de **`Booking`**.
- **4. (non-publique) L’instance de `PassengerRole`**

Collaboration ‘d’

- *Changer la destination d’un lien*
- **e.g. changer le `Airplane` de `SpecificFlight`, de `airplane1` à `airplane2`**
- **1. (publique) L’instance de `SpecificFlight`**
  - Détruit le lien vers **`airplane1`**
  - Créé un lien unidirectionnel vers une instance de **`airplane2`**
  - Appelle l’opération 2
  - Puis appelle l’opération 3.
- **2. (non-publique) `airplane1`**
  - Détruit le lien unidirectionnel vers l’instance de **`SpecificFlight`**.
- **3. (non-publique) `airplane2`**
  - Créé un lien unidirectionnel vers une instance de **`SpecificFlight`**.

Collaboration ‘e’



- *Rechercher une instance*
- **e.g. rechercher par nom pour un personnel de bord associé avec un `SpecificFlight`.**
-
- **1. (publique) L’instance de `SpecificFlight`**
  - Créé un `Iterator` à travers les liens de `crewMember` de `SpecificFlight`
  - Pour chacun de ceux-ci, appelle l’opération 2, jusqu’à trouver une correspondance.
- **2. (publique, peut-être) L’instance de `EmployeeRole` retourne son nom.**

Implémenter un diagramme de classes en Java

- Les attributs sont implementés en tant que variables d'instance
- Les généralisations sont implémentés en utilisant le mot-clé `extends`
- Les interfaces sont implémentés en utilisant le mot-clé `implements`
- Les associations sont normallement implémentées en utilisant des variables d'instance
  - Diviser chaque association en deux associations unidirectionnelles
    - Chacune des classe impliquées possède alors une variable d'instance
  - Pour une association où la multiplicité est de 'un' ou 'optionnel'
    - Déclarer une variable référant à cette classe
  - Pour une association où la multiplicité est de 'plusieurs':
    - Utiliser une collection réalisant l'interfaces `List`, telle que LinkedList

Exemple: SpecificFlight

```
class SpecificFlight
{
  private Calendar date;
  private RegularFlight regularFlight;
  private TerminalOfAirport destination;
  private Airplane airplane;
  private FlightLog flightLog;

  private ArrayList crewMembers;
   // of EmployeeRole
  private ArrayList bookings
  ...
}
```

Exemple: SpecificFlight

```
// Constructor that should only be called from
// addSpecificFlight
SpecificFlight(
  Calendar aDate,
  RegularFlight aRegularFlight)
{
  date = aDate;
  regularFlight = aRegularFlight;
}
```

Exemple: RegularFlight

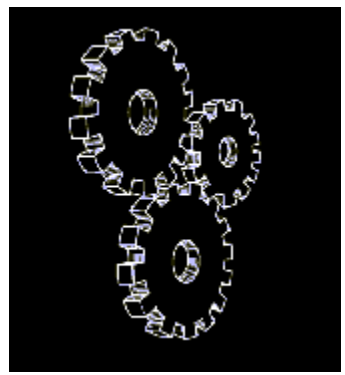
```
class RegularFlight
{
  private ArrayList specificFlights;
  ...
  // Method that has primary
  // responsibility

  public void addSpecificFlight(
    Calendar aDate)
  {
    SpecificFlight newSpecificFlight;
    newSpecificFlight =
      new SpecificFlight(aDate, this);
    specificFlights.add(newSpecificFlight);
  }
  ...
}
```

Modélisation dynamique

Vues dynamiques du système

- diagrammes de collaboration
- diagrammes de séquence
- diagrammes d'états-transitions
- diagrammes d'activités



Diagrammes d'interaction

- Les diagrammes d'interaction sont utilisés pour modéliser les aspects dynamiques d'un système
  - Ils aident à visualiser comment le système exécute ses tâches.
  - Un diagramme d'interaction est souvent construit à partir d'un diagramme de classes et d'un cas d'utilisation
    - L'objectif est de montrer comment un ensemble d'objets peut réaliser une tâche demandée par un acteur

Interactions et messages

- Un diagramme d'interaction montrent comment un ensemble d'objets et d'acteurs communiquent ensemble afin:
  - de réaliser un cas d'utilisation
  - de réaliser une certaine fonctionnalité
- L'ensemble des différentes étapes à accomplir s'appelle une *interaction*.
- Un diagramme d'interaction montre différents types de communication entre objets, acteurs et sous-systèmes:
  - E.g. appel à des méthodes, information envoyés sur un réseau
  - Ceux-ci sont appelés *messages*.

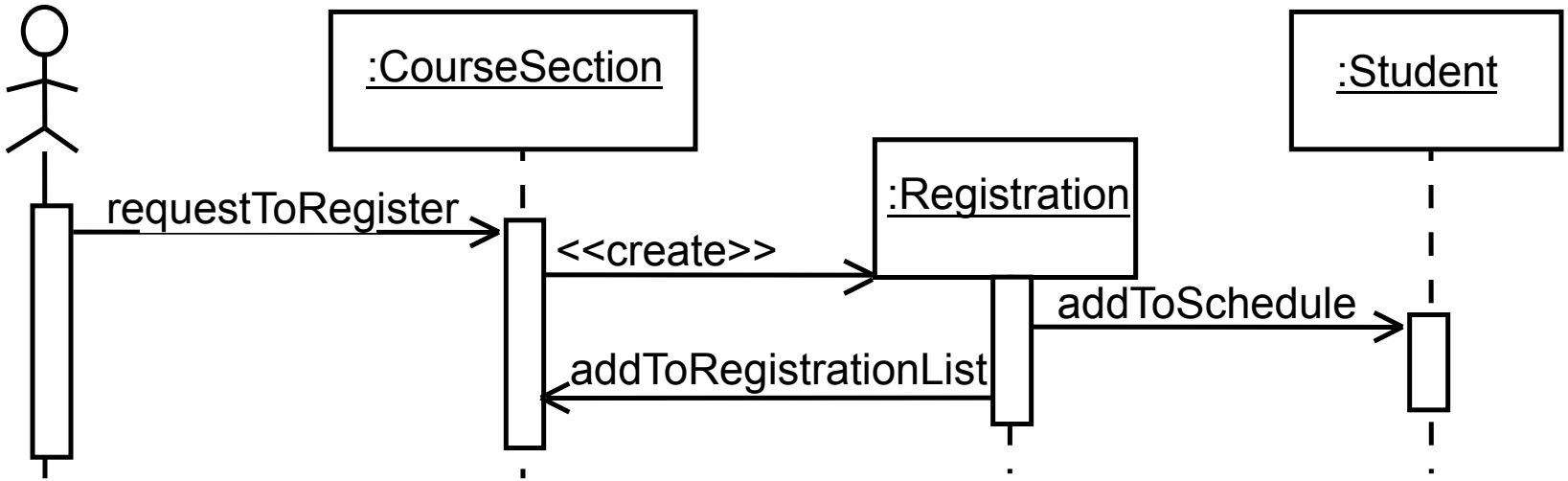
Les éléments se trouvant dans un diagramme d'interaction

- Des instances de classes
  - Représentés par des rectangles comme dans les diagrammes d'instances
- Acteurs
  - Représentés par des personnages-allumettes comme dans les diagramme de cas d'utilisation
- Messages
  - Représentés par des flèches horizontales

Création d'un diagramme d'interaction

- Un diagramme de classes et des cas d'utilisation doivent d'abord être élaborés
  - Il existe deux sortes de diagramme d'interaction
    - *Diagramme de séquence*
    - *Diagramme de collaboration*

Un exemple de diagramme de séquence



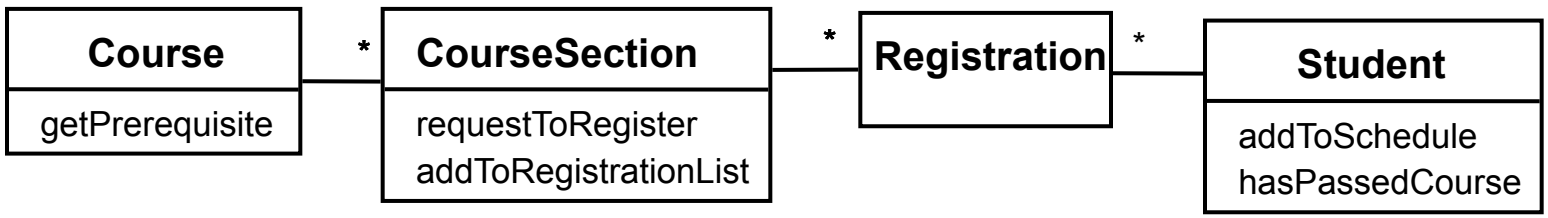


Diagramme de séquence

- Un diagramme de séquence montre la séquence temporelle d'échange de message entre l'acteur et les objets réalisant une certaines tâche
  - Les objets sont disposés horizontalement
  - L'acteur qui initie l'interaction se trouve généralement à l'extrême gauche
  - La dimension verticale représente le temps
  - Une ligne verticale, appelée *ligne de vie*, est accrochée à chaque objet ou acteur
  - La ligne de vie s'épaissie pour devenir une boite d'activation lorsque l'objet est actif, i.e., durant la *période d'activation*.
  - Un message est représenté par une flèche joignant deux boites d'activation.
    - Un message a un nom et peut aussi avoir une liste d'arguments et une valeur de retour.

Encore le même exemple, avec plus de détails

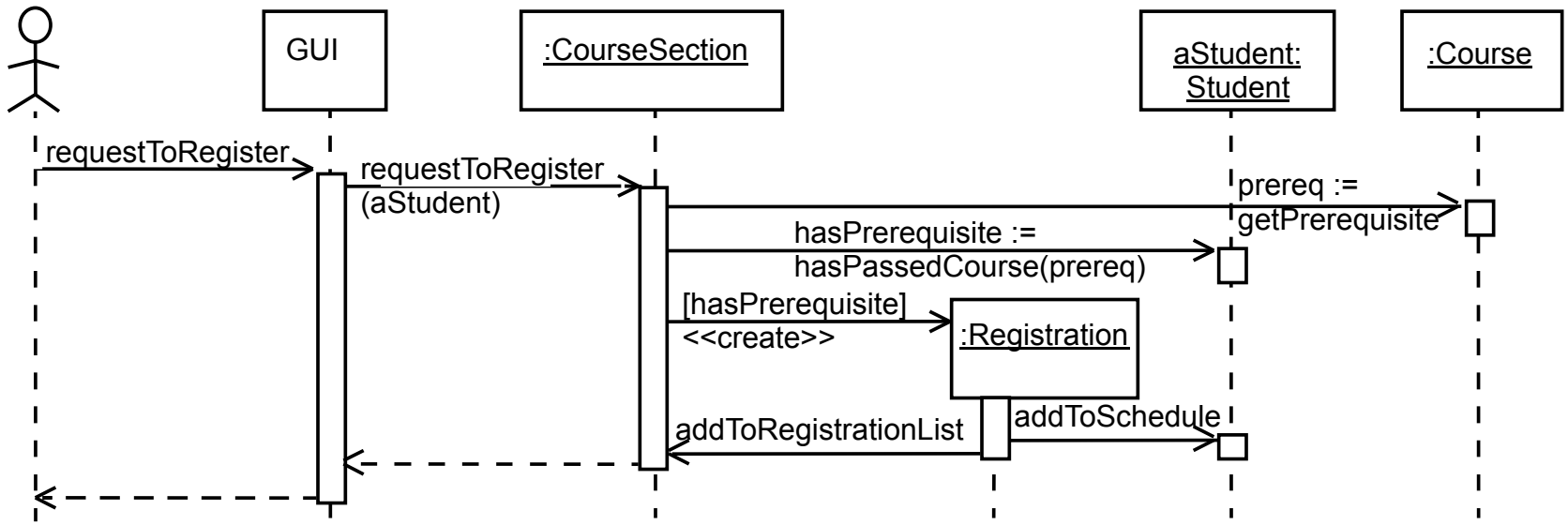


Diagramme de collaboration – un exemple

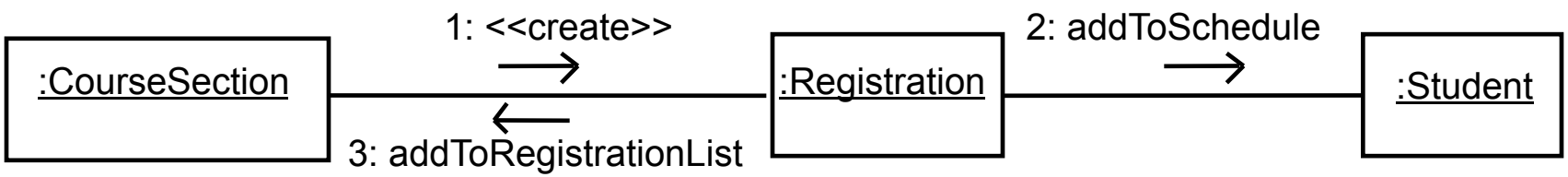


Diagramme de collaboration

- Un diagramme de collaboration illustre comment les objets coopèrent dans la réalisation d'une interaction
  - Un diagramme de collaboration est un graphe dont les objets sont les sommets.
  - Des liens de communication sont ajoutés entre ces objets
  - Les messages sont associés à ces liens.
    - Représentés par des flèches
  - L'ordonnancement temporel est indiqué à l'aide d'une numérotation

Liens de communication

- Un lien de communication existe entre 2 objets lorsqu'à un moment il est possible d'envoyer un message d'un objet à un autre.

- Plusieurs situations peuvent rendre cet échange possible:

1. Les classes sont en association
   - Il s'agit là de la situation la plus commune
   - Si tous les messages sont envoyés dans la même direction, l'association peut alors être rendue unidirectionnelles

D’autres liens de communication possibles

2. Le récepteur est contenu dans une variable locale de la méthode émettrice
   - Ceci se produit fréquemment lorsque l’objet a été créé par la méthode émettrice ou lorsqu’un résultat précédent a retourné un objet .
   - Le stéréotype a utiliser est «local» ou [L].

3. Une référence à l’objet récepteur a été reçu comme paramètre par la méthode émettrice
   - Le stéréotype a utiliser est «parameter» or [P].

D’autres liens de communication possibles

4. Le récepteur est un objet global
   - Ceci se produit lorsque la référence à l’objet peut être obtenue à l’aide d’une méthode statique
   - Le stéréotype a utiliser est «global», ou [G]

5. Les objets communiquent à travers un réseau
   - Nous suggérons d’utiliser «network».

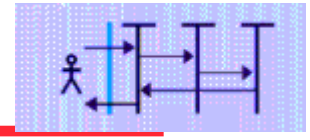
Comment choisir entre un diagramme de séquence et un diagramme de collaboration

- Les diagrammes de séquence
  - Rendent explicite la séquence temporelle dans l'interaction
    - Les cas d'utilisation ont aussi un tel ordre temporel
    - Les diagrammes de séquence constituent donc le choix naturel lorsque l'interaction est construite à partir d'un cas d'utilisation.
  - Facilite une écriture détaillée des messages
    - Moins d'espace est généralement disponible dans un diagramme de collaboration

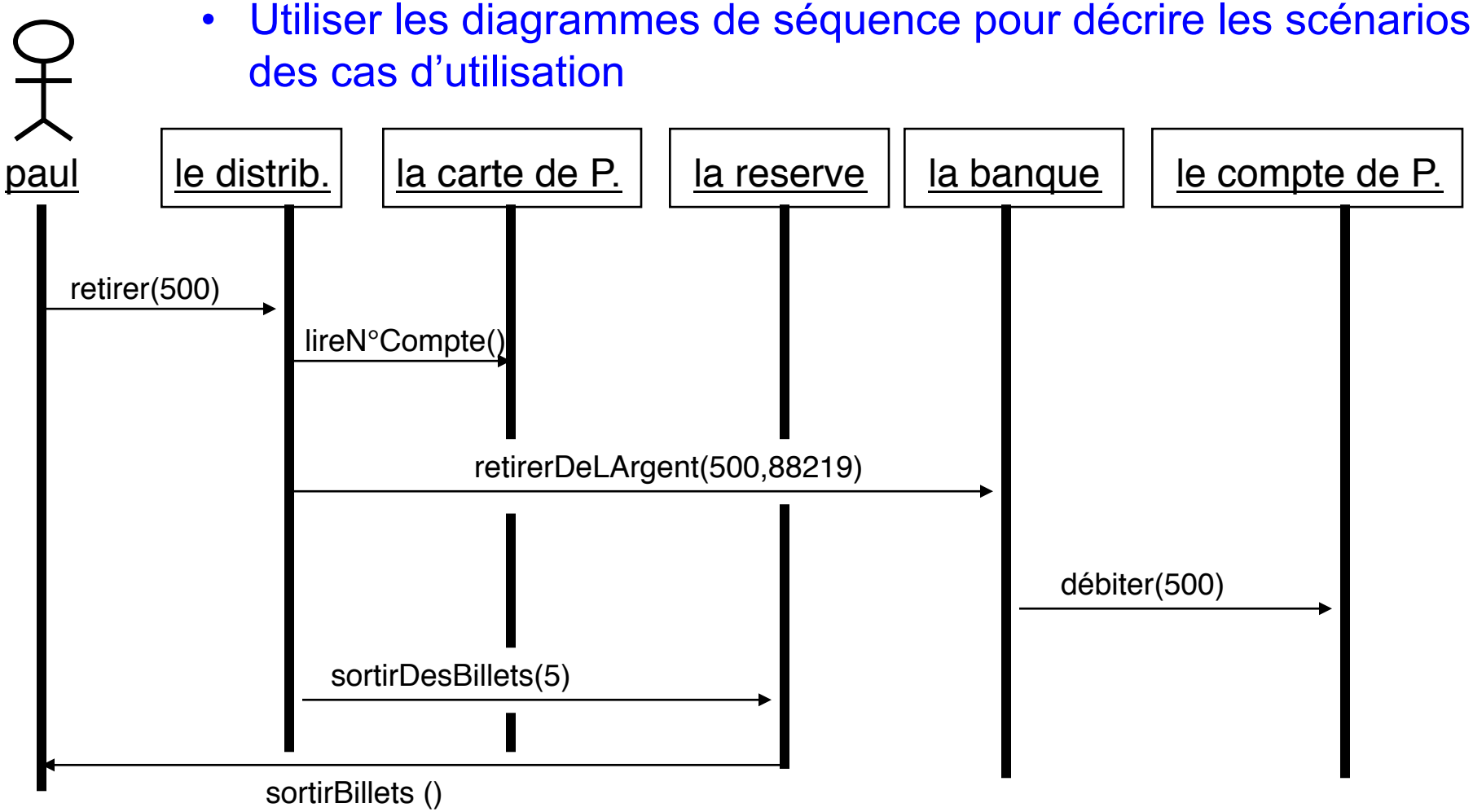
Comment choisir entre un diagramme de séquence et un diagramme de collaboration

- Diagramme de collaboration
  - Peuvent être vus comme la prolongation d'un diagramme de classes
    - Préférable lorsque l'interaction est déduite du diagramme de classes
    - Sont aussi très utiles pour valider des diagrammes de classes

Modélisation dynamique

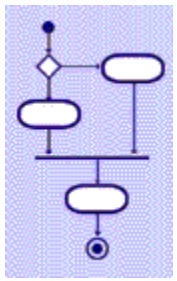


- Utiliser les diagrammes de séquence pour décrire les scénarios des cas d'utilisation

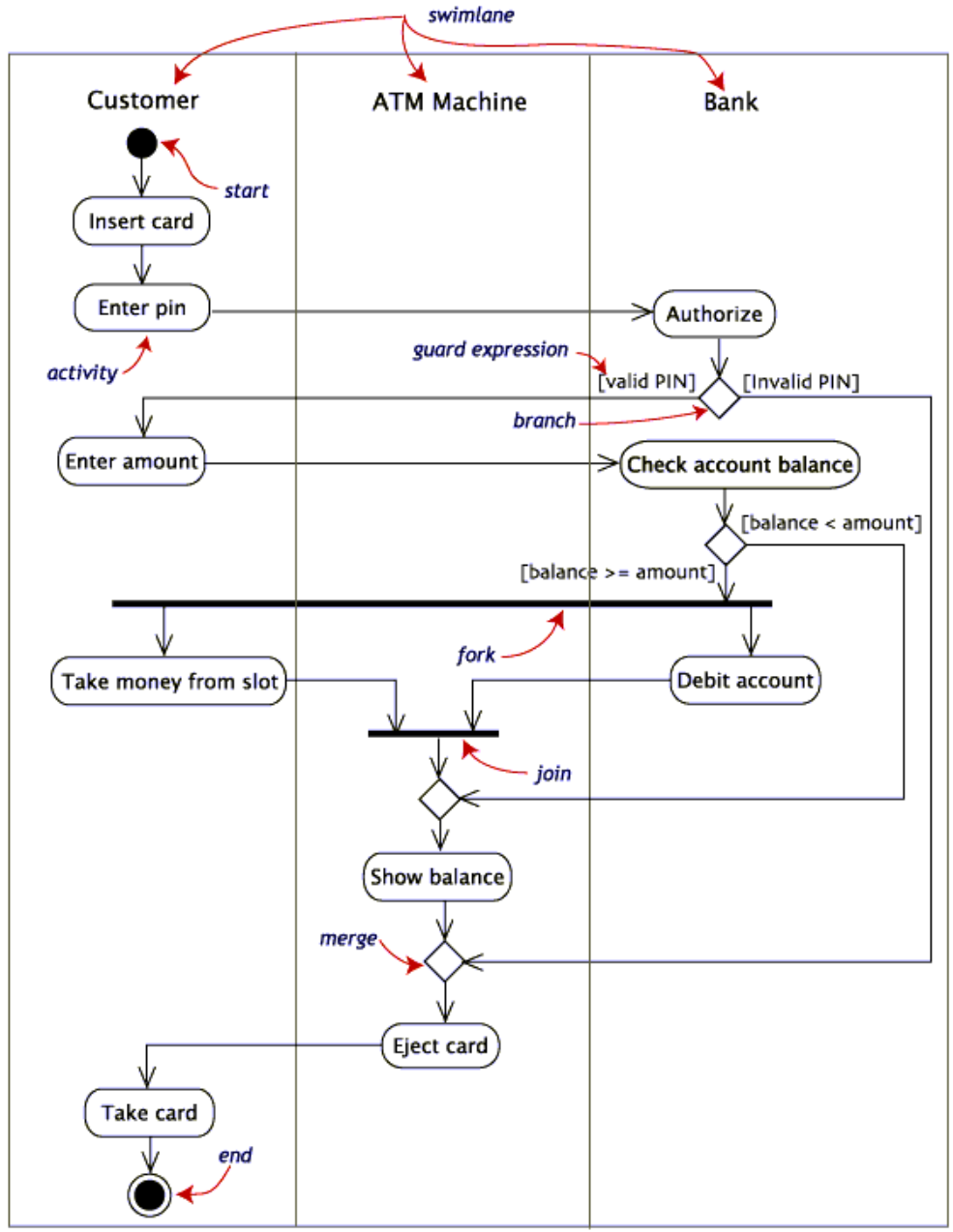


Analyse orientée objet

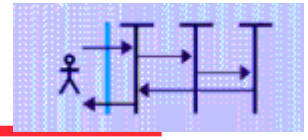
Utiliser les diagrammes d'activité pour décrire les scénarios des cas d'utilisation



Retirer l'argent d'une machine ATM



Analyse orientée objet



Étude de cas

Système de réservation de vols

Référence: UML par la pratique

De Pascal Roques

Risques et difficultés dans la modélisation des interactions et du comportement

- La modélisation dynamique est une tâche difficile
  - Dans un système de grande taille, il existe de nombreux chemins que le système peut emprunter.
  - Il est difficile d'attribuer le bon comportement aux bonnes classes:
    - *Le processus de modélisation devrait être supervisé par le développeur le plus expérimenté*
    - *Tous les aspects du modèle devrait être révisés attentivement.*
    - *Travailler de façon itérative:*
      - *Développer un premier diagramme de classes, puis les cas-types, les responsabilités, et ensuite les diagrammes d'interaction, d'activité et d'état;*
      - *Réviser et modifier ensuite les diagramme afin de les rendre consistants*
    - *Le fait de dessiner différents diagrammes représentant des aspects différents, mais très liés, aide à mettre en lumière les problèmes potentiels*

Points clés

- Le but de la phase d'analyse est de spécifier le système à développer
- Utiliser les diagrammes de cas d'utilisation pour décrire les fonctionnalité du système
- Utiliser les diagrammes de classes pour représenter les entités (objets) constituant du système
- Utiliser les diagrammes de séquence décrire des scénarios des cas d'utilisation