Pourquoi UML

Dominique Revuz

UML

- Un langage de Modélisation (standard)
- Basé sur une approche solide Objectory
- un support important de stratégies et d ’heuristiques
- Adapté au développement Objet
- Adapté au développement Itératif
- Adaptable au processus d ’une équipe
- supporté par des outils

Un langage de Modélisation

Formaliser les modèles:

- Le modèle des Exigences
- Le modèle d ’analyse
- Le modèle de conception
- Le modèle de test

Une aide à la conception

- Au tour d'UML un travail important à été réalisé pour matérialiser des stratégies et des heuristiques.
- Eviter les oublis, bien comprendre et bien répondre aux spécifications
- Analyser le risque, organiser le travail
- concevoir des architectures réutilisables extensibles efficaces

UML & OO

- UML utilise une approche objet des logiciels.
- Les entités métier sont identifies comme des objets
- les modules opérationnels sont aussi vus comme des objets
- La construction ce fait avec une approche YoYo (Top-down, Bottom-Up)

Itératif

L ’approche de la modélisation des besoins par cas d ’utilisation se prête très bien à un processus itératif de développement.

En effet les cas d ’utilisation offrent un découpage naturel en itération avec une validation directe par le client.

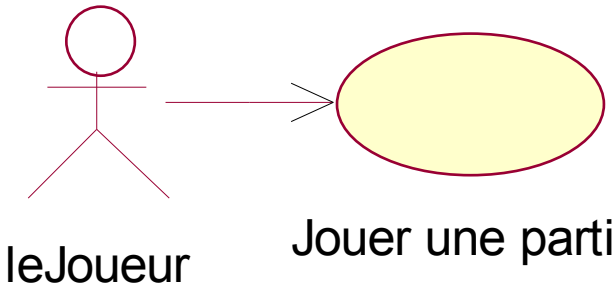
Adaptable

- UML n ’impose pas de méthode de développement. C ’est un standard de communication contenant un certain nombre d’artefacts utiles.
- Il est donc possible d ’intégrer l ’utilisation de UML à un processus de développement existant.

Des Outils

- UML est devenu le standard des outils CASE.
- Ces outils permettent de manipuler la plus part des artefacts UML.
- ROSE de Rational est un des meilleurs candidats

Un exemple



Première étapes: les Cas d'Utilisation

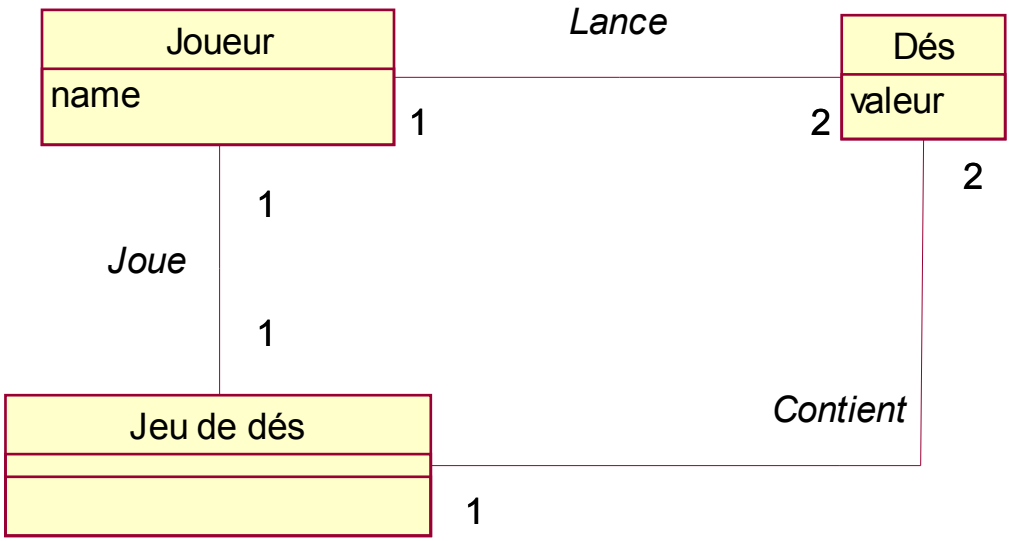
Use Case : Jouer une partie

Acteurs : Joueur

Description: Le jeu démarre quand le joueur prend les dés et les lance. Si le Total est égal à 7 le joueur gagne sinon il perd.

Deuxième étape le modèle conceptuel

c'est une description des objets du domaine, pas des éléments informatiques.



Construire un diagramme de Collaboration.

Ici, nous spécifions des objets logiciels qui réalisent les exigences grâce à une décomposition objet.

Les diagrammes de collaboration montrent le flot des messages entre instance et les méthodes invoquées.

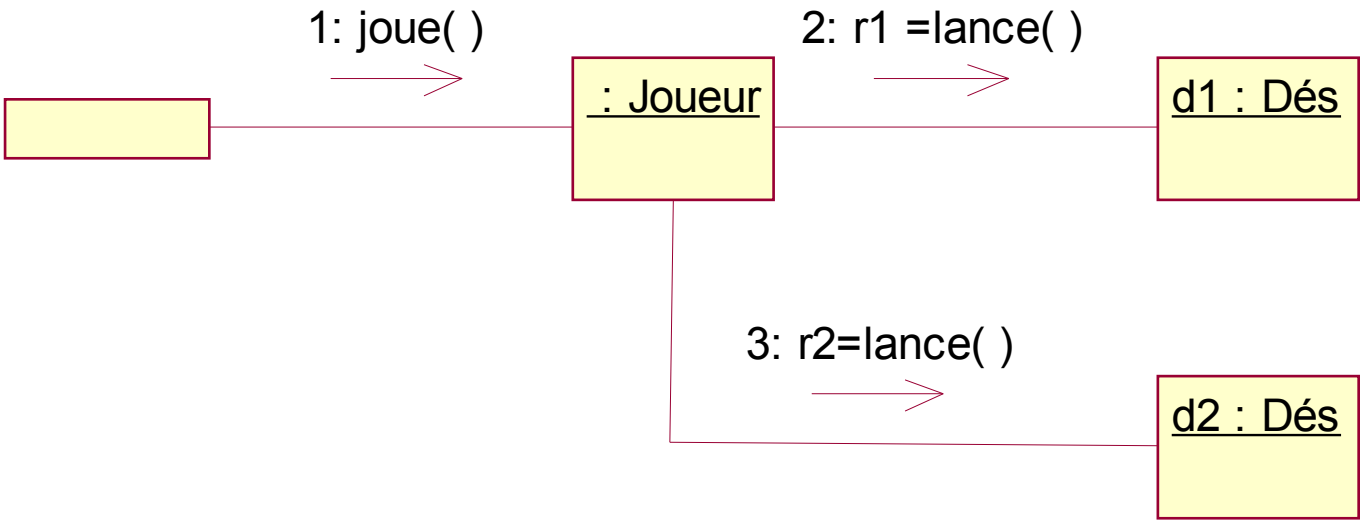
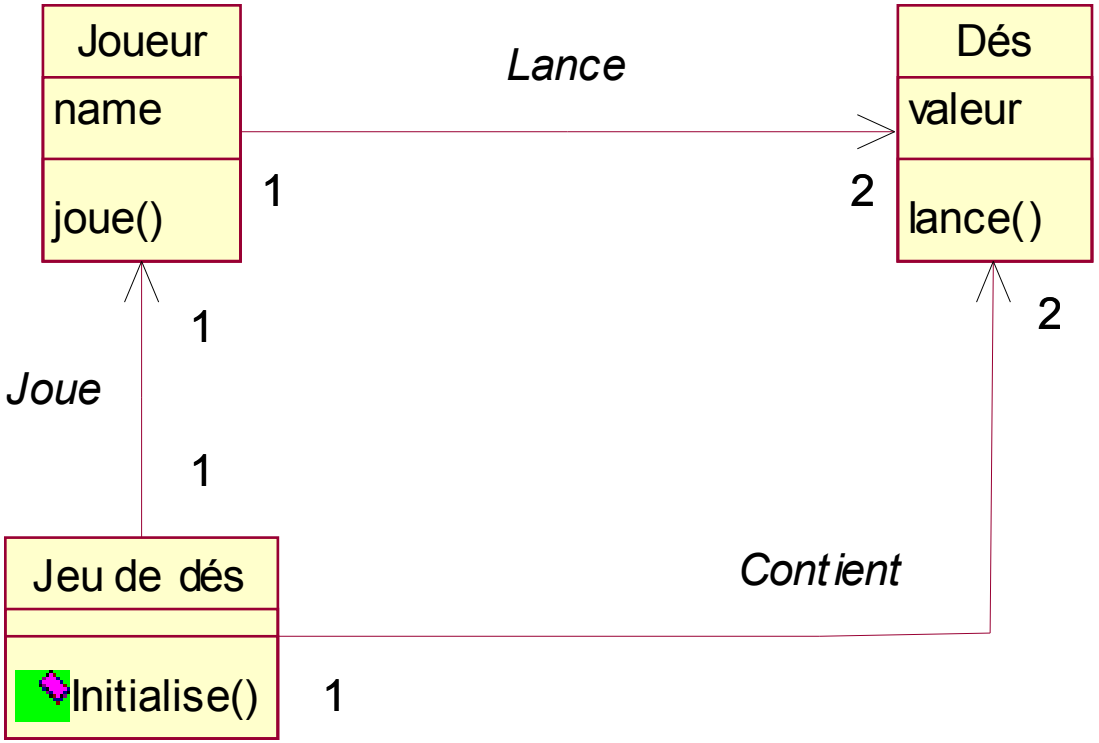


Diagramme de Classes

Comment les Objets sont connectés. Quels sont les dépendance fonctionnelles entre classes.

Quelles sont les Méthodes à définir.



UML

- un Langage pour la modélisation
  - Désambiguïser l'analyse et la conception
  - Formaliser les décisions stratégiques et tactiques
  - Outiller la compréhension du système

- Une méthode : Objectory de Ivar Jacobson

Pourquoi une Méthode

- Réduire les risques :
  - délais
  - coûts
  - faisabilité
  - adéquation
- Assurer la qualité logiciel

Et pourquoi itérative

- Permet de prendre en compte les changements.
- Les différents composants sont intégrés progressivement.
- Réduction des risques en amont et au plus tôt.
- Cela simplifie la réutilisation
- Construction plus robuste.
- Meilleur prise en main du système par les développeurs.
- Meilleur utilisation des ressources, parallèlisation (pipeline) des activités
- Evolution du processus de développement

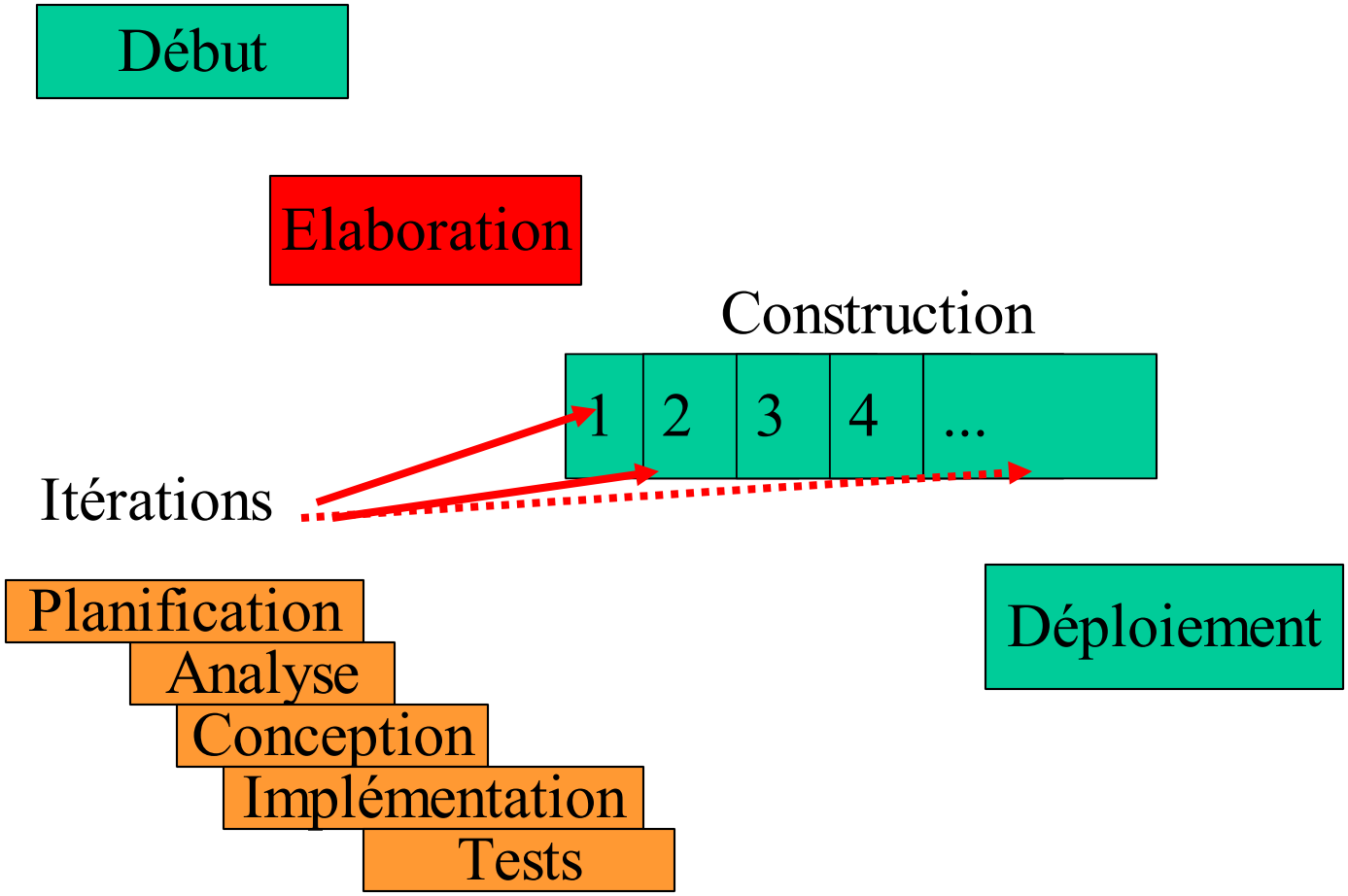
Les Points forts

- Stratégie de réduction des risques au plus tôt
- Compréhension du projet
  - Bonne définition de ce qu'il faut faire.
- Itérations
  - Des objectifs clairs à chaque étape
  - une mesure de l 'avancement

Résumé

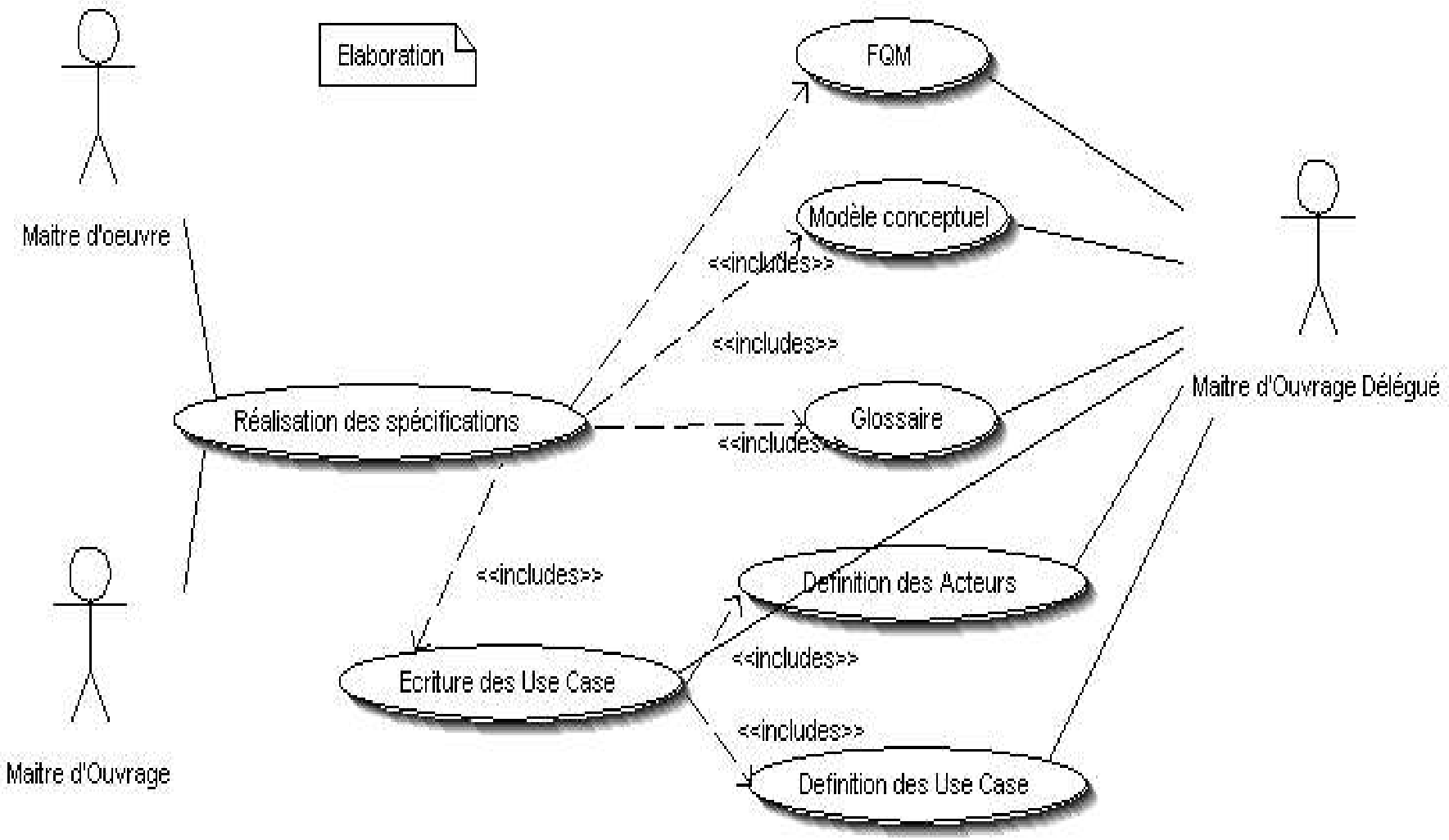
- Difficile => Des techniques pour organiser et simplifier le travail. Du savoir faire.
- Travail d'équipe => Un langage (UML). Des outils de travail collaboratif (CVS).
- Risqué => Méthodes

Le déroulement d'un projet



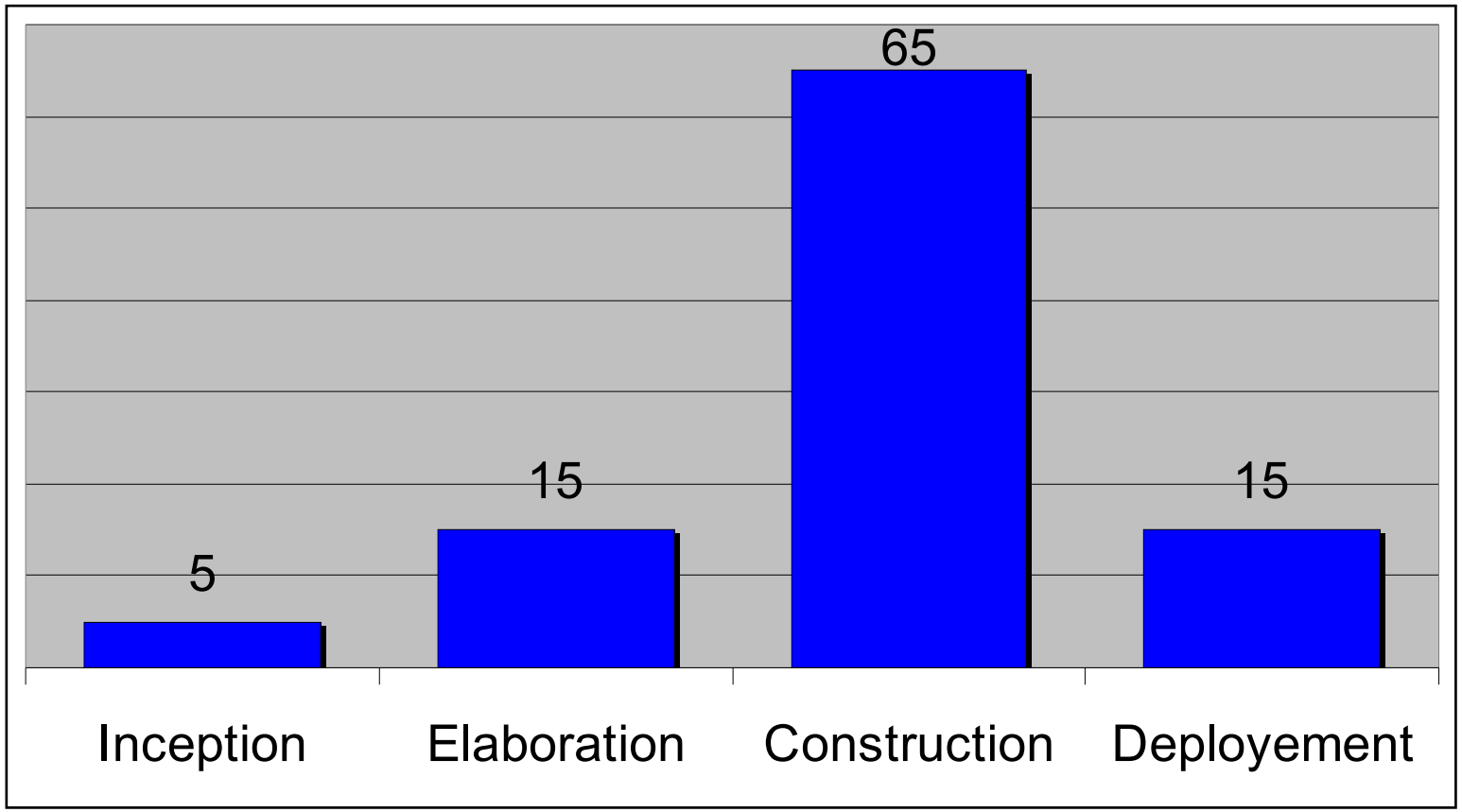
La Phase d'Elaboration avec UML

Dominique Revuz
dr@univ-mlv.fr


Elaboration
FQM
Modèle conceptuel
<<includes>>
<<includes>>
Maitre d'oeuvre
Réalisation des spécifications
Glossaire
<<includes>>
Maitre d'Ouvrage Délégué
<<includes>>
Definition des Acteurs
<<includes>>
Ecriture des Use Case
<<includes>>
Definition des Use Case
Maitre d'Ouvrage

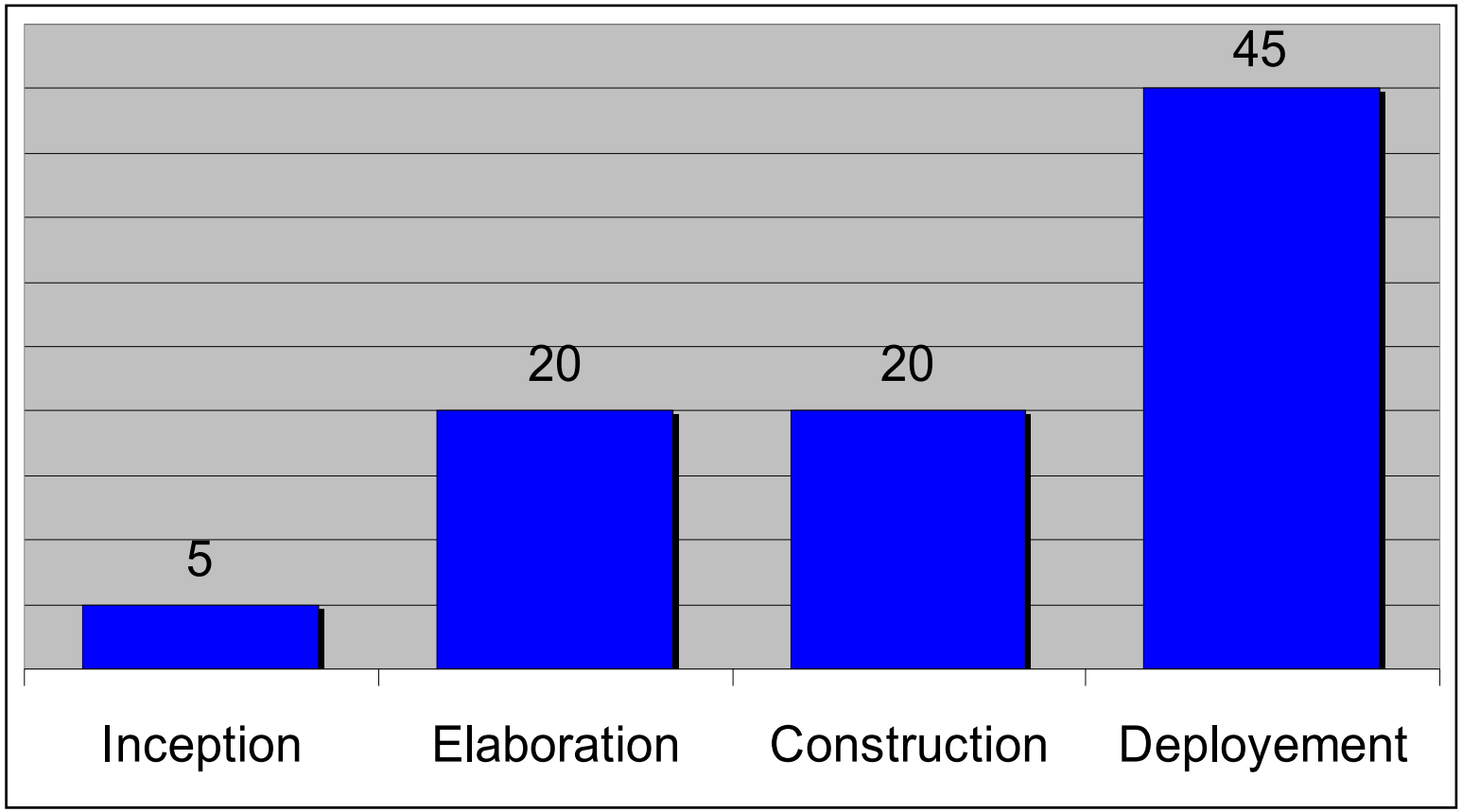
Organisation de l'Effort

Nouveau Développement



Organisation de l'Effort

Logiciels existants



L'état Virtuel

Inception

- Rapport d’opportunité
- L'équipe, les compétences, les experts.
- Planning
- Les ressources : personnel, budget, locaux etc.

La phase de démarrage

- Le rapport d'opportunité
  - Pourquoi ce projet, motivation
  - Ratio coût/rentabilité
    - marché, client, coût , trésorerie
  - Faisabilité
  - alternatives
  - disponibilités etc.

La phase de démarrage

- Faisabilité
  - Prototype
  - Difficultés spécifiques
  - Nouveauté
  - Expertise
  - Disponibilité des ressources
- Le Planning: intégration dans l'existant.

L'état Gazeux

Élaboration

- "Ce que le client veut"
- "Ce que l'on vas faire"
- "Comment on vas le faire"
- Spécifications des exigences (déclaratives)
- Glossaire, Prototypes
- Cas d’utilisation + diagrammes
- Brouillon de Modèle conceptuel

Élaboration

L'objectif de l'Élaboration est de réaliser les Spécifications (déclaratives) des exigences pour cella un certain nombre d'outils sont utilisés:

Glossaire, Cas d'utilisation + diagrammes

Modèle conceptuel, FQM, Prototypes.

Ceci proportionnellement à l'effort global.

Objectifs de l ’Elaboration

A. Construire le **modèle des exigences** du système : définir la cible

B. Ecrire les cas d ’utilisation et **planifier** la suite d’itérations qui va permettre de réaliser les objectifs de chaque cas d ’utilisation : atteindre la cible

ME: Quatre éléments

- Les fonctions / qualités / mesures
  - ce que fait le système
- Les acteurs, Les sous-systèmes
  - identifie ce qui est extérieur au système
- Les Cas d'utilisation
  - description des interactions avec le système
- Le glossaire & le Modèle conceptuel
  - une base pour la description du système

Modéliser les exigences

- Capturer l'ensemble des exigences
  - avec un point de vue utilisateur
  - en minimisant les oublis
- Synthétiser l'essentiel
  - pour organiser le développement
  - pour permettre la réutilisation
  - pour mesurer et réduire le risque

Différents points de vue

Point de Vue	***Exprime***	***Problématique***
Travailleur	Rôles et responsabilités des utilisateurs du système	• Activités des utilisateurs • Décisions d'automatisation • Interaction Homme/Machine • Spécification des compétences utilisateur
Logique	Décomposition logique (en UML) du système en un ensemble de sous système cohérents qui fournissent le comportement désiré	• Adapter les fonctionnalités du système pour réaliser les Cas d'utilisation • Extensibilité Maintenabilité • réutilisation interne • Cohésion et connectivité
Physique	Décomposition Physique du système et spécification des éléments physiques	Adéquation aux besoins du logiciel et réponses aux besoins
Données	Données stockées et traitées par le système	Capacités de stockage suffisantes Débits suffisants pour un temps de réponse adéquate
Processus	Activités (threads) qui réalise es éléments de calcul	Découpage des phase de calcul pour assurer les besoins de sûreté de fonctionnement

DR's Vending Machine

Notre logiciel de démonstration :)

Les activités du système: enregistrer les achats et gérer les paiements.

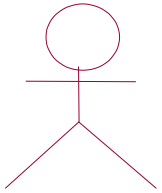
Le système est composé d'éléments matériels et logiciels.

Supposons que nous avons été sollicités pour réaliser le logiciel de ce système.

Première Tâche

Identifier les Acteurs,
Identifier les Cas d'Utilisation.

Les Acteurs et leur Rôle

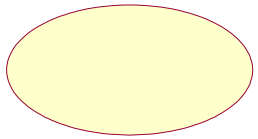


- Un acteur est une entité externe au système qui participe d'une certaine façon à un des Cas d'utilisation (Use case).
- Un acteur est représenté par son Rôle pour le système : Client, Caissier, OS, etc.
- Fait des accès en lecture ou écriture

Identifier les Use Case

Les Cas d'Utilisation (Use Case)

Un cas d'utilisation *est un document narratif qui décrit la séquence d'événements qui permet à un acteur de compléter un objectif en utilisant le système.*



Acheter des Articles

Use Case Name: Acheter des Articles

Acteurs: Client, Caissier

Type: Primaire (hiérarchisation)

Description:

Un client arrive à la caisse avec des Articles à acheter.

Le caissier enregistre les Articles et prend le paiement.

Le client peut ensuite partir avec les Articles.

Erreurs typiques du mod èle des cas d'Utilisation

Un seul diagramme	Découper le diagramme des cas d'Utilisation en plusieurs diagrammes en particulier un par cas d'utilisation !
Acteurs	oublier des système externes Inclure plusieurs acteurs qui ont exactement les mêmes objectifs quand il utilise le système et donc devrais être regroupés sous le même rôle.
Cas d'Utilisation	Inclure des fonctions auxiliaires qui ne font pas partie des besoins, oublier des cas d'utilisation distribuer des évènements sur plusieurs cas d'utilisation alors qu'ils participent a un même objectif d'acteur.
Contenu	Oublier des conditions environemental ou des informations clients (comme par exemple la fréquence du cas d'utilisation), oublier des résultats
Niveau de détail	Description du comportement interne du système, Donner une description trop courte ou incomplette
Réalisme	Inclure dans plusieurs cas d'utilisation des fonctionalités qui devrai être regroupées dans un seul cas d'Utilisation indépendant, Oublier des variations, ou oublier de spécifier ou ces variations appraisses dans le flot d'évènements.

Le Glossaire

Le glossaire est un outil utilisé tout au long du développement.

Il contient l'ensemble des termes employés dans les modèles utilisés au cours de la construction du système.

Les termes du glossaire deviennent en général la base des *objets métiers* importants du système.

Le glossaire permet de réduire l'ambiguïté des termes utilisés.

Le glossaire est construit au cours des différentes phases du projet. C'est surtout dans la phase de modélisation des exigences qu'il est le plus utile car il permet de travailler avec les *termes du client* ou de l'expert métier, qui sont ainsi expliqués pour tous.

La création du glossaire est une tâche coopérative, simple, avec un excellent rapport coût/valeur.

Le glossaire de DR's VM

- Article : ce qui est en vente dans le magasin
  - prix, Taux TVA, identifiant etc.
- Caisse : poste de travail du caissier, composé d'un ordinateur, un tiroir, un lecteur de code barre, etc.
- Reçu : ticket indiquant date, heure, articles, somme totale, etc.

Ou est la Cible ?

- Nous avons décrit pour le moment que les aspect Interactifs de notre Système.

- Le Client demande à l'agence de voyage d'être transporté de Paris à Lyon, sans préciser de moyen de transport.
- Budget, Vitesse, Date, Disponibilité ->

Deuxième Tâche

Expliciter les Exigences de façon mesurable

N'est Connu que ce qui est mesurable

Lord KELVIN disait clairement que :

Sans l'aptitude de mesurer, on ne peut progresser.

Le progrès de nos systèmes de mesure est également le progrès de la science.

FQM

- Exigence : Une condition ou une capacité a la quelle le système doit se conformer.
- Croiser les **Fonctions** et les **Qualités** nous permet d ’identifier les deux formes d ’exigences le quoi et le comment.
- Les **Mesures** permettent de valider que l ’objectif qualité est atteint.

Les fonctions du système

Tout ce qui entre dans le patron:

Le système doit faire <X> .

Catégorie	Sens
Evidente/service	**s'exécute et l'utilisateur en est conscient**
Cachée/technique	**s'exécute est nécessaire au fonctionnement**
Excitante/Estime	**Peu d'effet sur les coûts Mais aide la vente**

Exemple de cas

REF	Fonction	Catégorie
1.1	Enregistre la vente en cours c.a.d. les articles achetés	évidente
1.2	Calcul du total courant + tva	évidente
1.3	Mise à jour de l'inventaire	cachée
1.4	Enregistrer les ventes terminées.	cachée
1.5	Procédure d'identification du caissier.	évidente

Qualités Voulues

- Facile d ’utilisation
- Robuste
- Portable
- Temps de réponse
- Interface intuitive
- Scalable (supporte la montée en charge)

Lexi Qual :-)

Les qualités spécifiques au système

- Des réponses pertinentes
- Meilleur classement / évaluation (scoring)

Exemple de cas

Il faut croiser les fonctions avec les Qualités

REF	Fonction	Cat.	Qualité	Mesure	Cat.
1.9	Affiche la description et le prix de l'article	évidente	Temps de réponse	5s Max	Obligatoire
			Interface	Formulaires couleurs	Obligatoire voulu
2.4	Enregistrer les demandes de crédits	cachée	robuste	Doit être enregistré dans les 24h.	Obligatoire
			Temps de réponse	10s max	Obligatoire

Les cas d'utilisation de l'élaboration

Dans la phase d'élaboration, les cas d'utilisation sont rapidement esquissés, mais **tous** les cas d'utilisation doivent être décrits.

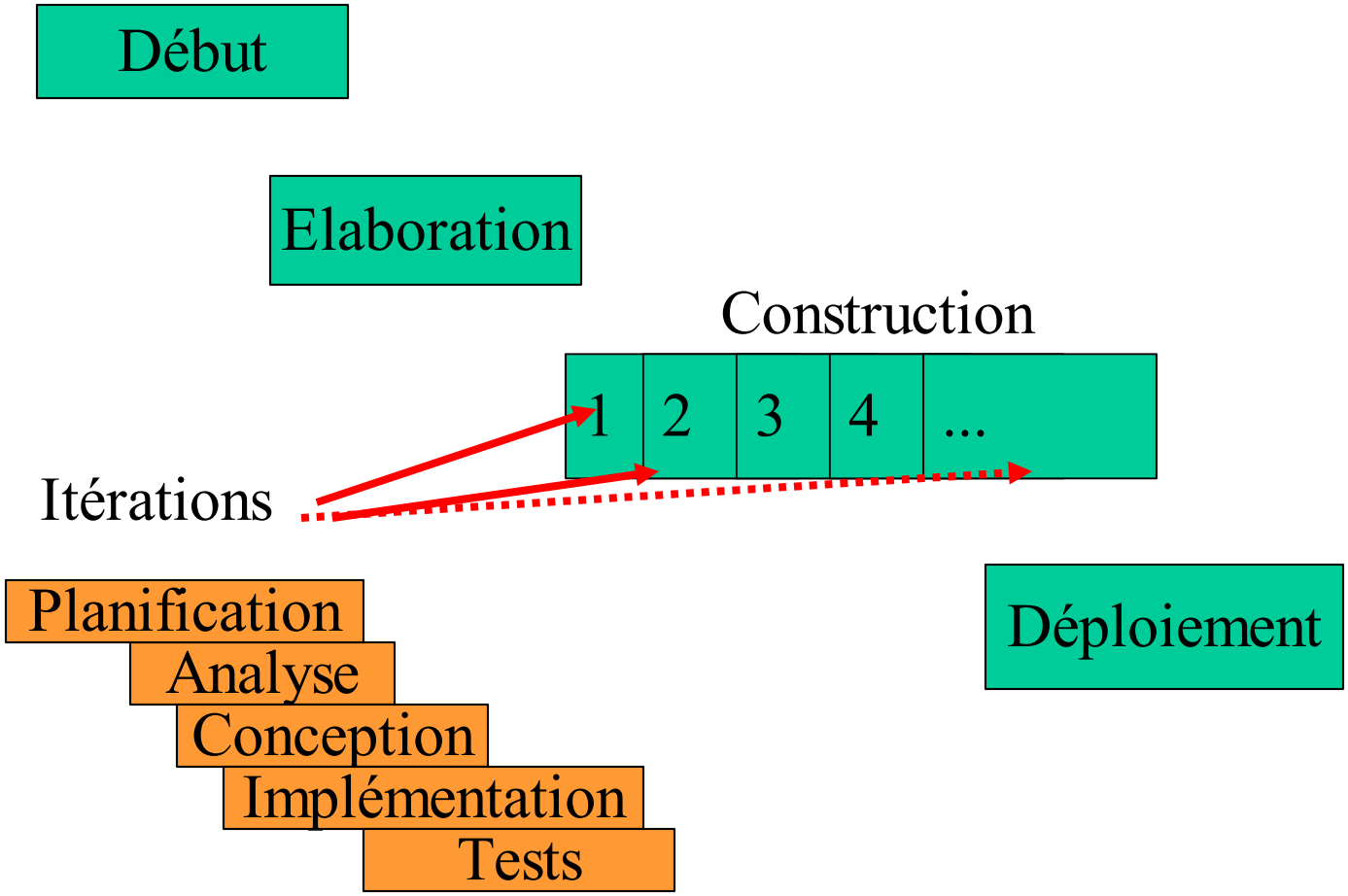
Il est possible, si le système le demande, de *détailler* les cas d'utilisation principaux.

Plus de détails => plus d'informations

Compromis : Meilleure compréhension, gestion des risques, estimation. Mais une plus grande complexité et plus de travail inutile en cas de changement.

Une Méthode itérative

Vue synthétique



Une approche itérative de la construction

- Raffiner la planification de l'itération
- analyse
- conception
- implémentation
- test

Une itération

L'ensemble des opérations sont réalisées dans la fenêtre de pilotage.

Une itération doit se terminer avec un logiciel évaluable.

Pour avoir une itération de bonne taille elle doit être définie avec l'équipe de développement qui doit satisfaire la contrainte temporelle.

Cas d'utilisation et Itérations

Les itérations sont définies sur des cas d'utilisation.

Les exigences d'un cas d'utilisation sont l'objectif d'une itération.

Les cas d'utilisation principaux sont réalisés dans les premières itérations.

Certains besoins non évidents doivent être placés dans les premières itérations, des services de support, persistance, sécurité, etc.

Création des Cas d'Utilisation

- Identifier et écrire les cas d 'utilisation
- dessiner les diagrammes de cas d 'utilisation

2 Axes

- Haut-Niveau vs Détaillé
- Essentiel vs Réel

HN vs Détaillés

Degré de description d ’un cas d ’utilisation

Concepts

Interactions précises

UC de Haut Niveau

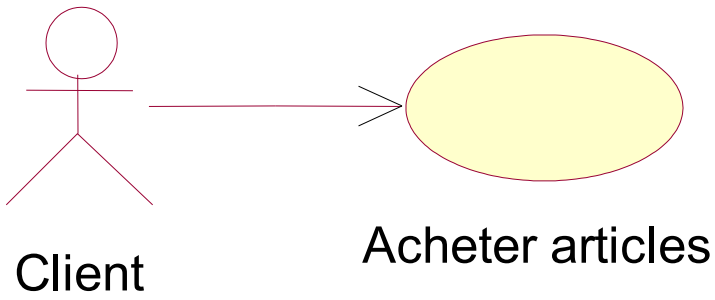
Use Case: Acheter des Articles

Acteurs: Client, Caissier

Type: Primaire (A discuter)

Description: Un client arrive à la caisse avec des articles à acheter. Le caissier enregistre les articles et prend le paiement. Le client peut en suite partir avec les articles.

Diagramme de UC (1)



UC Détaillé

Use Case: Acheter des articles avec du liquide

Acteur: Client(initiateur), caissier

Objectif: enregistrer une vente avec son paiement en liquide.

Résumé: Un client arrive à la caisse avec des articles à acheter. Le caissier enregistre les articles et prend le paiement. Le client peut en suite partir avec les articles.

Type: Primaire et essentiel

Références: Fonctions 1.1, 1.2, 1.3, …

Suite d'événements:

(le Style conversationnel : comme une conversation entre les acteurs et le système)

Actions de l'acteur	Réponses du système
1.Ce UC commence quand un client arrive à la caisse	
2.Le caissier enregistre chaque article. Si il y a plusieurs articles identiques leur nombre peut être entré par le caissier.	3. déterminer le prix de l'article mettre à jour la liste des articles. Le prix et la description de l'article courant sont affichés
4. le caissier indique que l'ajout d'articles est terminé	5. Calcul et affichage du total.
6. le caissier annonce le total.	
7. le client fourni un paiement en liquide supérieur ou égal au total.	
8. le caissier entre le montant du liquide fournit	9. Affichage de la monnaie a rendre. Imprime un reçu.
10. Le caissier range le liquide dans la caisse, en retire la monnaie. Fournit la monnaie et le reçu au client.	11. enregistre la vente.
12. Le client part avec ses articles.	

Diagramme d'Évènements

- Une technique de description du dialogue des acteurs avec le système et le diagramme d'évènements.

- -> dessin de achat article.

Un UC doit être complet

Un UC est une description de « bout en bout » d ’un processus assez large qui contient un certain nombre de transactions et d’étapes.

Une erreur commune est de définir un cas d ’utilisation pour une étape, une opération, ou une transaction.

Trouver les UC

- A partir des acteurs
  - Identifier les acteurs
  - pour chaque acteur identifier les UC qu'il initie ou auxquels il participe
- A partir des évènements
  - Identifier les évènements auxquels le système doit répondre
  - Relier l 'événement aux acteurs et aux UC.

Exemple de cas

Caissier

- Ouverture
- Fermeture

Client

- Acheter des articles
- Retourner des articles

UC et processus métier

- Retirer de l ’argent avec un DAB
- Commander un produit
- s’enregistrer a un enseignement
- Vérifier l ’orthographe d ’un document.
- Gérer un appel téléphonique
- Réserver une place

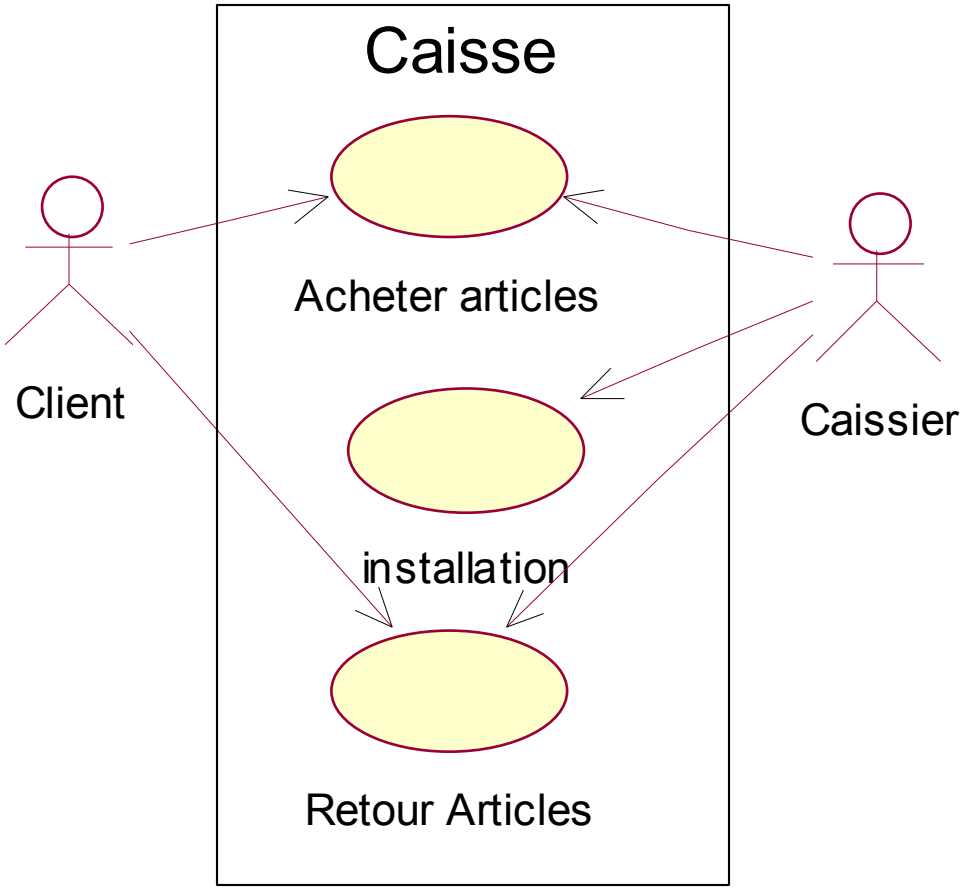
UC + Fonctions + Traçabilité

Les fonctions du système identifiées dans la première étape doivent être allouées aux cas d ’utilisation (les références dans les Use Case permettent de vérifier qu’il n ’y a pas d ’oubli).

Ceci donne un bon outil de traçabilité entre les différents artefacts.

Idéalement toutes les fonctions et cas d ’utilisation doivent pouvoir êtres tracés jusqu’à l ’implémentation et les tests.

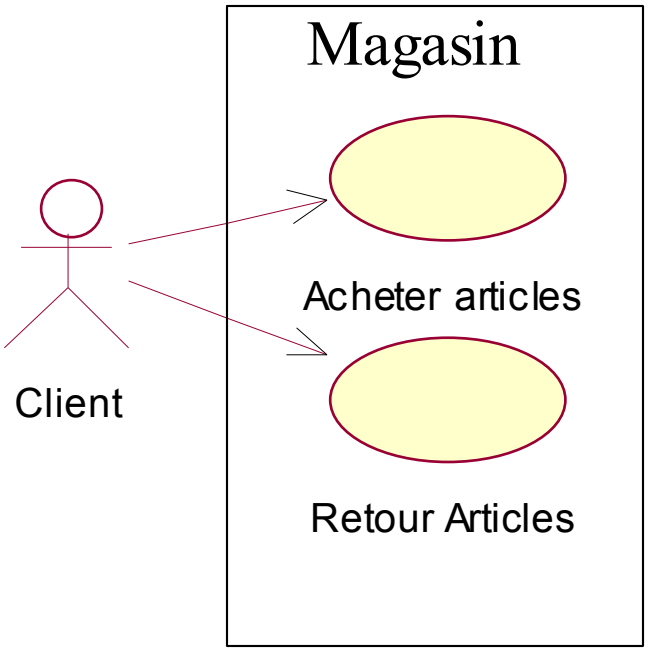
Diagrammes de Cas D'utilisation



Frontières

- Les UC décrivent des communications avec un système, quelles sont les frontières de ce système.
- Matérielles/Logicielles
- un département dans une organisation
- une organisation entière

Quelle frontière?



L ’objectif ici devient : modéliser le processus métier.

LexiGuide

- Quelle frontière ?
- La base linguistique (le « lingware ») est il un élément du système ou un sous-système (un acteur) avec lequel il communique.

Essentiels vs Réels

Degré de conception d ’un cas d ’utilisation

Essentiel
Abstrait

Réel
concret

Essentiel

- UC détaillés d ’une forme idéale qui ne prend pas en compte les questions technologiques.
- Les décisions de conceptions sont reportées et abstraites. Typiquement celles se reportant à l ’interface utilisateur.
- Montrer les activités essentielles et les motivations pour le cas d ’utilisation

- Les UC de haut-niveau sont par nature essentiels.

- Ex: Un Distributeur Automatique de Billets

Action de l'acteur	Réponse du système
1 Le client s'identifie	2. Le système propose des options
Et ainsi de suite	

- le système d 'identification peut changer au court du temps c'est une décision de conception.

Ils sont Essentiels

- Courts, créés au début, ils ont une longue durée de vie. Ils aident à la compréhension du système, ils posent les bases.
- Simples à construire car indépendants de la conception.
- Ils servent de base à la conservation du savoir et du savoir faire.

Les réels

- Description du processus dans les termes de la conception courante, définis sur des choix technologiques: d'entrées/sorties, C++, etc.
- Les interfaces sont décrites (dessinées)

Action de l'acteur	Réponse du système
1. Le client entre sa carte	2. Demande du code
3. Le client entre son code sur le clavier	4. Si le code est valide affiche le menu d'options
Et ainsi de suite	

(pas très « réel » mais bon :-)

Types d ’UC dans les phases de la conception

Phase d ’Analyse :

Une version détaillée et essentielle des UC développés pendant l ’itération

Phase de Conception :

Une version détaillée réelle

UC et planning

1. Liste des fonctions, frontière, identification des acteurs et des UC
2. Ecrire mode Haut-Niveau tous les UC
3. Dessiner les Diagrammes de UC
4. Relations entre UC
5. Versions détaillées pour les UC les plus risqués
6. Si nécessaire écrire des UC réels*
7. Ordonner les UC.

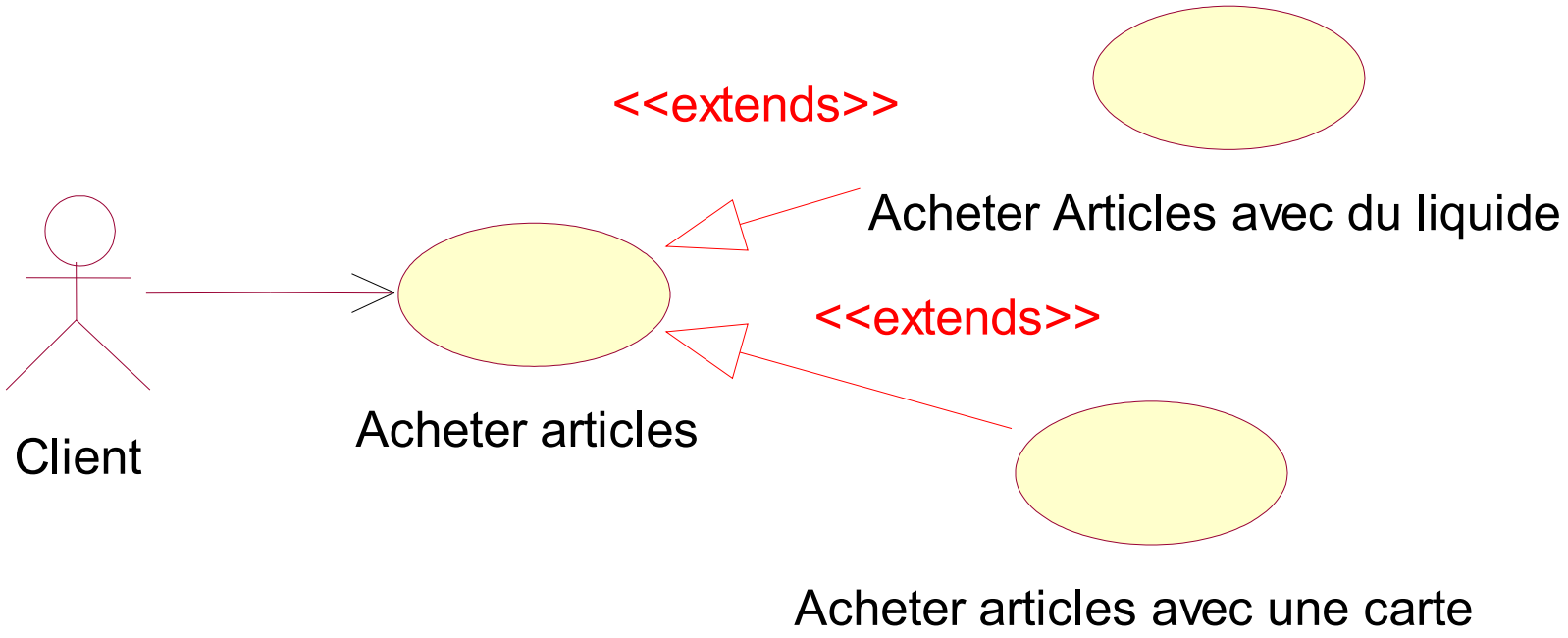
Les 4 productions essentielles de l ’Elaboration

- Les Fonctions
- Les Acteurs
- Les Cas d ’Utilisation
- Le glossaire et le Modèle conceptuel

Les autres Productions

- Diagrammes de Collaboration
- Diagrammes de séquences
- Diagramme de Use Case

Diagramme de UC (1)



Motivations de l'extension

- Conserver un cas d'utilisation Haut niveau simple et compréhensible
- Préparer la répartition des cas d'utilisation sur les itérations
- Conserver le concept « Bout en Bout »

Et notre point de vente ?

- Frontière : les limites du matériel
- Les acteurs (non exhaustif) et UC
  - Caissier (login , faire la caisse)
  - Client (acheter articles, retourner articles)
  - Directeur (Lancer , fermer)
  - Administrateur (ajouter un utilisateur)

Use case : lancer

Acteur : Directeur

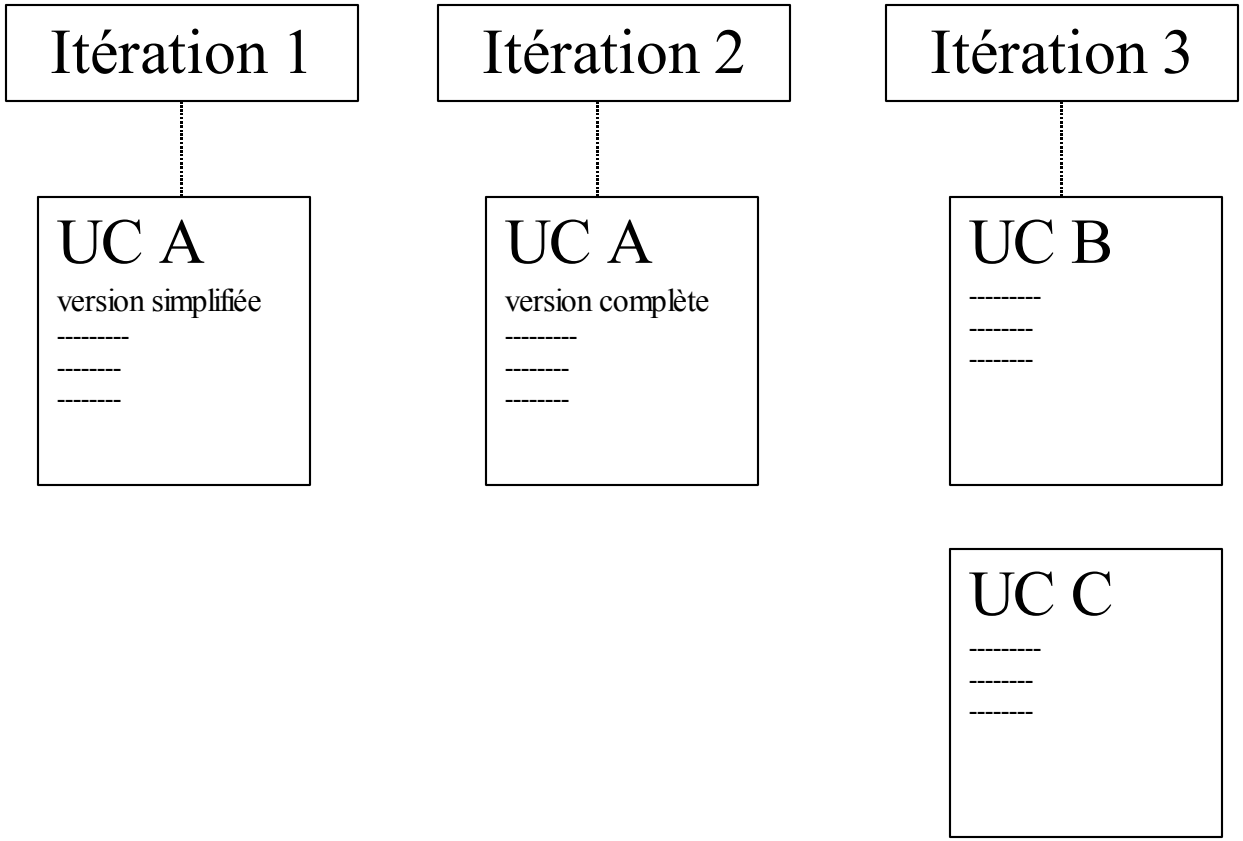
Type : primaire

Description: Le directeur allume le point de vente pour le préparer pour l ’utilisation par un caissier. Le directeur valide la date et l ’heure, après quoi le point de vente est près pour être utilisé par un caissier.

Ordonnancement des Cas d ’Utilisation

- Hiérarchiser les cas d ’utilisation
- allouer les cas d ’utilisation aux itérations
- éventuellement créer des cas d ’utilisation simplifiés

Allouer des UC aux itérations



Hiérarchiser les UC

- Objectif réaliser les UC les plus haut en premier de façons a réduire le risque
- Une bonne stratégie est de commencer par les UC qui ont le plus d ’impact sur architecture centrale (core).

Éléments de comparaison

A. impact important sur l ’architecture de conception. ex: ajout de beaucoup de classes ou services de persistance (données p.)

B. De bonnes informations et une bonne intuition sur la conception sont obtenues avec peu d ’efforts.

C. Contient des Fonctions risqués, complexes ou avec des contraintes de vitesse.

D. Nécessite un effort de recherche, d ’apprentissage ou une technologie risqué

E. Représente la base du processus métier réalisé

F. Augmente les revenus ou réduit les coûts

Évaluation

Pour chaque critère et chaque UC on évalue le critère
soit de façon floue : haut, moyen,bas
soit avec un score comme par exemple ceci :

Use Case	a	b	c	d	e	f	somme
Achat articles	5	3	2	0	5	3	18

Ordonner les cas d ’utilisation

- Ex:
  - 1 *Acheter articles* simplifié
  - 2, 3, 4 *Acheter articles* avec plusieurs mode de payement
  - 5 *Mise à jours de l ’inventaire*

Architecture

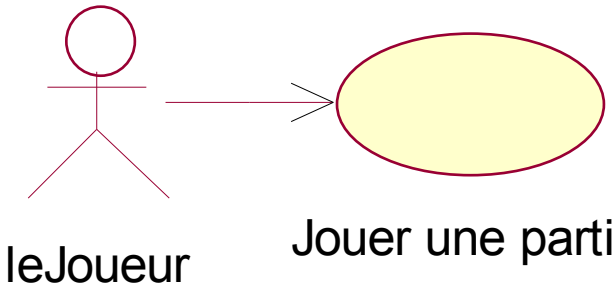
Par couches (layers)

Focalisation principale pour la définition des cas d 'utilisation

Logique Applicative

- Interface utilisateur
- Objets métier
- Objets logiciels
- Base de donnée

Un exemple



Première étape les Cas d'Utilisation

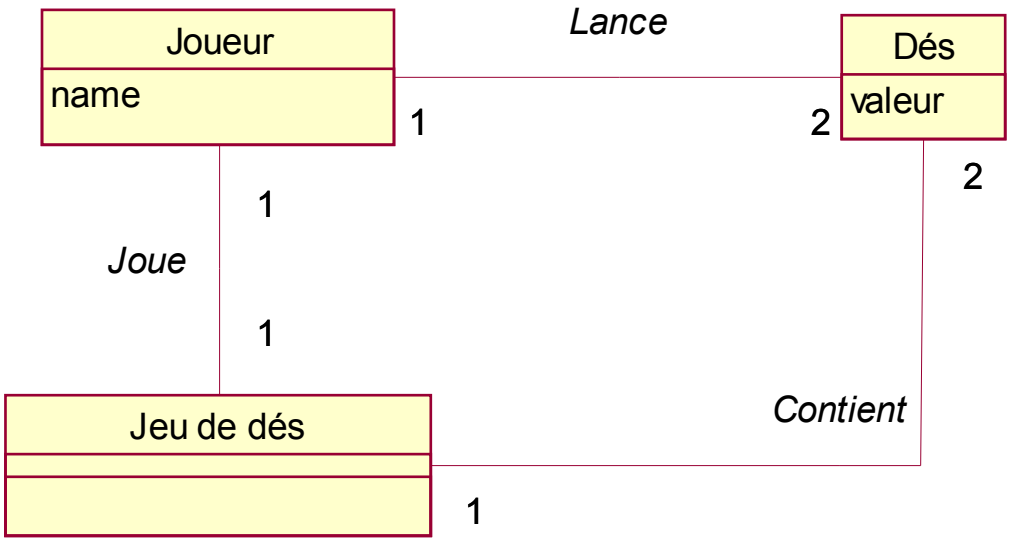
Use Case : Jouer une partie

Acteurs : Joueur

Description: Le jeu démarre quand le joueur prend les dés et les lance. Si le Total est égal à 7 le joueur gagne sinon il perd.

- Deuxième étape le modèle conceptuel

  c'est une description des objets du domaine, pas d'élément informatique.



Construire un diagramme de Collaboration.

Ici nous spécifions des objets logiciels qui réalisent les exigences grâce à une décomposition objet. Les diagrammes de collaboration montrent le flot des messages entre instances et les méthodes invoquées.

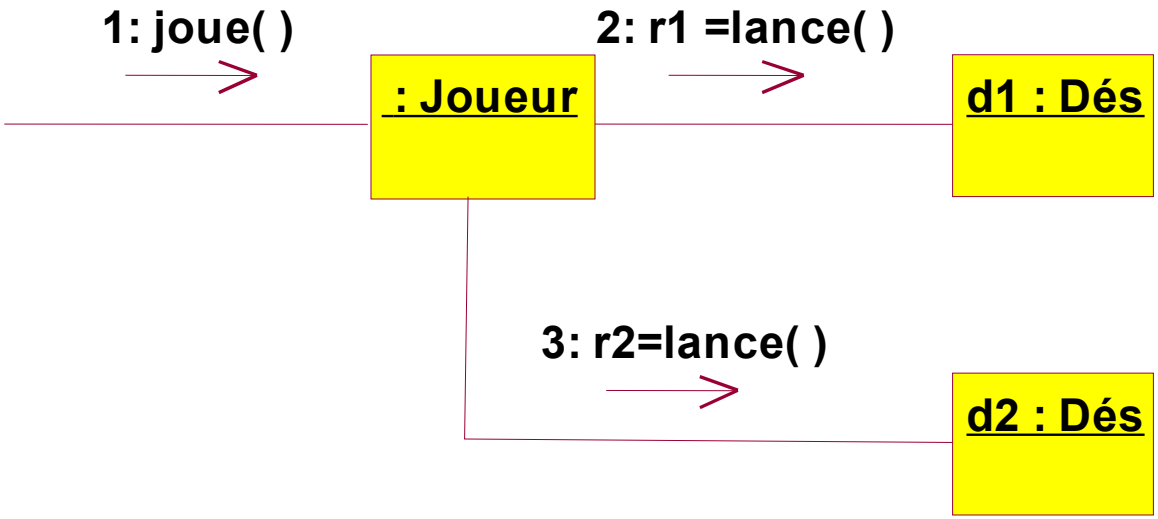
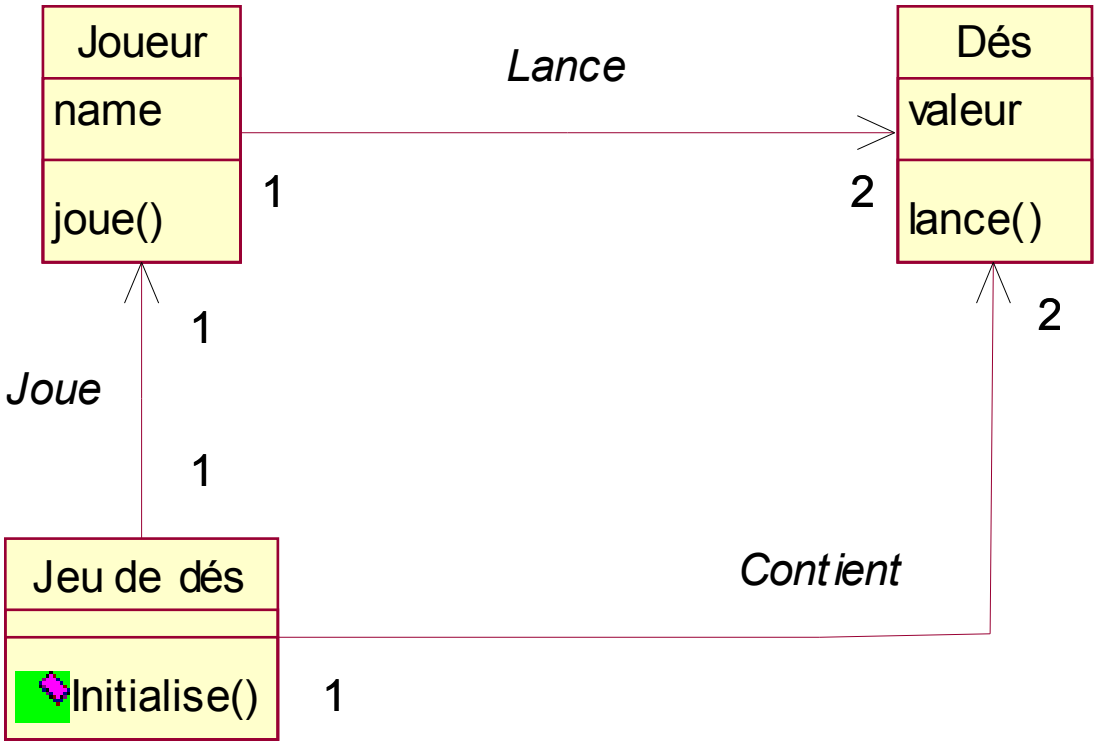


Diagramme de Classes

Comment les Objets sont connectés.

Quelles sont les Méthodes à définir



Choisir sa propre méthode

Le plus important est de créer une bonne conception.

Ceci demande une bonne maîtrise de principes et d'heuristiques pour identifier et abstraire les **bons objets** et leur affecter les bonnes **responsabilités**.

Le format détaillé

Use Case: nom **Acteur**: initiateur et tous les acteurs

Objectif:

Résumé: Récupération du UC de haut niveau

Références:

Type: Primaire/Essentiel

Références: Fonctions 1.1, 1.2, 1.3, …

Suite d'événements:

Modèle Conceptuel

dr@univ-mlv.fr

Construire un Modèle conceptuel

- Identifier les concepts relatifs au cycle de développement courant
- Créer un modèle conceptuel initial
- Distinguer entre des attributs corrects et incorrects
- Ajouter des concepts de spécification quand c'est approprié
- Différencier: concept, type, classe et interface

Première partie les Concepts

Concepts : le chemin vers les Objets

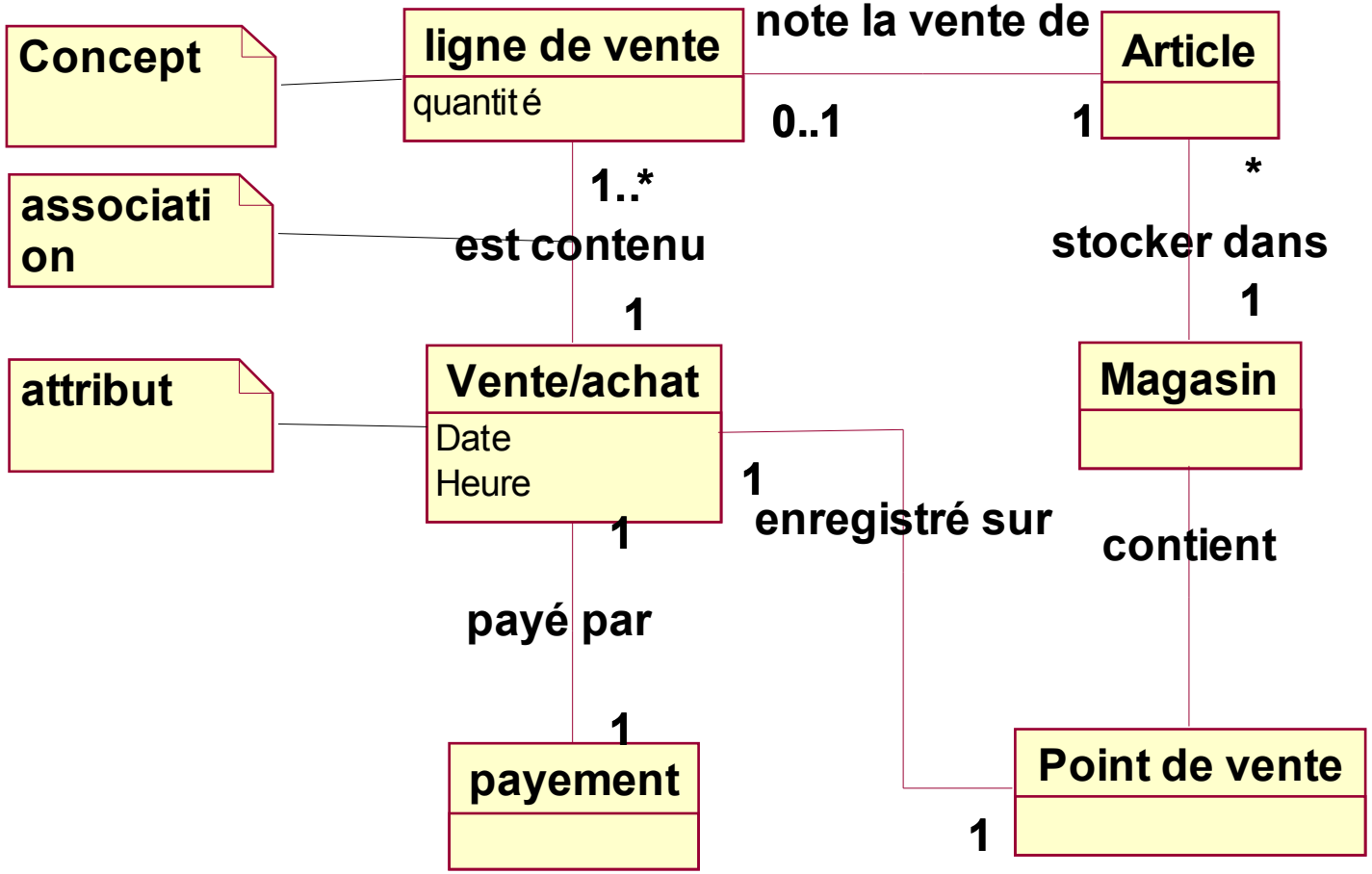
- Le travail de créer de bon concepts à l'analyse est payant en conception et en implémentation
- La Qualité d'un modèle conceptuel vient du fait qu'il représente bien le monde réel.
- Le modèle conceptuel initial est créé avec les Use Case pendant l 'Elaboration

Le modèle conceptuel ne contient pas d ’objets

- Le MC doit montrer
  - les concepts du domaine
  - les associations entre concepts
  - les attributs des concepts

- Le MC est matérialisé en UML par des diagrammes

Caisse: Un modèle conceptuel Partiel



Concept ≠ Classe

- Pas de Méthodes
- Pas de Concepts logiciels

- Le seul cas de figure ou l ’on parle de logiciel est le cas ou le domaine du système est celui du logiciel.

- Pas de responsabilités dans les concepts !

Concept

- Symbole = mots ou images représentant le concept
- Intention = la définition du concept

- Extension = un jeu d ’exemples auxquels le concept s ’applique.

Décomposition & stratégie

- Les concepts sont un outils de décomposition
- Ils permettent de réduire la complexité ou du moins de simplifier la manipulation des systèmes complexes

- Il faut mieux avoir un modèle conceptuel trop riche (détaillé) que trop pauvre.

Une Check liste pour les concepts

- Physique tangible
- spécifications, conceptions, descriptions de choses
- lieux
- transactions
- éléments de transactions
- Rôles de personnes
- conteneurs
- éléments dans un conteneur
- appareils
- Noms de Concepts abstraits
- Organisations
- événements
- processus
- règles & politiques
- catalogues
- documents financiers de travail, contractuels , légaux
- instruments financiers
- manuels livres

Les Groupes Nominaux

- Une méthode pour la recherche de concepts est d ’associer un concept ou un attribut à chaque groupe nominaux trouvés dans les descriptions du domaine (de la description des UC).
- C ’est léger
- C ’est dangereux du fait de l ’ambiguïté de la langue.
- Mais fait avec précaution cela peut être utile

Reçu $\in$ Modèle Conceptuel ?

- Reçu = description d ’une vente
  - inutile dans le Modèle conceptuel, en effet il est construit avec les autres éléments
- Reçu = document permettant le remboursement d ’article
  - doit apparaître dans le modèle

Construire un Modèle conceptuel

1 Faire la liste des Candidats concepts

2 Les dessiner en un modèle conceptuel

3 Ajouter les associations

4 Ajouter les attributs

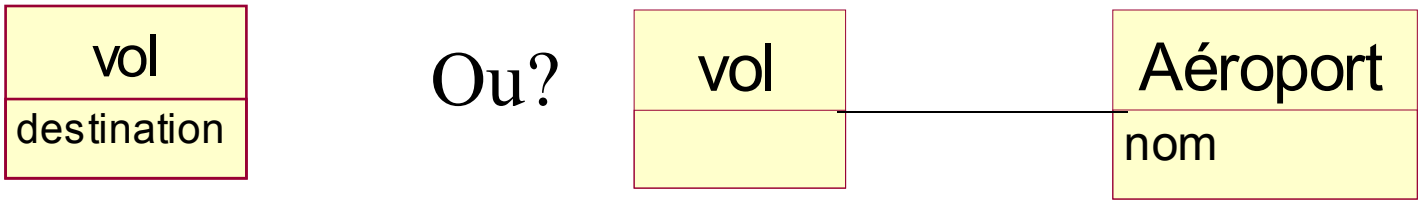
La stratégie du Cartographe

- Utiliser les Noms existants
- Exclure les éléments non pertinents (FPC)
- Ne rien ajouter !

Privilégier les concepts par rapport aux attributs

Si X ne peut être identifié comme une chaîne ou une valeur numérique alors c'est un concept sinon c'est un attribut.

Dans le doute c'est un concept



Le Vrai et le Faux

- Un modèle conceptuel n'est ni vrai ni faux
- il est plus ou moins utile

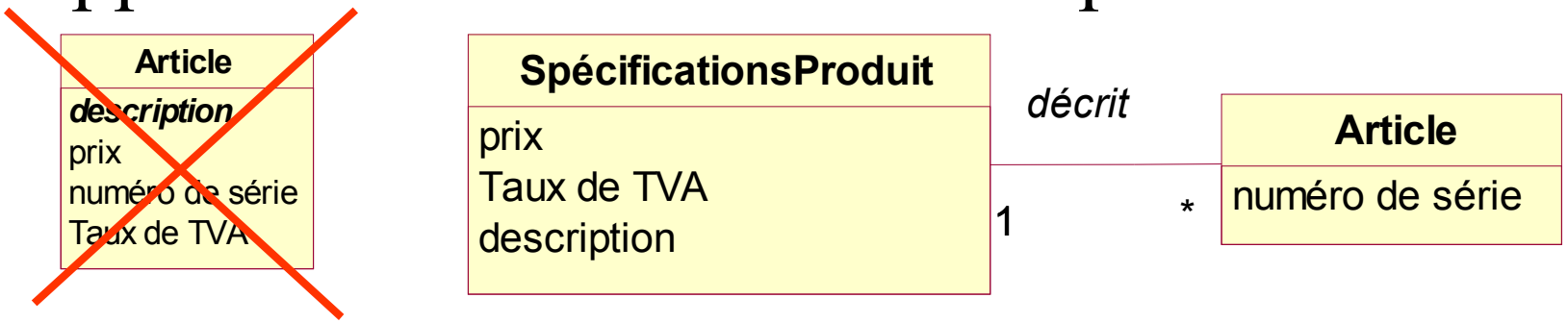
- Le modèle est peut être loin du monde réel quand le système s'applique a un domaine très abstrait (télécom, langue Naturelle, etc.)

Les Descripteurs

- Supposons que :
  - un Article représente un Objet physique contenu dans le magasin
  - un Article a une description, un prix, une tva, qui ne sont pas enregistrés ailleurs
  - Le personnel est amnésique
  - Tous les exemplaires d ’un article donné sont vendu
- Que ce passe t-il quand un client demande le prix de cet article ?

Les Descripteurs

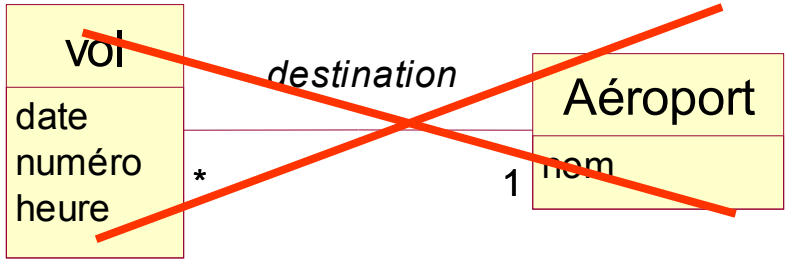
Les descripteurs qui sont des descriptions ou des spécifications d ’autres objets doivent apparaître dans le modèle conceptuel

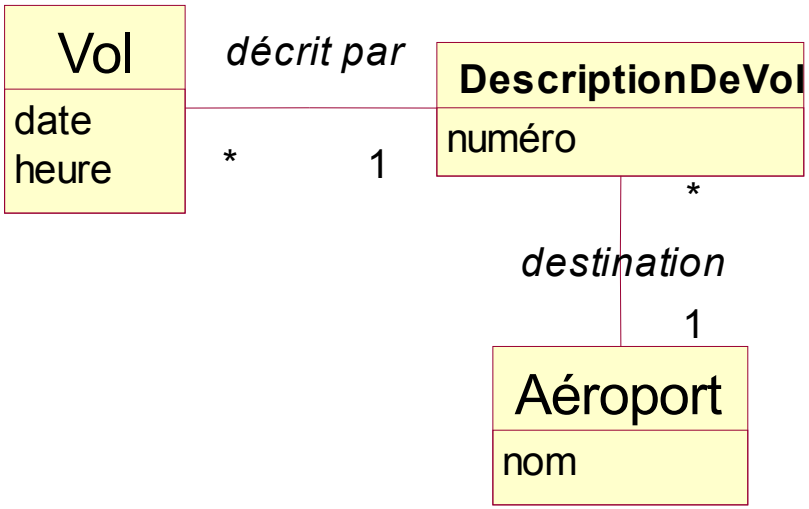


Quand utiliser les descripteurs ?

- Si la destruction de toutes les instances implique une perte d ’information
- Si cela permet de réduire la quantité d ’information redondante

exemple


vol
date
numéro
heure
destination
*
1
Aéroport
nom


Vol
date
heure
décrit par
*
1
DescriptionDeVol
numéro
*
destination
1
Aéroport
nom

Terminologie

- Concept : choses du monde réel
- Classe : description d'un ensemble d 'objets qui partagent les mêmes attributs, méthodes, relations et sémantique.
- Type : une classe sans méthodes
- Interface : ensemble des opérations/méthodes visibles (publiques)
- opération: définie sur un concept
- méthode : définie sur une classe

Deuxième partie les Associations

Objectifs

- Identifier les associations du modèle conceptuel
- Distinguer les deux types d ’association
  - need-to-know (doit-savoir)
  - comprehension-only (pour la compréhension)

Association

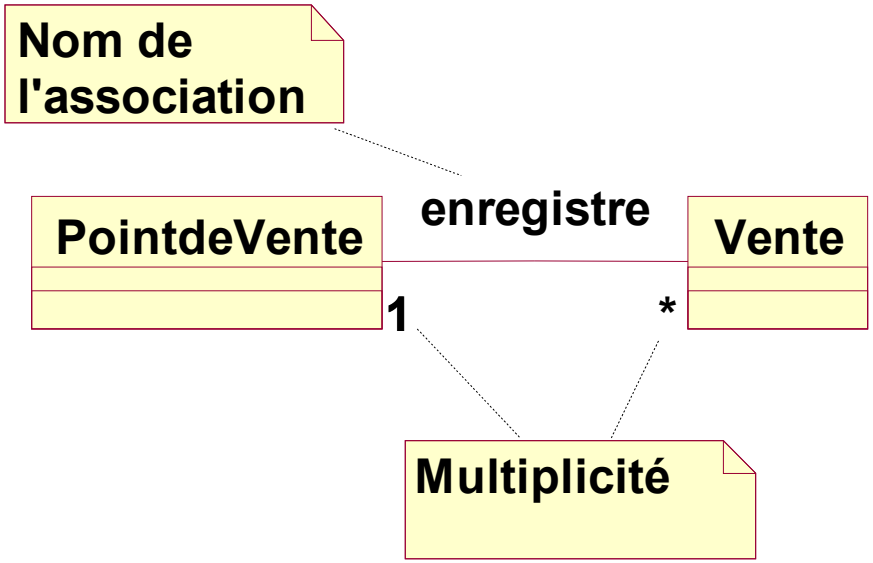
Une association est une relation entre concepts qui met en valeur une connexion intéressante ou significative

Associations utiles

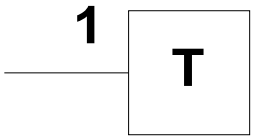
- Si la relation doit exister pendant un temps non nul, elle doit être identifiée

- C ’est une association de la liste des associations

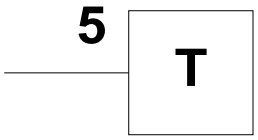
Notation UML



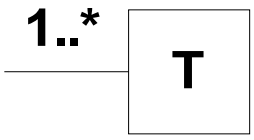
Multiplicité



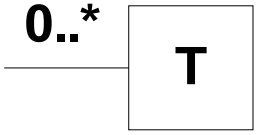
Exactement un



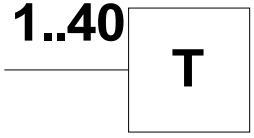
Exactement cinq



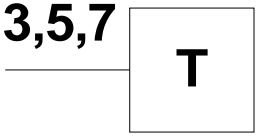
un ou plus



zéro ou plus



un à quarante



Exactement trois, cinq ou sept

Liste des Association les plus communes

- A partie physique de B
- A partie logique de B
- A contenu physiquement dans/sur B
- A contenu logiquement par B
- A est une description de B
- A élément de la transaction B
- A est membre de B
- A partie de l ’organisation B

- A utilise B
- A gère ou dirige B
- A communique avec B
- A est en relation avec une transaction B
- A est connu/enregistré / noté/capturé par B
- A est proche de B
- A est possédé par B
- A est une transaction en relations avec une transaction B

Concepts ou Associations ?

- Le plus important est de trouver les concepts
- On doit investir dans la recherche des concepts plus que dans la recherche des associations
- Identifier les relations « need-to-know »
- La transitivité est implicite ne surcharger

Rôles

- On identifie les deux éléments d ’une association par deux rôles
- Le rôle est défini par
  - un nom
  - une multiplicité
  - une navigabilité
  - une description textuelle si nécessaire

Troisième Partie
Ajouter des Attributs

Identifier les attributs

Identifier de bons attributs

Ce qui est un attribut et ce qui ne l'est pas

Un attribut

- Un attribut est une donnée qui est logiquement un élément constructif d ’un objet.
- On place dans le modèle conceptuel tous les attributs pour lesquels il existe un *besoins de mémoire* (dans les UC par ex)
- Un reçu contient une date et une heure d ’ou les attributs date et heure dans le concept vente.

Notation

KASS = Keep Attributs Simple S…!

Dans un modèle conceptuel il est préférable d ’utiliser des attributs simples, des donnée élémentaires :

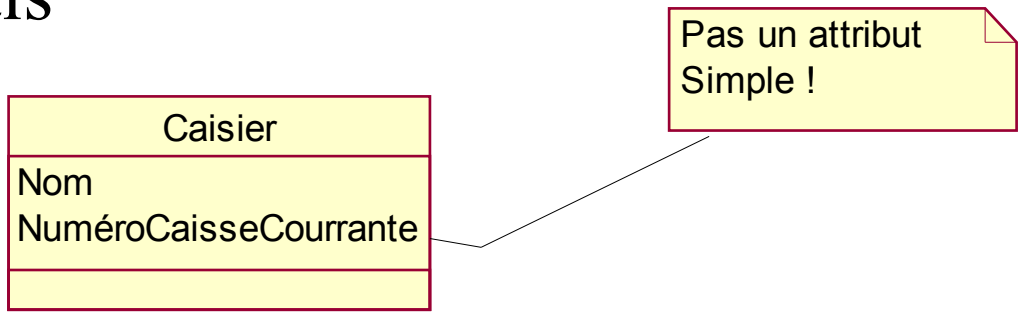
`Bool, Date, Nombre, Chaîne, Heure`

d ’autres exemples :

`Adresse, Couleur, Géométriques( Points, Rectangle), Téléphone, NSS, Code Postal, énumérations`

exemple

Mauvais



Meilleur

Types non Primitifs

- Certains type d ’attributs ne sont pas définis directement dans le langage.
- C ’est le cas pour des attributs comme :
  - Téléphone Nombre composé de plusieurs parties
  - NSS des opérations spéciales de comparaison sont définies dessus
  - Prix promotionnel (dates de début et de fin)
  - Quantité dans une unité
- A chaque fois il y a une information supplémentaire qui n’est pas dans le type primitif que l ’on associe directement à cet attribut.

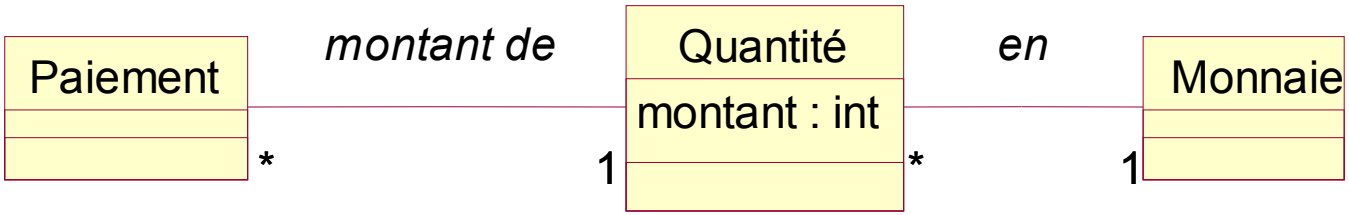
Un choix

- Pour choisir entre un attribut non-primitif et une association, le plus simple est de ce demander que veut on mettre en valeur ? Il n ’y a pas de réponses définitives
- Le Model conceptuel est un outils de communication les choix que l ’on fait doivent toujours prendre ceci en considération.

exemple

Modéliser des quantités et des unités

- Informellement un paiement se modélise avec un entier. Ce n'est généralement pas une solution ni robuste ni flexible car l 'unité dans laquelle est spécifié le nombre est souvent importante.



Approche Objet

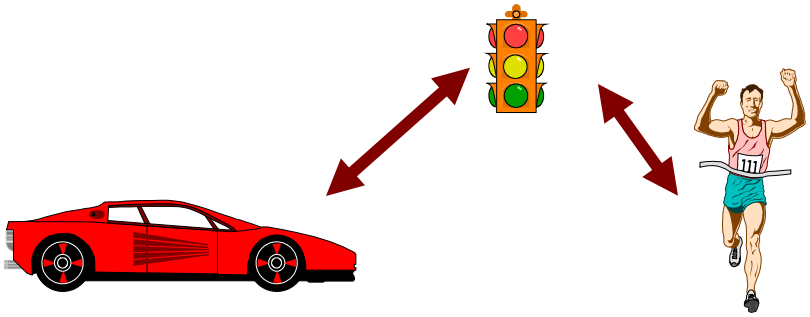
Dominique Revuz
dr@univ-mlv.fr

Pourquoi les Objets

- Plus proche des Utilisateurs
- Plus proche de la réalité
- Plus intuitifs
- Aide à l ’encapsulation
- Aide pour la modularité / décomposition
- Une approche YoYo du développement

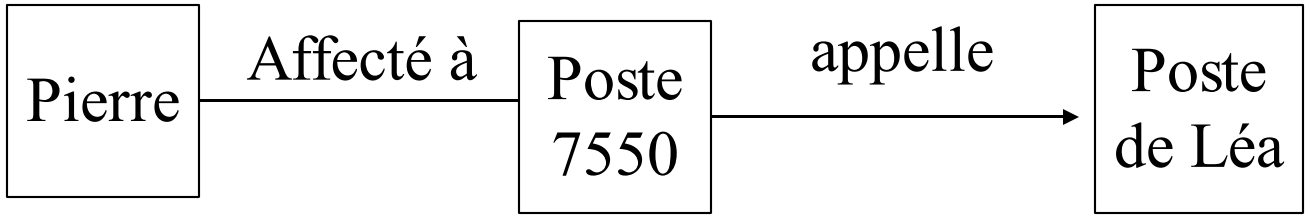
Le paradigme objet

Le monde réel est composé d ’entités autonomes qui sont en relations et coopèrent



Naturel

- Pas de fossé sémantique
- Localisation du sens et du comportement



Objet

Objet n.m. I. (Concret) Chose solide ayant unité et indépendance et répondant à une certaine destination.

II. (Abstrait) Tout ce qui se présente à l ’esprit, qui est occasion ou matière pour l ’activité de l ’esprit.

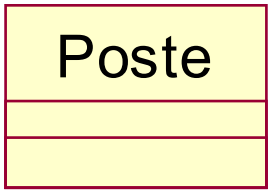
Robert Méthodique

Objet

Concept, abstraction ou chose

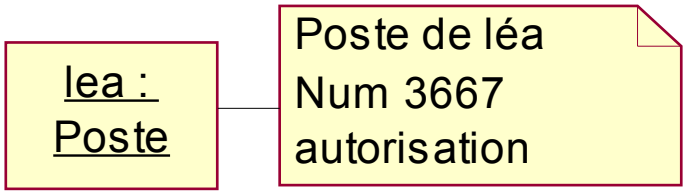
- **Responsabilité**
- **Comportement**
- **Etat**

Classe



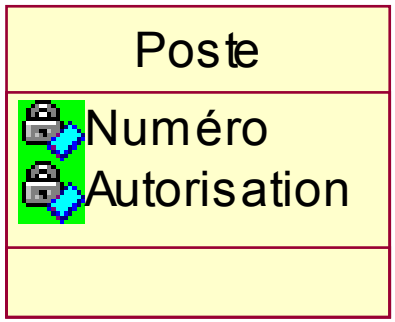
- Une abstraction qui permet de modéliser et/ou décrire un ensemble d ’objet d ’une façon générale.
- La classe décrit et/ou explicite
  - l ’espace de variation de l ’état
  - le comportement des instances
  - le rôle des instances

Instances



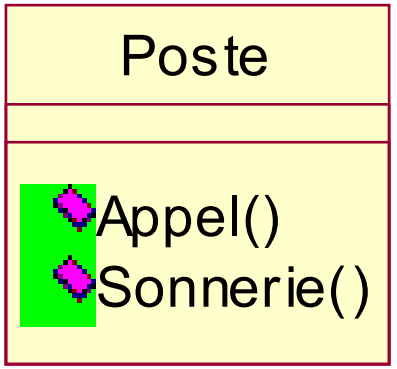
- Les instances d ’une classe sont les objets qui vérifient les contraintes définies par la classe.
- On décrit parfois la classe comme une usine qui fabrique des instances.
- Les instances sont différentes (Etat) mais ont toutes le même comportement.

Etat (Attributs)



- Caractéristiques instantanées d ’un Objet = ensemble des valeurs des attributs.
- Toutes les instances ont les mêmes attributs avec des valeurs variables, unique pour chaque instance.

Interface: Opérations&Messages



- Service offert par les Instances d ’une classe
- L ’interface ce définie en fonction de la responsabilité de la classe.
- C ’est la classe qui définie l ’interface des instances.
- Le concept de message s ’applique dans les cas asynchrones

Encapsulation

- Cacher le fonctionnement interne d ’un objet ou d ’un module
- Les changements interne du module n ’ont pas d ’influence sur le code extérieur à l ’objet.
- On cherche à obtenir des interfaces au sens de la chimie.

Interface

- Quand deux matériaux on une interface cela veut dire qu ’il n ’y pas d ’échange de matière entre les deux.
- exemple l ’huile et l ’eau
- contre exemple l ’or et l ’argent

Interface (2)

- On cherche à écrire des modules qui ont des interface.
- On veut que ni les corrections ni les bugs ne se propage d ’un module à l ’autre.
- En particulier les changements d ’implémentation d ’un module de doivent pas avoir d ’incidence sur les modules clients.

Module

- Un module est une unité de code
- De toute taille : classe, fichier, paquetage
- On parle de module quand l ’unité de code à une forte cohésion.
- Ex: string, stdio, vector<T>

Module ouvert

- Un module dans lequel il est encore possible de réaliser des modifications.
- Ajout d ’attributs, changement du code, ajout d ’une classe, etc.

- C ’est ce que demande le concepteur du module qui veut pouvoir : le corriger, l ’étendre, l ’adapter.

Module Fermé

- Un module utilisable par d ’autres.
- Compilable, validé, testé
- La valeur d ’un module est plus grande si il est fermé
  - avancement
  - debugging
  - réutilisation.

Un rêve Ouvert **ET** Fermé

- A la fois sécurisé et extensibles.
- Deux approche pour les modules O&F
- Interface
  - On ferme l ’interface pas l ’implémentation
- Héritage
  - On créer un nouveau module qui ne fait qu ’étendre le module précédant sans le changer. (les LOO offrent une syntaxe)

Module client

- Quand deux modules interagissent celui qui utilise l ’autre est appelé module client.
- Ex: Voiture/Conducteur
- Le conducteur utilise l ’interface de la voiture mais pas l ’inverse.
- Ex: Administration/Administré les deux modules font des demandes croisé. Les deux modules sont clients l’un de l’autre

Interface et Module O&F

- L ’interface est la partie publique d ’un module.
- Ce doit être la partie stable d ’un module.

- Quand un module utilise l ’interface d ’un autre module on dit qu ’il est un module client.

- Si l ’interface change les modules clients doivent changer.

- Il faut concentrer une bonne partie du travail de conception d ’un logiciel sur la définition de l ’interface des différents modules.
- Ex: Interface des Entrées/Sorties sous UNIX, en C , en C++, en Java.

Implémentation

- Un module est composé d ’une interface et d ’une implémentation.
- L ’implémentation d ’un module est le code qui réalise les fonctionnalités du module.
- En termes du C on a le « .h » et le « .c »
- En C++ et Java l ’interface est l ’ensemble des signatures de Méthode.
  L ’implémentation le code de celle-ci.

Polymorphisme

- Généralisation d ’une interface.
- On veut que le module client non seulement ne connaisse pas l ’implémentation d ’un module mais qu’il ne connaisse pas effectivement le type réel du module qu’il utilise.

Polymorphisme (2)

- Le polymorphisme est un mécanisme syntaxique qui permet à un module client d ’utiliser une interface générique sans connaître le module qui répondra effectivement à ces requêtes (messages, appels de méthodes).

Polymorphisme (3)

- Sous unix le système offre un service polymorphe d ’accès aux périphériques d ’entrée/sortie.
- Il est en effet possible d ’écrire un programme réalisant des E/S sans connaître le type de périphérique que l ’on vas utiliser effectivement (écran, disque, imprimante, tube, socket, mémoire, clavier, etc.) .

Interface

- Choisir une bonne interface est essentiel
- Risque à l ’ouverture
- compréhensibilité
- réutilisabilité
- Continuité $(s<\varepsilon \Rightarrow f(s)<\varepsilon)$

Construction des Modules

- Unités linguistiques
- peu d ’interfaces
- petites interfaces
- interfaces explicites
- encapsulation

Unités linguistique

- Les modules doivent correspondre à des unités syntaxiques du langage.
- Classe
- Paquetage
- Librairie

Peu d ’interface

- IL faut réduire le nombre d ’interface qu’utilise un module.
- Un module doit être le client d ’un minimum de module.
- + de client + de chance d ’avoir a changer
- + de client + grande complexité

Petites Interface

- Les interface pléthoriques vont à l'encontre de la compréhensibilité du module
- Une interface importante n 'aide pas à la réutilisabilité du module.
- De plus si l 'interface est grande le module doit être complexe

Interfaces explicites

- Les modules doivent avoir une interface explicite ou rien n ’est cacher.
- La documentation du module

Diagrammes de séquence

Séquence

Interaction

Collaboration

Le Comportement du Système

- Objectifs
  - Identifier les opérations et les événements du système
  - Créer les Diagrammes de séquence des cas d ’utilisation
- Les diagrammes de interaction montrent les interactions entre les acteurs et le système et entre les composants du système.

Activités + dépendances

- La création de diagrammes de séquences est une étape d'analyse :
  - Les cas d 'utilisation doivent être développés préalablement.
  - La création d 'un diagramme de séquence ce fait avec la création des Cas d 'utilisation détaillés

Comportement du Système

- Avant de se lancer dans la conception logique du système, il est nécessaire de définir son comportement comme une boite noire (vue des Acteurs).
- Le Comportement du système = est une description de ce que le système fait sans explication de comment il le fait.
- Le diagramme de séquence fait la base de cette description.

Les diagrammes de séquence

- Pendant l ’interaction de l ’acteur avec le système il produit des événements qui nécessitent des réponses du système.
- Exemple: quand un numéro d ’article est tapé par le caissier le système doit enregistrer la ligne de vente etc. .
- Il est désirable d’identifier, d’isoler et d’illustrer les opérations qui sont demandés par un acteur au système, car elles sont une part importante de la compréhension du système.

Un Diagramme de Séquence

- Un diagramme de séquence est un dessin qui montre pour un scénario particulier d'un cas d 'utilisation, les événements produits par les acteurs externes, leurs ordre, et les événements inter-système.
- Les événements les plus importants sont ceux qui **traversent** la frontière du système.

Le diagramme de séquence du système doit être fait pour le scénario principal du cas d'utilisation et pour les scénarios alternatifs les plus intéressants.

Le formalisme UML

Diagramme de cas d'utilisation

Diagramme de séquences

Diagramme de classes

Diagramme d'états - transitions

Diagramme d'activités

Diagramme de composants

Diagramme de déploiement

Diagramme de séquences

- Définition
- Objet
- Message
- Description avec Rose

Diagramme de séquences
Définition

- Un diagramme de séquence décrit une interaction particulière entre des objets et des acteurs du système
- Il s'agit d'une séquence d'événements ou de messages échangés pendant le déroulement d'un scénario
  - Scénario = instance d'un cas d'utilisation
  - Scénario principal
  - Scénarios secondaires

Diagramme de séquences

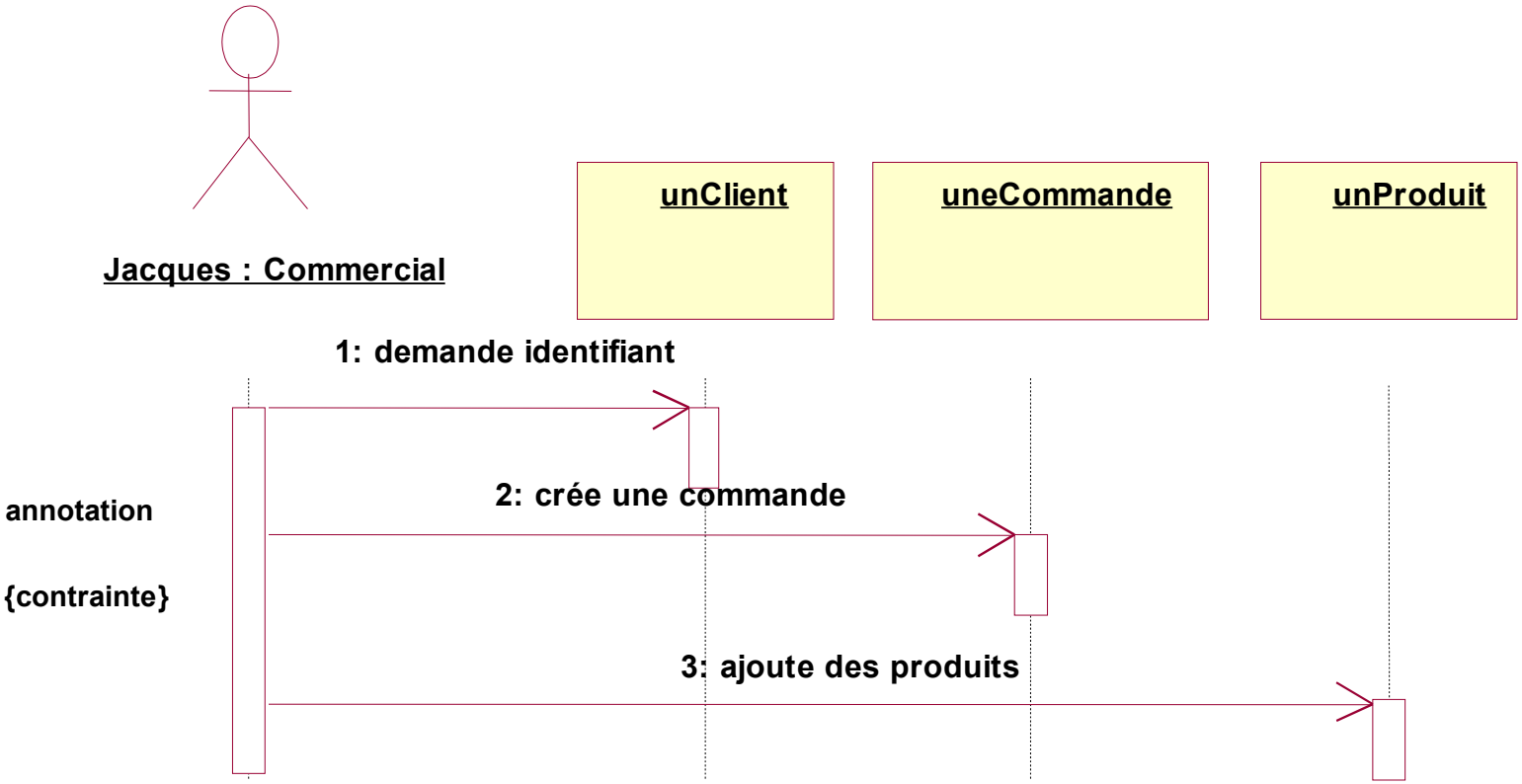


Diagramme de séquences

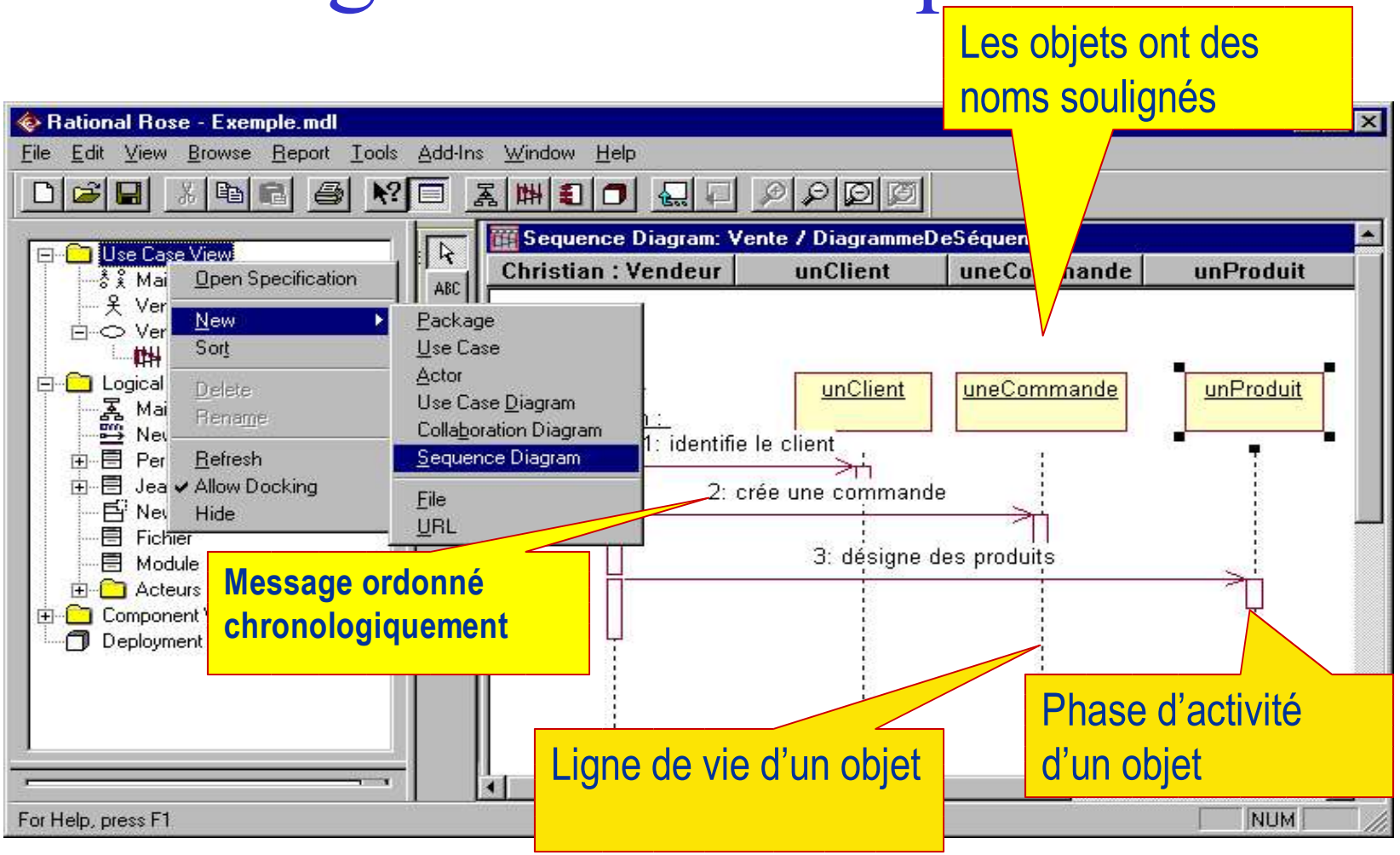


Diagramme de séquences

- Un objet est symbolisé par un rectangle et une barre verticale appelée ligne de vie de l'objet
  - Un nom souligné dans le rectangle désigne l'objet et la classe
  - La ligne de vie est un axe chronologique
  - L'activité de l'objet est caractérisée par un rectangle sur la ligne de vie

Diagramme de séquences

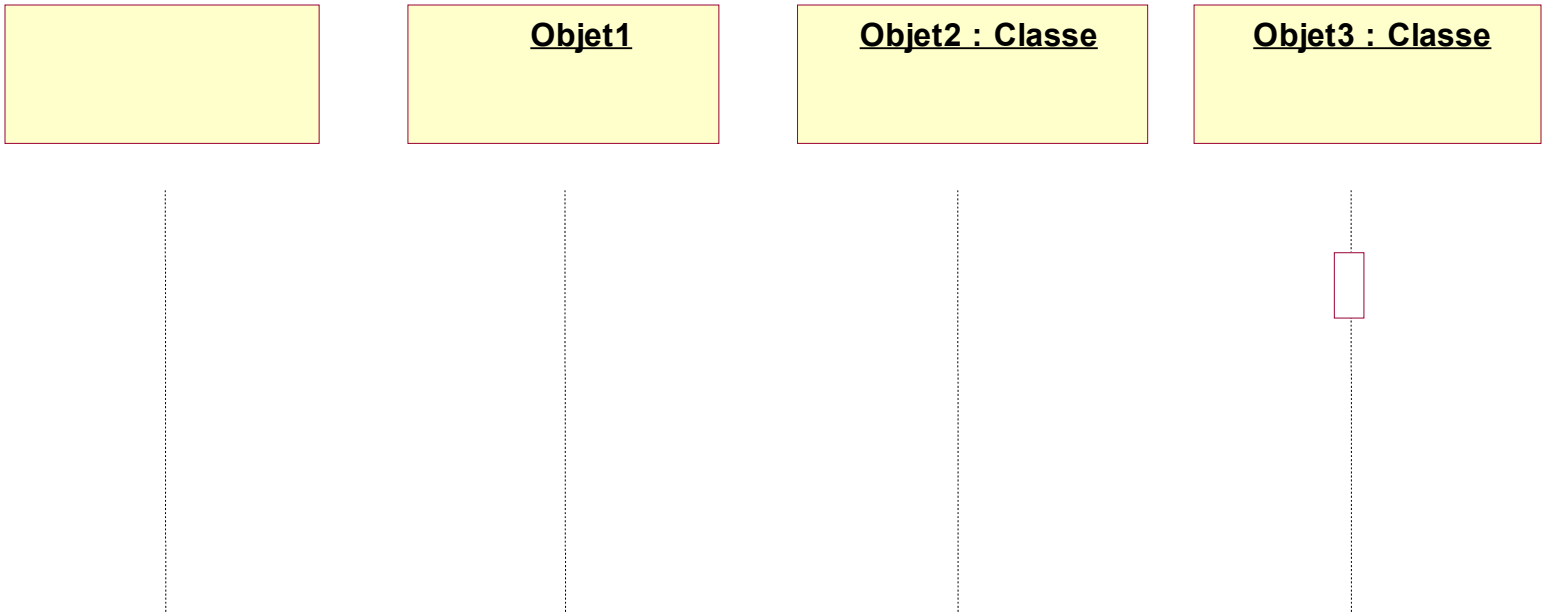


Diagramme de séquences

- La spécification d'un objet est constituée d'un texte et des propriétés
  - Class
  - Persistent
  - Static
  - Transcient
  - Multiple instances

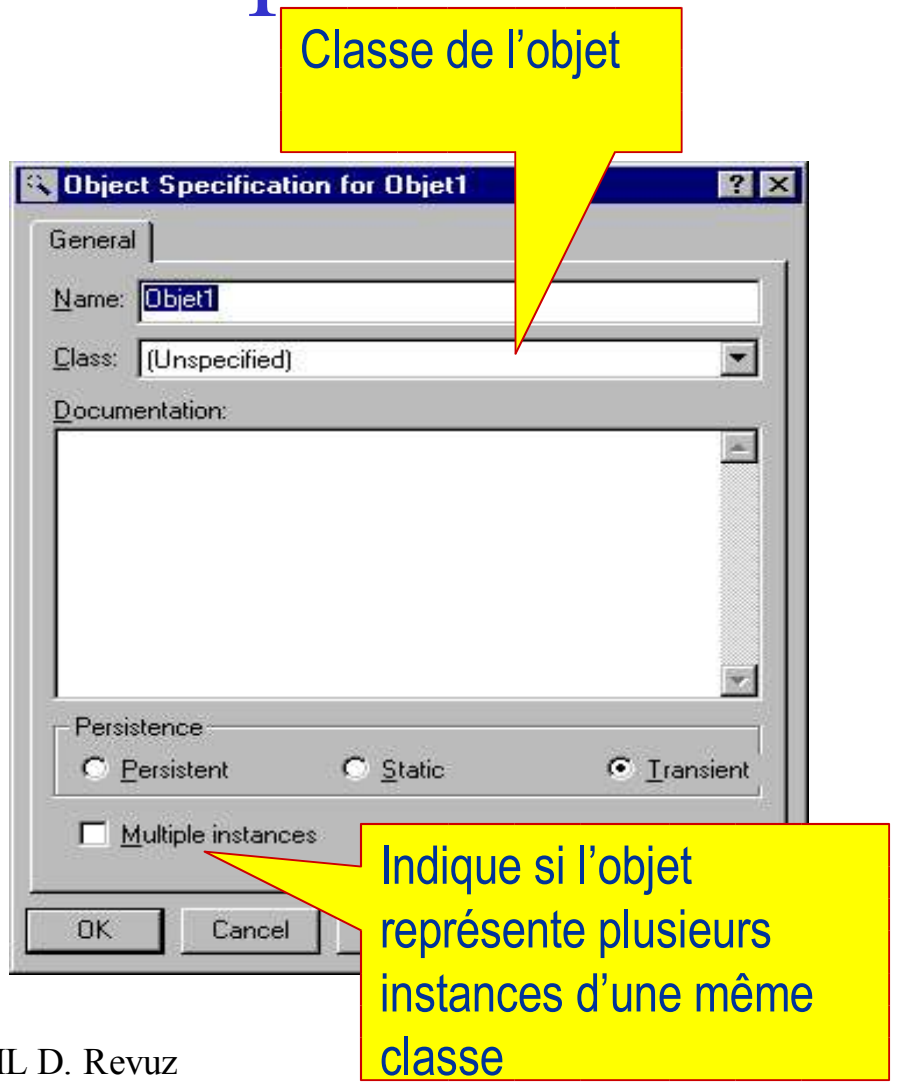


Diagramme de séquences

- **Un message entraîne un changement dans le système**
- **Un message est symbolisé par une flèche orientée de l'émetteur vers le destinataire**
- **La position sur la ligne de vie d'un objet ou un numéro donne l'ordre des messages**
- **Le rectangle représente l'objet qui détient le contrôle**

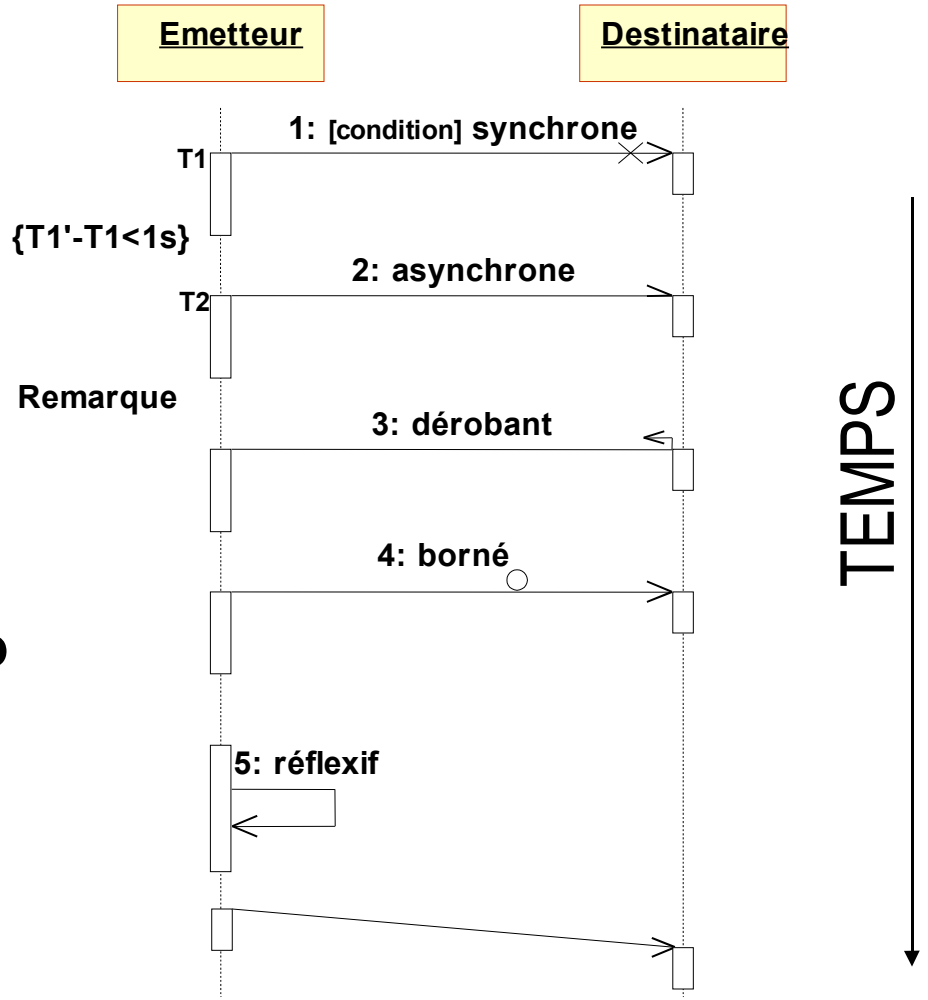


Diagramme de séquences

- Les annotations permettent de représenter toute forme d'interaction en utilisant un langage structuré
- L'envoi d'un message conditionnel est représenté par une expression entre crochets [ ] placée devant le message
- Le diagramme peut être annoté par des information textuelles, contraintes, structures

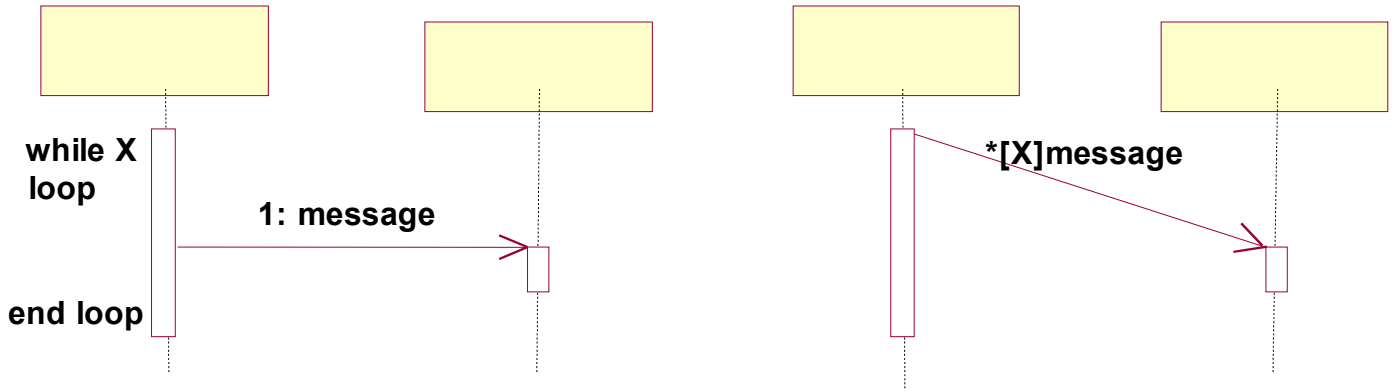
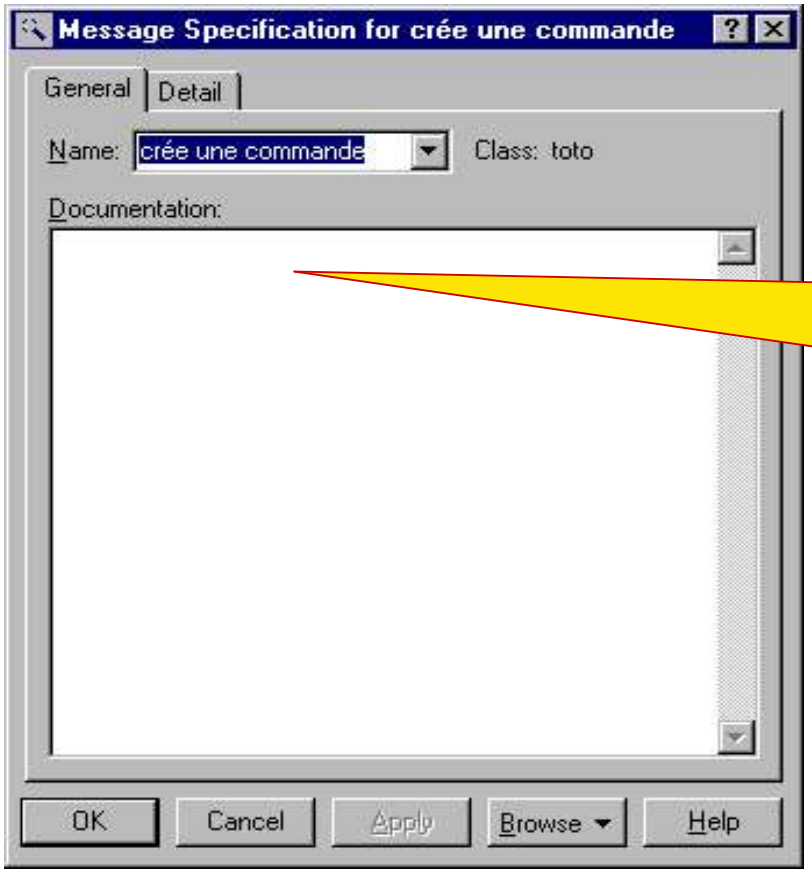
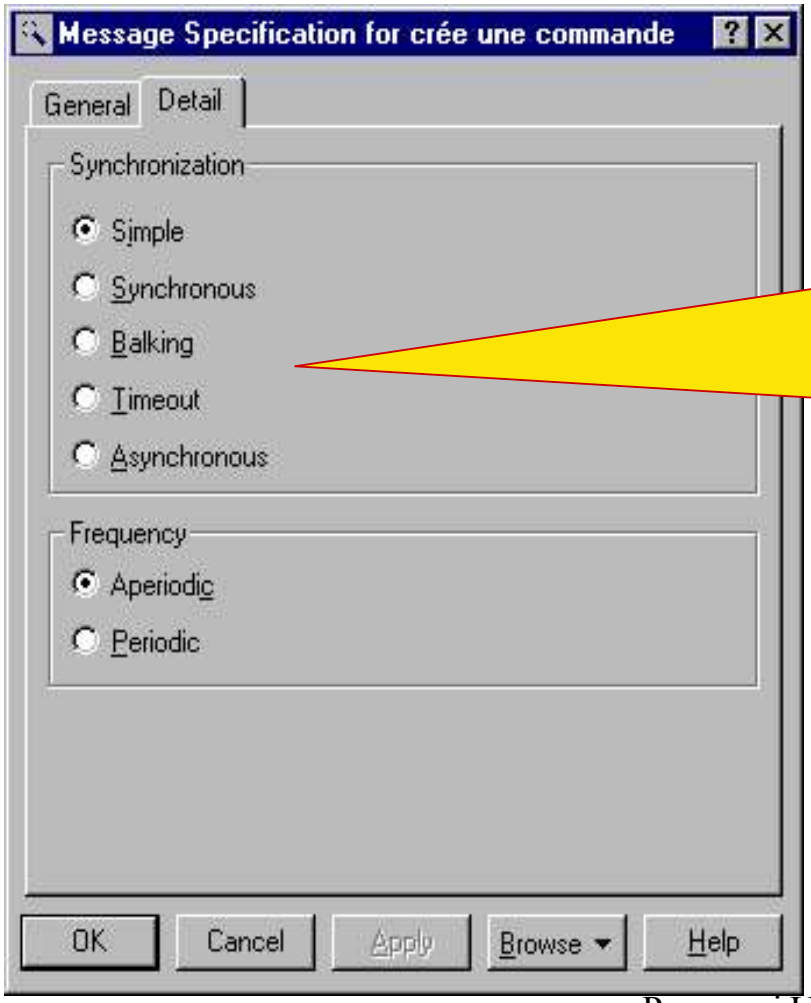


Diagramme de séquences



La documentation comporte des indications sur les rôles, les clés, les contraintes, les principaux comportements

Diagramme de séquences



Type de message

simple

synchrone

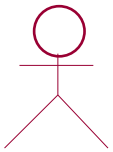
dérobant

borné

asynchrone

exemple

UC : Achat Article

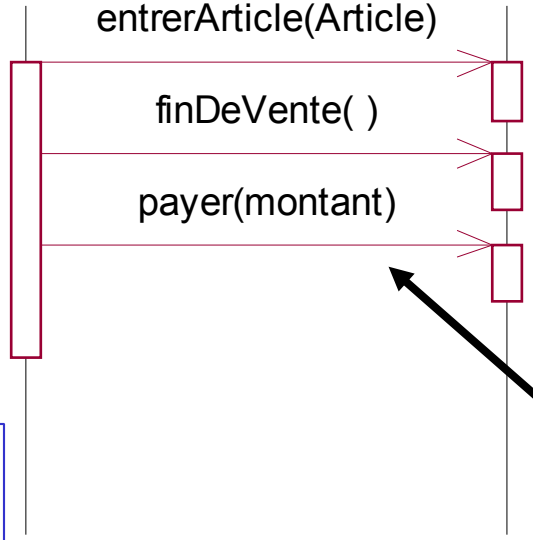


Client : Acteur

Système : Système

Boite Noire !

```
Répéter jusqu'à la
fin des articles =>
```



L'interaction acteur/système est décrite ici.

Texte qui clarifie la logique de contrôle, on peut prendre cette information directement dans la description du cas d 'utilisation.

Evénement système qui déclenche une opération du système

Diagramme de séquences
Exercice

- Faire un diagramme de séquence pour un scénario d'un cas d'utilisation :-)
- DUR DUR
- Bien donc exo1 : pour le TP tout à l 'heure
- faire le DdS de « Achat article »