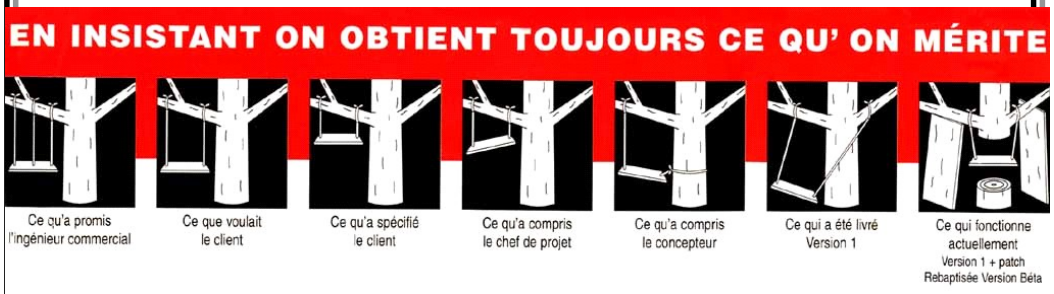
- *Introduction*
- *Diagramme de cas d'utilisation*
- *Fiche descriptive + Diagramme de séquence système*
- *Conception objet préliminaire*
- *Diagramme d'activité*

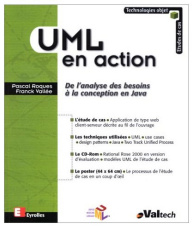
Méthodologie de la production d'applications

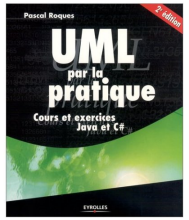


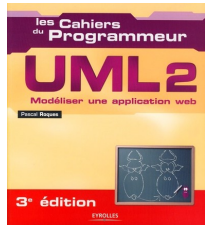
Modélisation objet

BIBLIOGRAPHIE

- UML en action, P. Roques, F. Vallée, Valtech, Eyrolles, 2003, ISBN 2-212-11213-0.
- UML par la pratique, P. Roques, Eyrolles, 2001, ISBN 2-212-11246-7.
- UML 2, Modéliser une application Web, P. Roques, Eyrolles, 3ème édition, 2007, ISBN 978-2-212-12136-0.

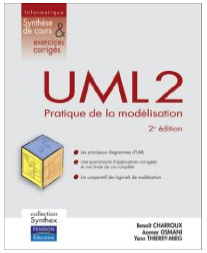


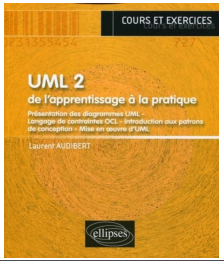




BIBLIOGRAPHIE

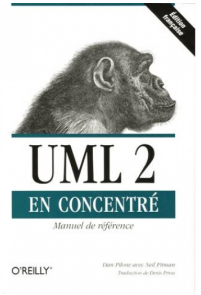
- UML2, Pratique de la modélisation, Benoît Charroux, Aomar Osmani, Yann Thierry-Mieg, 2ème édition, collection Synthex, Pearson Education, 2008, ISBN 2-7440-7287-1.
- UML2 de l'apprentissage à la pratique, Laurent Audibert, Ellipses Edition Marketing S.A., 2009, ISBN 978-2-7298-5269-6.





BIBLIOGRAPHIE

- UML2 en concentré, Dan Pilone avec Neil Pitman, O'Reilly, 2006, ISBN 2-84177-373-6.

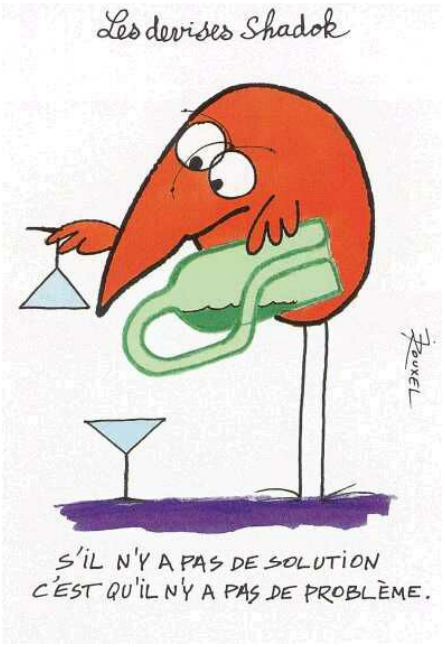


BIBLIOGRAPHIE

- **Tous les documents sur UML élaborés dans le cadre de l'OMG sont publics et disponibles sur le site :**

www.uml.org

INTRODUCTION



UML, un langage pour :

- Décrire, visualiser et comprendre le problème,
- Capturer, communiquer et utiliser des connaissances pour la résolution du problème,
- Spécifier, montrer et construire la solution,
- Documenter la solution,
- Quel que soit le domaine d'application de l'informatique.

LES BASES D'UML

Unified Modeling Language

- Aujourd'hui, le **standard industriel de la modélisation objet** est **UML**.

Il est sous l'entière responsabilité de l'OMG (*Object Management Group*).

- L'OMG est une association américaine à but non lucratif créée en 1989 (*groupement d'industriels*) dont l'objectif est la standardisation autour des technologies objet, afin de garantir l'interopérabilité des développements.

LES BASES D'UML

L'**OMG** comprend actuellement plus de *850* membres :

- les principaux acteurs de l'industrie informatique (*Sun, IBM, Microsoft, …*),
- les plus grandes entreprises utilisatrices dans tous les secteurs d'activité.

http://www.omg.org

LES BASES D'UML

UML (*Langage de Modélisation Unifiée*) est un langage de modélisation graphique et textuel destiné à :

- comprendre et décrire des besoins,
- spécifier et documenter des systèmes,
- esquisser des architectures logicielles,
- concevoir des solutions et communiquer des points de vue.

UML est un langage utilisable quelque soit :

- le type de système, logiciel, matériel, d'organisation,
- le type de métiers (*gestion , ingénierie*….)

LES 3 AMIGOS

James Rumbaugh

OMT

Grady Booch

Booch

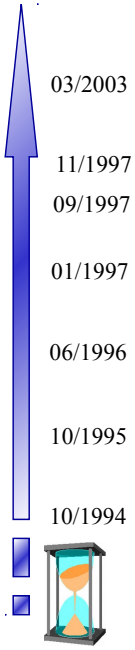
Ivar Jacobson

OOSE

Unified Method 0.8 1995

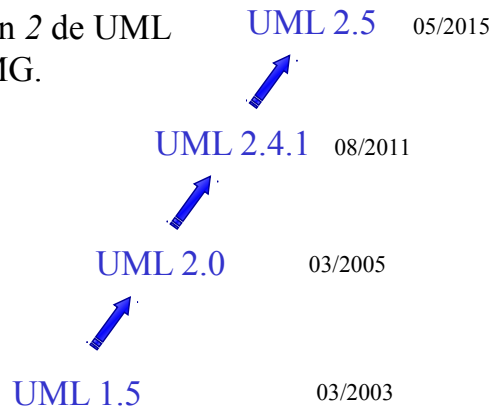
UML 0.9 1996

HISTORIQUE



HISTORIQUE

- *1994* : *50* méthodes objet différentes
- Juillet *2005* : première version *2* de UML validée par l'OMG.
- Août *2011* : version *2.4.1*
- Version *2.5* en *juin 2015*



LES BASES D'UML

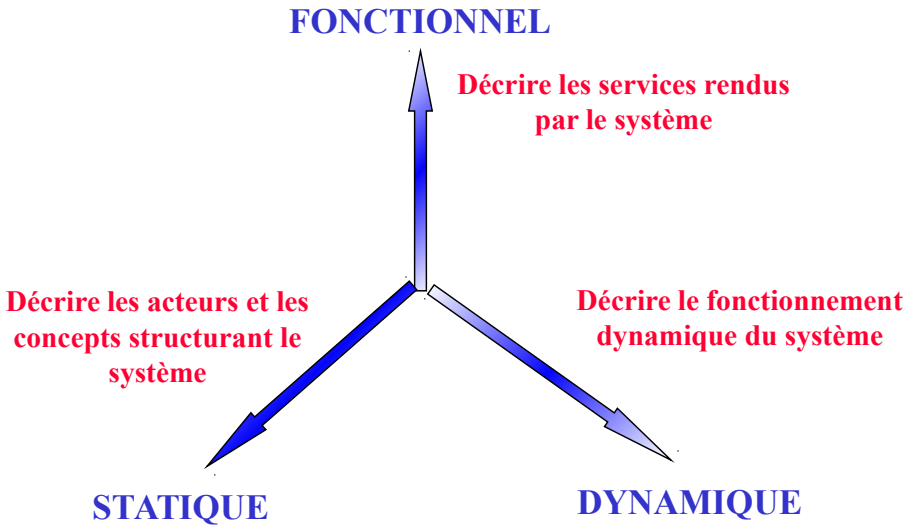
UML s'articule autour de **plusieurs types de diagrammes**, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel.

UML modélise le système suivant **trois modes de représentation** :

- le premier décrit les **services fonctionnels** rendus par le système,
- le second concerne la **structure statique** du système,
- le troisième concerne sa **dynamique de fonctionnement**.

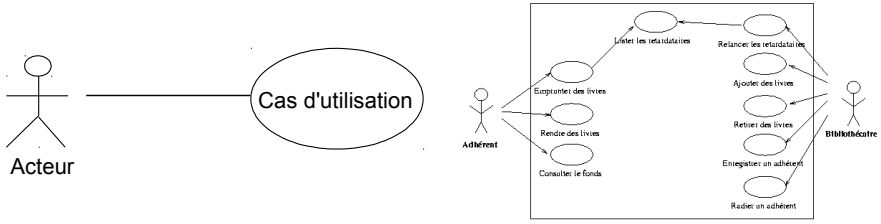
Les trois représentations sont **nécessaires** et **complémentaires** pour schématiser la façon dont est composé le système et comment ses composantes fonctionnent entre elles.

LES BASES D'UML



REPRÉSENTATION FONCTIONNELLE

Le mode de **représentation fonctionnel** s'appuie exclusivement sur le **diagramme de cas d'utilisation**.



Le **diagramme de cas d'utilisation** représente la structure des fonctionnalités nécessaires aux utilisateurs du système.

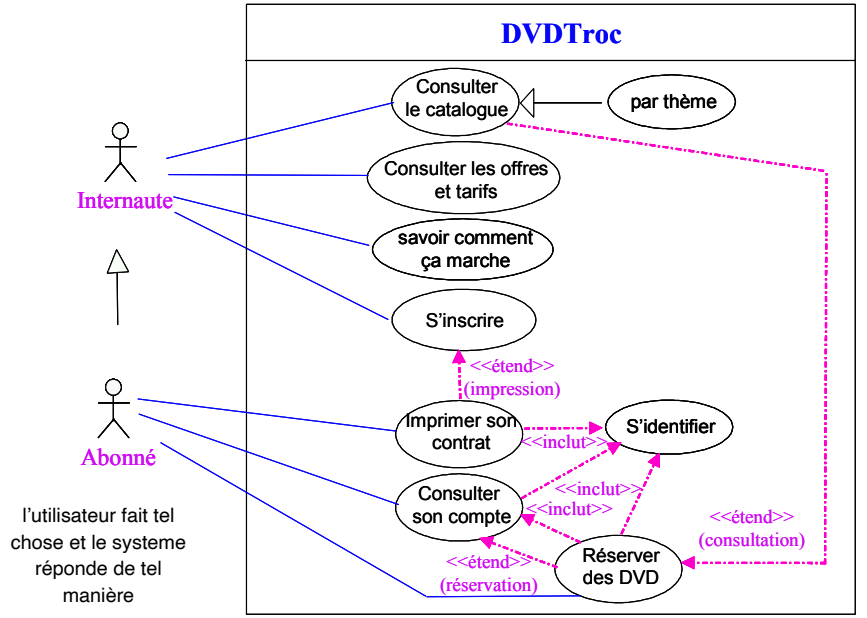
Il est utilisé dans l'activité de spécification des besoins.

REPRÉSENTATION FONCTIONNELLE

- Le diagramme de cas d'utilisation constitue la première étape UML d'analyse d'un système.

- **Il permet de modéliser les besoins des utilisateurs en identifiant les grandes fonctionnalités du système et en représentant les interactions fonctionnelles entre les acteurs et ces fonctionnalités.**

Exemple de diagramme de cas d'utilisation



REPRÉSENTATION STATIQUE

Le mode de **représentation statique** ou **structurel** s'appuie sur *7* diagrammes :

1- diagramme de classes
2- diagramme d'objets
3- diagramme de composants
4- diagramme de déploiement
5- diagramme de paquets
6- diagramme de structure composite
7- diagramme de profils

REPRÉSENTATION STATIQUE

1- diagramme de classes : point central dans un développement orienté objet.

✓ En analyse : décrit la structure des entités manipulées par les utilisateurs.

✓ En conception : représente la structure d'un code orienté objet.

REPRÉSENTATION STATIQUE

2- diagramme d'objets : illustre des structures de classes compliquées.

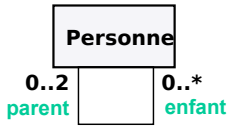
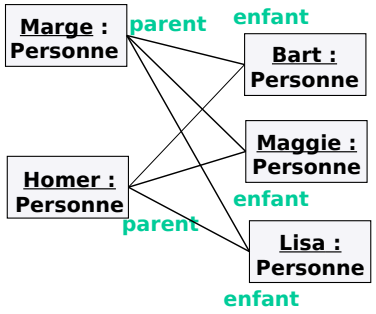


diagramme de classes



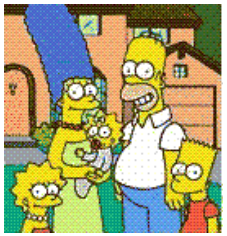


diagramme d'objets

REPRÉSENTATION STATIQUE

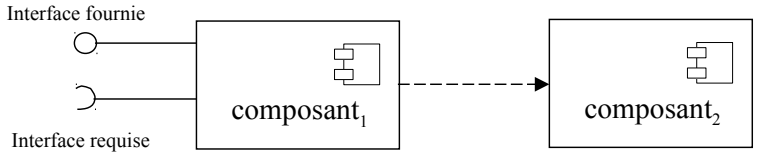
3- diagramme de composants : représente les concepts de configuration logicielle.

Il montre comment s'agencent des composants tels que :

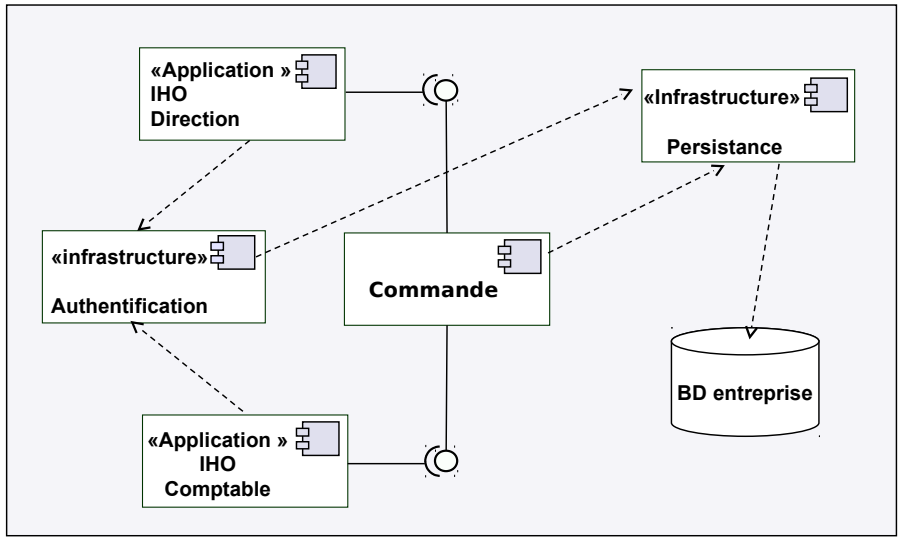
- ✓ les fichiers source,
- ✓ les paquetages de code ou les bibliothèques
- ✓ les bases de données

avec leur interface fournie et requise.

Il met en évidence les dépendances entre les composants (*qui utilise quoi*) et ainsi de mieux organiser les modules.



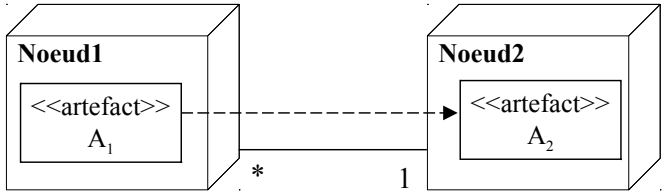
Exemple de diagramme de composants



REPRÉSENTATION STATIQUE

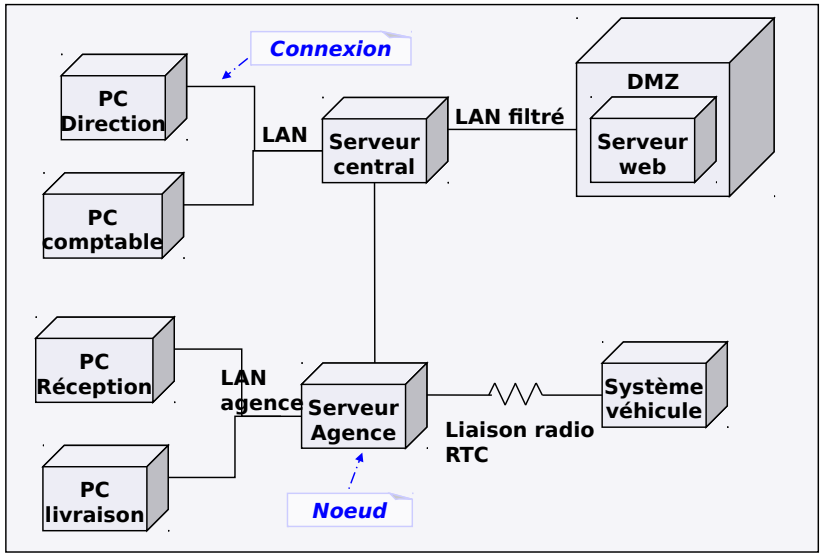
4- diagramme de déploiement montre la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels.

Les ressources matérielles sont représentées sous forme de noeuds. Les noeuds sont connectés entre eux, à l'aide d'un support de communication. La nature des lignes de communication et leurs caractéristiques peuvent être précisées.



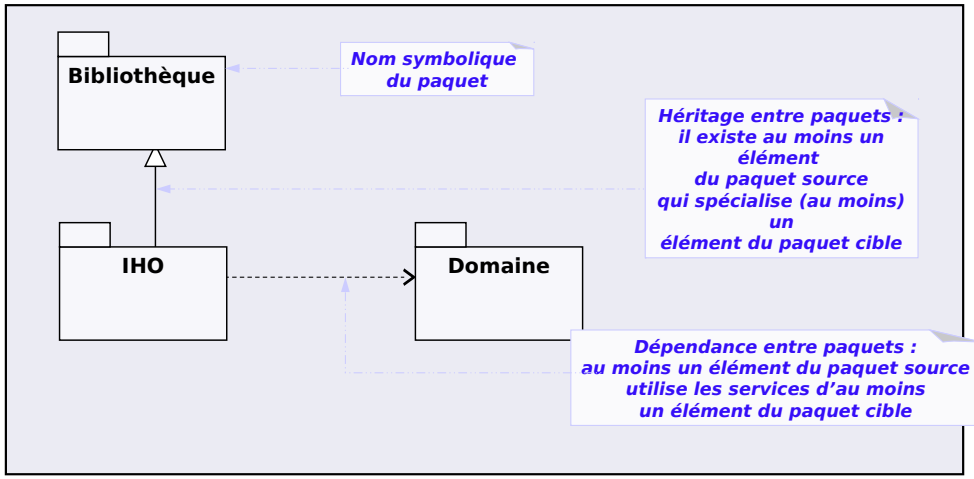
artefact : fichiers, exécutables, bases de données ...

Exemple de diagramme de déploiement



REPRÉSENTATION STATIQUE

5- diagramme de paquets montre l'organisation logique du modèle et les relations entre paquets.

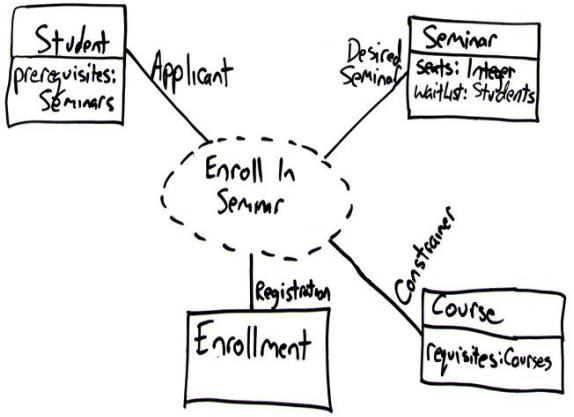


REPRÉSENTATION STATIQUE

6- diagramme de structure composite permet d'établir des liens entre les diagrammes de classes et les diagrammes de composants. Il montre comment les éléments d'un système se combinent pour former des composants plus complexes.

REPRÉSENTATION STATIQUE

6- diagramme de structure composite montre l'organisation interne d'un élément statique complexe sous forme d'un assemblage de parties, de connecteurs et de ports.



REPRÉSENTATION STATIQUE

7- diagramme de profils permet de spécialiser, de personnaliser pour un domaine particulier un meta-modèle de référence d'UML. (*depuis UML 2.2*)

REPRÉSENTATION DYNAMIQUE

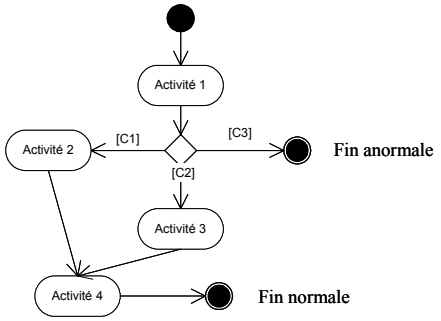
Le mode de **représentation dynamique** ou **comportemental** s'appuie *6* diagrammes :

1- diagramme d'activité
2- diagramme d'états-transitions
3- diagramme de séquence
4- diagramme de communication ou collaboration
5- diagramme global d'interaction
6- diagramme de temps ou de chronométrage

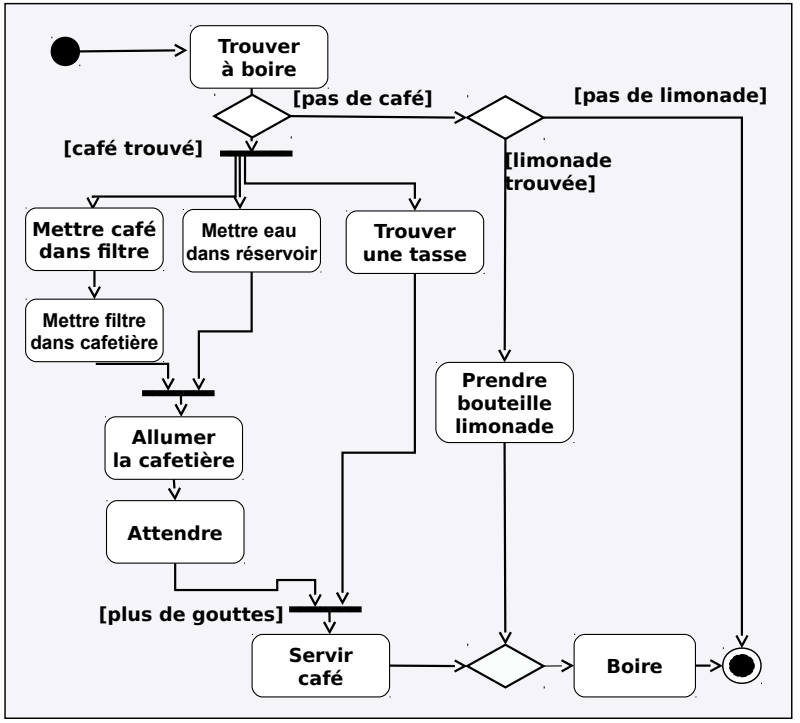
REPRÉSENTATION DYNAMIQUE

1- diagramme d'activité représente les règles d'enchaînement des activités dans le système.
Il permet :

- de consolider la spécification d'un cas d'utilisation
- de concevoir une méthode.
- de décrire également la navigation dans un site web.

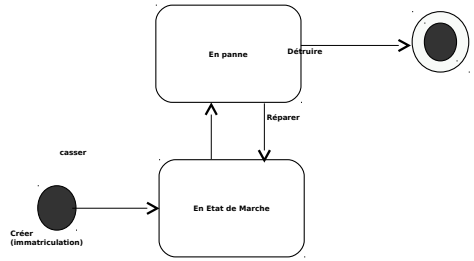


Exemple de diagramme d'activité



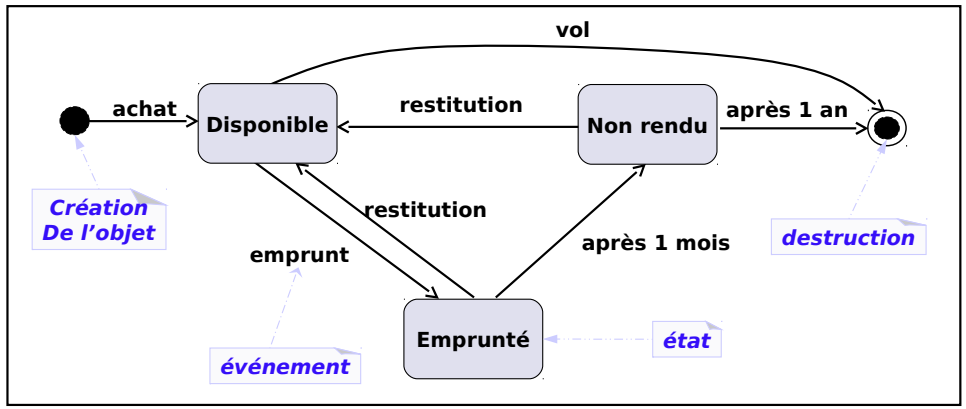
REPRÉSENTATION DYNAMIQUE

2- diagramme d'états-transitions représente le cycle de vie commun aux objets d'une même classe.
Ce diagramme complète la connaissance des classes en analyse et en conception en montrant les différents états et transitions possibles des objets d'une classe à l'exécution.



Exemple de diagramme d'états

Bibliothèque



REPRÉSENTATION DYNAMIQUE

3-4- diagrammes d'interaction : les **diagrammes de séquence** et les **diagrammes de communication ou collaboration**

✓ Ils représentent les échanges de messages entre objets, dans le cadre d'un fonctionnement particulier du système.

✓ Les diagrammes de séquence servent d'abord à développer en analyse les scénarios d'utilisation du système.

✓ Plus tard, les diagrammes de collaboration permettent de concevoir les méthodes.

Exemple de diagramme de séquences

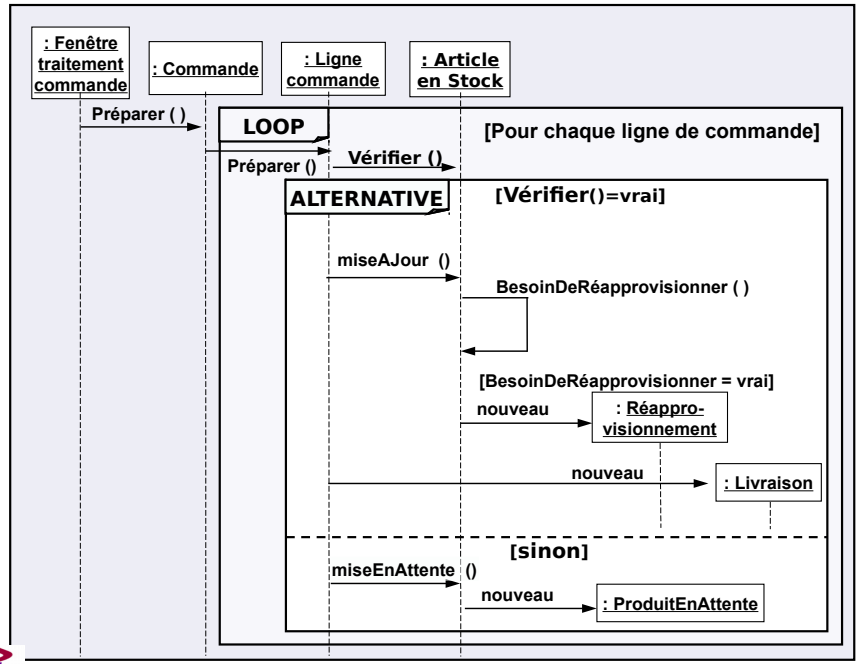
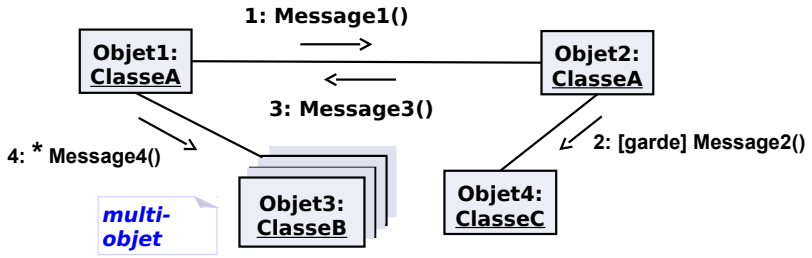
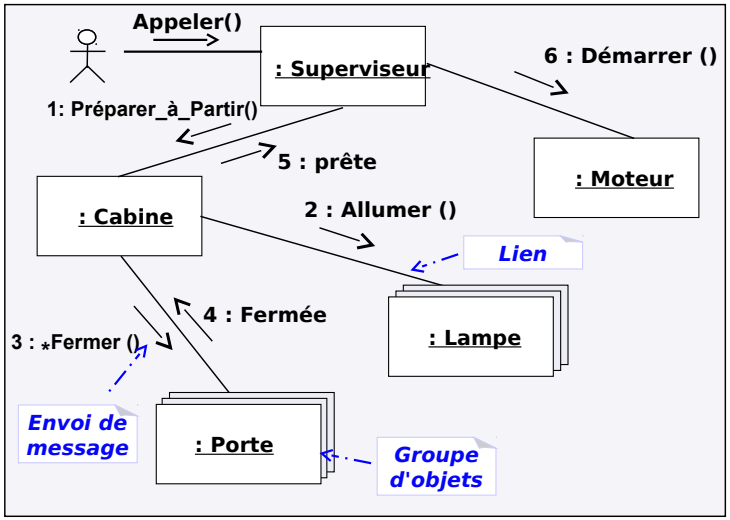


Diagramme de communication ou collaboration



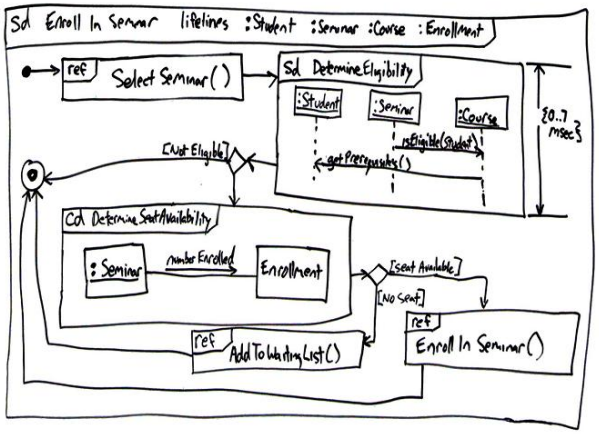
Exemple de diagramme de communication

Appel ascenseur



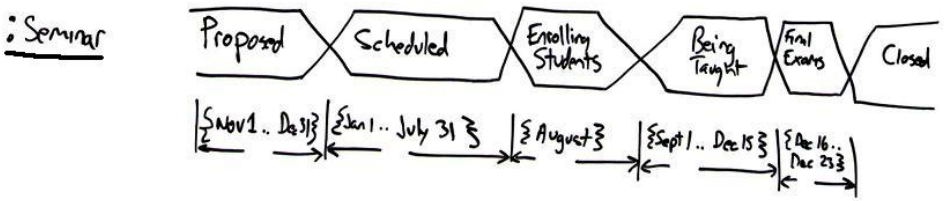
REPRÉSENTATION DYNAMIQUE

5- diagrammes global d'interaction : fusionne les diagrammes d'activités et de séquences pour combiner des fragments d'interaction avec des décisions et des flots.



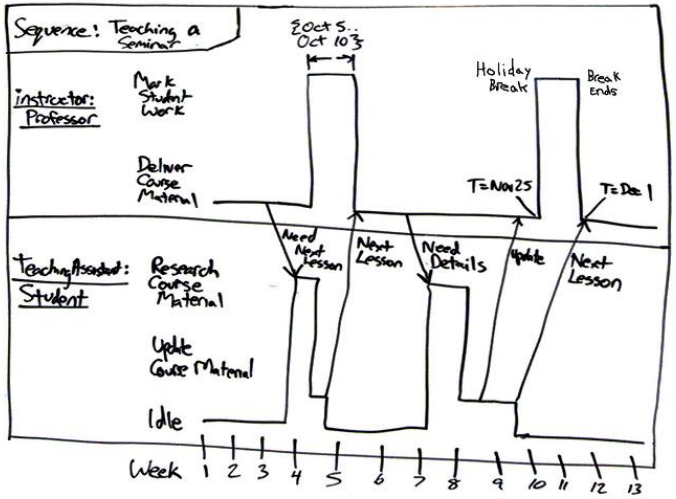
REPRÉSENTATION DYNAMIQUE

6- diagrammes de temps : fusionne les diagrammes d'états et de séquences pour montrer l'évolution de l'état d'un objet au cours du temps.



REPRÉSENTATION DYNAMIQUE

6- diagrammes de temps



LES BASES D'UML

L'ensemble des *14* types de diagrammes UML peut ainsi être réparti sur les trois axes de modélisation.

FONCTIONNEL

Diagramme de cas d'utilisation

Diagramme de classes
Diagramme d'objets
Diagramme de composants
Diagramme de déploiement
Diagramme de paquets
Diagramme de structure composite
Diagramme de profils

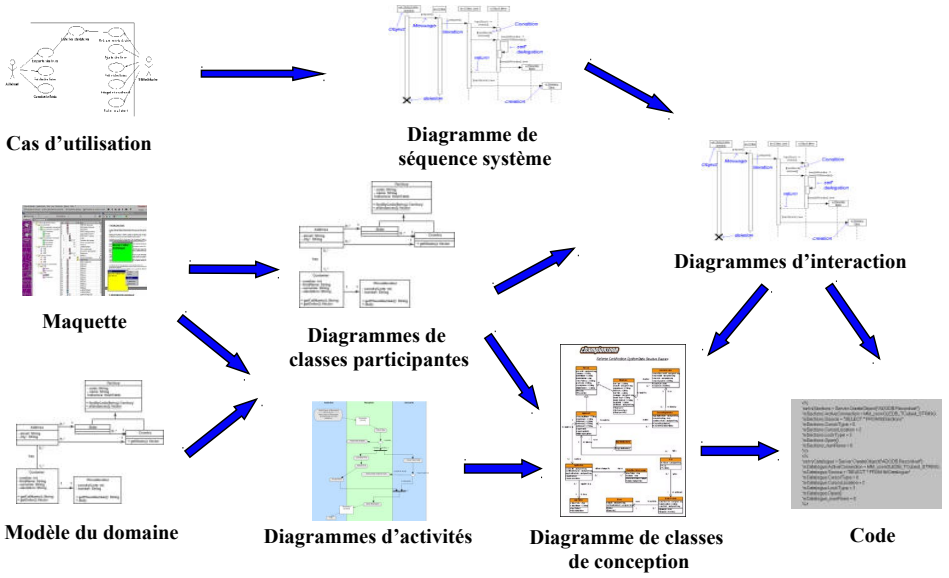
Diagramme d'activité
Diagrammes d'états-transitions
Diagramme de séquence
Diagramme de collaboration
Diagramme global d'interaction
Diagramme de temps

STATIQUE **DYNAMIQUE**

LES BASES D'UML

- Selon la situation, ces diagrammes sont d'une utilité variable.
- Ils ne sont pas tous utilisés lors d'une modélisation.

DÉMARCHE



DÉMARCHE

- On a obtenu un schéma montrant comment, en partant des besoins utilisateurs formalisés par des cas d'utilisation et une maquette, et avec l'apport du modèle du domaine, on peut aboutir à des diagrammes de conception qui permettent de dériver du code assez rapidement.

Diagramme de cas d'utilisation

L'utilisateur au moment de l'expression des besoins

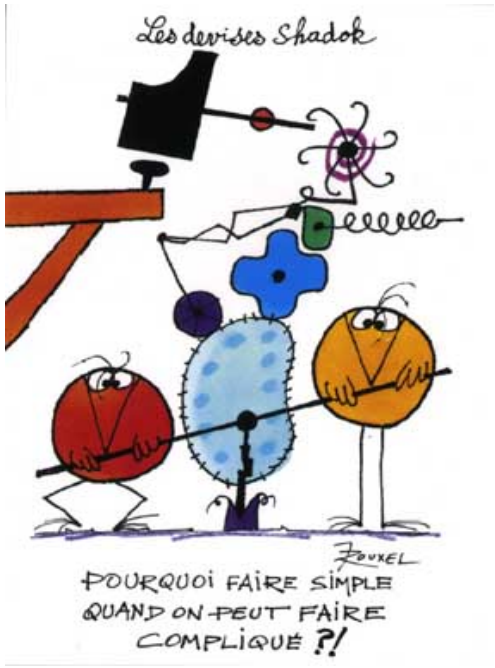
L'utilisateur lors de la revue d'analyse

L'utilisateur après lecture des documents de conception

L'utilisateur le jour de la livraison et installation

DIAGRAMME DE CAS D'UTILISATION

Description de haut niveau des fonctionnalités du système



L'expression des besoins

L'utilisateur au moment de l'expression des besoins

L'utilisateur lors de la revue d'analyse

L'utilisateur après lecture des documents de conception

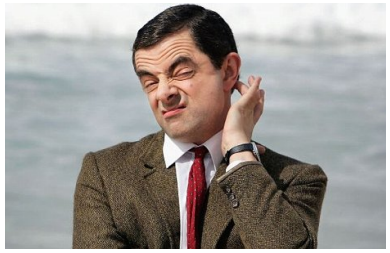
L'utilisateur le jour de la livraison et installation

SOMMAIRE

Première étape :

S'assurer que le logiciel va faire ce que le client souhaite

Comment comprendre ce qu'un client veut vraiment ?



Comment s'assurer qu'un client sait vraiment ce qu'il veut ?

recueil des exigences

recueil des exigences

Satisfaire votre client en livrant avec certitude ce qu'il a demandé

Écouter le client

Laisser parler le client

Se concentrer sur ce que le système doit faire

~~*Comment le système pourra le faire*~~ **NON !!**

Le client s'attend à ce que les choses fonctionnent même si des problèmes surviennent

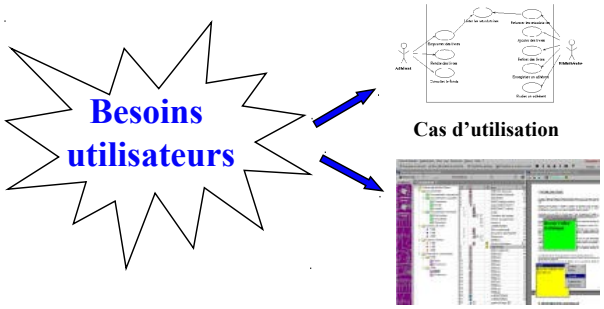
Il faut donc anticiper ce qui pourrait poser problème et ajouter des exigences pour régler ces éventuels problèmes

Bon ensemble d'exigences = au-delà de ce que votre client vous dit.

Il faut s'assurer que le système fonctionne même dans des circonstances inhabituelles ou inattendues

INTRODUCTION

- Expression préliminaire des besoins :
  - ✓ modélisation par les cas d'utilisation
  - ✓ maquette d'interface homme-machine (*IHM*)



- Maquette graphique : montre rapidement l'aspect visuel de l'application.

INTRODUCTION

- Dans le processus de standardisation des méthodes objet ayant abouti au formalisme UML, le concept des cas d'utilisation a été **repris de la méthode OOSE** de **Ivar Jacobson** (*un des pères de UML*).

- **But des cas d'utilisation** :

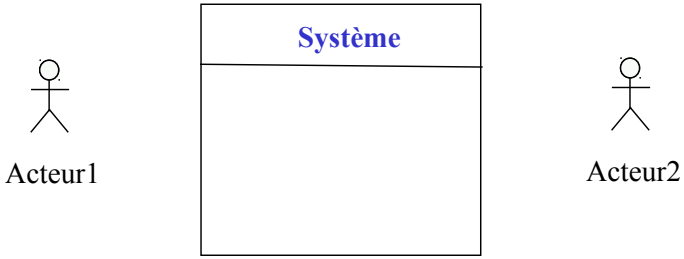
  - effectuer une bonne **délimitation du système**
  - améliorer la **compréhension de son fonctionnement**.

INTRODUCTION

- Pour permettre une **bonne délimitation** du système :

  - bien séparer les éléments constitutifs de l'application,
  - des éléments extérieurs à l'application.

Les acteurs représentent cette frontière.



INTRODUCTION

Objectif des cas d'utilisation

- Définir un **comportement** d'une partie du système **sans révéler la structure interne** du système.

- Un cas d'utilisation représente du point de vue fonctionnel une **fonction essentielle** du système.

INTRODUCTION

Qu'est ce qu'un cas d'utilisation ?

- Représente une interaction typique entre un utilisateur et un système informatique
- Identifie une fonctionnalité visible de l'utilisateur
- S'occupe d'un objectif élémentaire de l'utilisateur

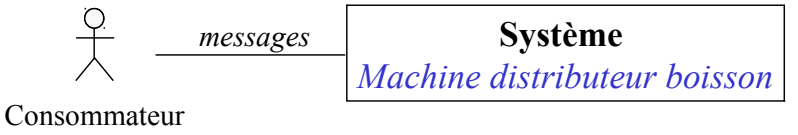
DÉMARCHE

- Identifier les acteurs
- Identifier les cas d'utilisation
- Structurer les cas d'utilisation en paquetages
- Ajouter les relations entre cas d'utilisation

SOMMAIRE

DÉFINITION

- Un **acteur** représente un **rôle** joué par une entité externe (*utilisateur humain, dispositif matériel ou autre système*) qui interagit **directement** avec le système étudié.



- Un acteur peut **consulter** et/ou **modifier** directement l'état du système, en émettant et/ou recevant des messages éventuellement porteurs de données.

TYPES DES ACTEURS

Les acteurs peuvent être de ***3* types** :

- **humains** : ce sont des utilisateurs du logiciel à travers son interface graphique par exemple.

- **logiciels** : ce sont des logiciels déjà disponibles qui communiquent avec le système grâce à une interface logicielle (*une Application Programming Interface par exemple*).

- **matériels, robots et automates** qui exploitent les données du système ou qui sont pilotés par le système.

ACTEURS À L'EXTÉRIEUR DU SYSTÈME

Il faut vérifier que les acteurs se trouvent bien ***à l'extérieur*** du système !

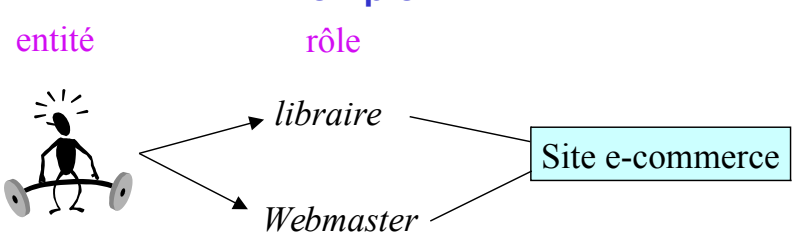
Une erreur fréquente consiste à répertorier des acteurs qui correspondent en fait à des composants du système étudié, voire même à de futures classes.

RÔLE ET ENTITÉ CONCRÈTE

Ne confondez pas rôle et entité concrète.

- Une **même entité concrète** peut jouer successivement **différents rôles** par rapport au système étudié, et être modélisée par plusieurs acteurs.

Exemple

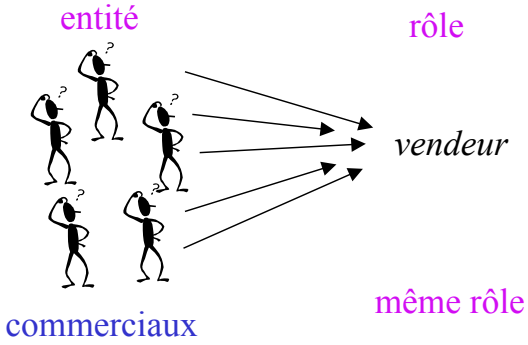


2 acteurs distincts (*2 profils différents*)

RÔLE ET ENTITÉ CONCRÈTE

Ne confondez pas rôle et entité concrète.

- Réciproquement, le **même rôle** peut être tenu simultanément par **plusieurs entités concrètes**, qui seront alors modélisées par le même acteur.



RÔLE ET ENTITÉ CONCRÈTE

On doit penser en termes de rôles et non de personnes ou d'intitulés de fonctions.

ACTEURS PHYSIQUES / LOGIQUES

Éliminez les acteurs physiques au profit des logiques

L'acteur est celui qui bénéficie de l'utilisation du système. Il a une autonomie de décision. Cela ne doit pas se réduire à un simple dispositif mécanique passif.
Cette règle simple permet de s'affranchir dans un premier temps des technologies d'interface et de se concentrer sur les acteurs "*métier*", nettement plus stables.

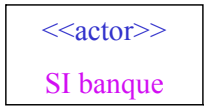
Exemple

- Les terminaux informatiques (*écrans, claviers, imprimantes,etc.*) des différents employés d'une entreprise, à la place des différents profils identifiés.

COMMENT LES REPRÉSENTER ?

1. La **représentation** graphique **standard** est :
   - l'icône appelée *stick man*,
   - avec le nom de l'acteur sous le dessin.

2. On peut également le représenter :
   - sous la forme rectangulaire d'une classe
   - avec le mot-clé <<actor>>.

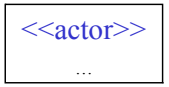


COMMENT LES REPRÉSENTER ?

3. Une troisième représentation :
   - intermédiaire entre les deux premières.

Bonne recommandation

Acteurs humains → 

Acteurs non-humains → 

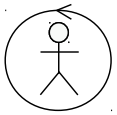
NOTATION DES ACTEURS

Icônes proposées par Ivar Jacobson (*un des pères d'UML*) :

Acteur externe à l'entreprise →

Acteur métier

Acteur interne à l'entreprise →



Travailleur métier

Intérêt : améliorer la lisibilité des diagrammes de cas d'utilisation.

TYPOLOGIE DES ACTEURS

- **Acteur principal** : celui pour qui le cas d'utilisation produit un résultat observable.

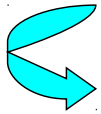
- **Acteur secondaire** : les autres participants du cas d'utilisation.

TYPOLOGIE DES ACTEURS

Acteurs principaux

- répondent à la question :
  - A qui va servir le système qu'on va développer ?
  - Qui le système doit-il aider ?

Ils sont la raison même de l'existence de ce système.

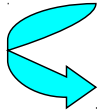


Celui pour qui le cas d'utilisation produit la **plus-value métier**. L'acteur principal est la plupart du temps le déclencheur du cas d'utilisation.

TYPOLOGIE DES ACTEURS

Acteurs secondaires

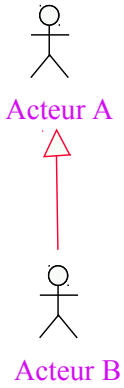
- Ils sont sollicités pour des **informations complémentaires**, ils peuvent uniquement consulter ou informer le système lors de l'exécution du cas d'utilisation.



Ce sont ceux qui paramètrent le système et qui lui fournissent toutes les informations nécessaires à son bon fonctionnement pour les acteurs principaux. Ils sont typiquement sollicités par le système pour obtenir des informations complémentaires.

RELATIONS ENTRE ACTEURS

- La seule relation possible entre deux acteurs est la généralisation.



L'acteur *B* est un acteur *A* avec un pouvoir supplémentaire.

ÉTUDE DE CAS

Étude d'un Guichet Automatique de Banque (GAB)

Le GAB offre les services suivants :

- Distribution d'argent à tout porteur de carte de crédit (*carte Visa ou carte de la banque*), via un lecteur de carte et un distributeur de billets.

- Consultation de solde de compte, dépôt en numéraire et dépôt de chèques pour les clients de la banque porteurs d'une carte de crédit de la banque.

ÉTUDE DE CAS

Étude d'un Guichet Automatique de Banque (GAB)

- Toutes les transactions sont sécurisées :

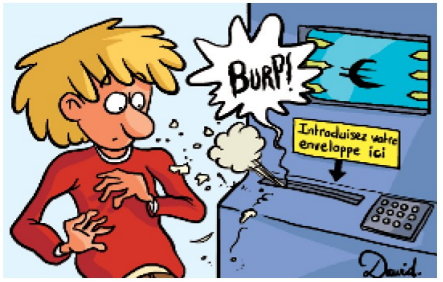
  ↳ Le système d'autorisation Visa, pour les transactions de retrait effectuées avec une carte Visa.

  ↳ Le système d'information de la banque, pour autoriser toutes les transactions effectuées par un client avec sa carte de la banque, mais également pour accéder au solde de ses comptes.

ÉTUDE DE CAS

Étude d'un Guichet Automatique de Banque (GAB)

- Il est nécessaire de recharger le distributeur, récupérer les cartes avalées, etc.

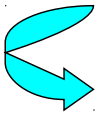


SOLUTION

- La carte bancaire est bien externe au GAB et elle interagit avec lui.

- On applique le principe : **éliminer autant que possible les acteurs physiques au profit des acteurs logiques**.

- C'est le porteur de carte qui retire de l'argent pour le dépenser ensuite, pas la carte !



carte bancaire : pas un acteur

SOMMAIRE

DÉFINITION

Un cas d'utilisation :

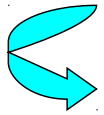
- un **ensemble de séquences d'actions**
  - réalisées par le système et
  - qui **produisent un résultat** observable
  - intéressant pour un acteur particulier.

- **modélise un service** rendu par le système.
  - Il exprime les interactions acteurs/système.

DÉFINITION

Un cas d'utilisation :

- spécifie un comportement attendu du système
  - considéré comme un tout,
  - sans imposer le mode de réalisation de ce comportement.

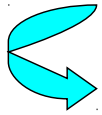


Il permet de décrire *ce que* le futur système devra faire, sans spécifier *comment* il le fera.

COMMENT LES IDENTIFIER ?

- L'ensemble des cas d'utilisation :

  ↳ doit décrire exhaustivement les exigences fonctionnelles du système.



- Chaque cas d'utilisation correspond donc
  - ↳ à une fonction métier du système,
  - ↳ selon le point de vue d'un de ses acteurs.

COMMENT LES IDENTIFIER ?

Pour chaque acteur, il faut :

- rechercher les différentes intentions *"métier"* selon lesquelles il utilise le système.

- déterminer dans le cahier des charges les services fonctionnels attendus du système.

NOM DES CAS D'UTILISATION

Verbe à l'infinitif suivi d'un complément

Utiliser le point de vue de l'acteur et non pas celui du système.

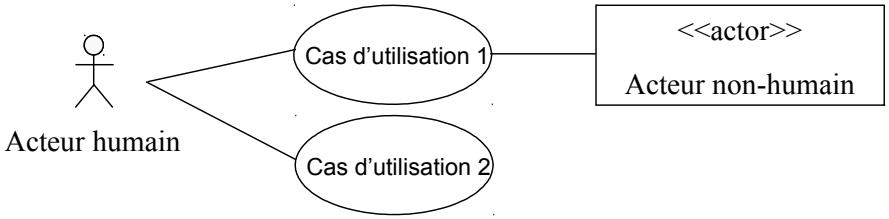
Exemple

Cas d'utilisation d'un distributeur de billets par l'acteur *"Porteur de CB"* doit être intitulé :
- *"Retirer de l'argent"* (*point de vue de l'acteur*),
- et non *"Distribuer de l'argent"* (*point de vue du système*).

COMMENT LES REPRÉSENTER ?

Le diagramme de cas d'utilisation est :

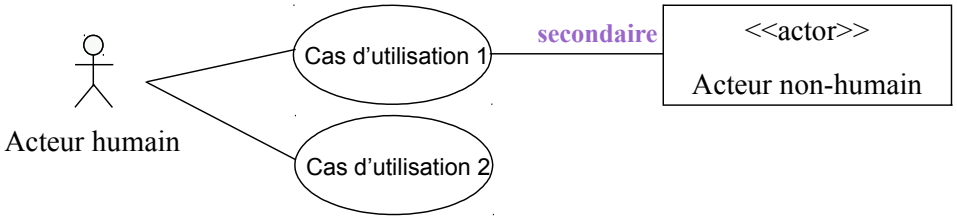
↳ un schéma qui montre les cas d'utilisation (*ovales*)
↳ reliés par des associations (*lignes*) à leurs acteurs.



COMMENT LES REPRÉSENTER ?

Conventions recommandées

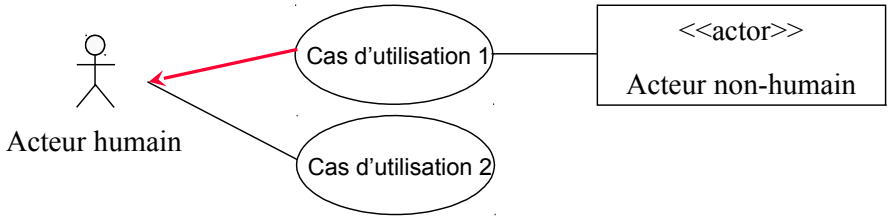
- Par défaut, le rôle d'un acteur est principal.
- Si ce n'est pas le cas, indiquez explicitement que le rôle est secondaire sur l'association, du côté de l'acteur.



COMMENT LES REPRÉSENTER ?

Conventions recommandées

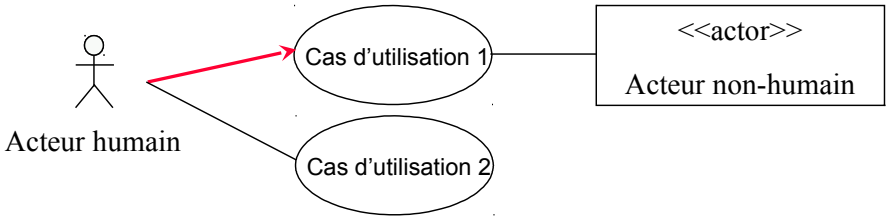
- Si un acteur a pour rôle unique de **consommer des informations** du système sans modifier l'état de celui-ci au niveau métier, représentez cette particularité en ajoutant une flèche vers l'acteur sur son association avec le cas d'utilisation.



COMMENT LES REPRÉSENTER ?

Conventions recommandées

- Si un acteur a pour rôle unique de **fournir des informations** au système sans en recevoir, représentez cette particularité en ajoutant sur l'association une flèche vers le cas d'utilisation.



OBJECTIF DES CAS D'UTILISATION

- dialoguer avec le client,
- analyser les besoins métier,
- disposer d'un support d'analyse,
- aider à démarrer l'analyse orientée objet en identifiant les classes candidates.

Les cas d'utilisation ne constituent pas une fin en soi.

LIMITATION DES CAS D'UTILISATION

Limiter à 20 le nombre des cas d'utilisation

Le niveau de granularité des cas d'utilisation étant très variable, cette limite arbitraire oblige à ne pas se poser trop de questions philosophiques et à rester synthétique.

Il est ainsi courant d'avoir une quinzaine de cas d'utilisation, comprenant chacun une dizaine de scénarios.
On peut considérer ces ordres de grandeur comme une limite pratique qui peut aider à mieux situer la frontière entre cas d'utilisation et scénario.

Une fois les cas d'utilisation identifiés, il faut encore les décrire !

REMARQUE

Pas de notion temporelle

- Il ne doit pas y avoir de notion temporelle dans un diagramme de cas d'utilisation.

- Il ne faut pas se dire que l'acteur fait ceci,
  puis le système lui répond cela,
  ce qui implique une réaction de l'acteur,
  et ainsi de suite.

ORGANISATION DES CAS D'UTILISATION

2 façons différentes et complémentaires :

- en ajoutant des relations d'inclusion,
  d'extension,
  de généralisation entre les cas d'utilisation.

- en les regroupant en paquetages,
  afin de définir des blocs fonctionnels de plus haut niveau.

SOMMAIRE

- Introduction
- Identification des acteurs
- Identification des cas d'utilisation
- **Relations entre cas d'utilisation**
- Structuration en paquetages

TYPES DE RELATIONS

3 types de relations standardisées

- **Une relation d'inclusion**, formalisée par un mot-clé <<inclut>> ou <<include>> : le cas d'utilisation de base en incorpore explicitement un autre, de façon **obligatoire**.

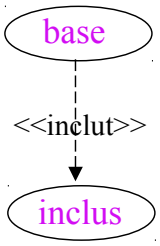
- **Une relation d'extension**, formalisée par un mot-clé <<étend>> ou <<extend>> : le cas d'utilisation de base en incorpore implicitement un autre, de façon **optionnelle**.

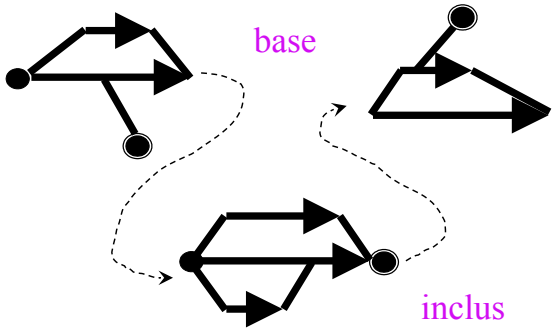
- **Une relation de généralisation/spécialisation** : les cas d'utilisation descendants héritent de la description de leur parent commun. Chacun d'entre eux peut néanmoins comprendre des interactions spécifiques supplémentaires.

RELATION D'INCLUSION

Relation <<INCLUT>>

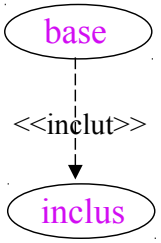
- Le cas de **base** en incorpore explicitement un autre, à un endroit spécifié dans ses enchaînements.
- Le cas d'utilisation **inclus** n'est jamais exécuté seul, mais seulement en tant que partie d'un cas de base plus vaste.





RELATION D'INCLUSION

Relation <<INCLUT>>



- Dans une relation <<inclut>>, le cas d'utilisation de base utilise systématiquement les enchaînements provenant du cas inclus.

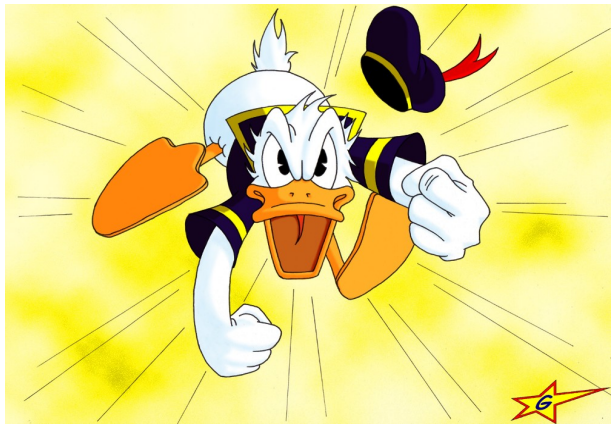
- On utilise fréquemment cette relation pour éviter de décrire plusieurs fois le même enchaînement, en factorisant le comportement commun dans un cas d'utilisation à part.

RELATION D'INCLUSION

Inclusion : Pas de décomposition fonctionnelle !

Une habitude malheureusement assez répandue consiste à utiliser la relation d'inclusion pour décomposer un cas d'utilisation en sous-cas d'utilisation, retombant dans le travers de la décomposition fonctionnelle.

RELATION D'INCLUSION



Inclusion : Pas de décomposition fonctionnelle !

EXERCICE

Exemple d'une entreprise de routage

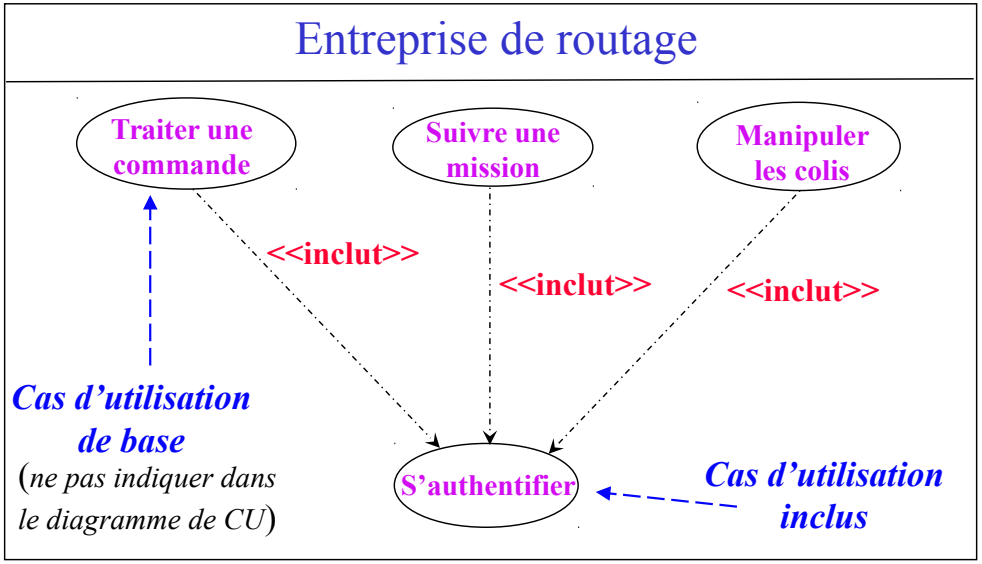
Dans les cas d'utilisation *"Traiter une commande"*, *"Suivre une mission"* et *"Manipuler les colis"*, on spécifie que l'acteur principal doit s'**authentifier**.

Le processus d'authentification implique un flot d'événements entre l'acteur et le système :

- ✓ saisie d'un login,
- ✓ saisie d'un mot de passe,

avec les différents cas d'erreur possibles.

EXERCICE



EXERCICE

Exemple d'une entreprise de routage

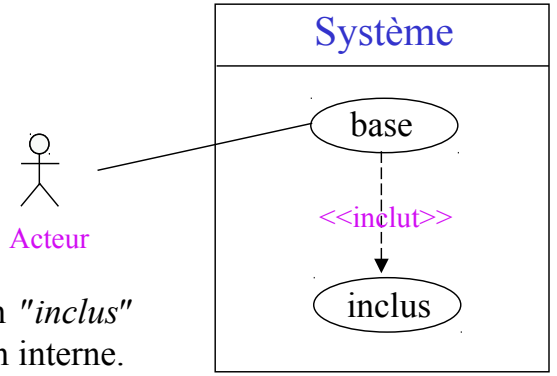
- L'authentification fait partie de la capture des besoins puisqu'il est visible de l'utilisateur final.

- Il correspond tout à fait à la notion de cas d'utilisation inclus, **ne s'exécutant jamais seul**, mais seulement lorsqu'il est appelé par un cas d'utilisation plus large.

RELATION D'INCLUSION

Cas interne

- Un cas d'utilisation est dit **interne** s'il n'est pas relié directement à un acteur.



- Le cas d'utilisation *"inclus"* est un cas d'utilisation interne.

RELATION D'EXTENSION

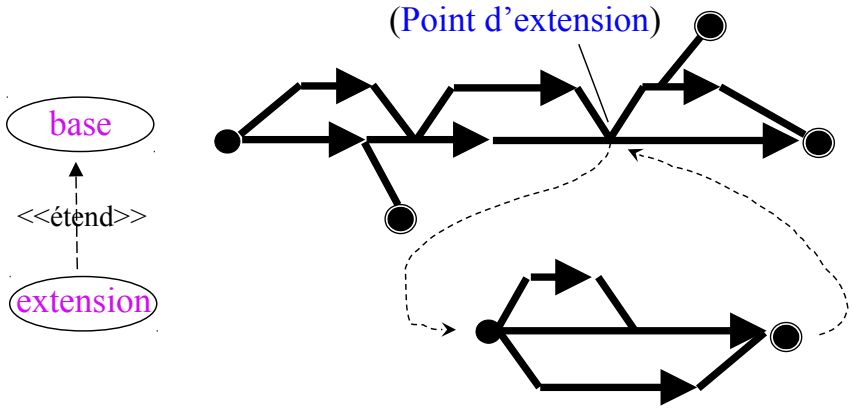
Relation <<ETEND>>

- Le cas de base en incorpore implicitement un autre, à un endroit spécifié indirectement dans celui qui étend.

- Le cas de base peut fonctionner tout seul, mais il peut également être complété par un autre, sous certaines conditions, et uniquement à certains points particuliers de son flot d'événements appelés **points d'extension**.

RELATION D'EXTENSION

Relation <<ETEND>>



RELATION D'EXTENSION

Relation <<ETEND>>

- Dans une relation d'extension, le cas d'utilisation de base recourt optionnellement aux enchaînements provenant du cas d'extension.

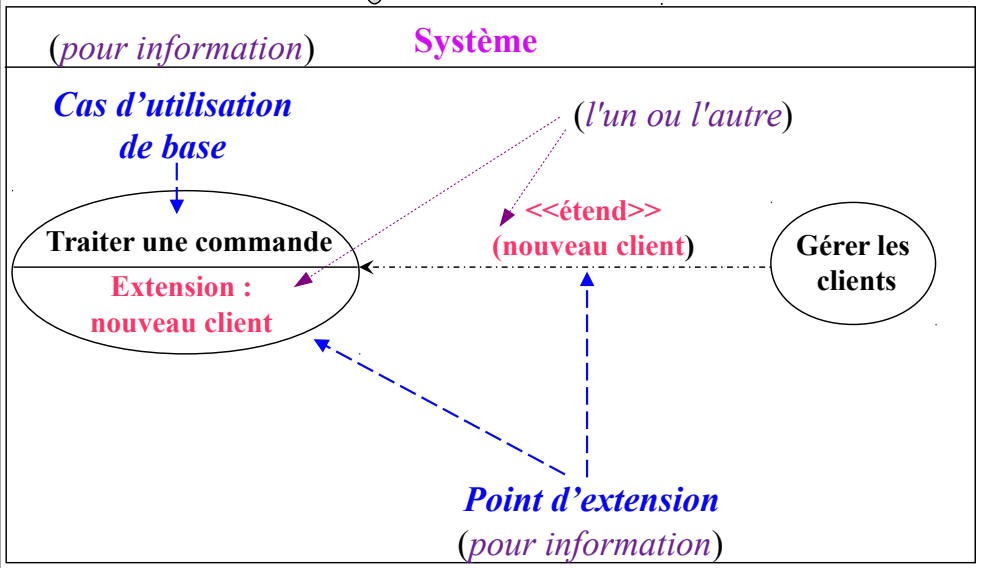
- On utilise principalement cette relation pour séparer le comportement ***optionnel*** (*les variantes*) du comportement obligatoire.

EXERCICE

Dans le cas d'utilisation "*Traiter une commande*", un des enchaînements principaux consiste à créer une nouvelle commande.
Or, si le client est inconnu du système, le réceptionniste va devoir interrompre son processus de création de commande pour tenter auparavant de créer un nouveau client.
Si ce processus se déroule sans encombre, il pourra alors continuer sa création de commande.
Le processus de création de client fait partie intégrante du cas d'utilisation "*Gérer les infos clients*".

Faire le diagramme de cas d'utilisation

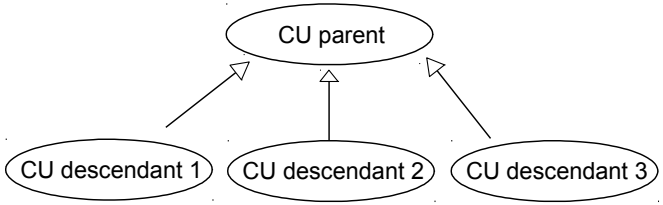
SOLUTION



RELATION DE GÉNÉRALISATION

- Les cas d'utilisation peuvent être hiérarchisés par généralisation / spécialisation.

- Les cas d'utilisation descendants héritent de la sémantique de leur parent. Ils peuvent comprendre des interactions spécifiques supplémentaires, ou modifier les interactions héritées.



SOMMAIRE

- Introduction
- Identification des acteurs
- Identification des cas d'utilisation
- Relations entre cas d'utilisation
- **Structuration en paquetages**

STRUCTURATION EN PAQUETAGES

- Regrouper les cas d'utilisation en ensembles fonctionnels cohérents. Pour ce faire, on utilise le concept général d'UML qui s'appelle le paquetage.

- **Paquetage** : mécanisme général de regroupement d'éléments en UML, qui peut être utilisé par exemple pour regrouper des cas d'utilisation, mais aussi des acteurs, des classes, etc.

- Les éléments contenus dans un paquetage :
  - ✓ doivent représenter un ensemble fortement cohérent,
  - ✓ sont généralement de même nature et de même niveau sémantique.

ÉTUDE DE CAS

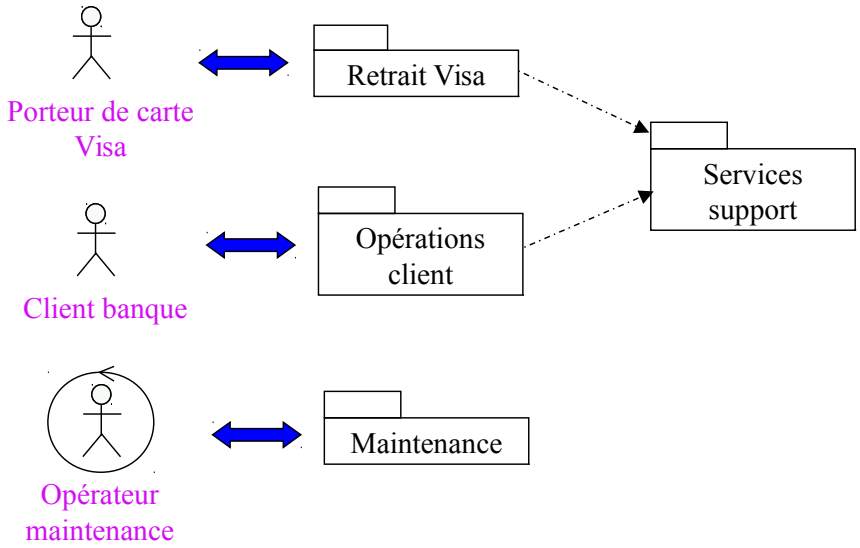
Proposer une structuration des cas d'utilisation en paquetages

SOLUTION

- Plusieurs stratégies sont possibles :
  - regroupement par acteur
  - regroupement par domaine fonctionnel

- Un regroupement par acteur principal est souhaitable car cela permet également de répartir les acteurs secondaires.

SOLUTION



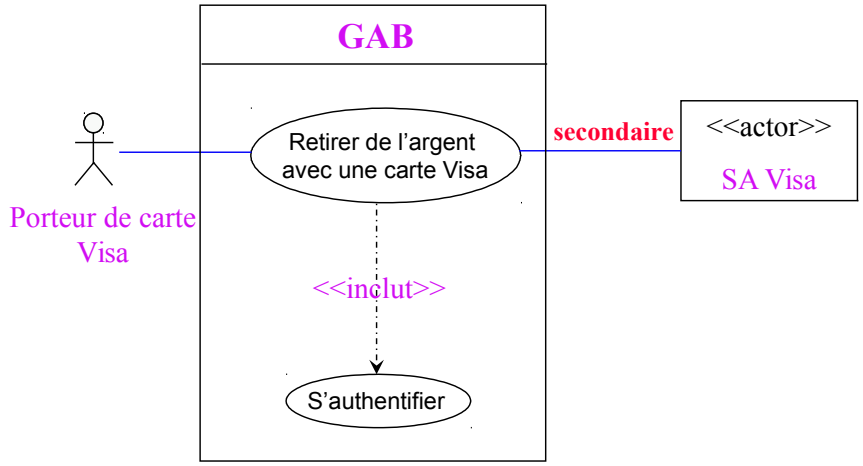
SOLUTION

Faire le diagramme de cas d'utilisation pour le paquetage "Retrait Visa"

Porteur de carte Visa

SOLUTION

Paquetage Retrait Visa

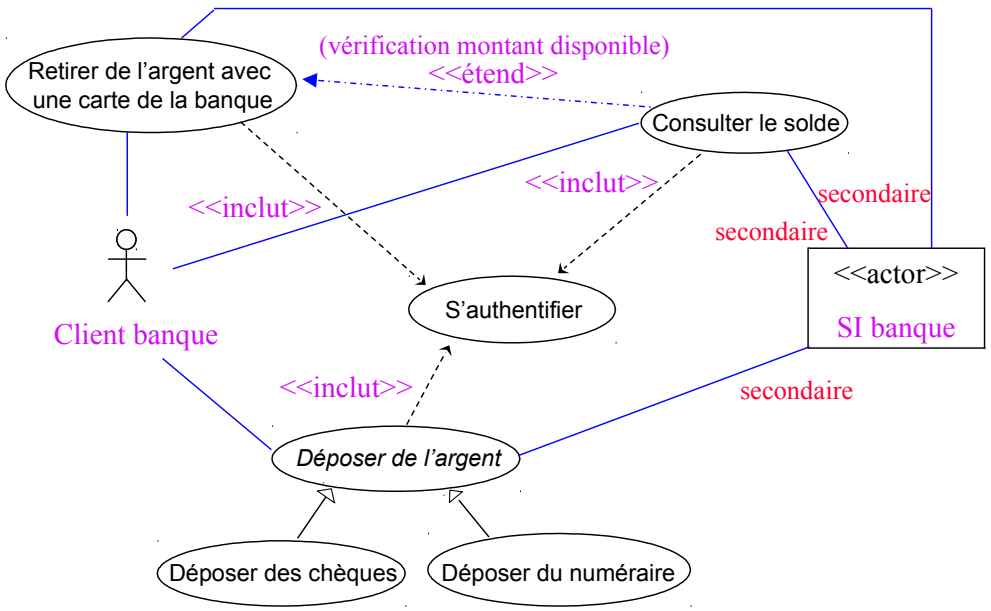


SOLUTION

Faire le diagramme de cas d'utilisation pour le paquetage "Opérations Client"

Client banque

Paquetage Opérations client



SOLUTION

Faire le diagramme de cas d'utilisation pour le paquetage "Maintenance"

Opérateur maintenance

SOLUTION

Paquetage Maintenance

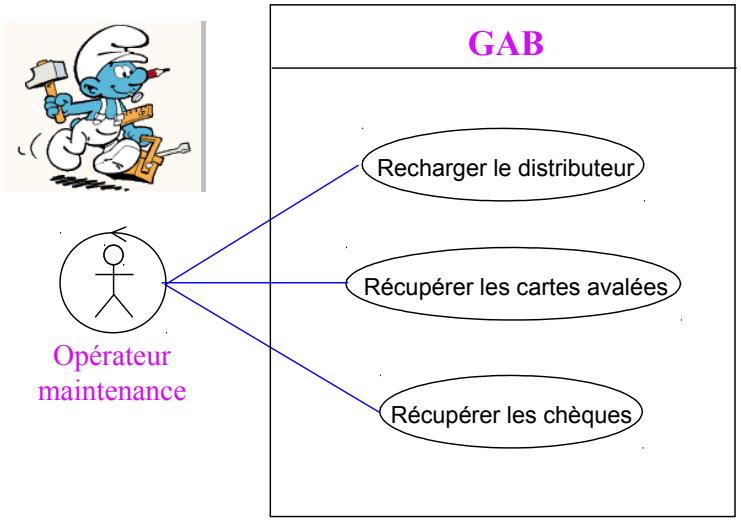
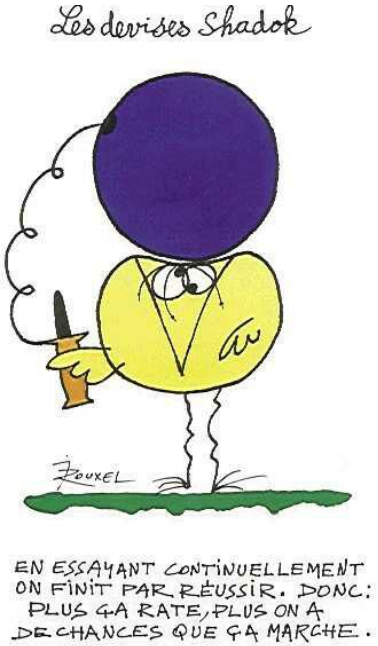


Diagramme de séquence système

DIAGRAMME DE SÉQUENCE SYSTÈME



SOMMAIRE

DÉMARCHE

- Description des cas d'utilisation :
  constitution d'une **fiche-type** pour chaque cas d'utilisation.

- Description textuelle complétée par une représentation graphique UML : le **diagramme de séquence système**.

SOMMAIRE

DESCRIPTION DES CAS D'UTILISATION

- Un cas d'utilisation représente un **ensemble de séquences d'interactions** entre le système et ses acteurs.

- Cela consiste à recenser toutes les interactions de façon **textuelle**.

CAS UTILISATION
Début - interaction 1 (*message*) - - interaction n **Fin**

- Le cas d'utilisation a un **début** et une **fin**.

DESCRIPTION DES CAS D'UTILISATION

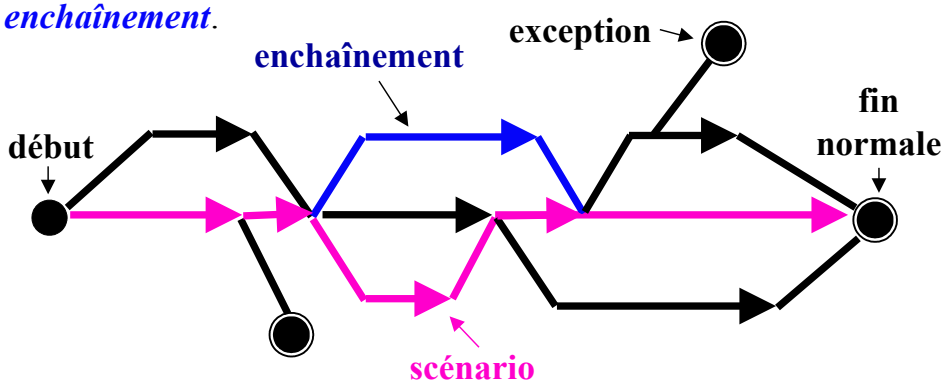
- Il faut préciser les variantes possibles :

  - les cas alternatifs,
  - les cas d'erreurs.

CAS UTILISATION
Cas Nominal **Cas Alternatifs** **Cas d'Erreurs**

DESCRIPTION DES CAS D'UTILISATION

- Chaque unité de description de séquences d'actions est appelée ***enchaînement***.



- Un ***scénario*** représente une succession particulière d'enchaînements, qui s'exécute du début à la fin du cas d'utilisation.

DESCRIPTION DES CAS D'UTILISATION

- Un **scénario** est une suite spécifique d'interactions entre les acteurs et le système.

- C'est une **instance du cas d'utilisation**, un chemin particulier dans sa combinatoire.

- Les scénarios sont aux cas d'utilisation ce que les objets sont aux classes : un scénario est une instance d'un cas d'utilisation.

CLASSE ⟶ *objets*

CAS UTILISATION ⟶ *scénarios*

DESCRIPTION DES CAS D'UTILISATION

- Une autre définition du cas d'utilisation : c'est une **collection de scénarios** de succès ou d'échec qui décrit la façon dont un acteur particulier utilise un système pour atteindre un objectif.

- Les **exceptions** décrivent les **interruptions** possibles d'exécution empêchant l'acteur d'obtenir sa plus-value métier.

DESCRIPTION DES CAS D'UTILISATION

Une erreur fréquente consiste à les rendre **dépendants** d'un choix prématuré d'**interface homme-machine**.

Conséquence

A chaque évolution d'interface : il faut entièrement les redocumenter
alors qu'il s'agit toujours du même cas d'utilisation.

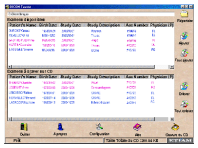
DESCRIPTION DES CAS D'UTILISATION

- La règle consiste à ne pas alourdir la description des séquences d'interactions de considérations techniques d'IHM.

- On peut annexer à la description du cas d'utilisation une proposition d'IHM si le client le souhaite, ou un descriptif des besoins d'IHM.

annexe

CAS UTILISATION
Cas Nominal Cas Alternatifs Cas d'Erreurs



DESCRIPTION DES CAS D'UTILISATION

La quantité de détails dont on a besoin dépend du risque que présente le cas d'utilisation.

Conséquence

Plus le risque est important, plus on doit détailler.

SOMMAIRE

- Démarche
- Description des cas d'utilisation
- **Scénarios**
- Étude de cas
- Exercice site Web
- Diagrammes de séquence système

SCÉNARIOS

- La fiche de description textuelle d'un cas d'utilisation n'est pas normalisée par UML.

On distingue :

- Le **scénario nominal**, celui qui permet de satisfaire les objectifs des acteurs par le chemin le plus direct de succès.

- Des **extensions**, qui comprennent tous les autres scénarios, de succès (*fin normale*) ou d'échec (*exception*).

cas alternatifs *cas d'erreurs*

SCÉNARIOS

Chaque scénario est composé d'étapes qui peuvent être de trois sortes :

- Un message d'un acteur au système.
- Une validation ou un changement d'état du système.
- Un message du système vers un acteur.

SCÉNARIOS

- Les **étapes** sont **numérotées séquentiellement** afin de pouvoir facilement indiquer par la suite les extensions possibles.

- Les extensions indiquent tous les autres scénarios ou branchements possibles, aussi bien de succès que d'échec.

- Comme les extensions doivent se brancher sur le scénario nominal, la convention de numérotation des étapes prend toute son importance.

SCÉNARIOS

- A une étape *X*, une première extension se note *Xa* :
  - elle identifie d'abord la **condition** qui provoque l'extension,
  - puis la **réponse** du système.
- On décrit la **condition** comme quelque chose qui peut être détecté par le système.
- La **réponse** du système peut être résumée en une seule étape ou comprendre une séquence d'étapes.

SCÉNARIOS

- Une seconde extension à la même étape se notera *Xb* et ainsi de suite.
- Si une condition d'extension peut survenir entre l'étape *X* et l'étape *Y*, elle sera notée *X-Ya*.
- Si elle peut arriver à tout moment du scénario nominal, elle sera notée **a*.

PRÉCONDITIONS

- Les **préconditions** définissent ce qui doit être **vrai en amont du cas d'utilisation** pour que celui-ci puisse démarrer.

Elles ne sont pas testées à l'intérieur du cas d'utilisation, mais sont tenues pour acquises.

- Souvent, une précondition implique que le scénario nominal d'un autre cas d'utilisation s'est déroulé normalement.

PRÉCONDITIONS

- Certaines préconditions triviales n'ont pas besoin d'être mentionnées
  - *par exemple, le système doit être alimenté en courant électrique*.
- Seules celles que le rédacteur juge importantes et dignes d'intérêt doivent être répertoriées.

POSTCONDITIONS

- Les **postconditions** définissent ce qui doit être **vrai lorsque le cas d'utilisation se termine avec succès**, qu'il s'agisse du scénario nominal ou d'un scénario alternatif.

SOMMAIRE

- Démarche
- Description des cas d'utilisation
- Scénarios
- **Étude de cas**
- Exercice site Web
- Diagrammes de séquence système

ÉTUDE DE CAS

Guichet Automatique de Banque

Décrire le cas d'utilisation Retirer de l'argent avec une carte Visa

- Résumé
- Acteurs
- Dates de création et de mise à jour
- Nom des responsables
- Numéro de version
- Préconditions
- Scénario nominal
- Extensions
- Postconditions

SOMMAIRE

- Démarche
- Description des cas d'utilisation
- Scénarios
- Étude de cas
- Exercice site Web
- **Diagrammes de séquence système**

DIAGRAMMES DE SÉQUENCE SYSTÈME

- On utilise le terme de **diagramme de séquence système** pour souligner le fait qu'on considère le **système informatique comme une boite noire**.

- Le comportement du système est décrit vu de l'extérieur, sans préjuger de ***comment*** il sera réalisé.
On ouvrira la boite noire seulement en conception.

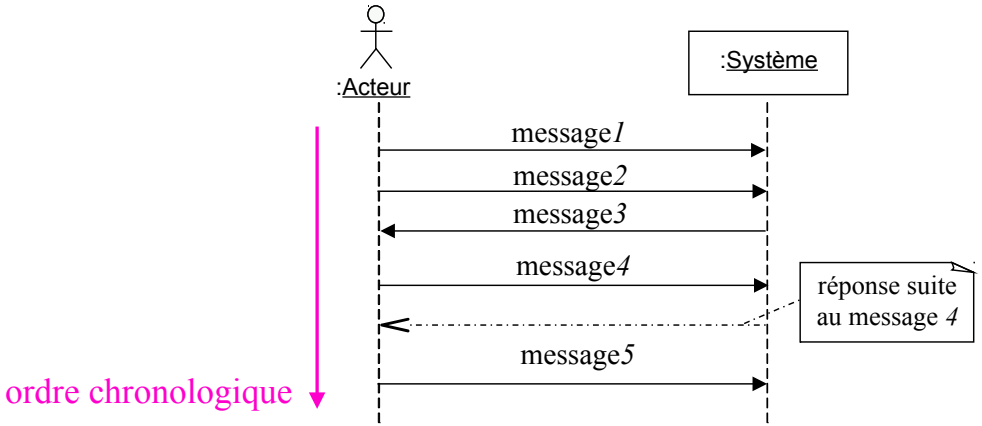
DIAGRAMMES DE SÉQUENCE SYSTÈME

- Les cas d'utilisation décrivent les **interactions des acteurs avec le système** qu'on veut spécifier et concevoir.

- Lors de ces interactions, les acteurs génèrent des **messages** qui affectent le système informatique et appellent généralement une **réponse** de celui-ci.

- On va isoler ces messages et les représenter graphiquement sur des diagrammes de séquence UML.

DIAGRAMMES DE SÉQUENCE SYSTÈME

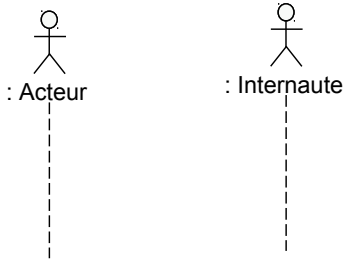


- L'ordre chronologique se déroule vers le bas et l'ordre des événements doit suivre la séquence décrite dans le cas d'utilisation.

DIAGRAMMES DE SÉQUENCE SYSTÈME

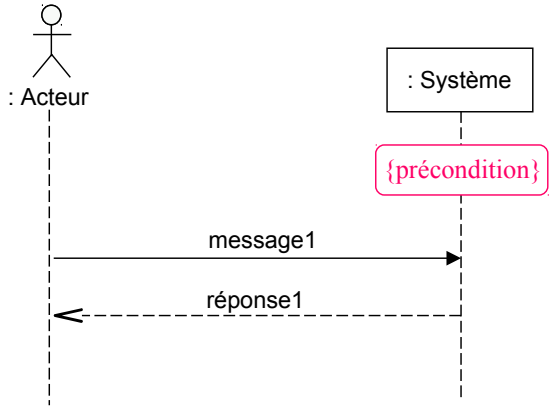
- Les diagrammes de séquence montrent des interactions entre instances (*objets*) et non pas entre classes qui ne sont que des descripteurs.

- La notation :Acteur indique qu'il s'agit d'une instance d'acteur et non de la classe.



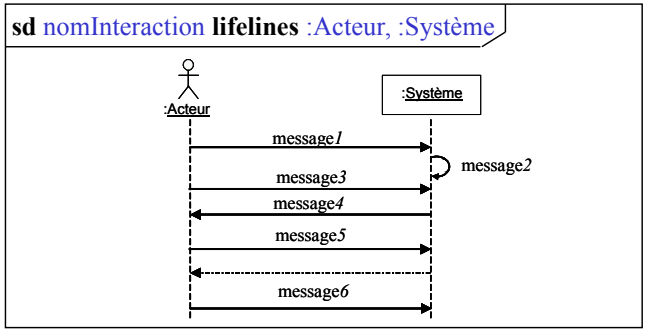
DIAGRAMMES DE SÉQUENCE SYSTÈME

- La précondition du cas d'utilisation peut être matérialisée graphiquement grâce au symbole d'état sur la ligne de vie.



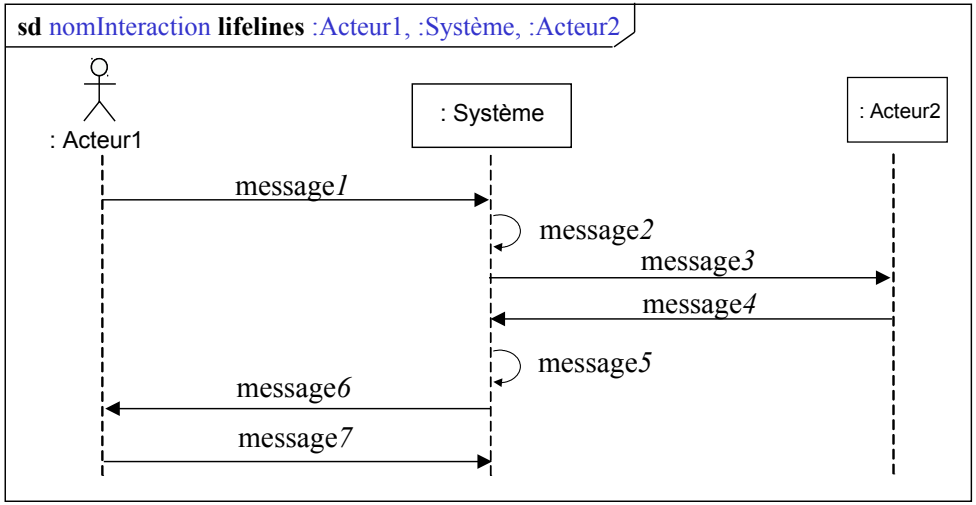
DIAGRAMMES DE SÉQUENCE SYSTÈME

- Un diagramme d'interaction se représente par un rectangle (*fragment d'interaction*) contenant, dans le coin supérieur gauche, un pentagone accompagné du mot-clé *sd* (*lorsqu'il s'agit d'un diagramme de séquence*) suivi du nom de l'interaction. La liste des lignes de vie qui figurent dans le diagramme peut suivre le nom de l'interaction.



DIAGRAMMES DE SÉQUENCE SYSTÈME

Exemple de diagramme avec plus d'un acteur



DIAGRAMMES DE SÉQUENCE SYSTÈME

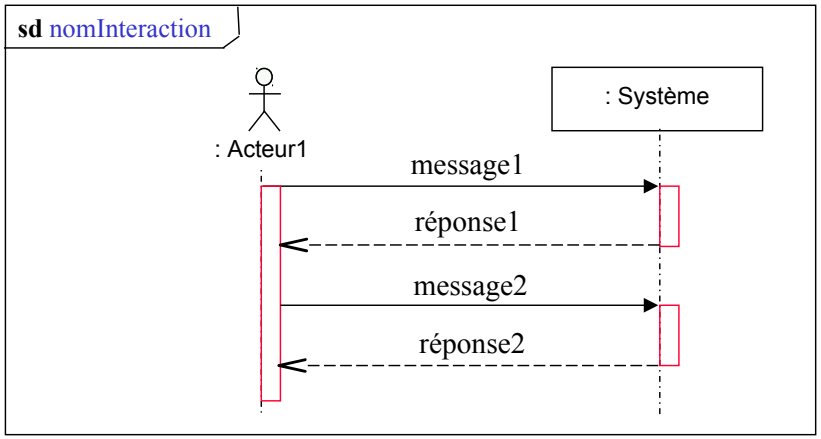
- La spécification de l'exécution d'une réaction se fait par un rectangle blanc ou gris placé sur la ligne de vie.

- Cette notation d'activation est optionnelle mais nous aide à mieux comprendre la flèche pointillé du message de retour.

- En général, on ne fait figurer que les retours intéressants et non les retours triviaux.

DIAGRAMMES DE SÉQUENCE SYSTÈME

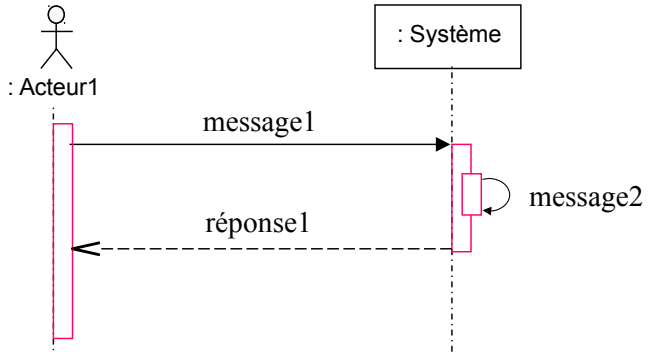
- Le système informatique n'est actif que lorsqu'il est sollicité par un acteur, alors que les acteurs sont a priori toujours actifs.



FLÈCHE BOUCLANT SUR LE SYSTÈME

- Elle permet de représenter graphiquement un comportement interne majeur sur lequel on veut mettre l'accent.

- Il ne faut pas en abuser car ce n'est pas l'objectif premier de ce type de diagramme.



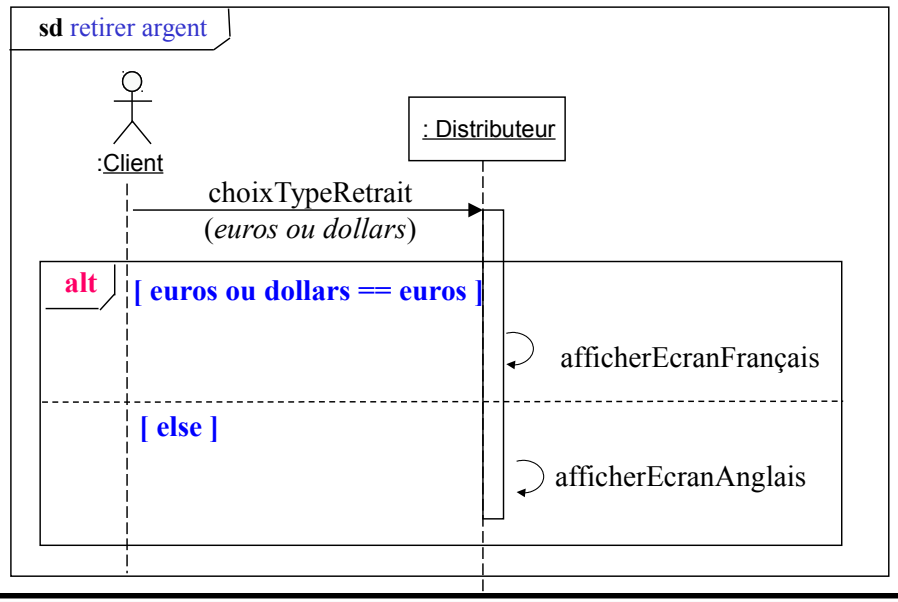
OPÉRATEURS D'INTERACTION

- Choix et boucle : *alternative*, *option*, *break* et *loop*.
- Envoi de messages : *assertion* et *negative*.
- Envoi de messages en parallèle : *parallel* et *critical region*.
- Ordre des messages : *weak sequencing* et *strict sequencing*.

ALTERNATIVES

- Les alternatives permettent de sélectionner un comportement en fonction de conditions de garde placées avant chaque opérande.
- L'opérateur d'interaction est **alt**.
- On peut inclure une condition de garde de type else qui exécute l'opérande associé lorsque toutes les autres conditions ont pour valeur false.
- C'est un peu l'équivalent d'une exécution à choix multiple (*switch*).
- Si plusieurs opérandes prennent la valeur vraie, le choix est non déterministe.

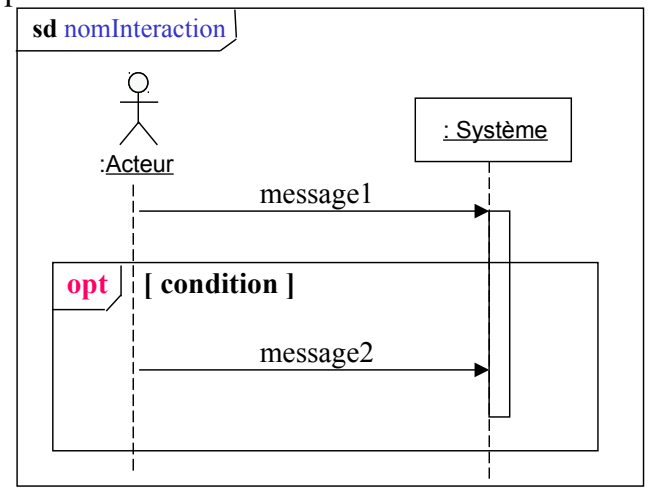
ALTERNATIVES



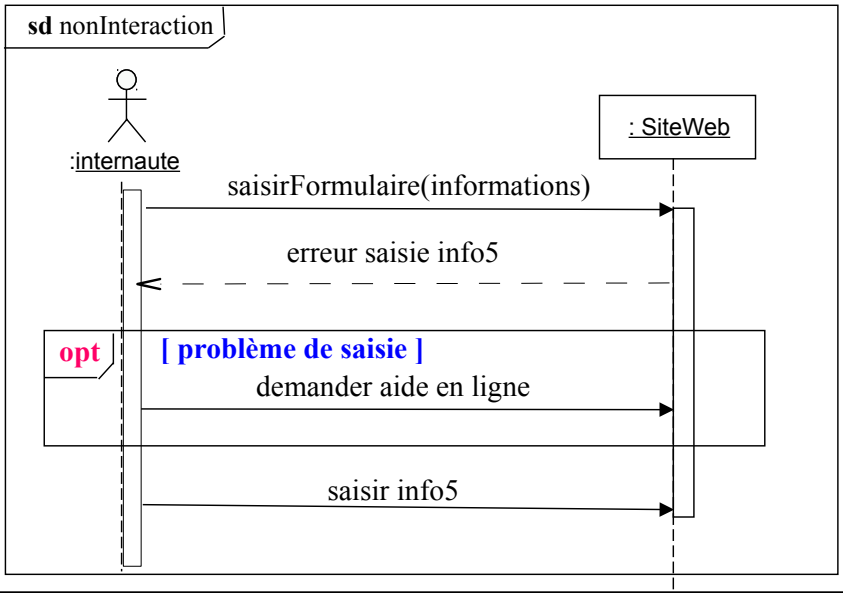
OPTION

- L'opérateur *option* est identique à un opérateur d'alternative, mais avec un choix unique.

le sous-fragment s'exécute si la condition de garde est vraie et ne s'exécute pas dans le cas contraire.



OPTION



BOUCLES

- Une boucle indique qu'un ensemble d'occurrences événementielles doit être exécuté un certain nombre de fois.
- L'opérateur d'interaction est **loop**.
- La notation inclut une valeur minimale et une valeur maximale de répétition.
- On peut aussi utiliser une condition de garde qui est évaluée à chaque itération de la boucle afin de sortir éventuellement de celle-ci.

loop (min, max) où min et max sont optionnelles.

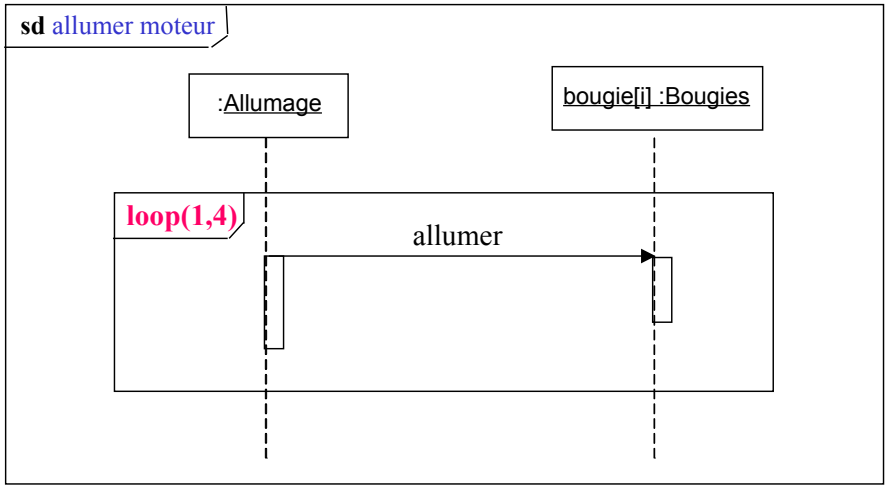
BOUCLES

- Si max est omis, alors il a implicitement la même valeur que min.

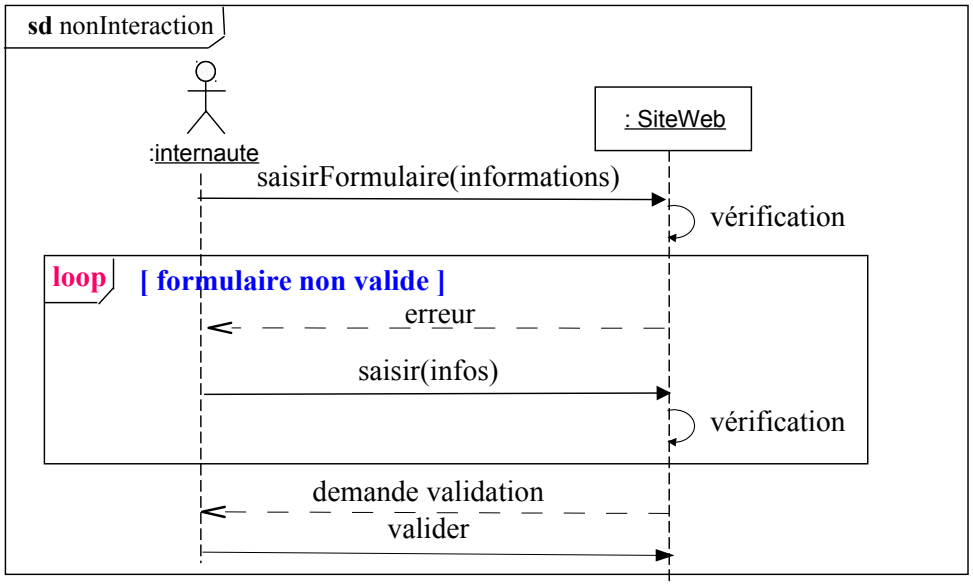
- La valeur de max peut être remplacée par un astérisque pour indiquer une boucle infinie, tout au moins tant que la condition de garde retourne true.

- Si les deux valeurs min et max sont omises, alors min et max sont supposés valoir 0 et l'infini. Dans ce cas, une condition de garde est indispensable pour éviter une boucle infinie.

BOUCLES



BOUCLES



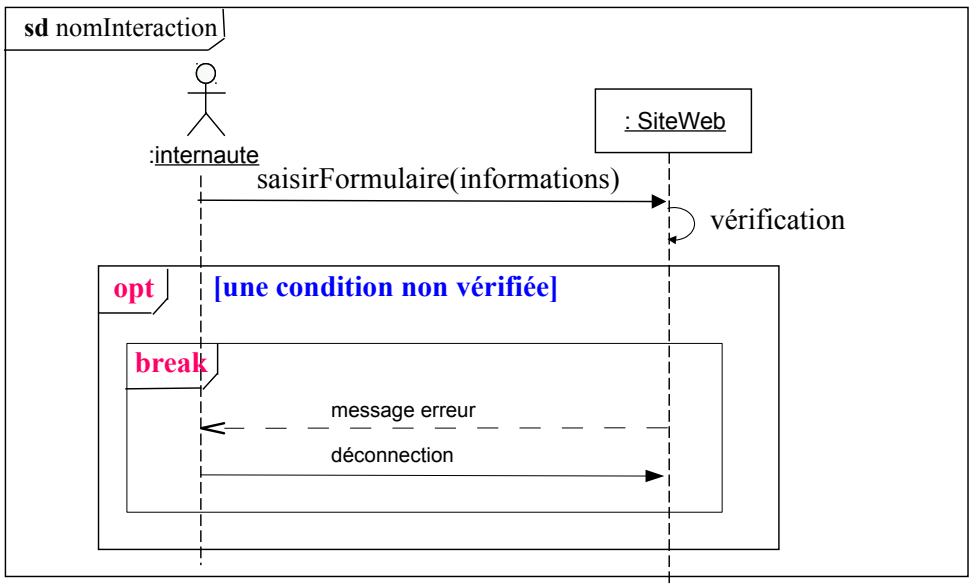
BREAK

- Une rupture (*break*) indique que l'opérande du fragment d'interaction associé doit s'exécuter puis mettre fin à l'interaction englobante.

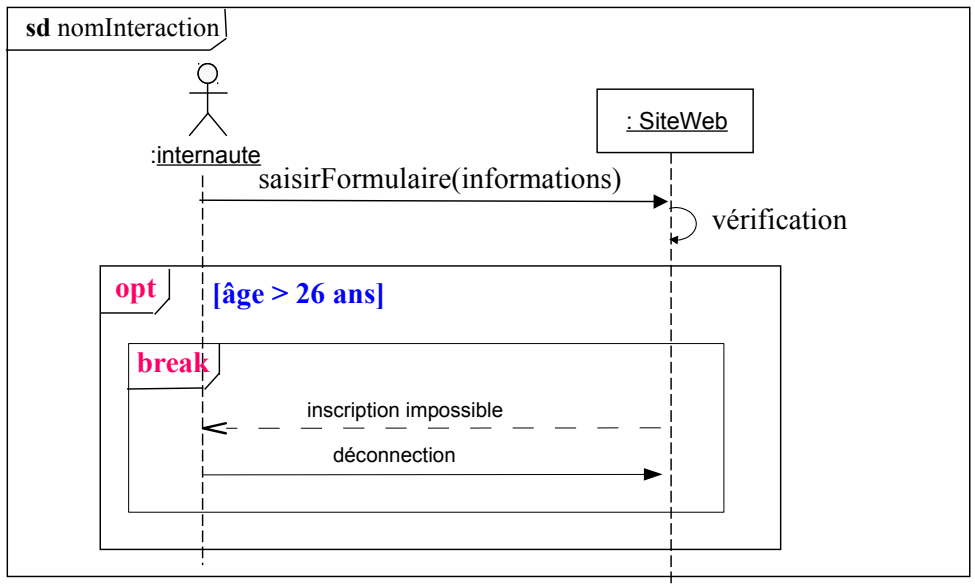
- L'opérateur d'interaction est **break**.

- Une rupture est similaire au bloc de code suivant :
  if (conditionDeGarde) { … ; return ;}

BREAK



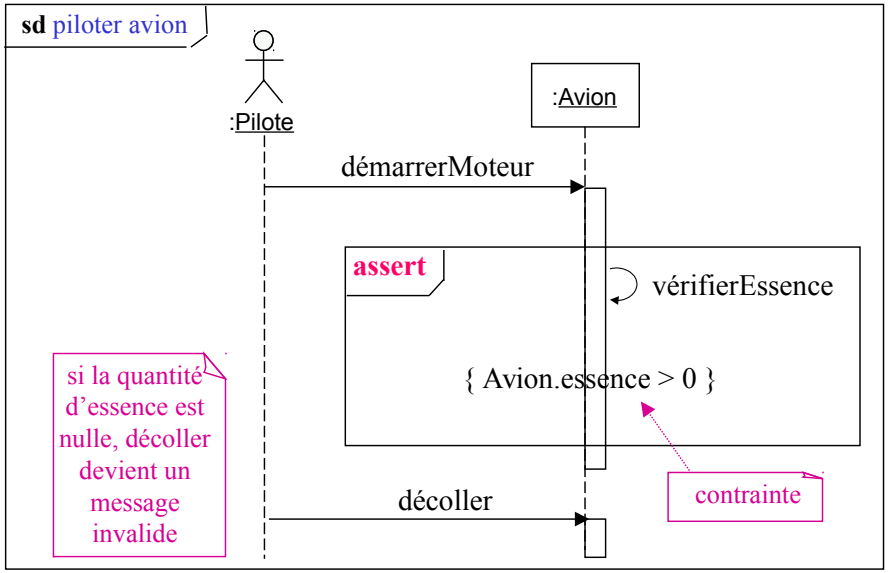
BREAK

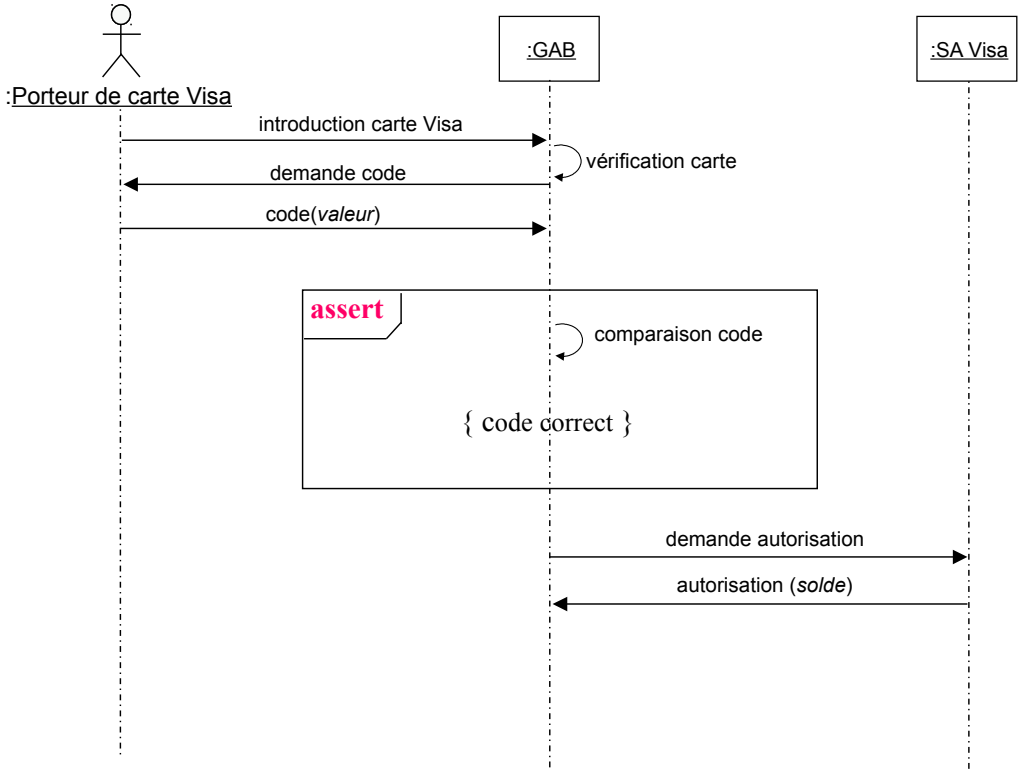


CONTRAINTES

- Une contrainte est indiquée sur une ligne de vie par un texte entre accolades.
- Une contrainte est évaluée au cours de l'exécution de l'interaction.
- Si la condition de la contrainte n'est pas vérifiée, les occurrences d'événement qui suivent cette contrainte sont considérées comme invalides, alors qu'une contrainte qui se vérifie rend valides les événements à suivre.

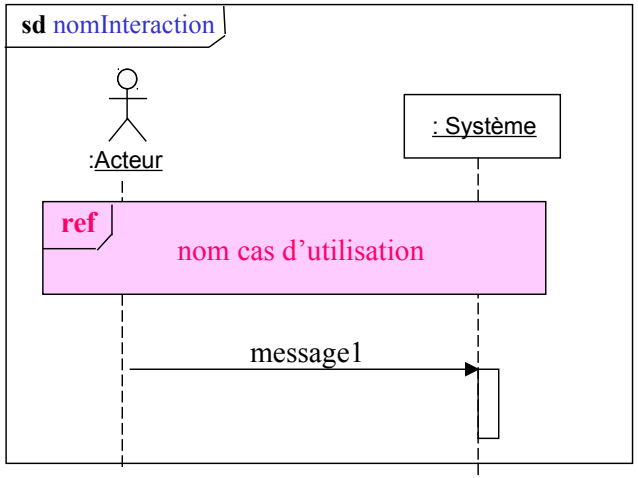
CONTRAINTES



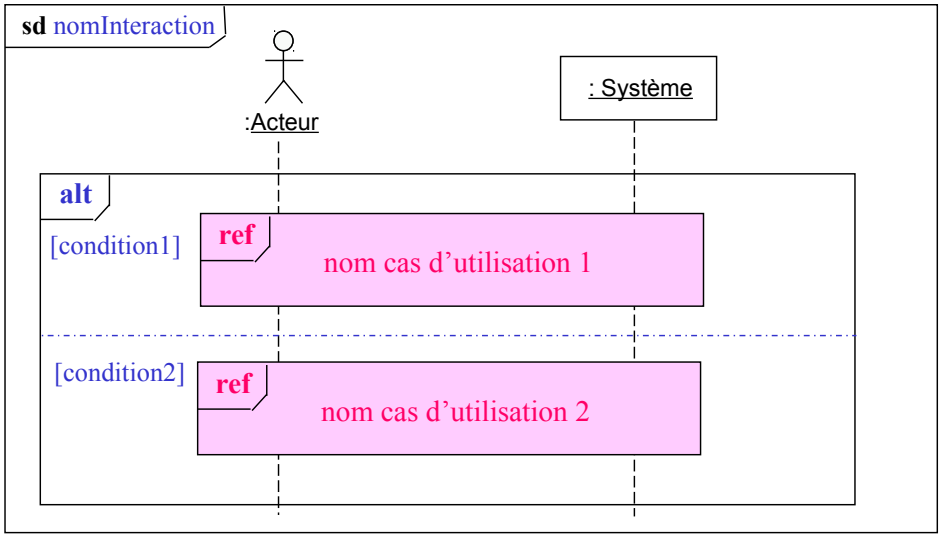


RENVOI VERS D'AUTRES CAS D'UTILISATION

- Une interaction peut référencer un cas d'utilisation et cela sera matérialisé par un cadre avec le mot-clé *ref*.



RENVOI VERS D'AUTRES CAS D'UTILISATION



Conception objet préliminaire

CONCEPTION OBJET PRÉLIMINAIRE

SOMMAIRE

INTRODUCTION

- Les diagrammes de cas d'utilisation montrent des interactions entre des acteurs et les grandes fonctions d'un système.

- Cependant, les interactions ne se limitent pas aux acteurs : les objets au cœur du système interagissent en s'envoyant des messages.

INTRODUCTION

- Apprentissage d'**attribution des responsabilités** de comportement des classes d'analyse (*entités* + *dialogues* + *contrôles*).

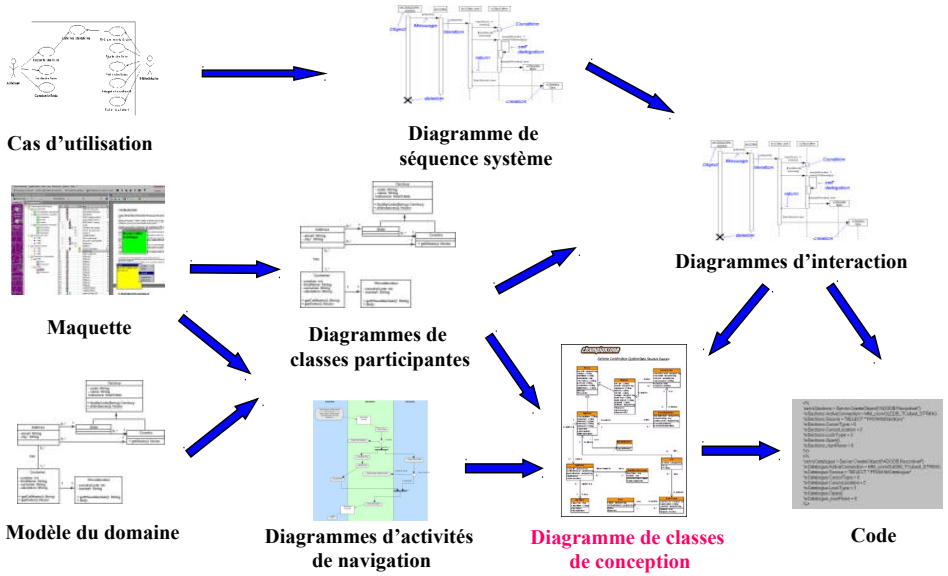
- Présentation du résultat de cette étude dans des diagrammes d'interactions UML (*séquence ou communication*).

Vue statique complétée

(*sous forme de diagrammes de classes de conception préliminaire*)

indépendante des choix technologiques

DÉMARCHE



INTRODUCTION

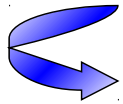
Déjà réalisé :

- Identification d'un certain nombre d'opérations potentielles dans les classes "*dialogues*" et "*contrôles*" :

Premier jet, une base de travail.

Actuellement :

- Conception d'un ensemble de classes faiblement couplées entre elles et fortement cohérentes.



Étude détaillée de la dynamique des objets grâce aux diagrammes d'interaction.

INTRODUCTION

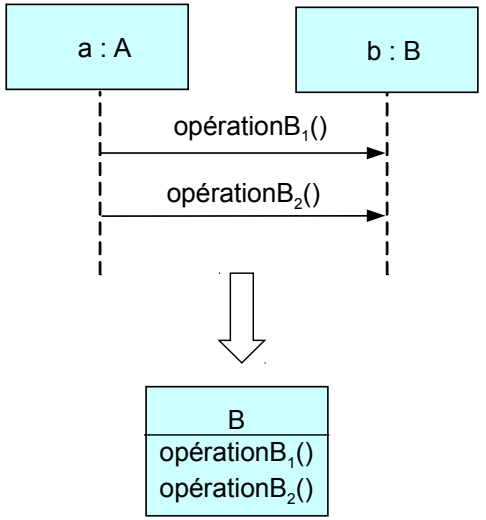
- L'attribution des bonnes responsabilités aux bonnes classes = un des problèmes les plus délicats de la conception orientée objet.

- Pour chaque service ou fonction, il faut décider quelle est la classe qui va le contenir.

- Les diagrammes d'interaction permettent au concepteur de représenter graphiquement ces décisions d'allocation de responsabilités.

- Chaque diagramme va représenter un ensemble d'objets de classes différentes collaborant dans le cadre d'un scénario d'exécution du système.

INTRODUCTION



INTRODUCTION

- Les objets communiquent en s'envoyant des messages qui invoquent des opérations (*ou méthodes*) sur les objets récepteurs.

- Suivi visuel des interactions dynamiques entre objets, et les traitements réalisés par chacun.

- Par rapport aux diagrammes de séquence système, on va remplacer le système vu comme une boite noire par un ensemble d'objets en collaboration.

- Utilisation des trois types de classes d'analyse, à savoir les dialogues, les contrôles et les entités.

INTRODUCTION

- Respect des règles fixées sur les relations entre classes d'analyse

✓ les acteurs ne peuvent interagir (*envoyer des messages*) qu'avec les dialogues.

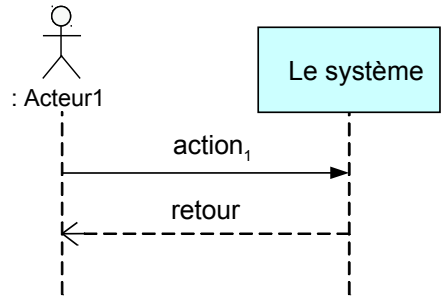
✓ les dialogues peuvent interagir avec les contrôles ou, exceptionnellement, avec d'autres dialogues.

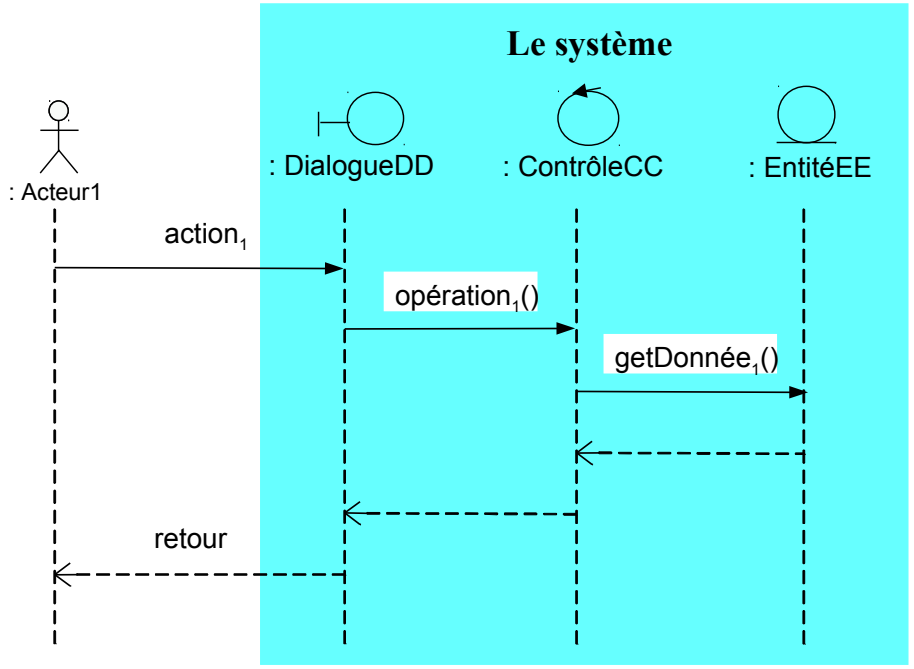
✓ les contrôles peuvent interagir avec les dialogues, les entités, ou d'autres contrôles.

✓ les entités ne peuvent interagir qu'entre elles.

INTRODUCTION

- Changement du niveau d'abstraction par rapport au diagramme de séquence système :





SOMMAIRE

- Introduction
- **Diagrammes d'interaction**
- Diagrammes pour le site Web
- Classes de conception préliminaire

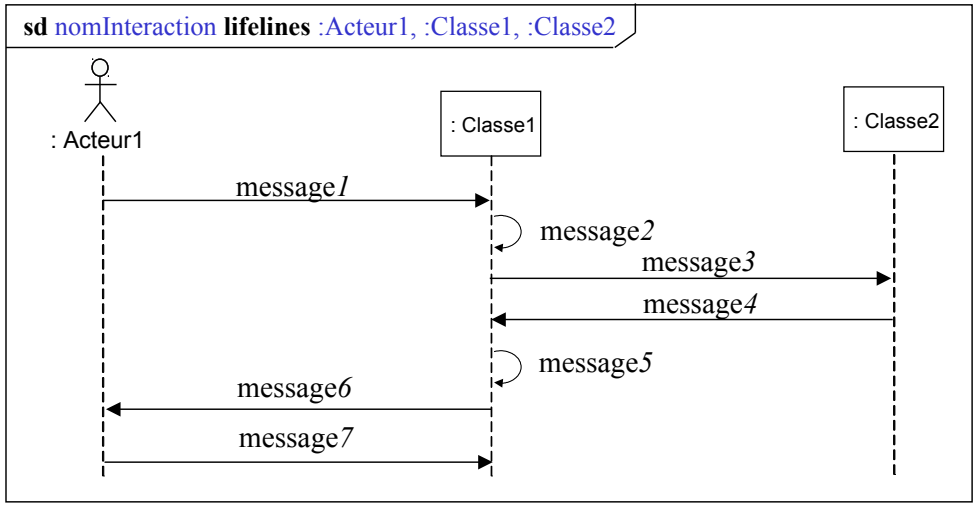
TYPES DE DIAGRAMMES D'INTERACTION

- *2* types de diagrammes d'interaction :
  - ✓ les diagrammes de séquence,
  - ✓ les diagrammes de communication (UML 2) ou de collaboration (UML 1).
- Chaque type de diagramme a ses points forts et ses points faibles.
- Si on manque de place en largeur : les diagrammes de communication sont préférables (*ils permettent l'extension verticale des nouveaux objets*).
  En revanche, la lecture des séquences de messages y est plus difficile.

DIAGRAMMES DE SÉQUENCE

- Les **diagrammes de séquence** représentent les interactions dans un format où chaque nouvel objet est ajouté en haut à droite.
- On représente la ligne de vie de chaque objet par un trait pointillé vertical.
- Chaque ligne de vie sert de point de départ ou d'arrivée à des messages représentés eux-mêmes par des flèches horizontales.
- Par convention, le temps coule de haut en bas.
- Il indique ainsi visuellement la séquence relative des envois et réceptions de messages, d'où la dénomination : diagramme de séquence.

DIAGRAMMES DE SÉQUENCE



DIAGRAMMES DE SÉQUENCE

Syntaxe des messages

[<attribut> =] <message> [: <valeurRetour>]

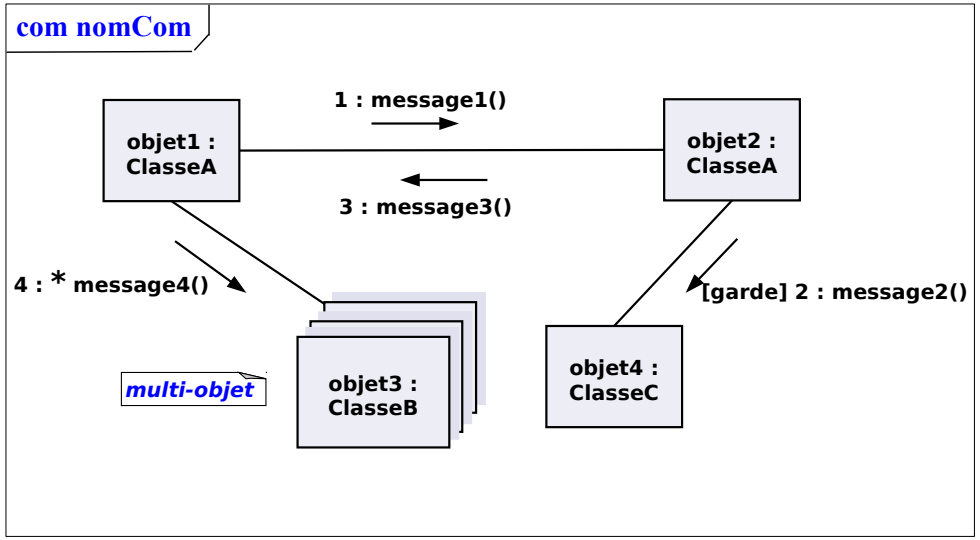
optionnel *optionnel*

nombreLivres = chercher(" Tintin ") : 42

DIAGRAMMES DE COMMUNICATION

- Les **diagrammes de communication** illustrent les interactions entre objets sous forme de graphes ou de réseaux.
- Les objets peuvent être placés en tout point du diagramme.
- Ils sont connectés par des liens qui indiquent qu'une forme de navigation et de visibilité entre ces objets est possible.
- Tout message entre objets est représenté par une expression et une petite flèche indiquant son sens de circulation.
- Chaque lien permet le trafic de plusieurs messages et chaque message est assorti d'un numéro d'ordre.

DIAGRAMMES DE COMMUNICATION



DIAGRAMMES DE COMMUNICATION

Syntaxe des messages

[<cond>] <seq> [<iter>] : [<var>] <message> [<par>]

optionnel *optionnel* *optionnel* *optionnel*

- <cond> : condition sous forme d'expression booléenne entre crochets.
- <seq> : numéro de séquence du message.
  Les messages 1.2a et 1.2b sont envoyés en même temps.

DIAGRAMMES DE COMMUNICATION

[<cond>] <seq> [<iter>] : [<var>] <message> [<par>]

- <iter> : spécifie, en langage naturel et entre crochets, l'envoi séquentiel (* <iter>) ou en parallèle (* || <iter>) de plusieurs messages.
La spécification <iter> peut être ignorée.

- <var> : valeur de retour du message, qui sera, par exemple, transmise en paramètre à un autre message.

- <message> : nom du message.

- <par> : paramètres (*optionnels*) du message.

DIAGRAMMES DE COMMUNICATION

Exemple de messages

[<cond>] <seq> [<iter>] : [<var>] <message> [<par>]

- 1 : mes()
- 2.1 : mes(args)
- 3.2b : var = mes()
- [a > b] 3 : mes()
- 2.1 *[i : =0 .. 10] : mes()
- 4 *|| [pour toutes les portes] : fermer()
- [heure = heureDépart] 1.1a * || [i:= 0 .. 10] : ok[i] = fermer()

DIAGRAMMES D'INTERACTION

- Les diagrammes d'interaction montrent des instances (*des occurrences*) et non pas des classes :

  ✓ à chaque élément UML (*classe, acteur, etc.*) correspond une instance qui utilise le même symbole graphique que le type.

  ✓ on peut identifier l'instance par un nom univoque (*exemple objet1:ClasseA*). En l'absence d'un tel nom, on fait précéder le nom de classe par le symbole ":" .

DIAGRAMMES D'INTERACTION

Types de messages

- Envoi d'un signal
- Invocation d'une opération
- Création ou destruction d'une instance

Envoi d'un signal

- L'envoi d'un signal déclenche une réaction chez le récepteur, de façon asynchrone.
- L'émetteur du signal ne reste pas bloqué le temps que le signal parvienne au récepteur et il ne sait pas quand, ni même si le message sera traité par le destinataire.

Invocation d'une opération

- Type de message le plus utilisé en programmation objet.
- L'invocation peut être synchrone (*l'émetteur reste bloqué le temps que dure l'invocation de l'opération*) ou asynchrone.
- La plupart des invocations sont synchrones.

Représentation des messages

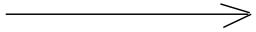
- UML fait la différence entre un message synchrone et asynchrone.

ATTENTION NOTATION

flèche pleine

message synchrone

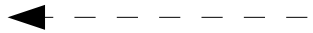
flèche évidée

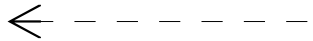


message asynchrone

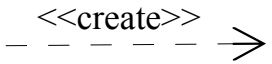
Représentation des messages

message réponse





message de création



message de suppression

rien d'indiqué

Types de messages

- **message synchrone** : bloque l'expéditeur jusqu'à la prise en compte du message par le destinataire. Le flot de contrôle passe de l'émetteur au récepteur (*l'émetteur devient passif et le récepteur actif*) à la prise en compte du message.

- **message asynchrone** : n'interrompt pas l'exécution de l'expéditeur. Le message envoyé peut être pris en compte par le récepteur à tout moment ou ignoré (*jamais traité*).

Création ou destruction d'une instance

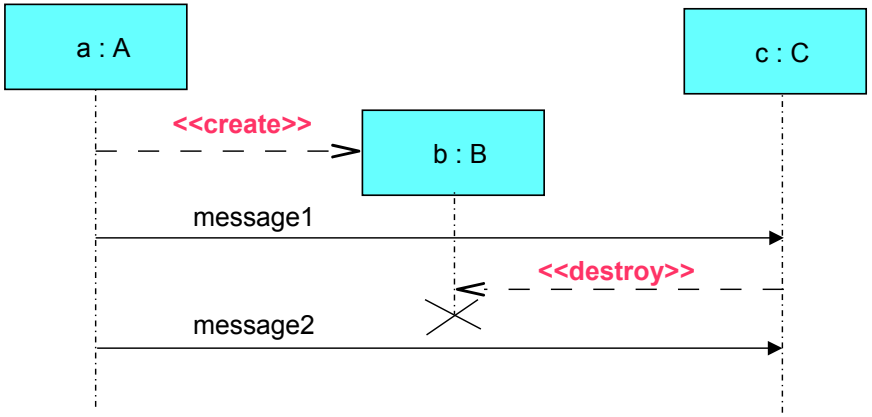
- Création d'une instance : message standardisé *"create"*.

- Le message *"create"* peut comprendre des paramètres d'initialisation.

- Destruction d'un instance : message standardisé *"destroy"*.

- La création et la destruction d'instances ont des représentations particulières sur le diagramme de séquence, mais pas sur le diagramme de communication.

- L'objet *b* est créé et détruit durant le scénario, contrairement aux objets *a* et *c* qui préexistent et survivent au scénario concerné.



- Certains langages objet comme Java et C# ont un "*garbage collector*" qui détruit automatiquement les objets qui ne sont plus utilisés.

- Ce n'est pas le cas en C++ où les objets doivent être détruits explicitement.

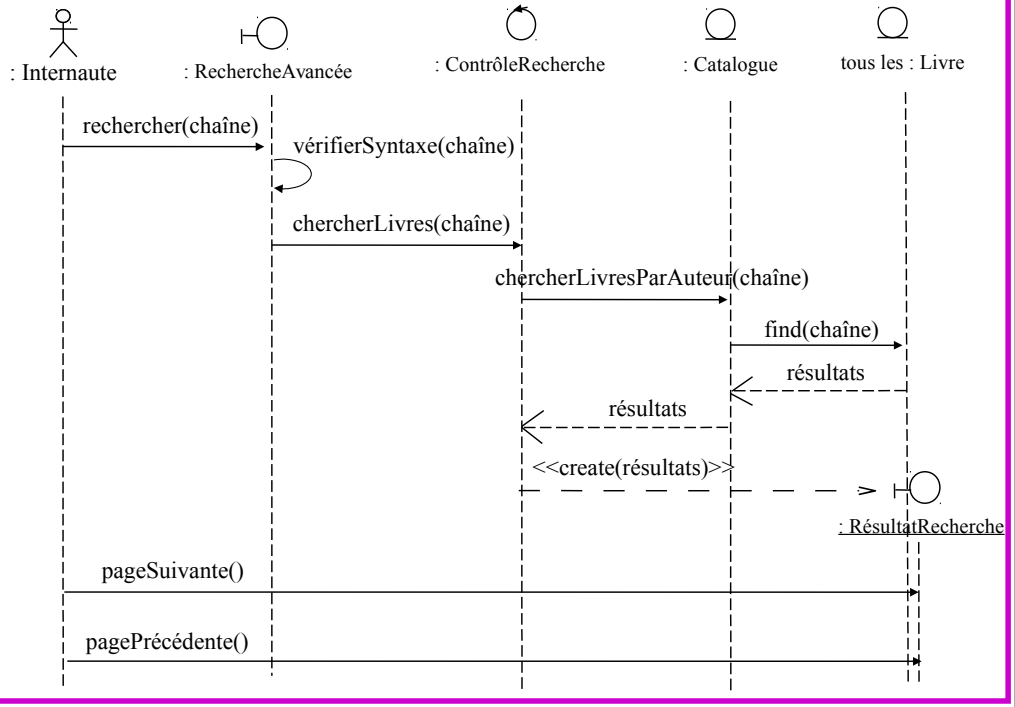
SOMMAIRE

- Introduction
- Diagrammes d'interaction
- **Diagrammes pour le site Web**
- Classes de conception préliminaire

DIAGRAMMES D'INTERACTION SITE WEB

Faire un diagramme de séquence
Rechercher des ouvrages

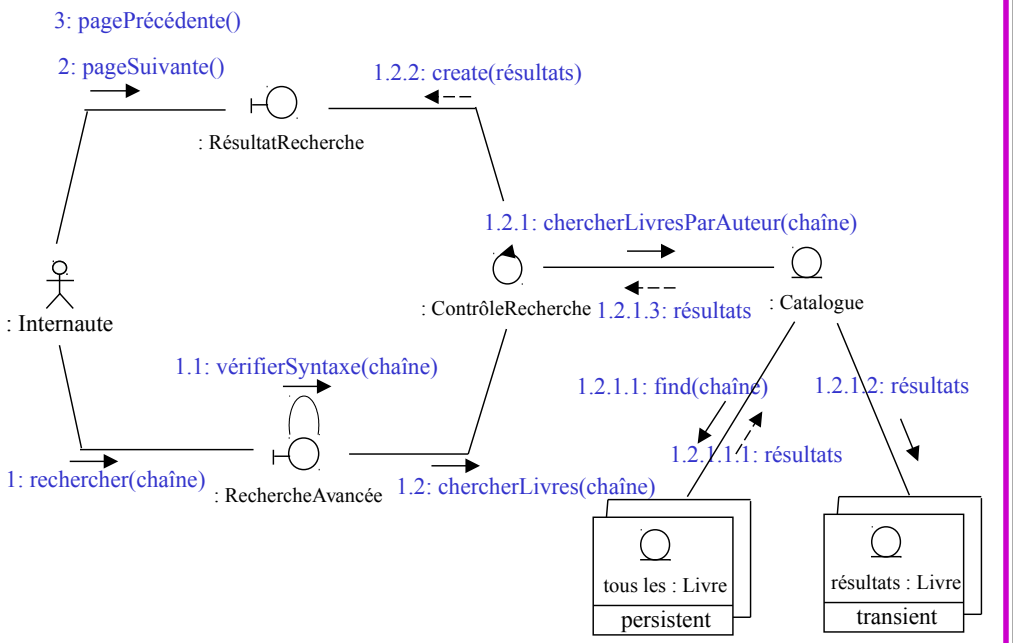
Scénario nominal de recherche avancée par nom d'auteur.

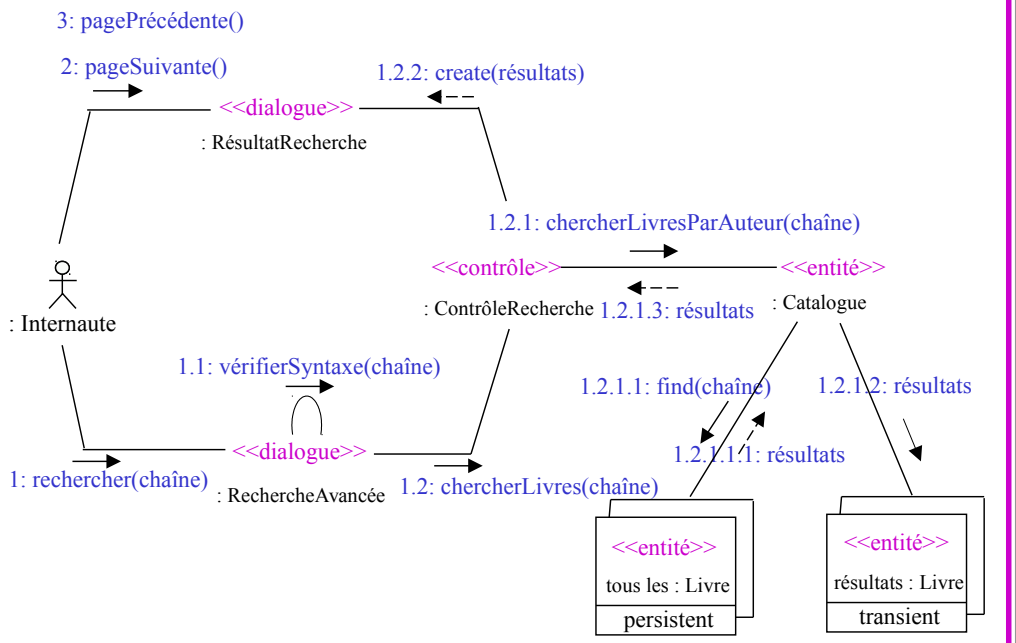


DIAGRAMMES D'INTERACTION SITE WEB

Faire un diagramme de communication
Rechercher des ouvrages

Scénario nominal de recherche avancée par nom d'auteur.





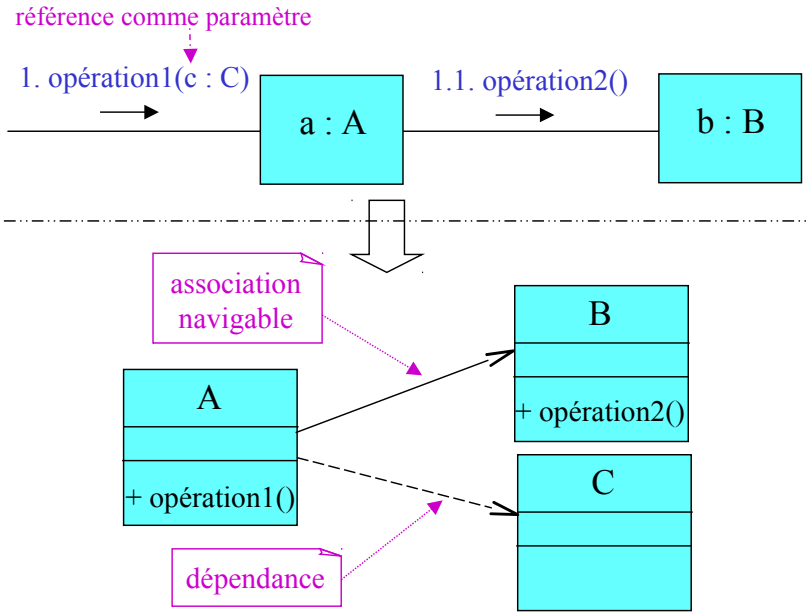
SOMMAIRE

CLASSES DE CONCEPTION PRÉLIMINAIRE

- On va affiner et compléter les diagrammes de classes déjà obtenus (*analyse du domaine + classes participantes*).

- Pour cela, on utilise les diagrammes d'interaction qu'on vient de réaliser pour :

- **ajouter** ou **préciser les opérations dans les classes**
  (*un message ne peut être reçu par un objet que si sa classe a déclaré l'opération publique correspondante*) ;
- **ajouter des types aux attributs,**
  aux paramètres et
  aux retours des opérations ;
- **affiner les relations entre classes**
  (*indication de navigabilité, dépendances*).



Liens durables ou temporaires

- Un **lien durable** entre objets va donner lieu à une association navigable entre les classes correspondantes.
Le lien entre l'objet a et l'objet b devient une association navigable entre les classes A et B.

- Un **lien temporaire** va donner lieu à une relation de dépendance.
Le fait que l'objet a reçoive en paramètre d'un message une référence sur un objet de la classe C induit une dépendance entre les classes A et C.

Les types ne sont pas encore ceux d'un langage de programmation, puisqu'on veut rester **indépendants des choix technologiques** à ce niveau.

On utilisera bien des noms comme *String* ou *Date* dans les diagrammes qui suivent ; ce ne sont pas les types Java, mais bien des types génériques.

Les multi-objets ne sont pas représentés en tant que classes collections de façon à rester le plus longtemps possible **indépendants du langage de programmation cible**.
De ce fait, les opérations génériques sur les collections (comme *find* ou *add*) n'apparaissent pas.

On ne fait pas figurer les opérations systématiques de création ou destruction d'instances ainsi que les accesseurs des attributs (*get* et *set*). Cela permet de garder des diagrammes lisibles et qui contiennent seulement les informations les plus importantes.

Diagramme d'activité

DIAGRAMME D'ACTIVITÉ

Diagramme comportemental

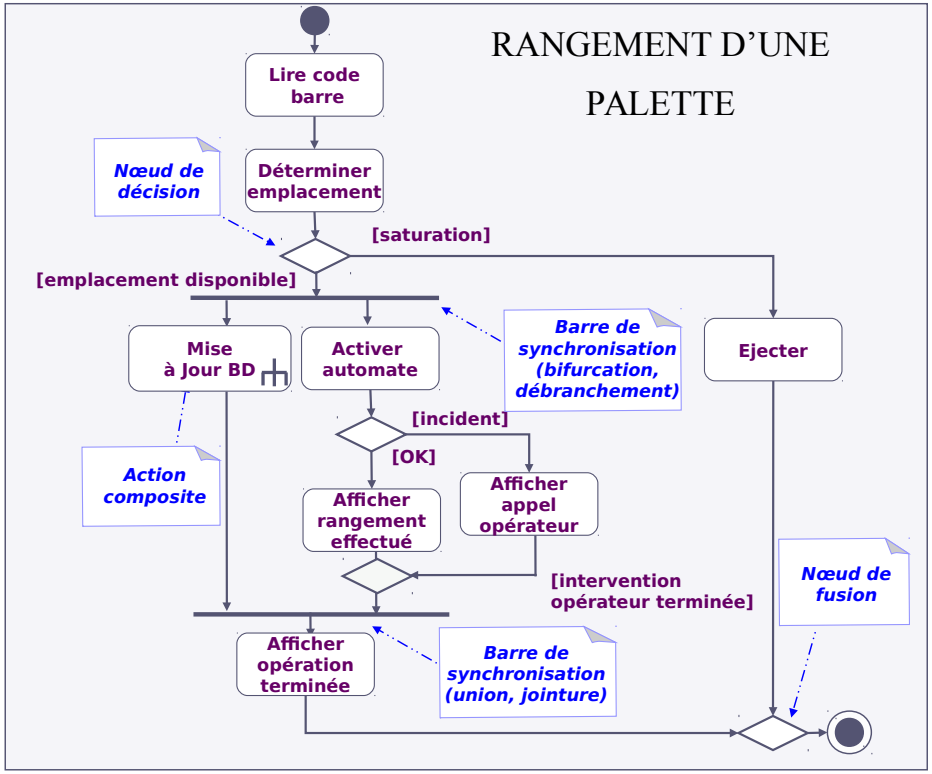
Enchaînement des actions au sein d'une activité à l'aide de flots de contrôle et d'objets

INTRODUCTION

- **Diagramme comportemental** d'UML.
- Permet de modéliser tout système ayant un comportement.
- Représentation **proche de l'organigramme**
- UML permet de représenter graphiquement :
  - ✓ le **comportement d'une méthode** ou
  - ✓ le **déroulement d'un cas d'utilisation**,

  à l'aide de diagrammes d'activité
  (*une variante des diagrammes d'états-transitions*).
- Réservé à des **opérations complexes** ou **sensibles**.

RANGEMENT D'UNE PALETTE



INTRODUCTION

- La **description d'un cas d'utilisation** par un diagramme d'activité correspond à sa **traduction algorithmique**.
- On se concentre ici sur les **activités telles que les voient les acteurs** qui collaborent avec le système dans le cadre d'un processus métier.
- La différence principale entre les **diagrammes d'interaction** et les **diagrammes d'activité** est que les premiers mettent l'accent sur le **flot de contrôle d'un objet à l'autre**, tandis que les seconds insistent sur le **flot de contrôle d'une action à l'autre**.

DIAGRAMME D'ACTIVITÉ

- Un **diagramme d'activité de base** contient un nombre restreint d'éléments :
  - ✓ des **actions**
  - ✓ des **transitions** entre actions, pouvant porter des **conditions**
  - ✓ des **branchements conditionnels** ou **décisions**
  - ✓ un **début** et une ou plusieurs **terminaisons** possibles.

DIAGRAMME D'ACTIVITÉ

- Dans un diagramme, **une seule action** est en cours (*ou active*) **à la fois**.

- Si plusieurs transitions sortent d'une même action, cela signifie qu'il y a un choix disponible pour l'utilisateur.
Une seule des transitions sera déclenchée lors d'une navigation particulière.

ACTIVITÉ / ACTION

râteau en bas à droite

- Une **activité** représente l'exécution d'un calcul complexe, alors qu'une action est un comportement élémentaire.

- Une **activité** définit un comportement décrit par un séquencement organisé d'unités dont les éléments simples sont les actions.

- Une **activité** peut être décomposée en **plusieurs actions**.

ACTIVITÉ / ACTION

- Une **action** est le plus petit traitement qui puisse être exprimé en UML. Cela représente une **étape simple de l'activité**.

- Une étape simple signifie que l'**action** n'est **pas décomposée** en éléments plus simples. Ne veut pas dire que l'action elle-même est simple.

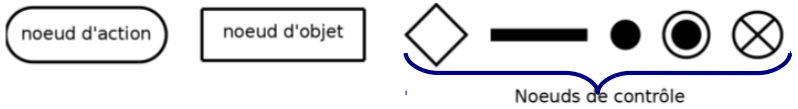
- **Exemples d'actions** :
  - fonctions mathématiques
  - appels à d'autres comportements (*activités*)
  - traitements de données

ACTIVITÉ / ACTION

- **Exemples plus concrets d'actions** :
  - calcul de TVA
  - envoi d'une commande à un fournisseur
  - constitution d'une liste d'articles à commander par le fournisseur car son stock s'épuise.

- Les **activités** sont destinées à être **réutilisées** au sein d'une application alors que les **actions** sont typiquement **spécifiques** et utilisées seulement dans le cadre **d'une activité** donnée.

NŒUDS D'ACTIVITÉ



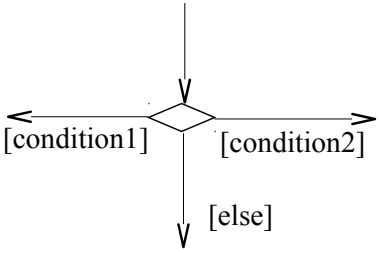
Nœuds de contrôle :

- nœud de décision ou de fusion
- nœud de bifurcation ou d'union
- nœud initial
- nœud final
- nœud final de flot

NŒUD DE DÉCISION

- Un **nœud de décision** est un nœud de contrôle qui permet de **faire un choix** entre plusieurs flots sortants.

- Il possède un arc entrant et plusieurs arcs sortants. Ces derniers sont généralement accompagnés de conditions de garde pour conditionner le choix.



NŒUD DE DÉCISION

- Si, quand le nœud de décision est atteint, aucun arc en aval n'est franchissable (*i.e. aucune condition de garde n'est vraie*), c'est que le **modèle est mal formé**.

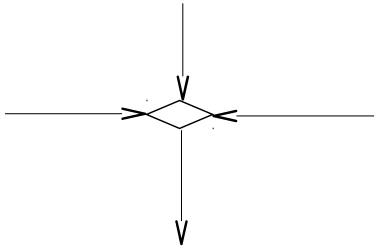
- L'utilisation d'une garde [else] est recommandée après un nœud de décision, car elle garantit un modèle bien formé. En effet, la condition de garde [else] est validée si et seulement si toutes les autres gardes des transitions ayant la même source sont fausses.

- Dans le cas où plusieurs arcs sont franchissables *(i.e. plusieurs conditions de garde sont vraies*), seul l'un d'entre eux est retenu et ce choix est non déterministe.

NŒUD DE FUSION

- Un **nœud de fusion** est un nœud de contrôle qui rassemble **plusieurs flots** alternatifs **entrants** en **un seul flot sortant**.

- Il n'est pas utilisé pour synchroniser des flots concurrents (*c'est le rôle du nœud d'union*), mais pour **accepter un flot parmi plusieurs**.

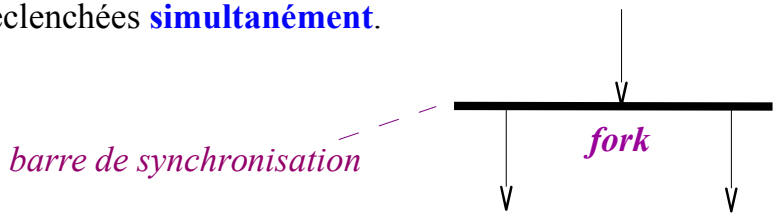


NŒUD DE BIFURCATION

- Un **nœud de bifurcation**, également appelé **nœud de débranchement**, est un nœud de contrôle qui sépare un flot en plusieurs flots concurrents.

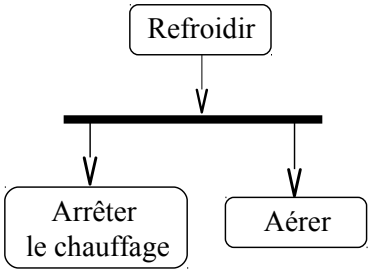
- Un tel nœud possède donc un arc entrant et plusieurs arcs sortants.

- Les **transitions** au départ d'une **barre de synchronisation** sont déclenchées **simultanément**.



NŒUD DE BIFURCATION

Exemple



- Pour refroidir, il faut **simultanément**
  - ✓ arrêter le chauffage et
  - ✓ ouvrir les fenêtres

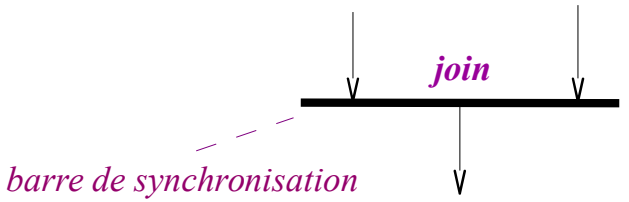
- Cela signifie également que l'**ordre d'exécution des actions n'a pas d'importance**.
- Le diagramme se borne à énoncer les règles de séquencement minimales qui doivent être appliquées.

NŒUD D'UNION

- Un **nœud d'union**, également appelé **nœud de jointure**, est un nœud de contrôle qui synchronise des flots multiples.

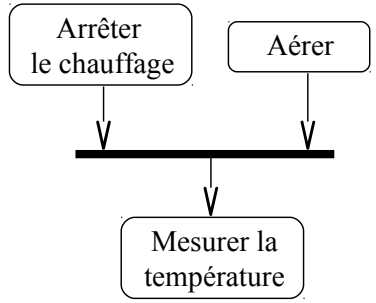
- Un tel nœud possède donc plusieurs arcs entrants et un seul arc sortant.

- Lorsque tous les arcs entrants sont activés, l'arc sortant l'est également.



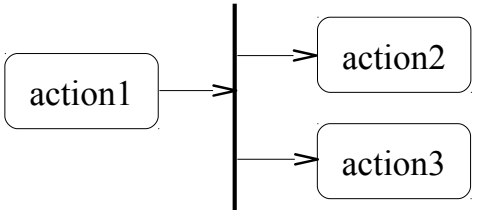
NŒUD D'UNION

- Inversement, un nœud d'union ne peut être franchie que lorsque toutes les transitions en entrée sur la barre ont été déclenchées.



- La mesure de température est effectuée une fois que :
  - ✓ le chauffage est arrêté **et**
  - ✓ la pièce aérée.

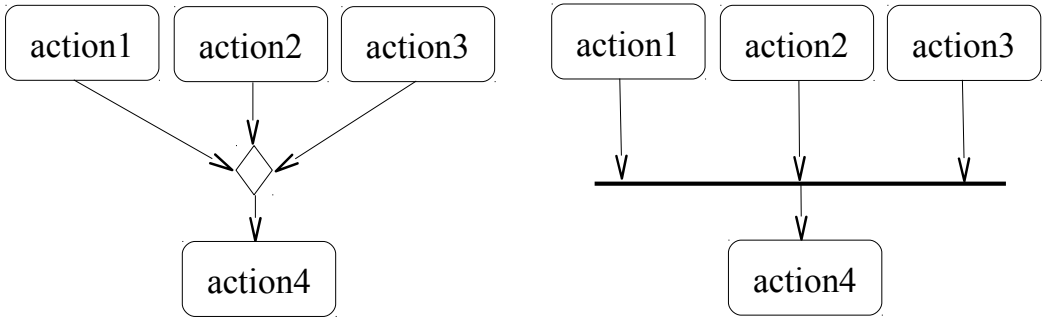
EXERCICE



L'action1 se termine. Dire ce qui est vérifié :

- ☐ L'action2 et l'action3 s'exécutent simultanément.
- ☐ L'action2 uniquement s'exécute.
- ☐ L'action2 s'exécute puis l'action3 s'exécute sans pour autant que l'action2 soit terminée.
- ☐ L'action3 uniquement s'exécute.
- ☐ L'action3 s'exécute puis l'action2 s'exécute sans pour autant que l'action3 soit terminée.

EXERCICE



Que se passe-t-il lorsque les actions suivantes se terminent ?

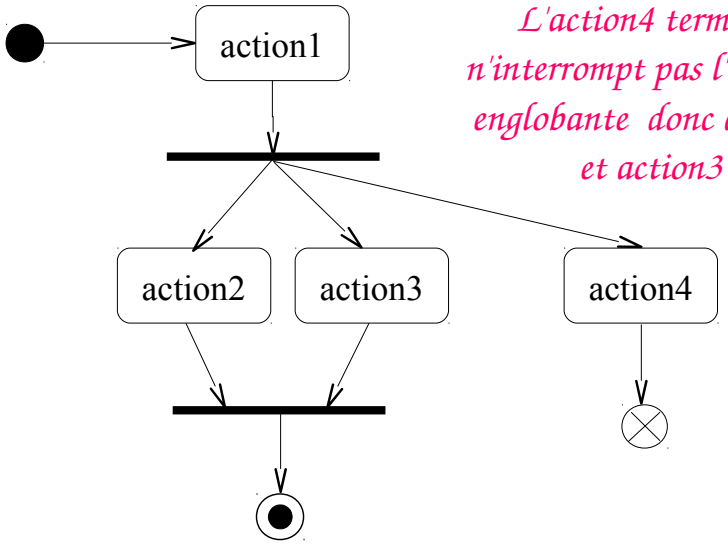
(action1), (action2), (action3),
(action1 et action2),
(action1 et action2 et action3)

NŒUD DE FIN DE FLOT

- Lorsqu'un flot d'exécution atteint un **nœud de fin de flot**, le **flot** en question est **terminé**, mais cette fin de flot n'a **aucune incidence sur les autres flots** actifs de l'activité enveloppante.

- Graphiquement, un nœud de fin de flot est représenté par un cercle vide barré d'un X.

NŒUD DE FIN DE FLOT

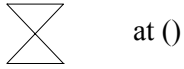


L'action4 terminée n'interrompt pas l'activité englobante donc action2 et action3

SIGNAUX

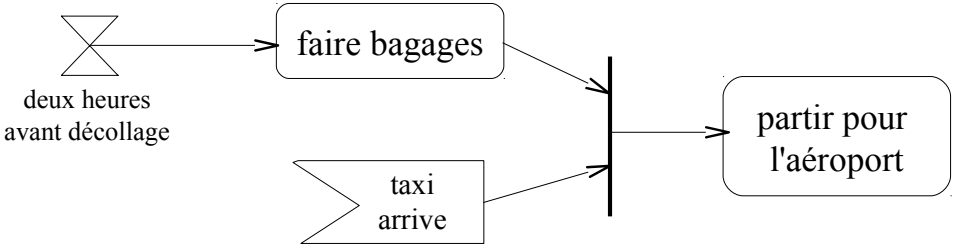
- Les actions peuvent répondre à des signaux et en émettre.

- On peut également avoir des signaux temporels.

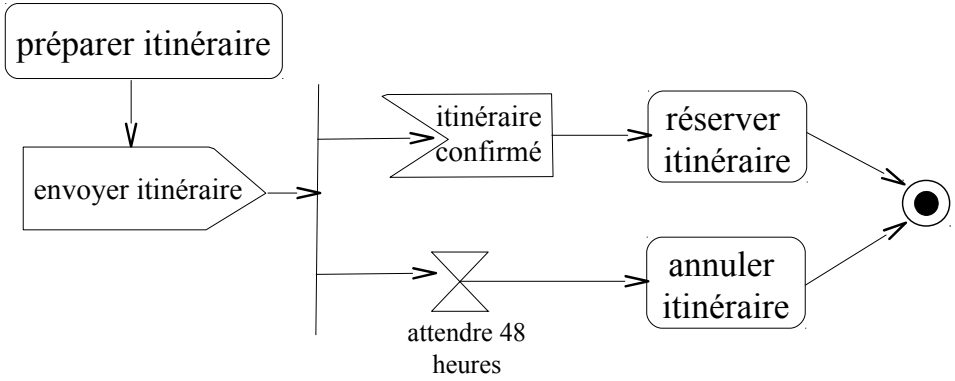


SIGNAUX

- Les **signaux temporels** sont liés au **passage du temps**.
- Ils peuvent par exemple indiquer des fins de mois dans une année fiscale ou des microsecondes dans un contrôleur temps réel.

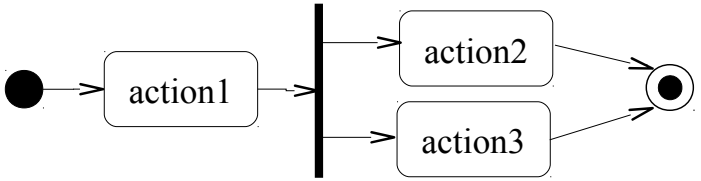


SIGNAUX



Les deux flots sont en compétition : le premier qui atteindra l'état final gagnera et mettra fin à l'autre.

EXERCICE



L'action2 se termine. Que se passe-t-il ?

ZONE D'EXPANSION

- Une **zone** ou région d'**expansion** est utilisée lorsque la sortie d'une action déclenche plusieurs invocations d'une autre action.

- Une région d'expansion marque une zone dans laquelle des actions se déroulent une fois pour chaque élément d'une collection.

ZONE D'EXPANSION

- Une **zone** ou région d'**expansion** est un nœud dans une activité qui accepte un ensemble d'objets en entrée, les traite individuellement et finalement les retourne en sortie tous traités.

- Cela permet de représenter finement le traitement de chaque objet d'un ensemble, dans le cas d'un algorithme commun.

- La région d'expansion se représente comme la région interruptible, mais avec un port multiple (*quatre carrés au lieu d'un*).

ZONE D'EXPANSION

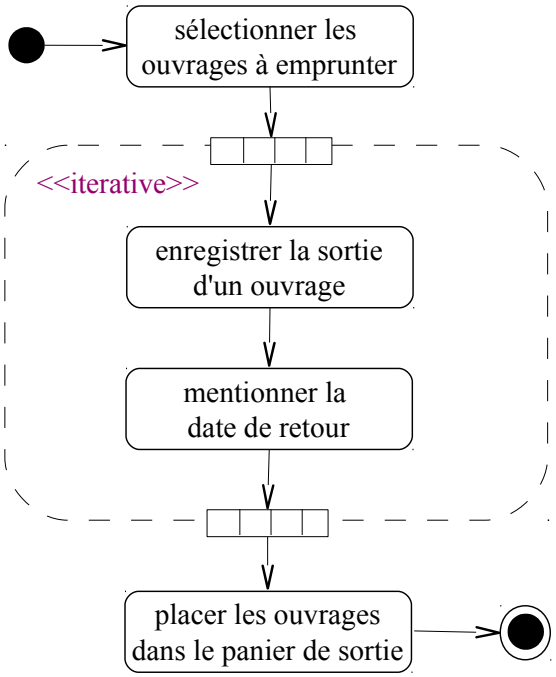
- **Zone d'expansion** = nœud d'activité structurée qui s'**exécute une fois pour chacun des éléments d'une collection**.

- La **collection** est **passée** à la zone d'expansion **à travers une punaise** d'expansion représenté graphiquement par une juxtaposition de punaises pour évoquer la notion de collection.

- Chaque exécution de l'activité interne **insère une valeur de retour dans la collection de sortie**.

- La zone d'expansion est stéréotypée <<parallel>>, <<iterative>> ou encore <<stream>> en fonction du comportement souhaité pour l'ordonnancement des exécutions de l'activité interne.

ad enregistrer la sortie des ouvrages



ZONE D'EXPANSION

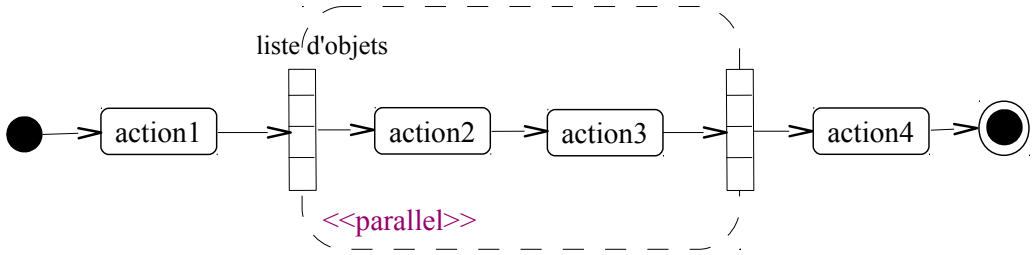
- **<<parallel>>** (*parallèle*) : les objets de l'ensemble sont traités en parallèle.
adaptée quand les exécutions de l'activité interne sont indépendantes et peuvent être réalisées dans un **ordre indifférent, voire simultanément**.

- **<<iterative>>** (*séquentielle*) : les objets de l'ensemble sont traités séquentiellement.
impose que les différentes exécutions de l'activité interne soient réalisées séquentiellement dans l'**ordre des éléments de la collection**.

ZONE D'EXPANSION

- **<<stream>>** (*continu*) : les objets de l'ensemble sont traités comme un seul flot, un peu comme sur une chaîne de montage. Chaque objet n'a pas à attendre que le traitement de l'objet précédent soit complètement terminé.
utilisé quand la **collection est un flux de données**.

EXERCICE



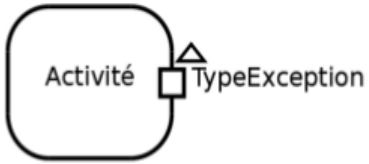
On suppose que la liste des objets se compose des objets o1 et o2.

Donnez l'ordre d'exécution des actions.

EXCEPTIONS

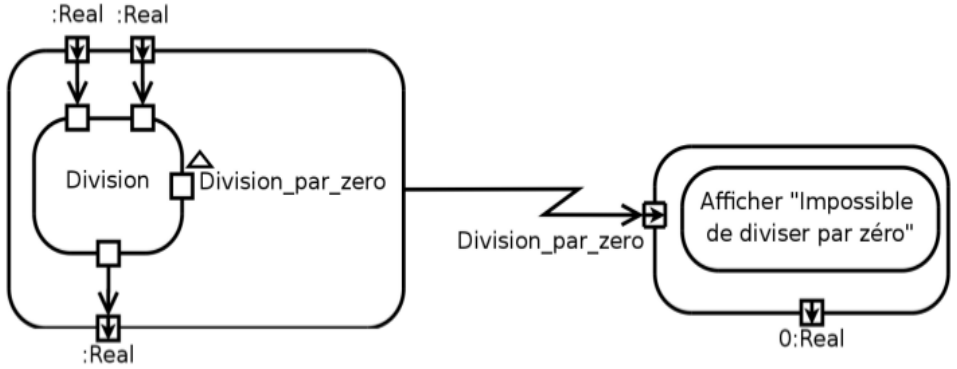
- Une **exception** est générée quand une **situation anormale entrave le déroulement nominal** d'une tâche.

- Elle peut être **générée automatiquement** pour signaler une erreur d'exécution (*débordement d'indice de tableau, division par zéro, . . .*), ou être **soulevée explicitement par une action** (*RaiseException*) pour signaler une situation problématique qui n'est pas prise en charge par la séquence de traitement normale.



Exemple : division par zéro

GESTIONNAIRE D'EXCEPTION

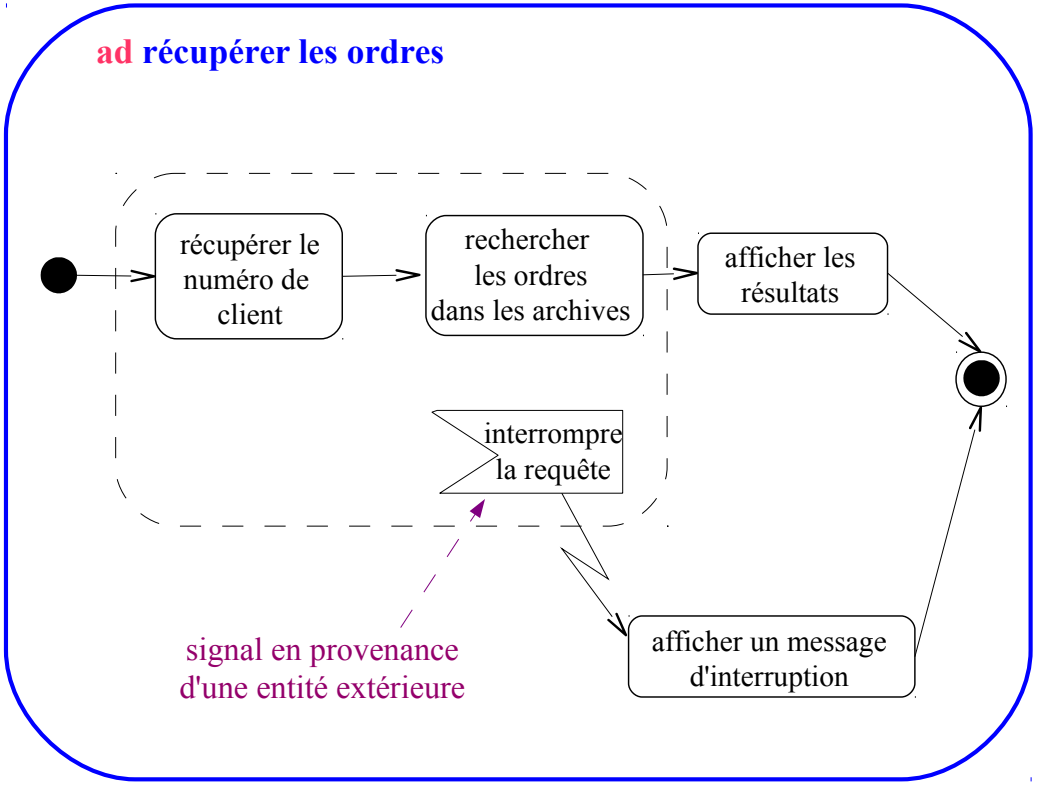


RÉGION INTERRUPTIBLE

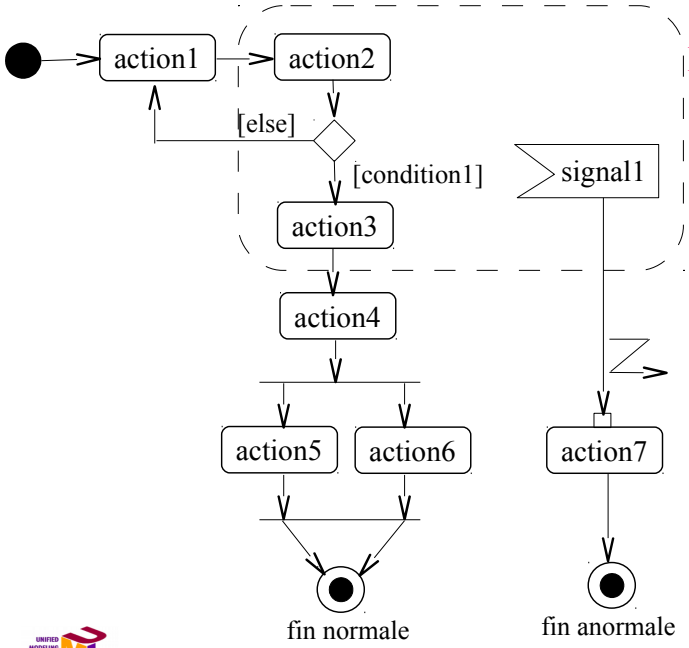
- Il peut arriver qu'on veuille **terminer** seulement **un sous-ensemble des actions** pour une raison particulière.

- UML propose le concept de **région interruptible** pour modéliser cette situation.

- Elle est représentée graphiquement par un rectangle aux bords arrondis en pointillés qui englobe un sous-ensemble des actions de l'activité.



EXERCICE

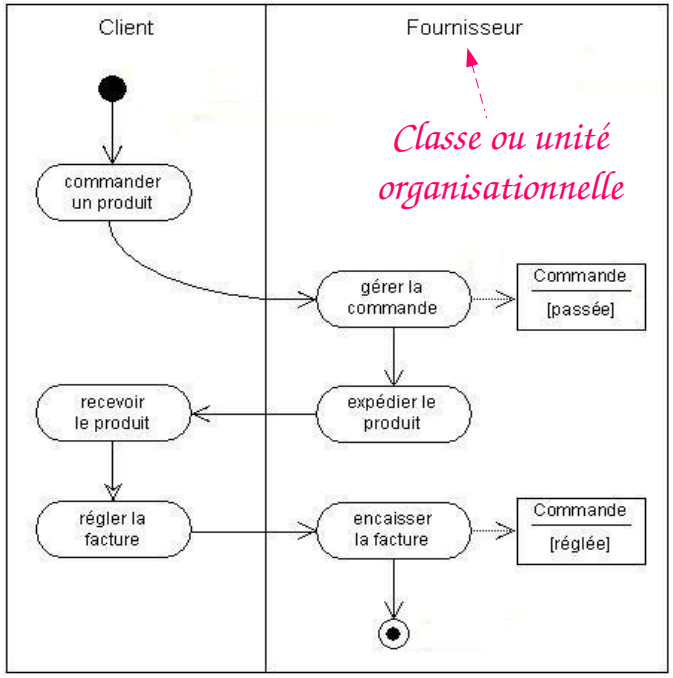


Le signal signal1 survient. Que se passe-t-il dans chacune des actions suivantes ?

action1
action2
action3
action4

PARTITION

- Les diagrammes d'activité **indiquent ce qui se passe mais ne disent pas qui fait quoi**.
- En termes de programmation, cela signifie qu'ils ne précisent pas quelle classe est responsable de chaque action.
- En termes de modélisation des processus métier, ils ne permettent pas de savoir quelle partie de l'organisation exécute chaque action.
- Ce n'est pas nécessairement un problème : il est souvent plus significatif de se concentrer sur ce qui est fait que sur les responsables de telle ou telle partie d'un comportement.



Les diagrammes d'activité peuvent être découpés en partitions, pour montrer qui est responsable de quoi, au sein d'un mécanisme ou d'une organisation.

Utile pour modéliser des processus d'entreprise

PARTITION

- Afin d'organiser un diagramme d'activité selon les différents responsables des actions représentées, il est possible de définir des "*couloirs d'activité*" ou "*lignes d'eau*".
- Les **partitions**, appelées aussi **couloirs d'activité** ou **lignes d'eau** (*swimlane*), permettent d'**organiser les nœuds d'actions** dans un diagramme d'activité en opérant des regroupements.
- Les partitions n'ont pas de signification bien arrêtée, mais correspondent souvent à des unités d'organisation du modèle.

PARTITION

- Par exemple, on peut les utiliser pour spécifier la classe responsable d'un ensemble tâche. Dans ce cas, la classe est responsable de l'implémentation du comportement des nœuds inclus dans ladite partition.

- Les nœuds d'une activité peuvent être organisés en partitions verticales ou horizontales permettant de séparer les responsabilités.

- C'est particulièrement utile pour modéliser des processus d'entreprise.