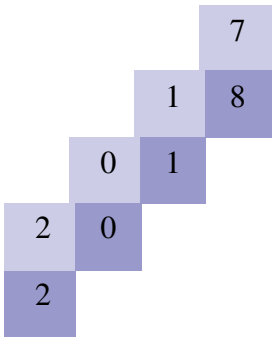


Les concepts de base

Langage UML

Démarches Associées

Les patrons de conception

Diagramme d'états / transitions

Enseignant : Dr. Mhamed SAIDANE

Diagramme d'états-transitions

Introduction

- Les diagrammes d'états-transitions d'UML décrivent le comportement interne d'un objet à l'aide d'un automate à états finis.
- Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements (invocations de méthode).
- Le diagramme d'états-transitions est le seul diagramme UML, à offrir une vision complète de l'ensemble des comportements de l'élément auquel il est attaché.
- Concrètement, un diagramme d'états-transitions est un graphe qui représente un automate à états finis, c'est-à-dire une machine dont le comportement des sorties ne dépend pas seulement de l'état de ses entrées, mais aussi d'un historique...

Diagramme d'états-transitions

Introduction

- Exemple :
  - Cet automate possède deux états (Allumé et Eteint) et deux transitions correspondant au même évènement : l'appuie sur un bouton d'éclairage domestique.
  - Cet automate à états finis illustre le fonctionnement d'un interrupteur dans une maison. Lorsque l'on appuie sur un bouton d'éclairage, la réaction de l'éclairage associé dépendra de son état courant (de son historique)
    - s'il la lumière est allumée, elle s'éteindra,
    - si elle est éteinte, elle s'allumera

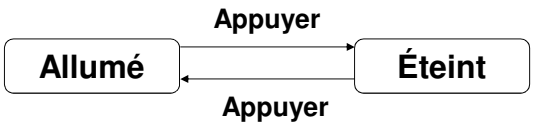


Diagramme d'états-transitions

- Un diagramme d'états-transitions rassemble et organise les états et les transitions d'un classeur donné.
- Le modèle dynamique du système comprend plusieurs diagrammes d'états-transitions. Il est souhaitable de construire un diagramme d'états-transitions pour chaque classeur (qui, le plus souvent, est une classe) possédant un comportement dynamique important.
- Attention ! Un diagramme d'états-transitions ne peut être associé qu'à un seul classeur.
- Le diagramme d'états présente les états dans lesquels un objet peut se trouver, les ***transitions*** entre ses ***états*** et le point de ***départ*** et de ***fin*** d'une séquence de changement d'état.

↳ **C'est donc une vue synthétique du fonctionnement dynamique d'un objet.**

Diagramme d'états-transitions

Représentation

- Le rectangle arrondi représente un ***état*** , les flèches des ***transitions***, le cercle plein représente le point de ***départ*** et le cercle mi-plein le point de ***fin***.

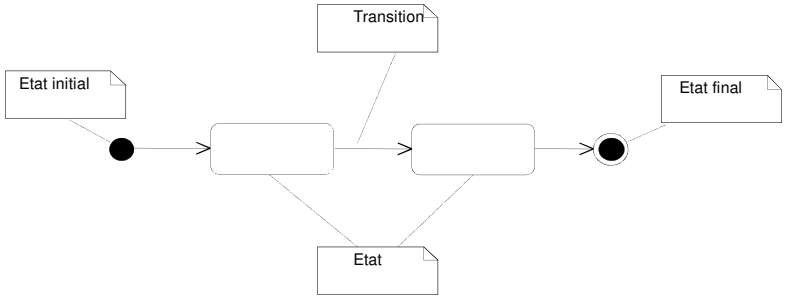


Diagramme d'états-transitions

Etat initial et Etat Final

- L'état initial est un pseudo-état qui définit le point de départ par défaut pour l'automate ou le sous-état . Lorsque un objet est créé, il entre dans l'état initial.

- L'état final est un pseudo-état qui indique que l'exécution de l'automate ou du sous-état est terminé.

Diagramme d'états-transitions

Évènement

- Les transitions d'un diagramme d'états-transitions sont déclenchés par des **évènements déclencheurs**.
- L'évènements déclencheur est indiqué à côté de la flèche représentant ta transition.
- **Remarques :**
  - Une transition peut avoir lieu en réponse à un événement.
  - Une transition peut avoir lieu après qu'un état ait fini une activité. On parle d'une transition automatique.

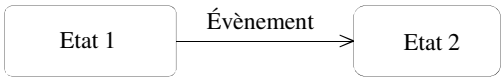


Diagramme d'états-transitions

Évènement

- Un événement se produit à un instant précis et est dépourvu de durée. Quand un événement est reçu, une transition peut être déclenchée et faire basculer l'objet dans un nouvel état.
- On peut diviser les événements en plusieurs types explicites et implicites :
  - signal,
  - appel,
  - changement et temporel.

Diagramme d'états-transitions

Évènement

- **Le type signal « Signal »**
  - Un signal est destiné explicitement à véhiculer une communication asynchrone à sens unique entre deux objets.
  - L'objet expéditeur n'attend pas que le destinataire traite le signal pour poursuivre son déroulement. La réception de signal est un événement pour le destinataire.
- **Le type appel « Call »**
  - Les événements d'appel sont des **méthodes** déclarées au niveau du **diagramme de classes**.
  - La syntaxe d'un signal ou d'un évènement d'appel (call) est la suivante :

<nom_événement> ([<paramettre> : <type> [; <paramettre> : <type> ...]])

Diagramme d'états-transitions

Évènement (suite)

- **Le type changement « Change »**
  - Un événement de changement est généré par la satisfaction (i.e. passage de faux à vrai) d'une expression booléenne sur des valeurs d'attributs. Il s'agit d'une manière déclarative d'attendre qu'une condition soit satisfaite.
  - Un événement de changement est évalué continuellement jusqu'à ce qu'il devienne vrai, et c'est à ce moment-là que la transition se déclenche.
  - La syntaxe d'un événement de changement est la suivante :

when (<condition_booléenne>)

Diagramme d'états-transitions

Évènement (suite)

- **Le type temporel « After» ou « when »**
  - Les événements temporels sont générés par le passage du temps. Ils sont spécifiés soit de manière absolue (date précise), soit de manière relative (temps écoulé). Par défaut, le temps commence à s'écouler dès l'entrée dans l'état courant.
  - La syntaxe d'un événement temporel spécifié de manière relative est la suivante :

    after (<durée>)
  - Un événement temporel spécifié de manière absolue est défini en utilisant un événement de changement :

    when (date = <date>)

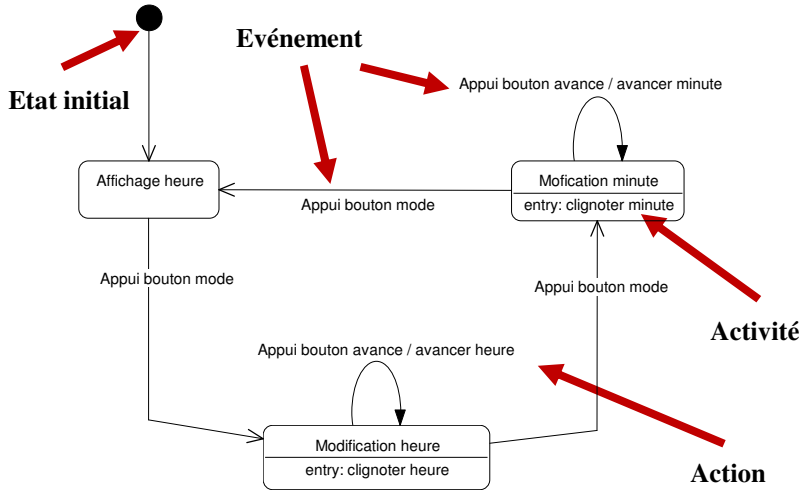
Diagramme d'états-transitions

Transition

- Une transition définit la réponse d'un objet à l'occurrence d'un événement. Elle lie, généralement, deux états E1 et E2 et indique qu'un objet dans un état E1 peut entrer dans l'état E2 et exécuter certaines activités, si un événement déclencheur se produit et que la condition de garde est vérifiée.
- La syntaxe d'une transition est la suivante :

  [<événement>]['[' <garde> ']'] ['/' <activité>]
- La garde désigne une condition qui doit être remplie pour pouvoir déclencher la transition.
- L'activité désigne des instructions à suivre au moment du déclenchement de la transition (on parle aussi de tir d'une transition).
  - *La façon de spécifier l'activité à réaliser est laissée libre (langage naturel ou pseudo-code).*

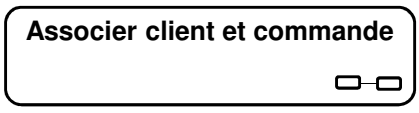
Exemple : notation complète «Fonctionnement d'une montre digitale »



État composite

- Un état dit composite, opposition à un état dit « simple », est décomposé en deux ou plusieurs sous-états.
  - Tout état ou sous-état peut ainsi être décomposé en sous-états imbriqués sans limite à priori de profondeur.
  - Un état composite est un état décomposé en régions contenant chacune un ou plusieurs sous-états.

- Une notation abrégée permet d'indiquer qu'un état est composite et que sa définition est donnée sur un autre diagramme.



Notation abrégée

Diagramme d'états-transitions

État composite : Exemple « transmission d'une voiture »

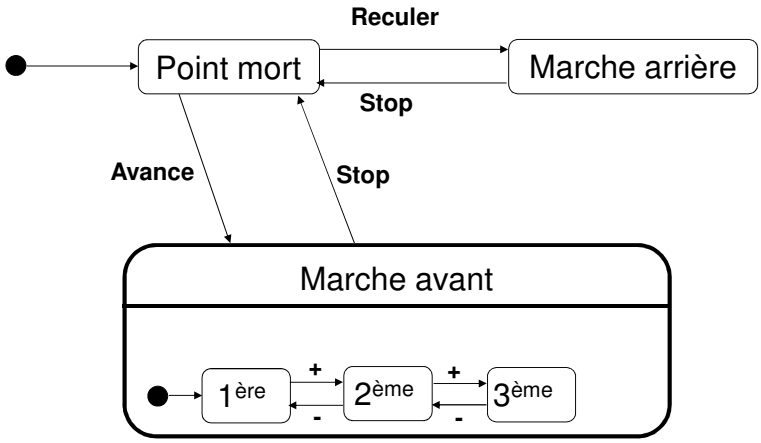


Diagramme d'états-transitions

État concurrent

- On peut représenter plusieurs automates s'exécutant indépendamment.
- Un objet peut être alors simultanément dans plusieurs **états concurrents**.

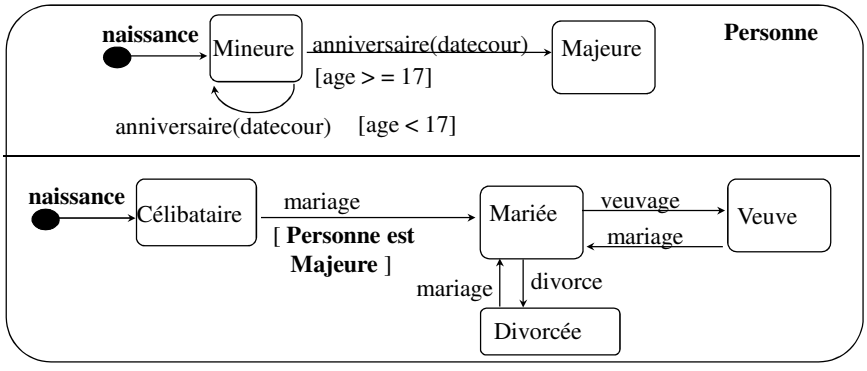
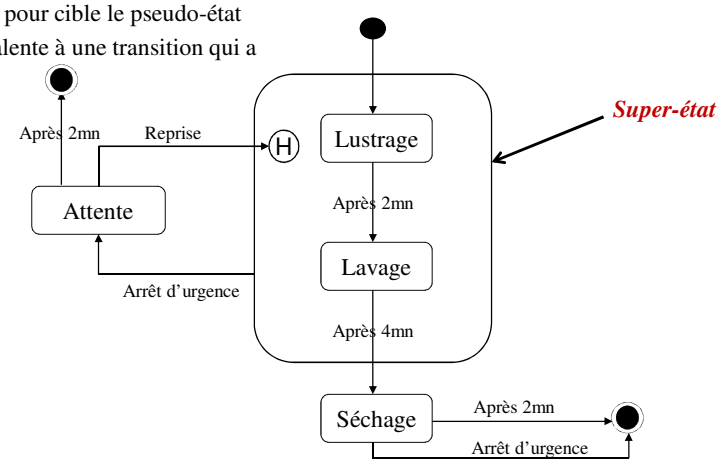


Diagramme d'états-transitions

Historique

- Un pseudo-état historique est noté par un **H** cerclé.
- Une transition ayant pour cible le pseudo-état historique est équivalente à une transition qui a pour cible le dernier état visité dans la région contenant le H.



EPI ECOLE PLURIDISCIPLINAIRE INTERNATIONALE

7
1 8
0 1
2 0
2

Diagramme de composants

Les concepts de base

Langage UML

Démarches Associées

Les patrons de conception

Diagramme de composants

- Un composant est une partie physique du système. Il peut être du code, un fichier, une table, etc.
- Un composant logiciel est une unité logicielle autonome au seins d'un système global ou d'un sous-système.
- Un composant peut être une implémentation logicielle d'une ou de plusieurs classes.
- Un composant est éventuellement connecté à d'autres composants via des dépendances ou des compositions.

Pourquoi les composants?

- Ils fournissent aux clients une vue de la structure du système final
- Ils fournissent aux développeurs la structure à mettre en œuvre
- Ils facilitent la documentation technique
- Ils favorisent la **réutilisation**

Diagramme de composants

- Un objet cache ce qu'il fait aux autres objets et au monde externe. Cet objet doit présenter une « facette » au monde externe pour que les autres objets puissent lui demander d'exécuter ses opérations. Cette facette est dite « interface ».
- Un composant peut réaliser un ensemble d'interfaces qui définissent le comportement offert à d'autres composants.
- Un composant peut accéder aux services d'autres composants à travers leurs interfaces.
- Un composant peut être réutilisé dans un autre système si ce dernier peut accéder aux interfaces du composant réutilisé.

Diagramme de composants

- Un diagramme de composants décrit les composants, leurs interfaces et leurs dépendances dans l'environnement de réalisation.

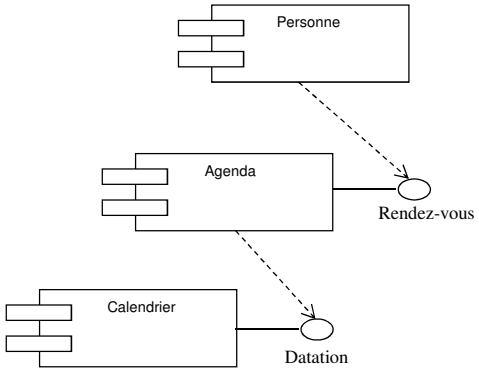
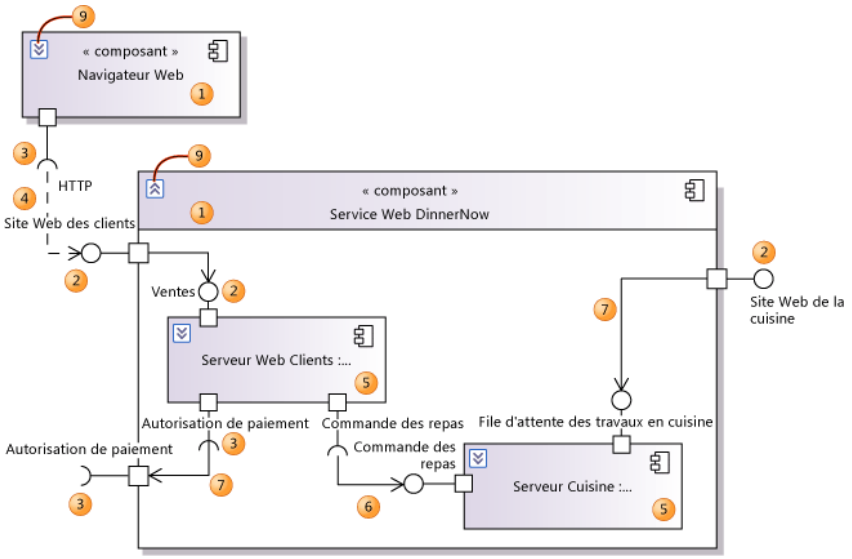


Diagramme de composants

- Les composants peuvent être décomposés en sous-composants.
- Les liens entre composants sont spécifiés à l'aide de **dépendances entre leurs interfaces**.
  - Le « câblage interne » d'un composant est spécifié par les **connecteurs de délégation**. Un tel connecteur connecte un port externe du composant avec un port de l'un de sous-composants internes.

Diagramme de composants



EPI ECOLE PLURIDISCIPLINAIRE INTERNATIONALE

7
1 8
0 1
2 0
2

Les concepts de base

Langage UML

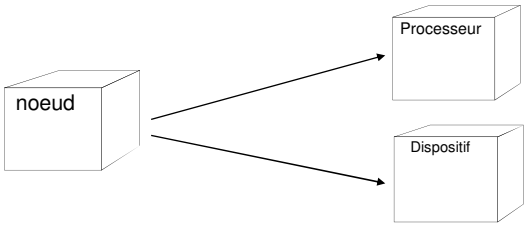
Démarches Associées

Les patrons de conception

Diagramme de déploiement

Diagramme de déploiement

- En dernier lieu le système doit s'exécuter sur des ressources matérielles dans un environnement matériel particulier.
- Uml permet de représenter un environnement d'exécution ainsi que des ressources physiques grâce aux **diagrammes de déploiement**.
- Deux types de nœud sont possibles :
  - Nœud « processeur » : c'est un nœud qui peut exécuter un composant.
  - Un nœud « dispositif » (imprimante ou écran) : un nœud qui ne peut pas exécuter un composant.



Exemple 1

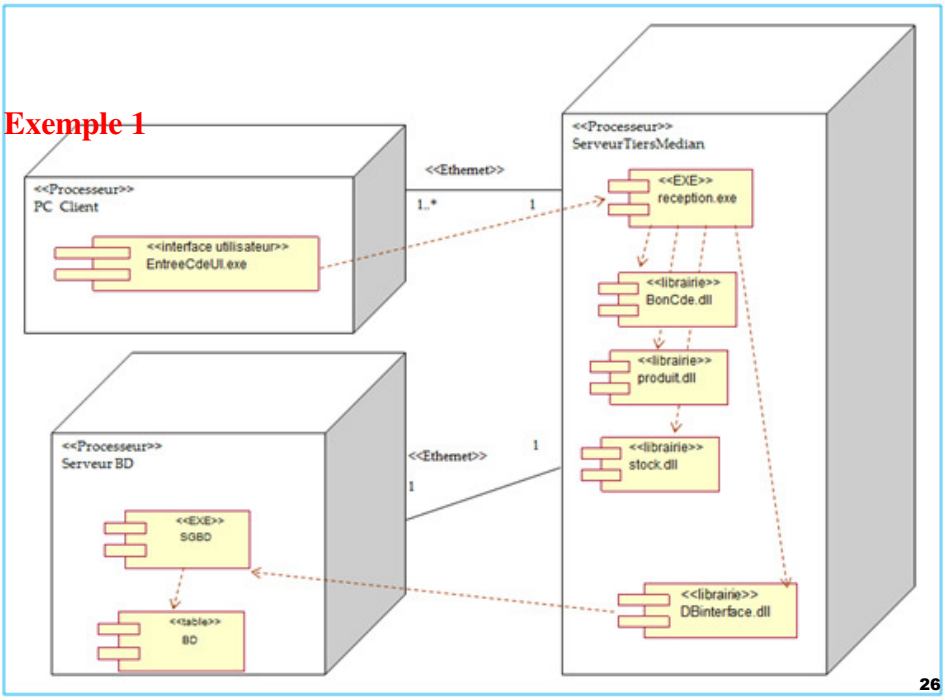
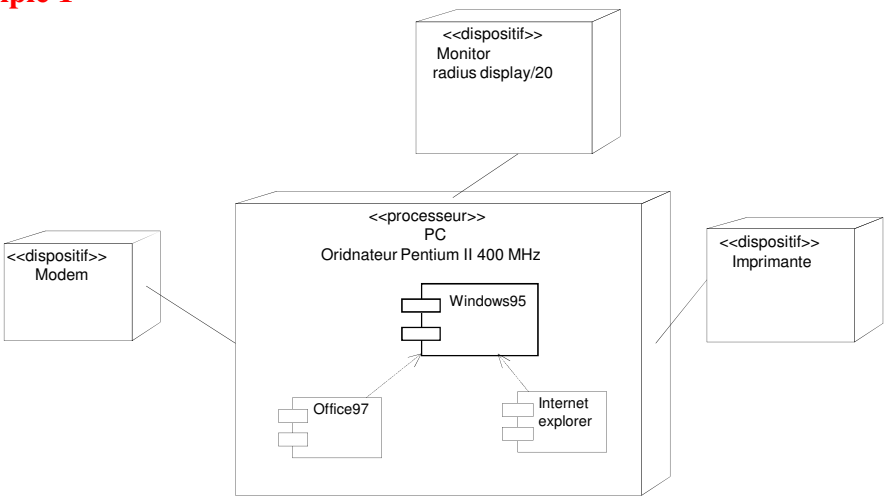


Diagramme de déploiement

Exemple 1



ECOLE PLURIDISCIPLINAIRE INTERNATIONALE

7
1 8
0 1
2 0
2

UML et méthodes

1. Des besoins au code avec UML : **une méthode minimale**

Les concepts de base

Langage UML

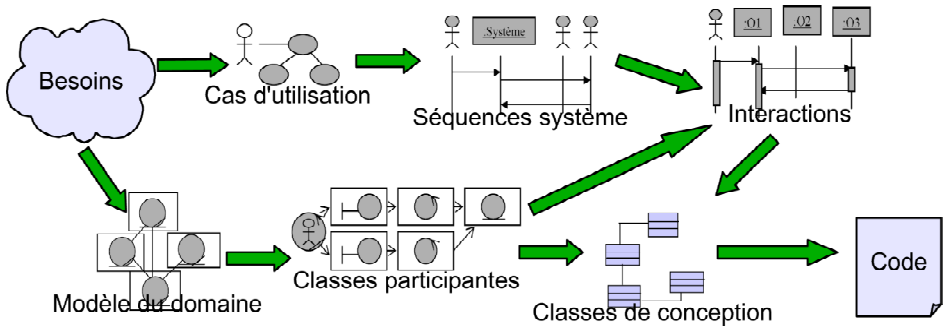
Démarches Associées

Les patrons de conception

Méthode minimale

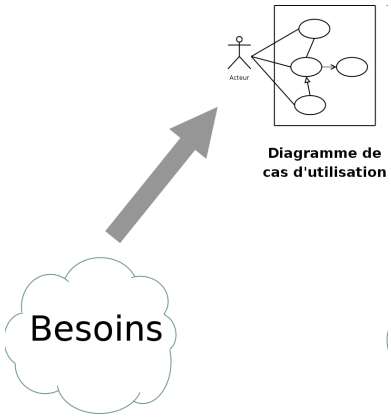
- Objectif
  - Résoudre 80% des problèmes avec 20% d'UML.
- Proposition d'une méthode archi-minimale :
  - Très nettement moins complexe que RUP ;
  - Adaptée pour des projets modestes ;
  - Minimum vital pour qui prétend utiliser un peu UML.
- Inspirée de : « UML 2 - Modéliser une application web », Pascal Roques, Editions Eyrolles (2006)

Méthode minimale



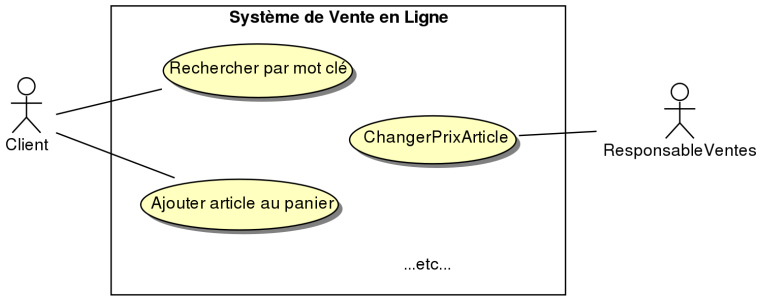
Méthode minimale

- **1. Identification des besoins et spécifications des fonctionnalités**
  - **1.1 Identification et représentation des besoins : diagramme de cas d'utilisation**



Méthode minimale

- **Cas d'utilisation**
  - Comment définir les besoins ?
    ① Identifier les limites du système ;
    ② Identifier les acteurs ;
    ③ Identifier les cas d'utilisation ;
    ④ Structurer les cas d'utilisation en packages ;
    ⑤ Ajouter les relations entre cas d'utilisation ;
    ⑥ Classer les cas d'utilisation par ordre d'importance



Méthode minimale

- Modèle du domaine

Le modèle du domaine est constitué d'un ensemble de classes dans lesquelles aucune opération n'est définie.

- Le **modèle du domaine** décrit les **concepts invariants** du domaine d'application.
  - Exemple : Pour un logiciel de gestions de factures, on aura des classes comme Produit, Client, Facture...
    - Peu importe que le logiciel soit en ligne ou non.
    - Peu importent les technologies employées.

Méthode minimale

- Modèle du domaine (*suite*)
  - Etapes de la démarche :
    ① Identifier les concepts du domaine ;
    ② Ajouter les associations et les attributs ;
    ③ Généraliser les concepts ;
    ④ Structurer en packages : structuration selon les principes de cohérence et d'indépendance.

- Les concepts du domaine peuvent être identifiés directement à partir de la connaissance du domaine ou par interviews des experts métier.

Méthode minimale

- Exemple de classement :

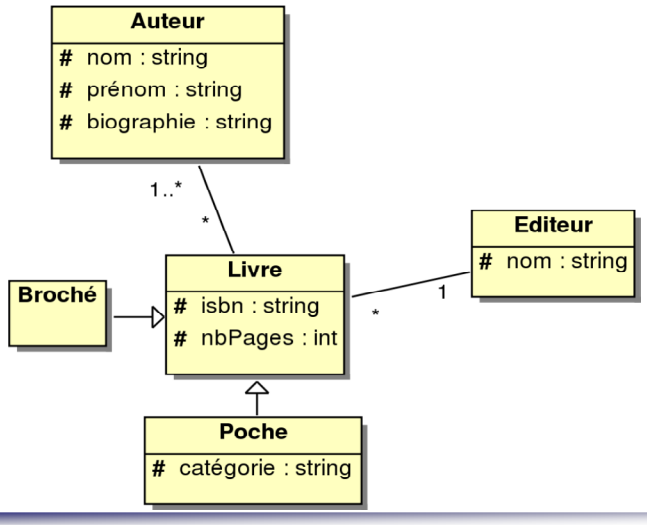
  Dans le cadre d'une approche itérative et incrémentale, il faut affecter un degré d'importance et un coefficient de risque à chacun des cas d'utilisation pour définir l'ordre des incréments à réaliser.

Cas d'utilisation	Priorité	Risque
Ajouter article au panier	Haute	Moyen
Changer prix Article	Moyen	Moyen
Rechercher par mots-clés	Bas	Moyen
Etc…	Etc…	Etc…

- Un tel classement permet de déterminer les cas d'utilisation centraux en fonction :
  - De leur priorité fonctionnelle ;
  - Du risque qu'il font courir au projet dans son ensemble
- Les fonctionnalités des cas les plus centraux seront développées en priorité

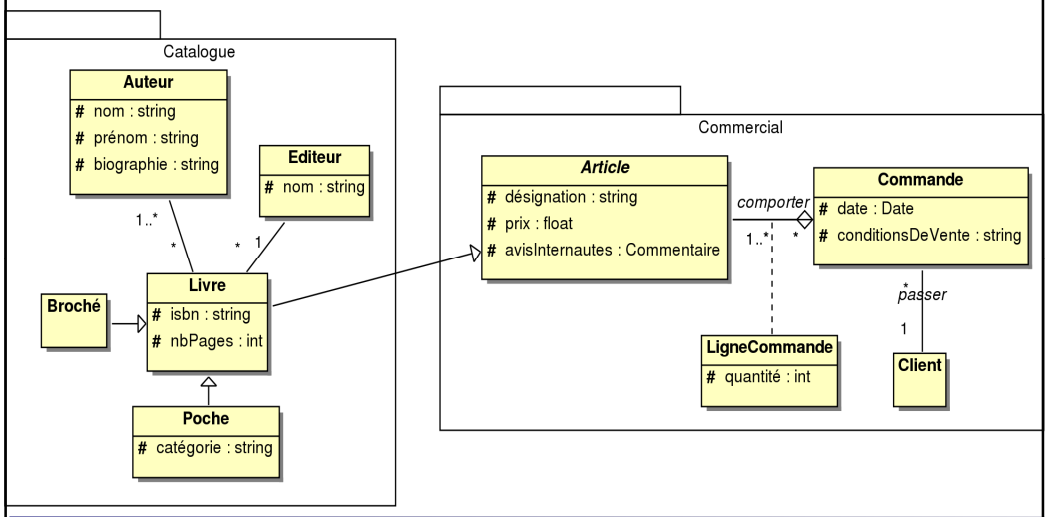
Méthode minimale

- Exemple de modèle du domaine



Méthode minimale

- Structuration en packages



Méthode minimale

- Séquences système

Les séquences système formalisent les descriptions textuelles des cas d'utilisation, en utilisant des diagrammes de séquence

- Construire les **Diagrammes de Séquences Système** (DSS) implique souvent la mise à jour des cas d'utilisation à la lumière des réflexions que nous inspirent la production des DSS
- Les DSS permettent de spécifier les **opérations système**
- Le système est considéré comme un tout :
  - On s'intéresse à ses interactions avec les acteurs ;
  - On utilise une classe Système qui à part les acteurs donnera lieu à la seule ligne de vie des DSS.

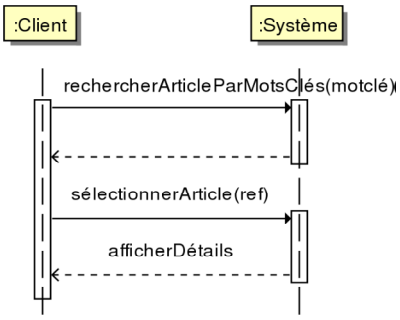
Méthode minimale

- Séquences système (*suite*)

Un nouveau DSS est produit pour chacun des cas d'utilisation.

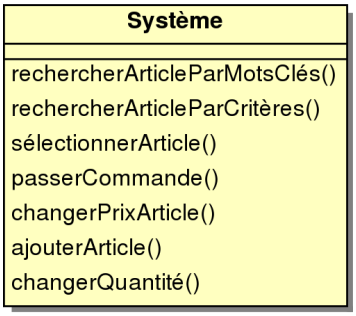
- Les DSS sont parfois très simples mais ils seront enrichis par la suite

Exemple de diagramme de séquence système



Méthode minimale

- Opérations système



...etc...

- Les opérations système sont des opérations qui devront être réalisées par l'une ou l'autre des classes du système.
- Elles correspondent à tous les messages qui viennent des acteurs vers le système dans les différents DSS.

Méthode minimale

- Classes participantes
  - Pour chaque cas d'utilisation, on définit les classes d'analyse mises en œuvre pour sa réalisation effective.
  - Typologie des classes d'analyse :
    - Les **classes métier** (ou entités) représentent les objets métier. Elles correspondent aux classes du modèle du domaine.
    - Les **classes de dialogue** sont celles qui permettent les interactions entre les acteurs et l'application.
    - Les **classes de contrôle** permettent d'abstraire les fonctionnalités du système :
      - Elles font le lien entre les classes dialogue et les classes métier.
      - Elles permettent de contrôler la cinématique de l'application, c'est-à-dire l'ordre dans lequel les choses doivent se dérouler.

Méthode minimale

- Diagrammes de classes participantes

Le **Diagramme des Classes Participantes** (DCP) est un diagramme de classes décrivant toutes les classes d'analyse.

Le DCP est une version enrichie du modèle du domaine, auquel on adjoint les classes d'interaction et de contrôle.

- A ce point du développement, seules les classes de dialogue ont des opérations, qui correspondent aux opérations système, c'est-à-dire aux messages échangés avec les acteurs, que seules les classes de dialogues sont habilitées à intercepter ou à émettre.

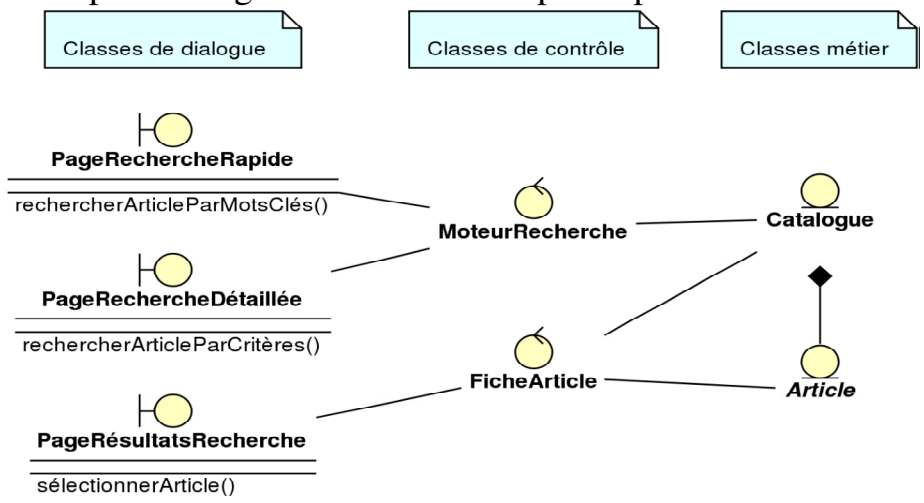
Méthode minimale

- Diagrammes de classes participantes (*suite*)

- Architecture en couches :
  - Les dialogues ne peuvent être reliés qu'aux contrôles ou à d'autres dialogues (en général, associations unidirectionnelles).
  - Les classes métier ne peuvent être reliées qu'aux contrôles ou à d'autres classes métier.
  - Les contrôles peuvent être associés à tous les types de classes..

Méthode minimale

- Exemple de diagrammes de classes participantes



Méthode minimale

- Diagrammes d'interaction
  - Dans les diagrammes de séquence système, le système était vu comme une boîte noire (ligne de vie « Système »).
    - On sait maintenant de quels objets est composé le système (diagramme de classes participantes).
    - Le système n'est plus une boîte noire.

Chaque diagramme de séquence système donne lieu à un diagramme d'interaction. Il y en a donc autant que de cas d'utilisation.

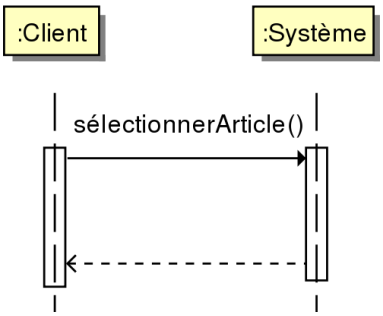
En plus des interactions du système avec l'extérieur, les Diagrammes d'Interaction montrent les interactions internes provoquées

Méthode minimale

- Diagrammes d'interaction (*suite*)
  - Les DSS sont repris mais l'objet Système est éclaté pour donner le détail des classes d'analyse :
    - **Les lignes de vie correspondent aux classes participantes**.

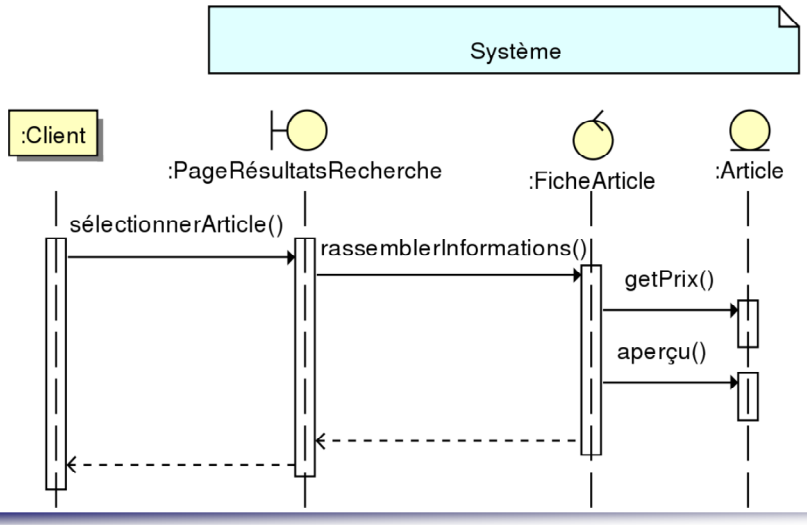
Méthode minimale

- Des séquences système aux interactions internes



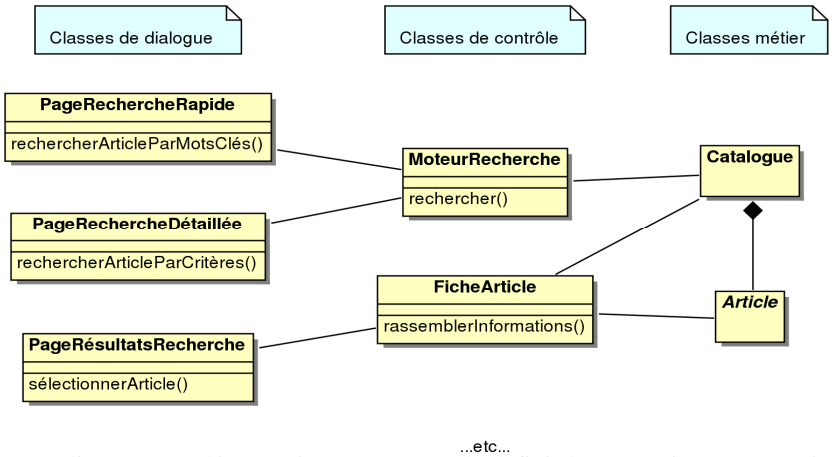
Méthode minimale

- Des séquences système aux interactions internes



Méthode minimale

- Diagramme des classes de conception



  - Les diagrammes d'interaction permettent de définir les opérations (nécessaires et suffisantes) des classes métier et de contrôle (messages synchrones).