Génie Logiciel

- Merise
- Modélisation objet
- UML

Ce support n'est qu'une adaptation des supports rédigés par **Isabelle VALEMBOIS** (Thales-services)

MERISE

- Les modèles de la méthode Merise :
  - Le modèle de flux
  - **Le modèle conceptuel de données**
  - Le modèle conceptuel de traitements
  - Le modèle organisationnel de traitements
  - Le modèle logique de données
  - Le modèle opérationnel des traitements
  - **Le modèle physique des données**
- En savoir plus sur Merise :
  - La méthode MERISE - H. Tardieu, A. Rochefeld, R. Colletti

MODELISATION OBJET (1/6)

- Principaux concepts objet :
  - Classe
    - *Type d'objet caractérisé par sa structure de données (attributs) et son comportement (méthodes)*
  - Objet
    - *Un objet représente un individu, une entité identifiable réelle ou conceptuelle avec un rôle bien défini dans le domaine du problème ou dans un système (instance de classe)*
  - Attribut
    - *Un attribut est une propriété, une caractéristique, une qualité inhérente ou distinctive d'un objet (exemples : la couleur ou l'immatriculation d'une voiture)*
  - Méthode
    - *Une méthode est une action qu'un objet effectue de sa propre initiative ou à la demande (réaction à un événement ou à un envoi de message). Ces méthodes décrivent les propriétés dynamiques d'un objet.*

MODELISATION OBJET (2/6)

- Principaux concepts objet :
  - Quelques méthodes "classiques" :
    - *Un **constructeur** est une méthode qui permet de construire et d'initialiser un objet (instanciation d'un objet)*
    - *Par opposition, un **destructeur** est une méthode qui permet de détruire un objet instancié*
    - *Un **accesseur** est une méthode qui permet de récupérer la valeur d'un attribut d'un objet (get… et is…)*
    - *Un **modificateur** est une méthode qui permet de modifier la valeur d'un attribut d'un objet (set…)*
    - *Un **observateur** est une méthode qui permet de retrouver des informations sur l'histoire (l'état) d'un objet*
    - *Un **itérateur** est une méthode qui permet d'appliquer à chaque partie d'un objet (par exemple dans le cas d'un objet de type collection) une action déterminée*

MODELISATION OBJET (3/6)

- Principaux concepts objet :
  - Association
    - *Une association permet de préciser une **relation** entre différents objets. Une association possède généralement un nom qui sert à décrire la nature de la relation*
    - *Une association est également caractérisée par le nombre d'objets pouvant être reliés par l'intermédiaire d'une instance d'association : **0..1, 0..n, 1..n, n..n***
  - Agrégation
    - *Une agrégation est une **relation binaire** entre deux objets qui spécifie une **inclusion** entre une partie et un tout. Une partie peut appartenir à d'autres agrégations et exister indépendamment*
  - Composition
    - *Une composition est un **type particulier d'agrégation** qui spécifie une inclusion entre une partie et un tout plus forte que l'agrégation : quand on supprime l'élément composite, il y a obligatoirement suppression des composants*

MODELISATION OBJET (4/6)

- Principaux concepts objet :
  - Héritage
    - *Mécanisme permettant à une classe d'objets de bénéficier de la structure de données et du comportement d'une classe "mère", tout en lui permettant de les affiner et ce, afin de prendre en compte les spécificités de la classe "fille", sans avoir cependant à redéfinir ce que les deux classes ont de commun*
  - Généralisation
    - *Dans le cas d'un héritage, on dit qu'une classe "mère" est une généralisation des propriétés de ses classes "fille"*
  - Spécialisation
    - *Dans le cas d'un héritage, on dit qu'une classe "fille" est une spécialisation des propriétés de sa classe "mère"*

MODELISATION OBJET (5/6)

- Principaux concepts objet :
  - Encapsulation (interface)
    - *Mécanisme permettant de dissimuler les détails du fonctionnement interne d'une classe aux autres classes*
  - Abstraction (classe abstraite)
    - *Mécanisme permettant la dissociation entre la déclaration d'une classe et son implémentation (interface)*
  - Polymorphisme
    - *Mécanisme permettant d'associer à un comportement, une implémentation différente en fonction de l'objet auquel on se réfère (par exemple : dessiner à l'écran un carré ou un cercle). L'émetteur n'a pas besoin de connaître la classe du receveur, seulement que la sémantique du message sera la même pour toutes les classes similaires (par exemple : la méthode toString en JAVA)*

MODELISATION OBJET (6/6)

- Principaux concepts objet :
  - Message
    - *Mécanisme caractéristique des langages objet. Le message est l'unique support de communication entre objets. L'arrivée d'un message provoque l'exécution d'une méthode de l'objet*
  - Type d'accès, portée, visibilité
    - *Public*
    - *Private*
    - *Protected*
  - Surcharge de méthode
    - *Mécanisme permettant de déclarer plusieurs constructeurs ou plusieurs fois une même méthode (même nom) dans une classe, à condition que tous aient une* ***signature différente*** *(valeur de retour et/ou paramètres différents)*

UML (1/24)

- UML (Unified Modeling Language) :
  - Auteurs
    - *James Rumbaugh, Grady Booch et Yvar Jacobson*
  - Objectifs
    - *Faciliter la communication entre les différents acteurs d'un projet*
    - *Faciliter la communication avec la machine*
    - *Documenter un projet de bout en bout*
    - *Spécifier et donc limiter les ambiguïtés*
    - *Construire (interpréter les diagrammes pour code)*
  - Définition
    - *Langage de description des objets permettant une modélisation rigoureuse des systèmes complexes*
    - *Langage Unifié pour la Modélisation objet*

UML (2/24)

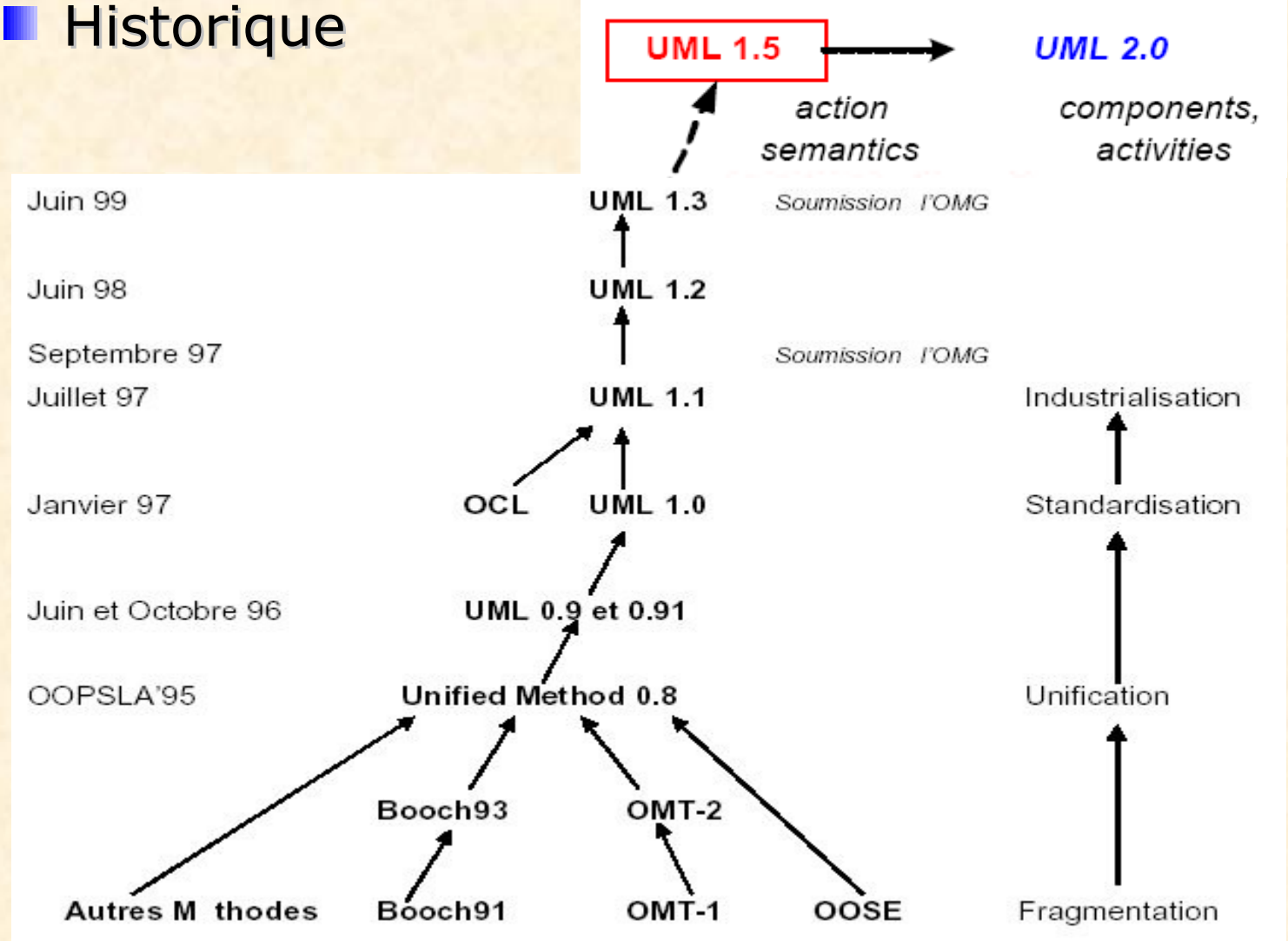
- UML n'est pas une mode :
  - Issu du terrain et d'un large consensus (industriels, informaticiens, méthodologistes), fruit d'un travail d'experts (HP, Microsoft…)
  - Norme riche (permet de spécifier du début à la fin) et ouverte (en constante évolution depuis sa création)
  - Indépendant de la cible
  - Adapté à la programmation orienté objet
  - Industrialisé (il existe de nombreux d'outils)
- Plus d'informations sur UML : http://uml.free.fr

UML (3/24)

- Avantages d'UML :
  - Formel et normalisé (garantit stabilité et performance d'un projet, **réduit les risques**)
  - Support de communication performant et éprouvé (permet de cadrer l'analyse et de faciliter la compréhension de représentations abstraites : c'est « l'esperanto » de l'analyse)
- Inconvénients d'UML :
  - Période d'apprentissage
  - Processus de production non couvert

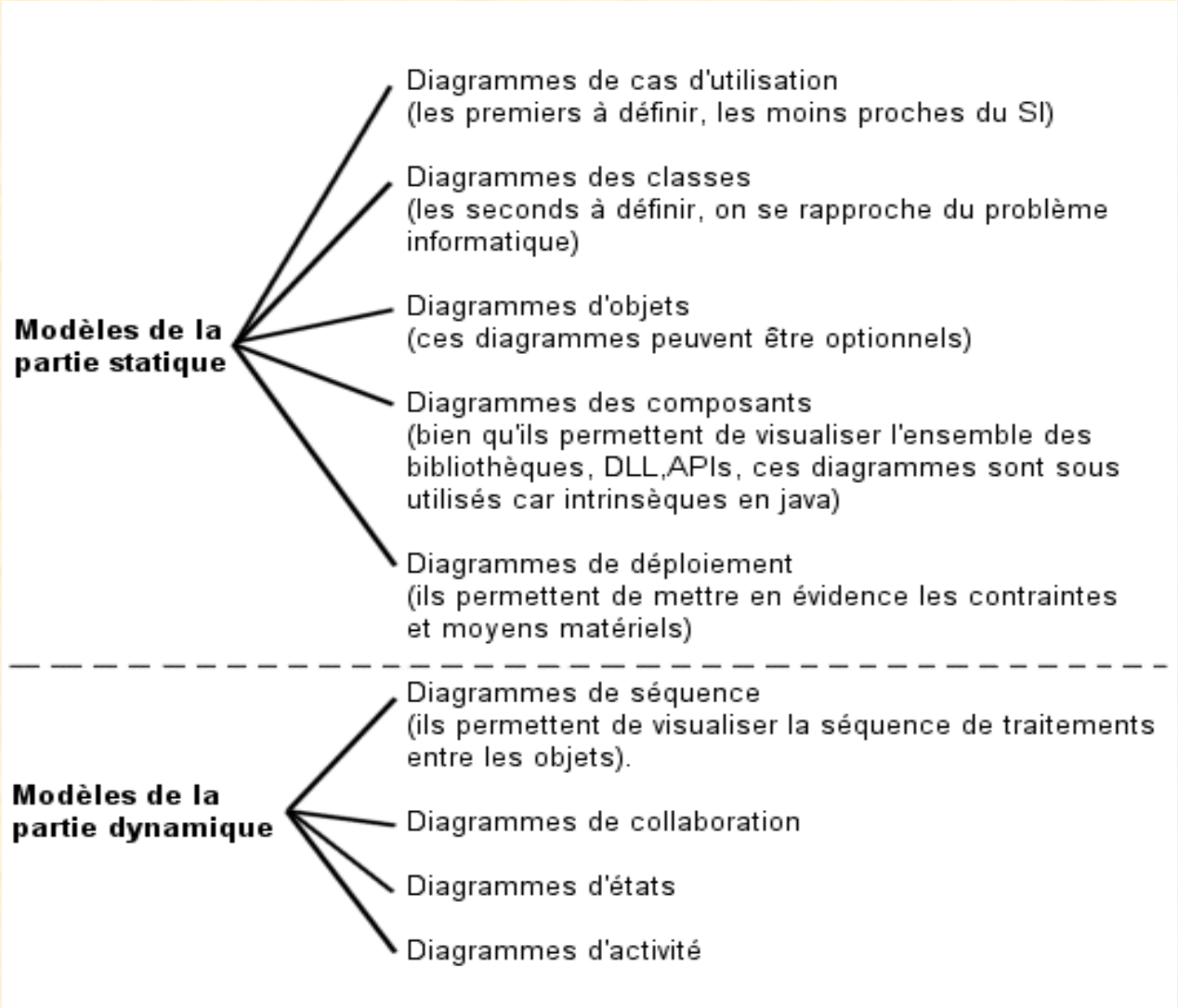
UML (4/24)

- Historique



UML (5/24)

- Diagrammes UML :



UML (5’/24)

- Découverte des besoins
  - Diagramme de cas d’utilisation : décrit les fonctions du système (point de vue de ses futurs utilisateurs -Jacobson)
  - Diagramme de séquence : représentation des interactions temporelles entre objets dans la réalisation d’une IHM
- Analyse
  - Diagramme de classes : structure des données
  - Diagramme d’objets : illustration
  - Diagramme collaboratif : représentation des interactions entre objets
  - Diagramme d’états : représentation du comportement des objets d’une classe en terme d’états et de transitions d’états
  - Diagramme d’activités : structure d’une opération en actions

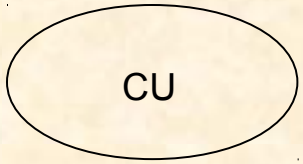
UML (5”/24)

- Conception
  - Diagramme de séquence : représentation des interactions temporelles entre objets dans la réalisation d’une opération
  - Diagramme de déploiement : description du déploiement des composants sur les dispositifs matériels
  - Diagramme de composants : architecture des composants physiques d’une application

UML (6/24)

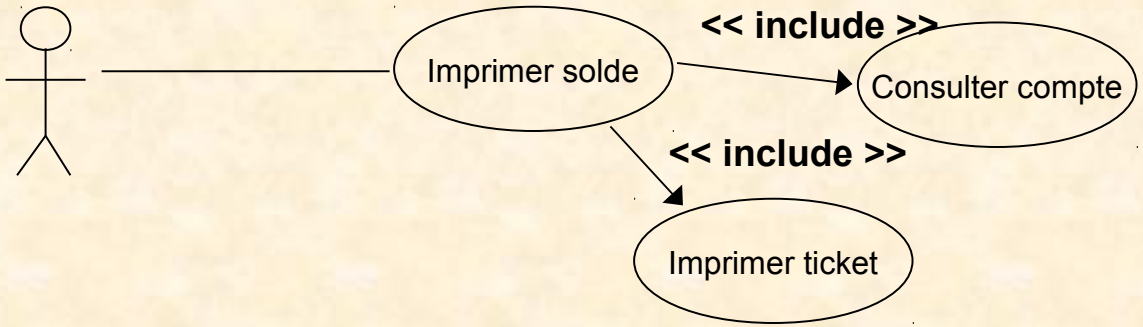
- Le diagramme de cas d'utilisation :
  - Il permet d'identifier et de décrire les utilisateurs du système (acteurs) et leur interaction avec le système
  - Démarche :
    - *Identification des acteurs*

    - *Identification des cas d'utilisation*

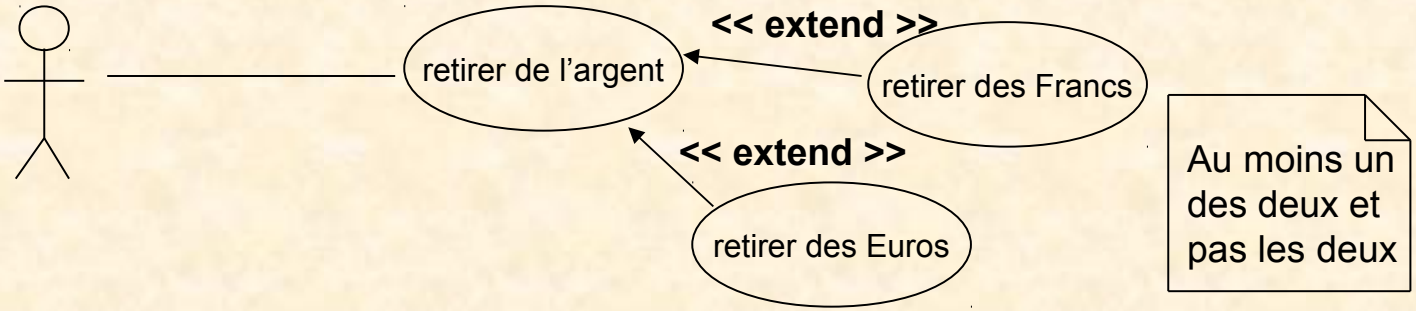


UML (7/24)

- Les relations dans un diagramme de cas d'utilisation :
  - Utilisation : la relation **include** indique qu'un CU utilise un autre CU. Si le CU source est vide, alors « décomposition »

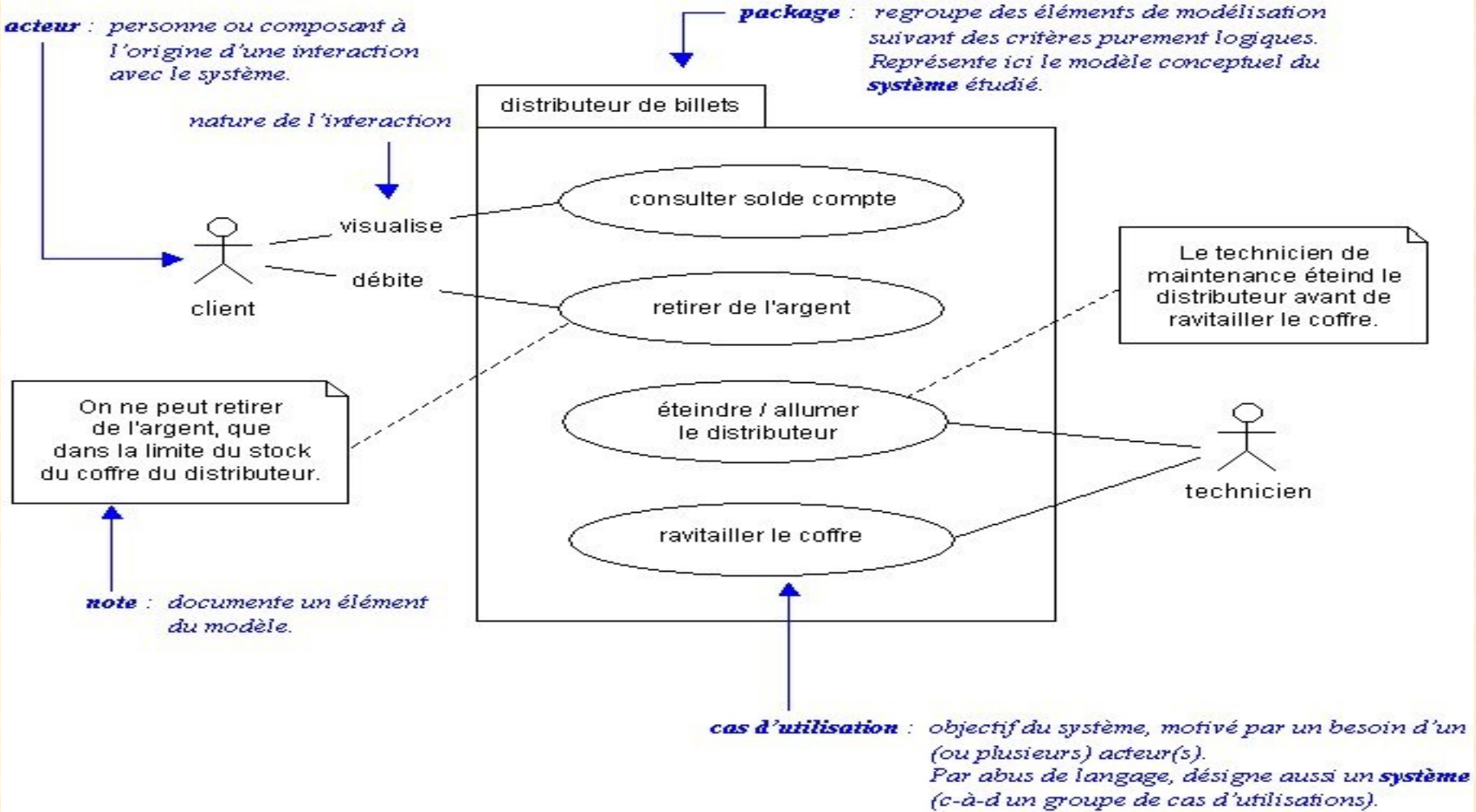


  - Extension : la relation **extend** du CU A vers le CU B indique que le CU B peut aussi faire A. Ceci permet aussi les CU avec garde.



UML (8/24)

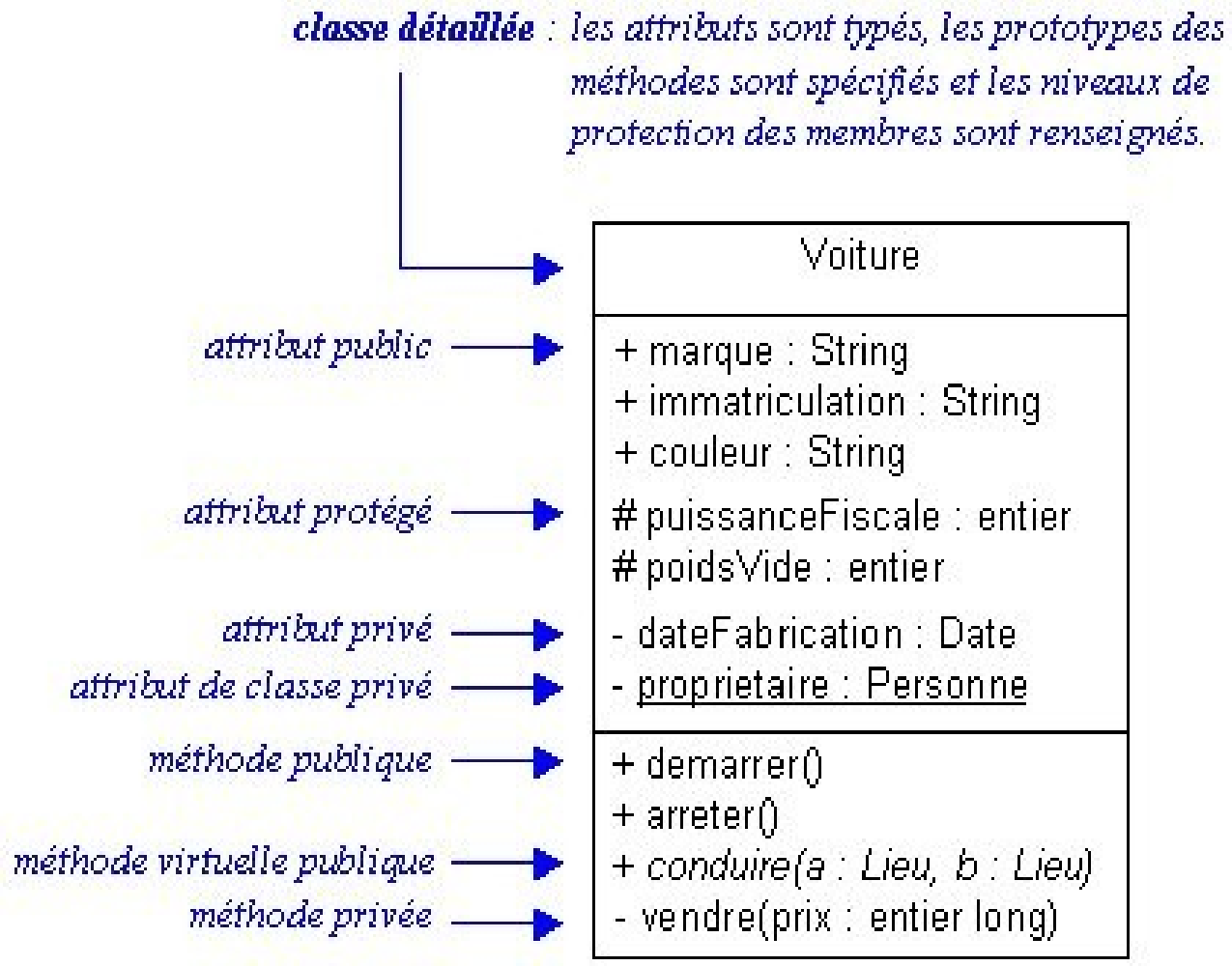
- Exemple de diagramme de cas d'utilisation :



UML (9/24)

- Les Classes

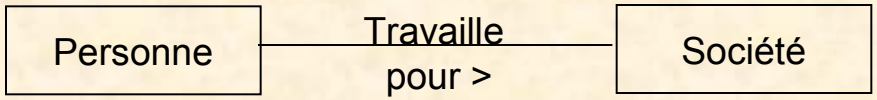
+ attributs dérivés ou calculables



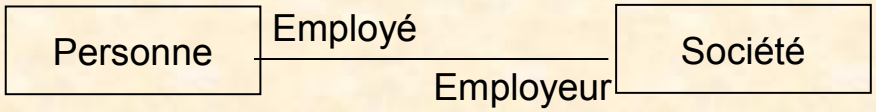
UML (10/24)

- Le diagramme de classes :
  - Il permet de décrire les classes d'un système ainsi que les associations et les relations d'héritage entre ces classes
    - *Les associations (forme verbale…)*

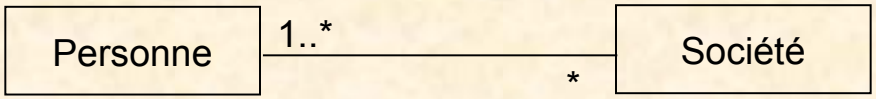
active,



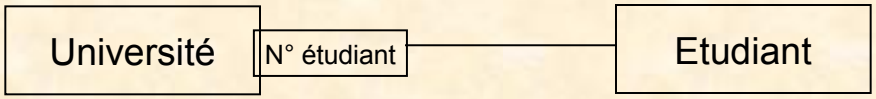
rôles,



cardinalité,

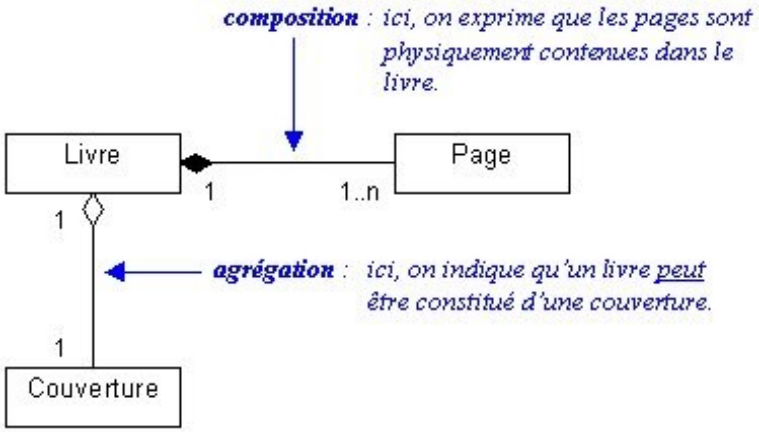


qualification ou restriction

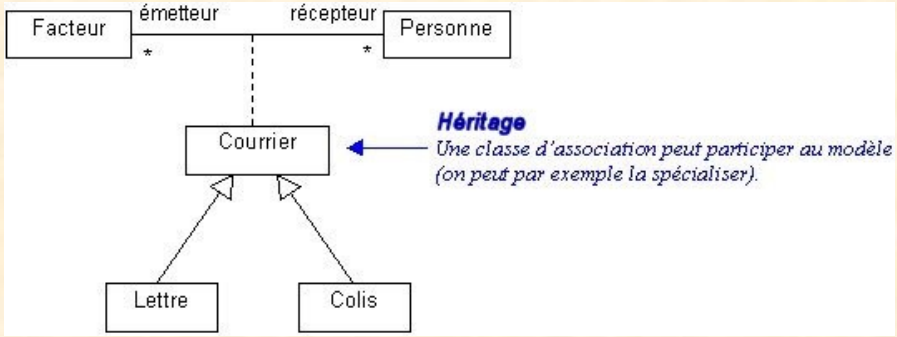


UML (11/24)

- Le diagramme de classes :
  - Il permet de décrire les classes d'un système ainsi que les associations et les relations d'héritage entre ces classes
    - *Composition et agrégation*

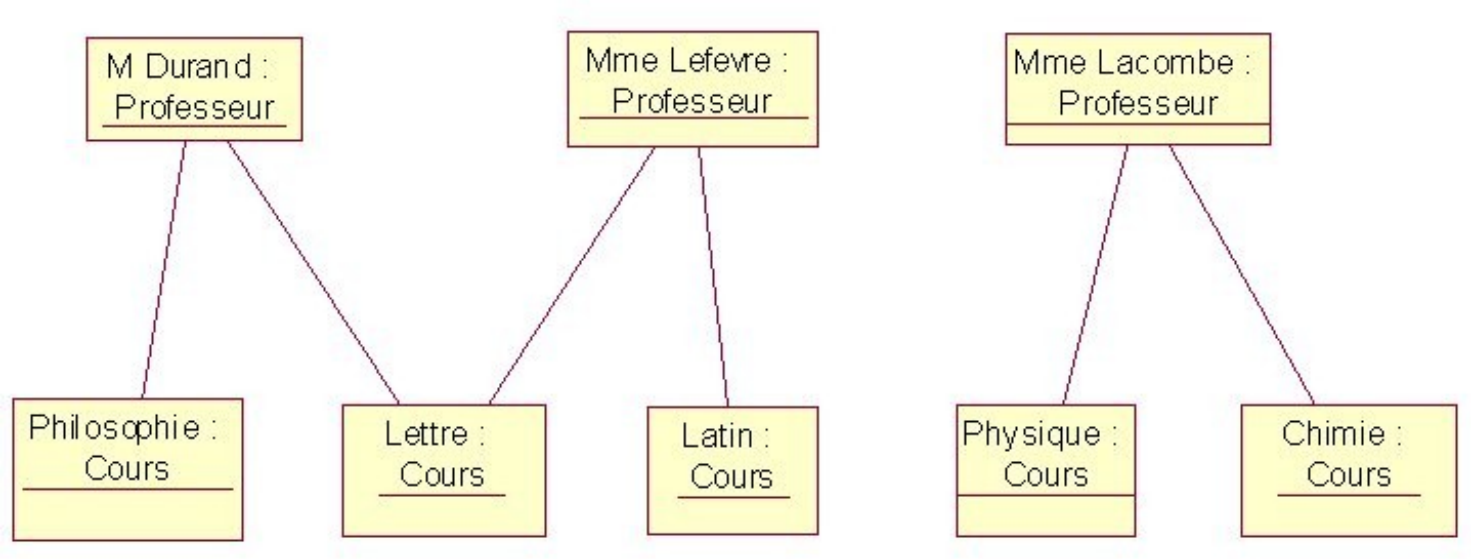


    - *L'héritage*



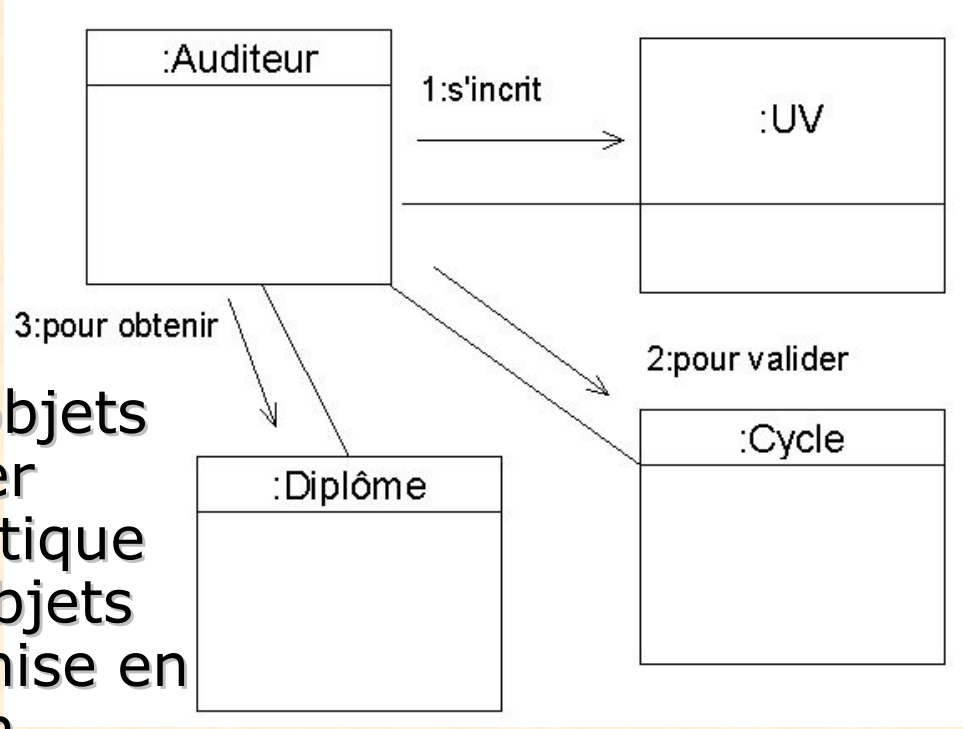
UML (12/24)

- Le diagramme d'objets :
  - Le diagramme d'objets est un cas illustré d'un diagramme de classes. Selon le contexte, il montre la relation entre les objets



UML (13/24)

- Le diagramme de collaboration :
  - Le diagramme de collaboration est une extension des diagrammes de classes
  - Il montre les interactions entre objets et vise à représenter du point de vue statique et dynamique les objets impliqués dans la mise en place d'une fonction applicative



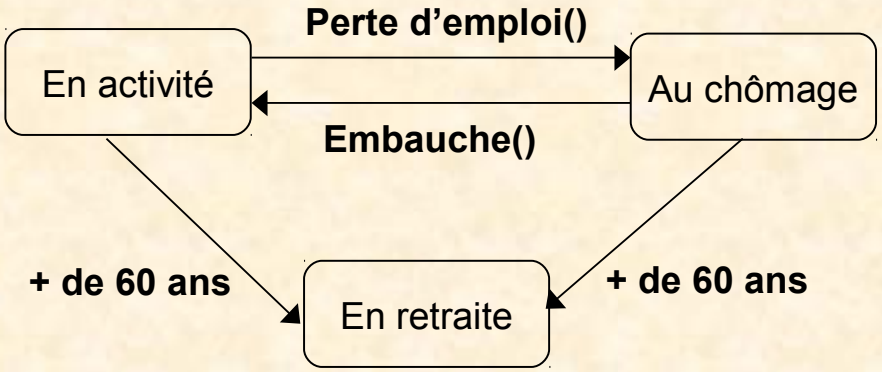
UML (14/24)

- Le diagramme d'états (transitions) :
  - Il permet de décrire les changements d'états d'un objet, en réponse aux interactions avec d'autres objets ou avec des acteurs

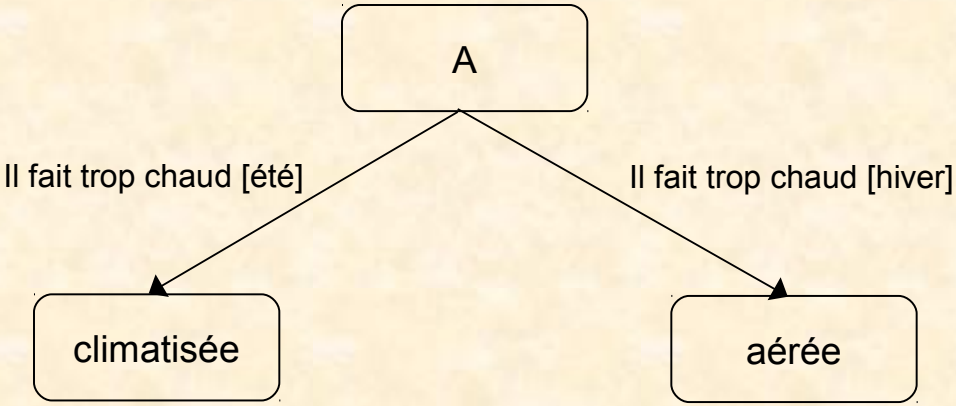
  - Un diagramme d'états possède
    - *Un état initial*
    - *Au moins un état final*

UML (15/24)

- Exemples de diagrammes d'états :
  - Événement

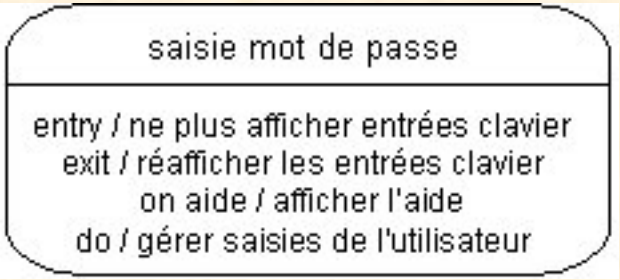


  - Transition gardée

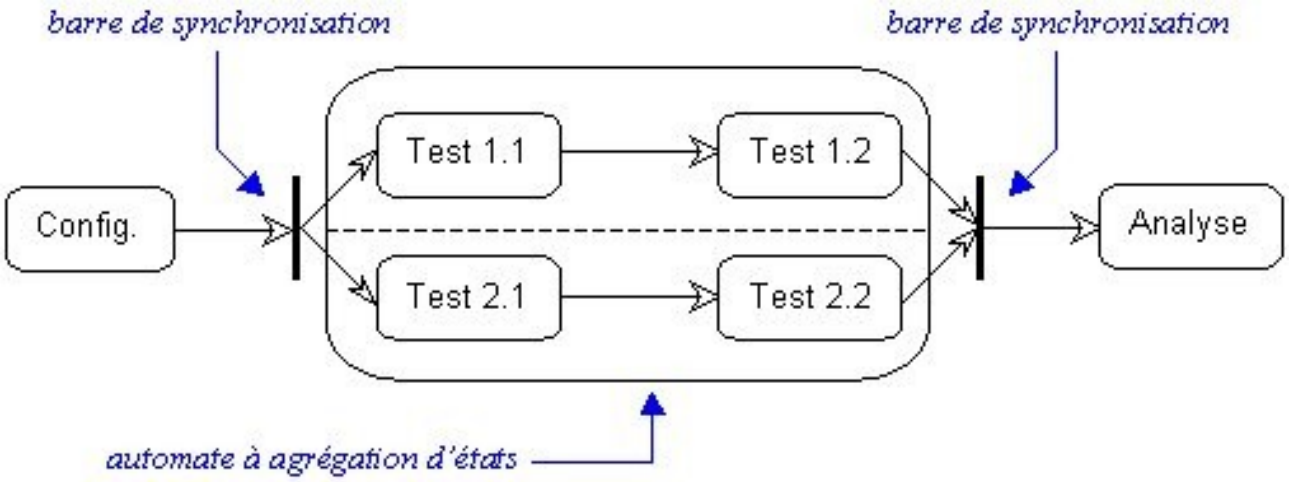


UML (16/24)

- Le diagramme d'états (suite) :
  - Actions dans un état

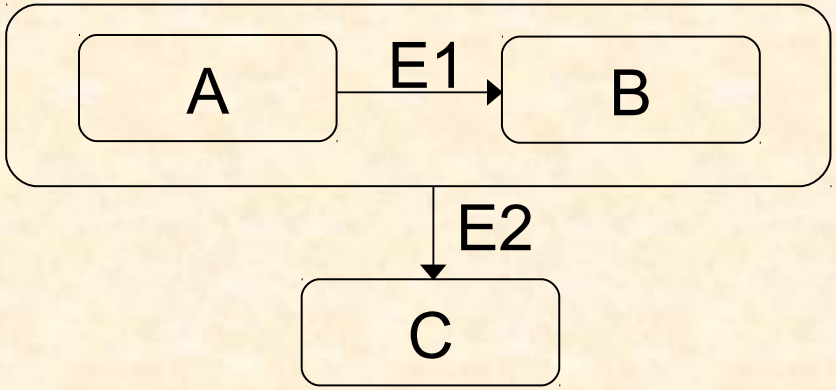


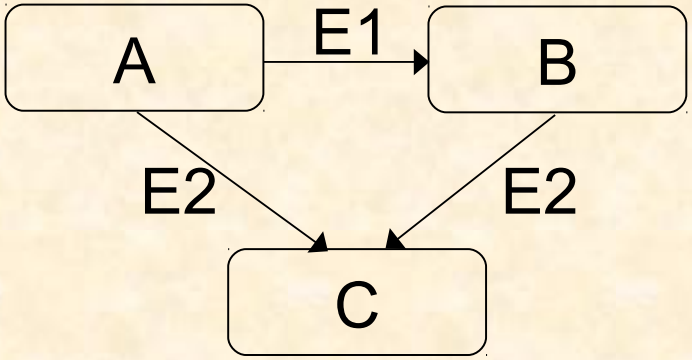
  - Synchronisation



UML (17/24)

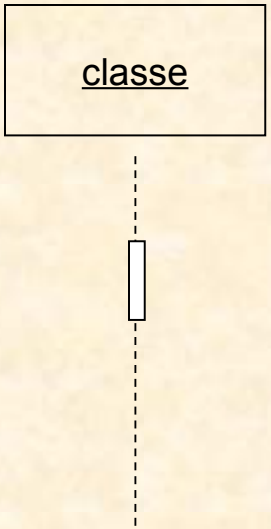
- Le diagramme d'états (suite) :
  - Généralisation





UML (18/24)

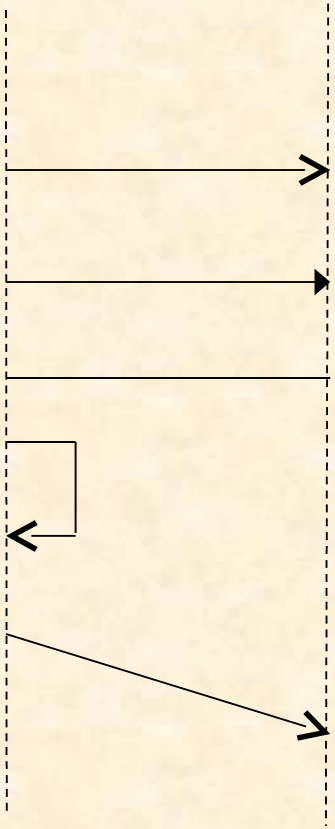
- Le diagramme de séquences :
  - Il permet de représenter des interactions entre les objets selon un point de vue temporel
  - Représentation d'un objet :
    - *Classe (grand rectangle)*
    - *Axe chronologique (ligne)*
    - *Début et fin de l'interaction (petit rectangle sur l'axe)*



  - NB : Il présente un certain nombre d'analogies avec le diagramme de collaboration

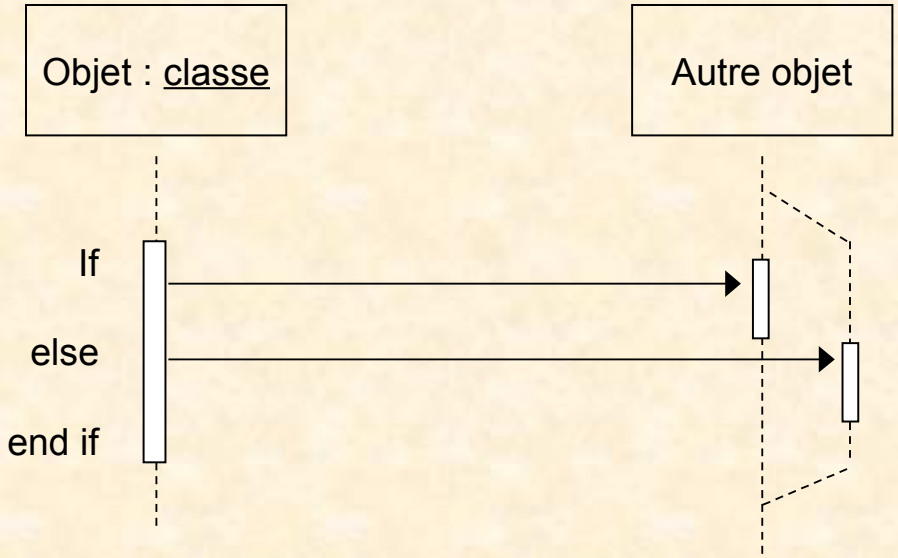
UML (19/24)

- Le diagramme de séquences :
  - Les messages :
    - *simples*
    - *synchrones*
    - *asynchrones*
    - *réflectifs*
    - *avec délai*



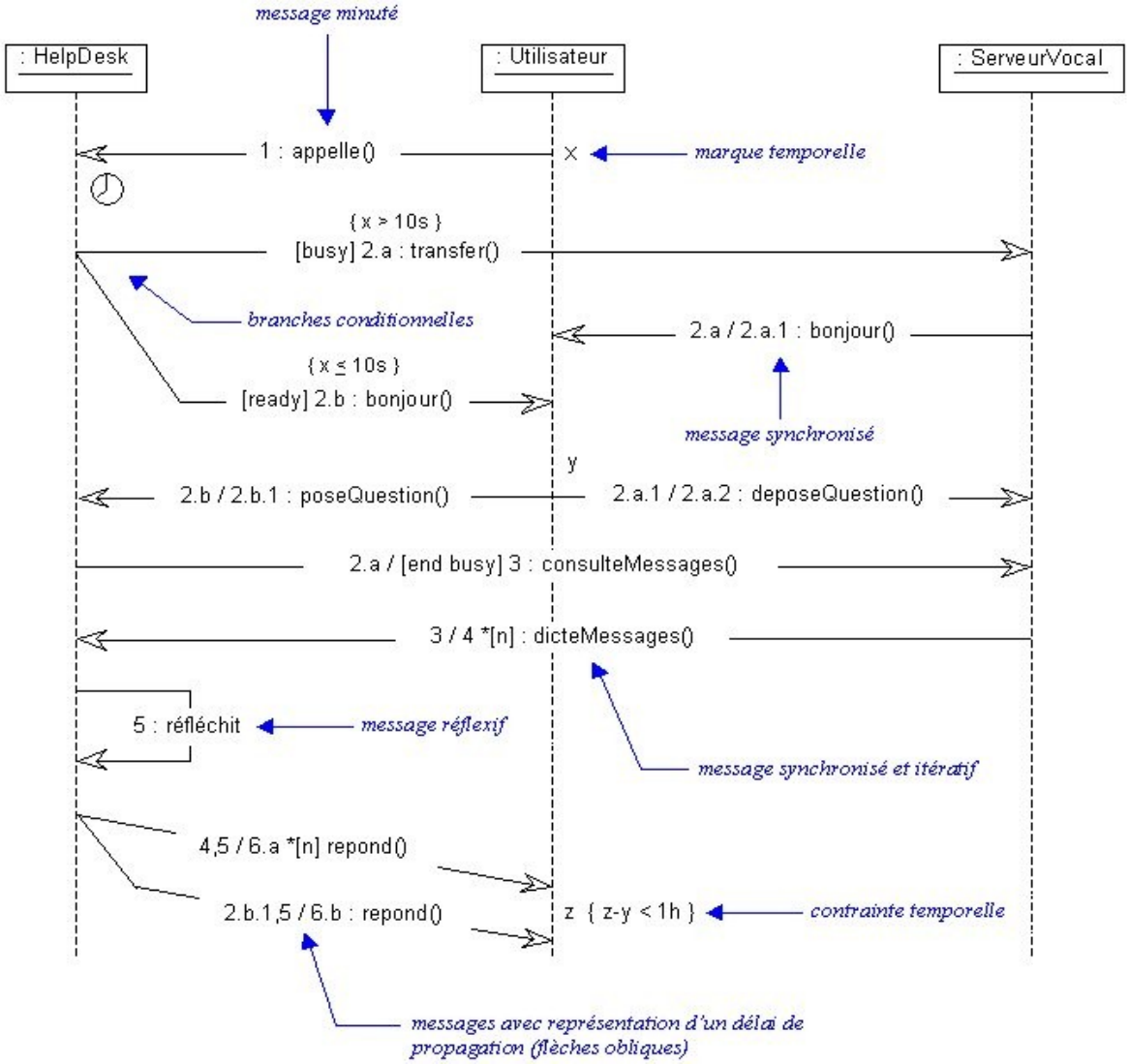
UML (20/24)

- Le diagramme de séquences :
  - Ajout de pseudo-code
  - Branche conditionnel



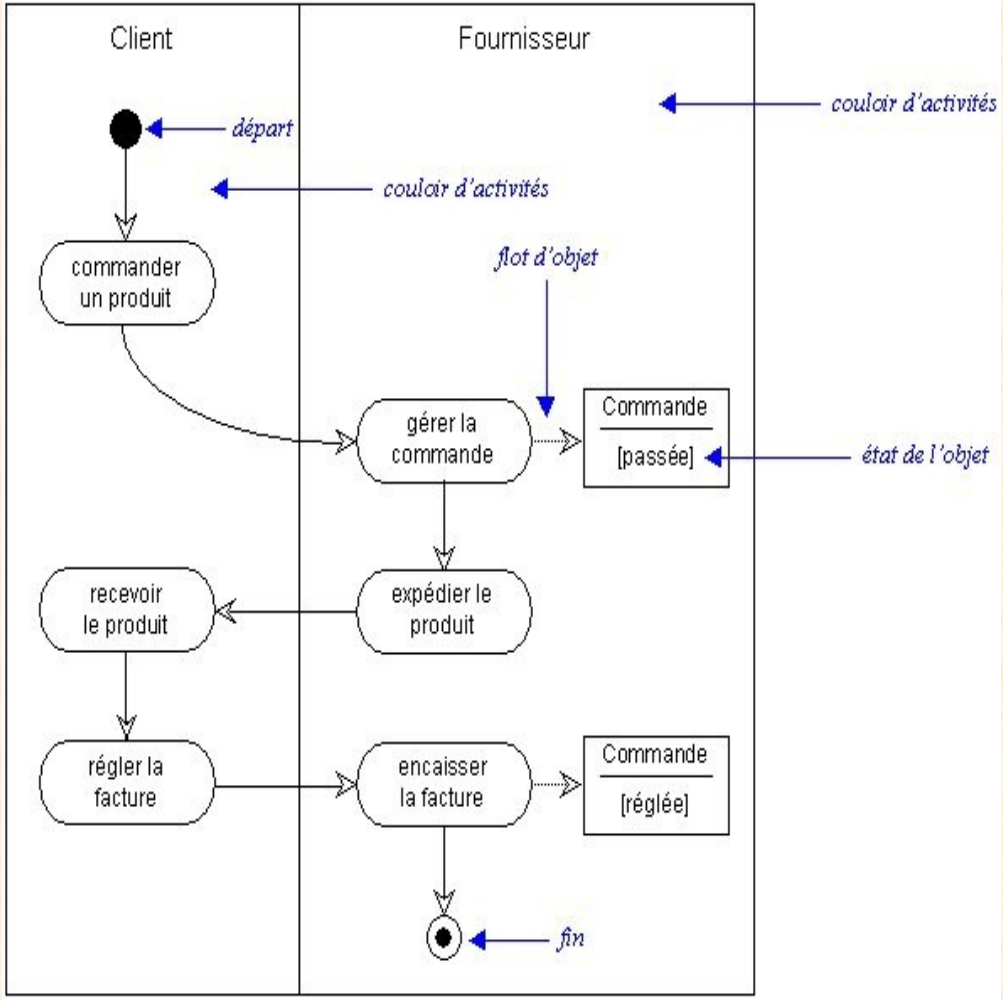
UML (21/24)

- Exemple de diagramme de séquences :



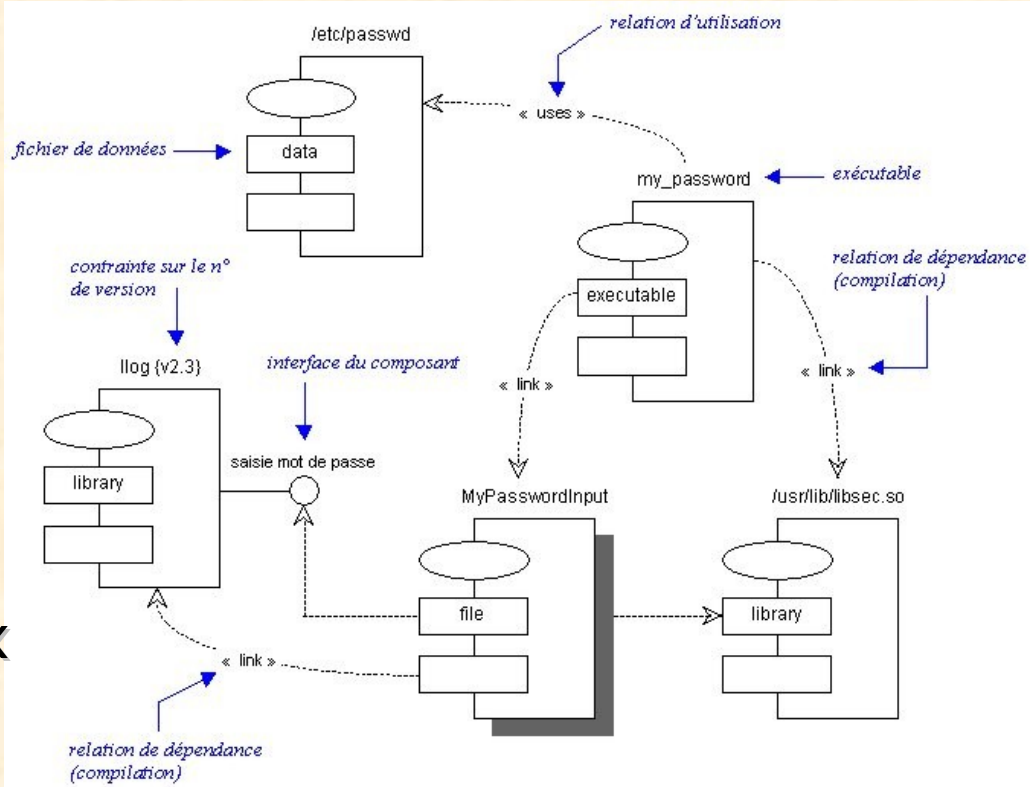
UML (22/24)

- Le diagramme d'activités :
  - Il s'agit d'une variante du diagramme d'état-transition représentant la dynamique du système
  - Il sert à représenter le comportement interne d'un cas d'utilisation. Son intérêt réside dans la représentation simplifiée des activités.



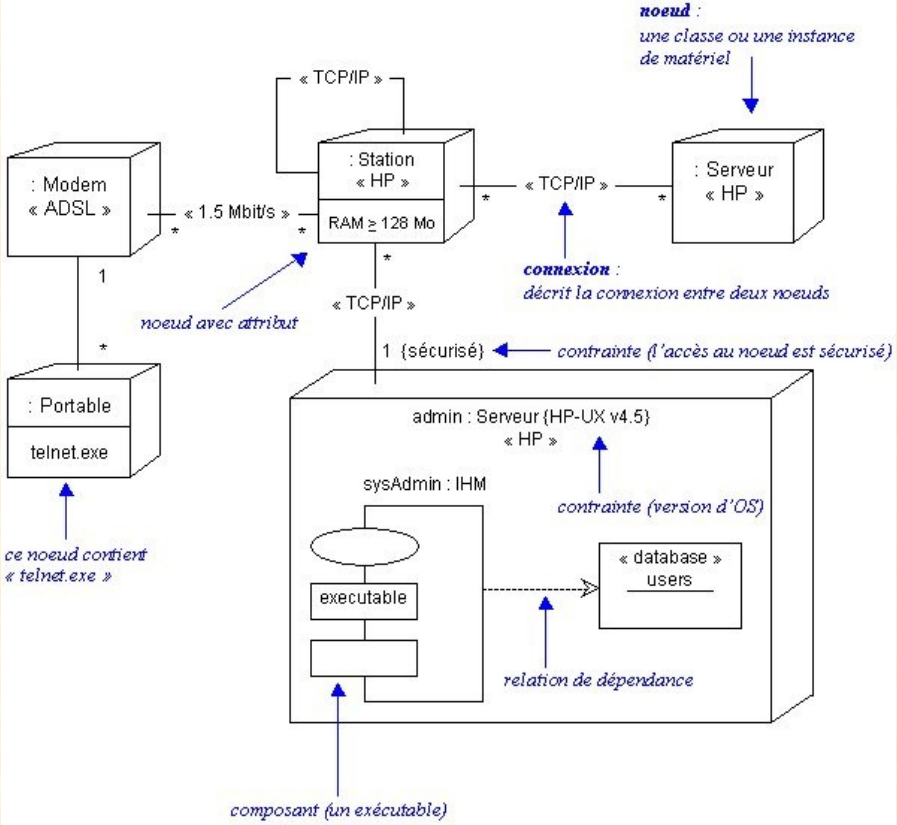
UML (23/24)

- Le diagramme de composants :
  - Il décrit les éléments physiques et leurs relations dans l'environnement de réalisation
  - Il montre les choix de réalisation



UML (24/24)

- Le diagramme de déploiement :
  - Il montre la disposition physique des différents matériels (nœuds) qui entrent dans la composition d'un système et la répartition des programmes exécutables sur ces matériels
  - Il montre les choix de réalisation



OUTILS (1/3)

- Outils Merise :
  - Power AMC
  - MS-Designer…
- Outils UML :
  - Rational Rose (racheté et intégré par IBM)
  - Objecteering
  - Together ControlCenter
  - ArgoUML, Poséidon…
- Outils de gestion de tests :
  - Mercury TestDirector
  - Compuware QADirector
  - Rational TestManager…

OUTILS (2/3)

- Critères de sélection des outils UML :
  - Respect des normes UML
  - OS supportés
  - Stabilité
  - Exhaustivité des diagrammes
  - Correspondance relationnel
  - Génération de code
  - Format de documentation
  - Gestion de configuration
  - Reverse engineering
  - Gestion de tests

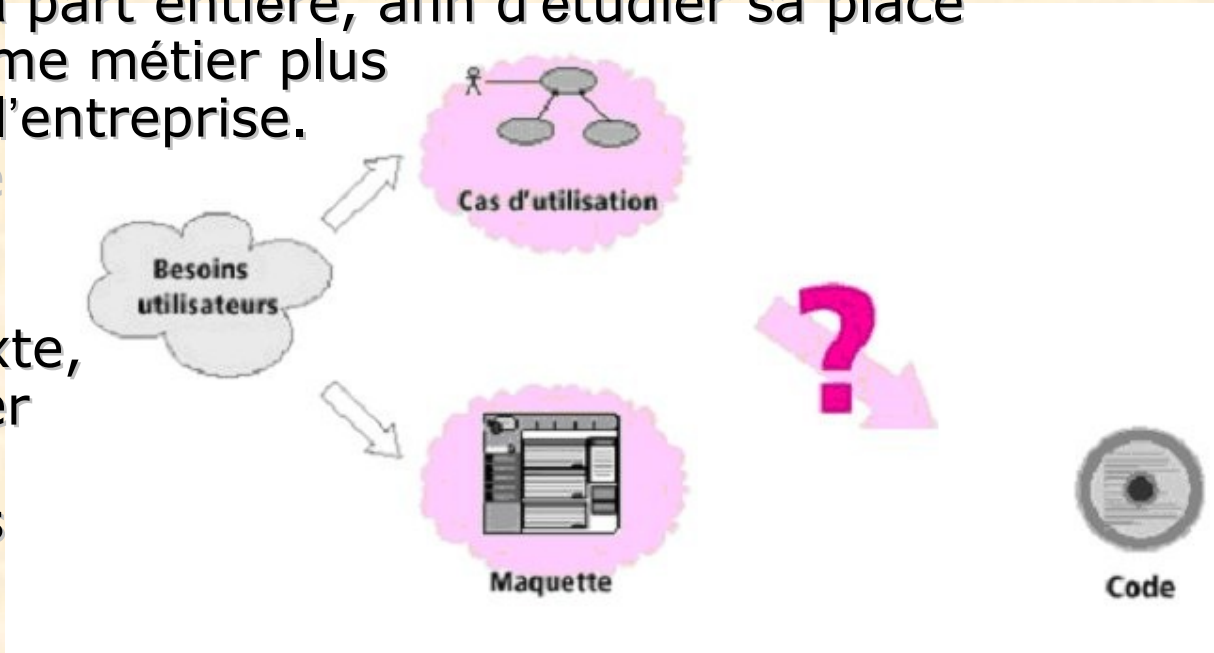
OUTILS (3/3)

- Caractéristiques des principaux produits de modélisation :

	Nor.	OS.	Sta.	Exh.	Gén.	For.	Conf.	Rev.	Rel.	Tes.
Rational (leader)	X	Windows Unix	X	X	X	Word	SourceSafe ClearCase	X	-	-
Objecteering	X	Windows Linux	X	X	X	?	X	X	-	X
Together ControlCenter	X	Windows Unix	X	X	X	HTML	X	X		X
Mega Development		Windows	X	X	X	Word HTML	X	X	X	-
ArgoUML et Poséidon	X	Windows Unix	-	-	-	?	-	-	-	-
Xfig et Dia	-	Linux	X	-	-	EPS	-	-	-	-

SPECIFICATIONS (1/5)

- La spécification des besoins (ou des exigences)
  - **Point de départ**
  - Dans les activités de spécification des exigences, il convient d'abord de considérer le système comme une **boîte noire** à part entière, afin d'étudier sa place dans le système métier plus global qu'est l'entreprise. On développe pour cela un modèle de niveau contexte, qui va préciser les frontières fonctionnelles du système.

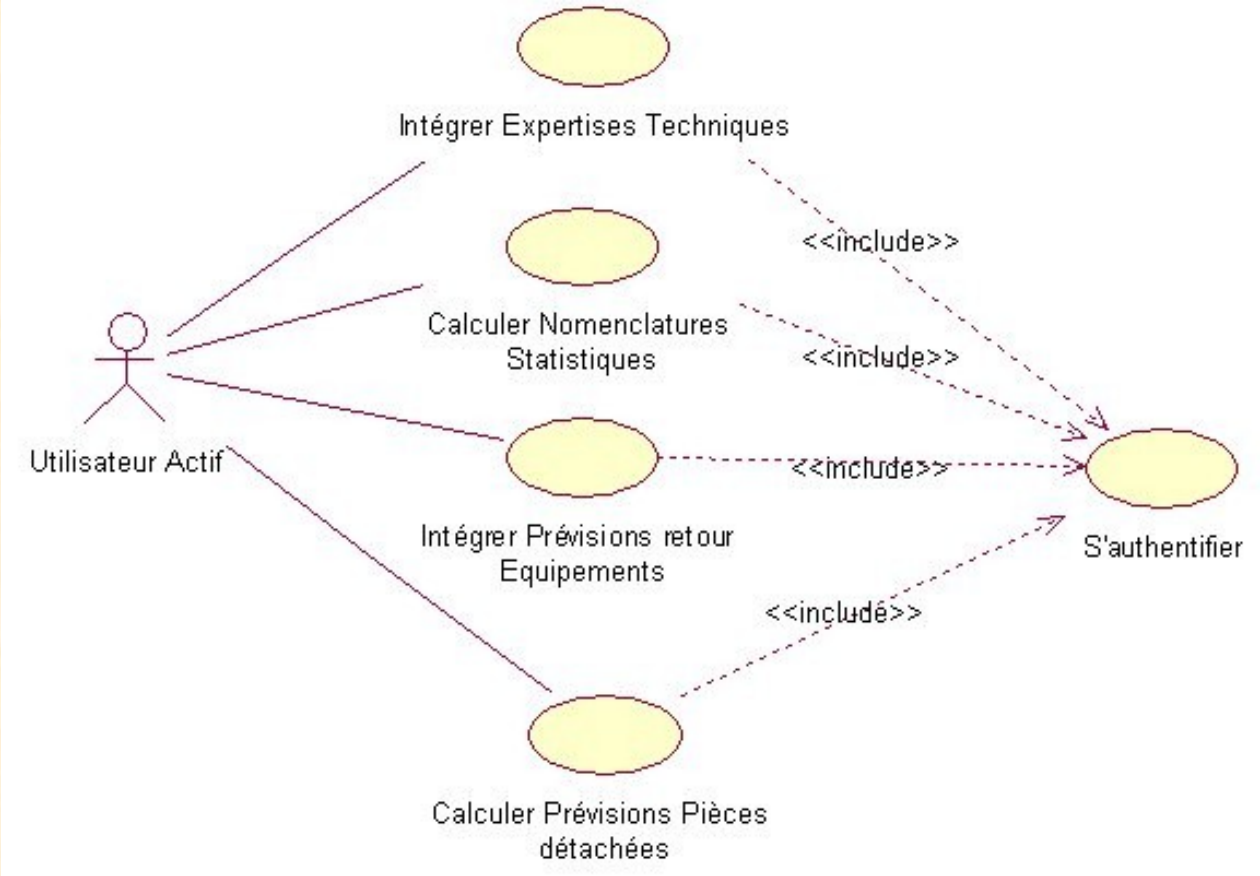


SPECIFICATIONS (2/5)

- La spécification des besoins (suite)
  Cette activité met donc en œuvre un mode de représentation fonctionnel s'appuyant sur :
  - Les **diagrammes de cas d'utilisation** qui permettent d'identifier et de formaliser les différents acteurs ainsi que les services attendus de la future application.
  - La **maquette** pour sa part, permet de présenter de manière concrète ces services via les IHM (Interface Homme Machine) de l'application. Elle est destinée plus particulièrement à faire réagir les utilisateurs de la future application afin d'y apporter les mises au point nécessaires à une complète adéquation entre les services rendus par l'application et les besoins des utilisateurs.

SPECIFICATIONS (3/5)

- La spécification des besoins (suite)
  - Exemple de diagramme de cas d'utilisation :



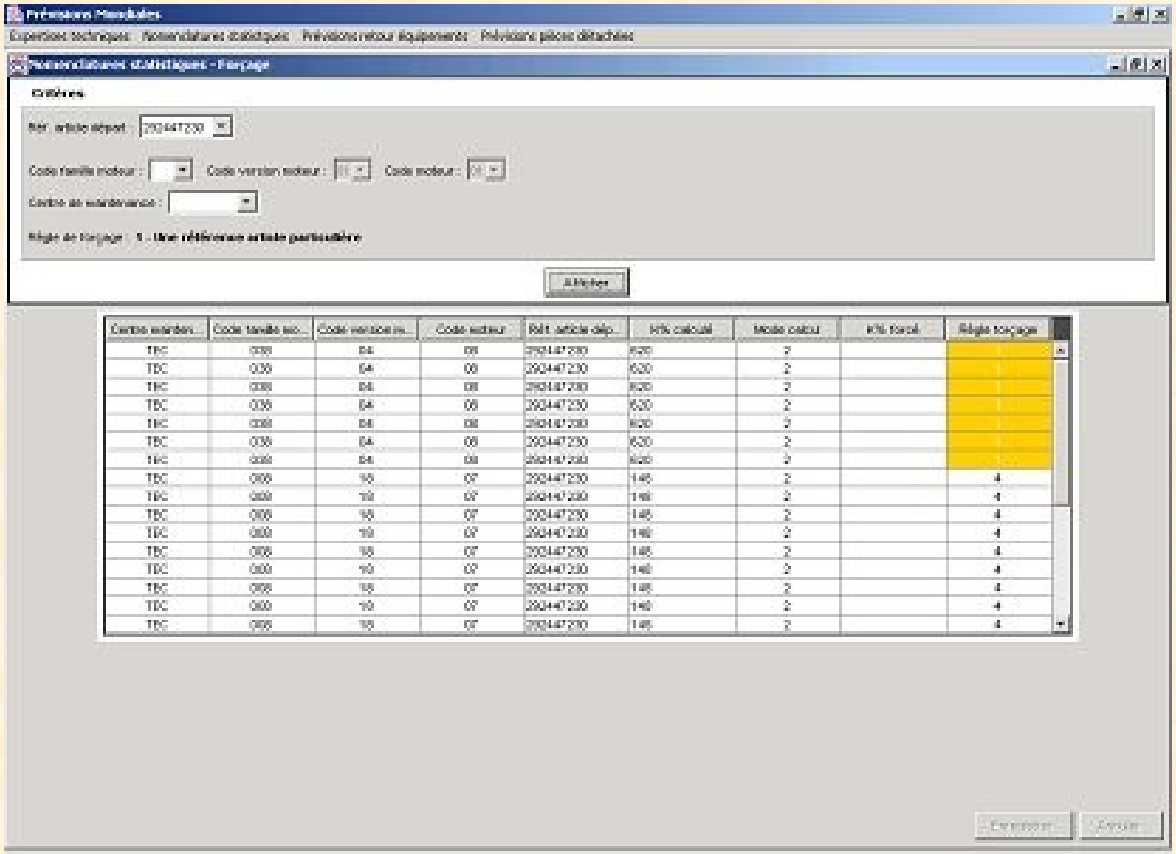
SPECIFICATIONS (4/5)

- La spécification des besoins (suite)
  - Exemple de description pour le cas d'utilisation ***s'authentifier***

Acteur principal	Utilisateur Identifié
Acteur secondaire	Utilisateur Inconnu
Objectifs	L'Utilisateur Identifié souhaite se connecter à l'application
Préconditions	L'Utilisateur Identifié possède un login et un mot de passe
Postconditions	1. L'Utilisateur Identifié est authentifié 2. L'Utilisateur Inconnu n'est pas authentifié
Scénario nominal	1. L'Utilisateur Identifié saisit un login et un mot de passe depuis l'écran de connexion 2. Le système vérifie en base la cohérence du login et du mot de passe 3. L'Utilisateur Identifié est connecté.
Extensions	2a. Le login ou le mot de passe n'est pas saisi. 1. Le système affiche un message d'erreur «Login (ou mot de passe) manquant ». 2. Le système revient à l'étape 1 du scénario nominal. 2a. Le login ou le mot de passe est incorrect. 1. Le système affiche un message d'erreur «Il est impossible de se connecter à l'application : login ou mot de passe erroné ». 2. Le système revient à l'étape 1 du scénario nominal
Besoins IHM	Cacher la saisie du mot de passe

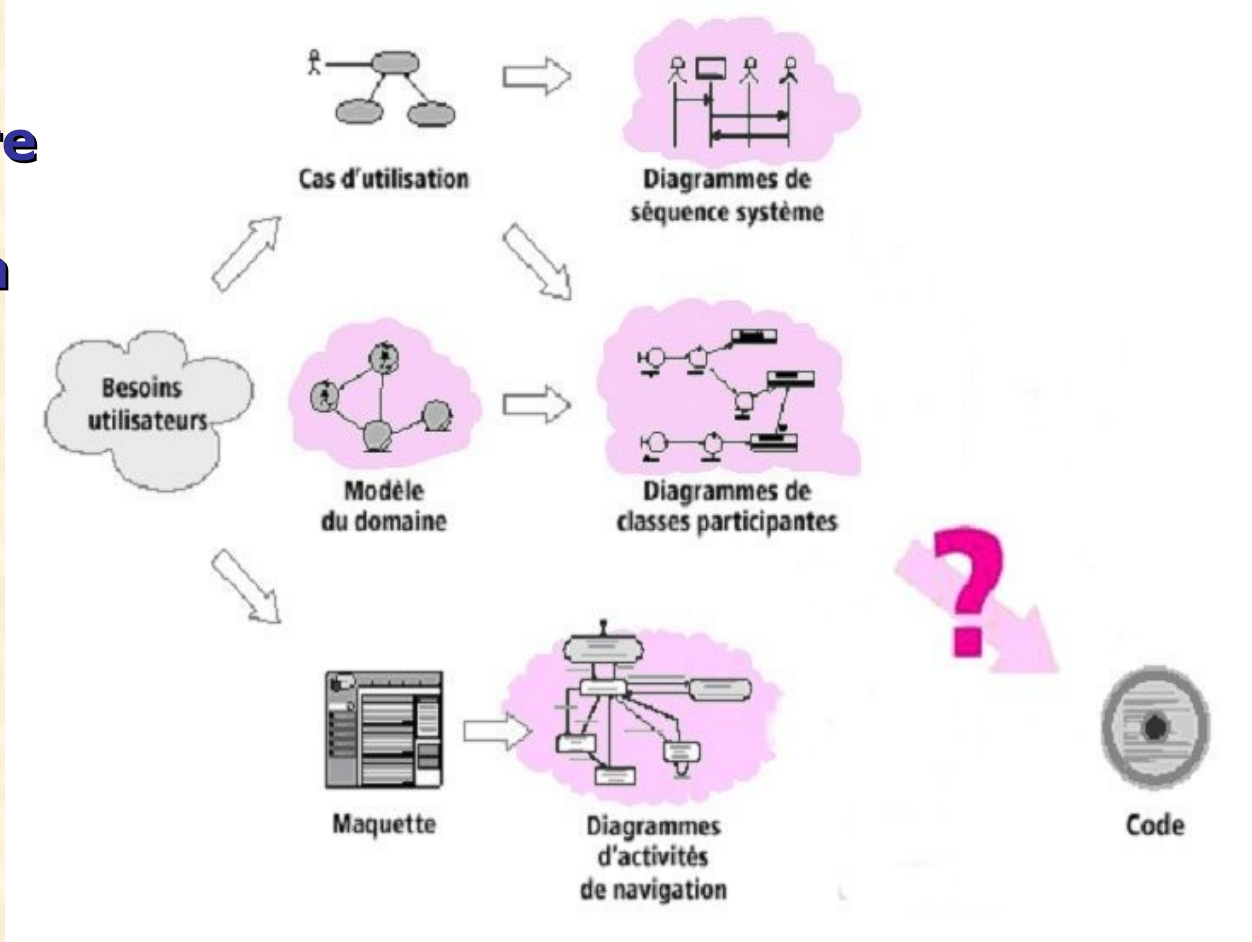
SPECIFICATIONS (5/5)

- La spécification des besoins (suite)
  - Exemple de maquette :



ANALYSE (1/8)

- L'analyse
  - **Étape intermédiaire entre la spécification des besoins et la conception du système**



ANALYSE (2/8)

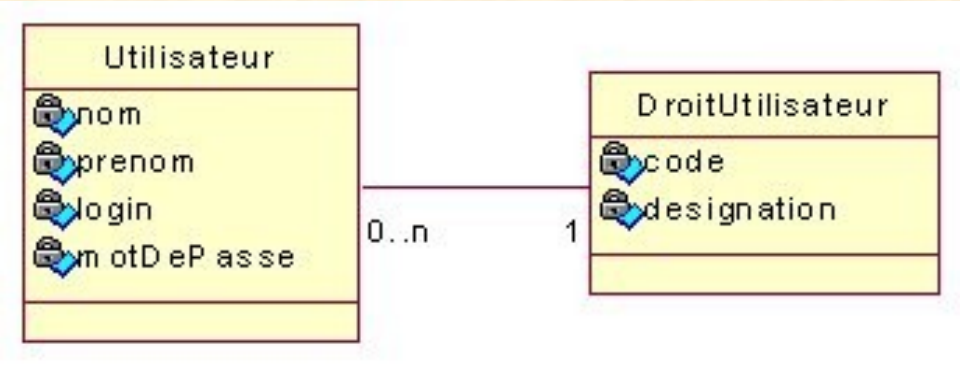
- L'analyse statique
  - Pour identifier les classes de conception intervenant dans les diagrammes d'interaction du modèle conceptuel, deux sous-étapes sont nécessaires :
    - *Construire le **modèle du domaine.** Sorte de glossaire formalisé des concepts fondamentaux de l'espace du problème, ce modèle fournit une partie des classes de conception : **celles correspondant directement aux concepts métier manipulés par les experts du domaine et les utilisateurs**. Ces concepts, leurs attributs et leurs relations vont être décrits en UML par des diagrammes de classes simplifiées.*
    - *Construire les **diagrammes de classes participantes**. Afin de **prendre en compte également l'IHM et la cinématique de l'application**, les diagrammes de classes participantes font la jonction entre les cas d'utilisation, le modèle du domaine, la maquette… et les diagrammes de conception logicielle. Il s'agit de diagrammes de classes qui décrivent, **par cas d'utilisation**, les trois principales classes d'analyse et leurs relations.*

ANALYSE (3/8)

- L'analyse statique (suite)
  - Les diagrammes de classes participantes :
    - *Les classes de type **dialogue** : elles permettent les interconnexions entre l'IHM et ses utilisateurs. Ce sont typiquement les écrans proposés à l'utilisateur : les formulaires de saisie, les résultats de recherche… Elles proviennent directement de l'analyse de la maquette.*
    - *Les classes de type **métier** : elles représentent les **règles métier** et proviennent directement du modèle du domaine mais sont **confirmées et complétées pour chaque cas d'utilisation**.*
    - *Les classes de type **contrôle** : elles contiennent la cinématique de l'application et font la **transition entre les dialogues et les classes métier**, en permettant aux écrans de manipuler des informations détenues par un ou plusieurs objets métier.*

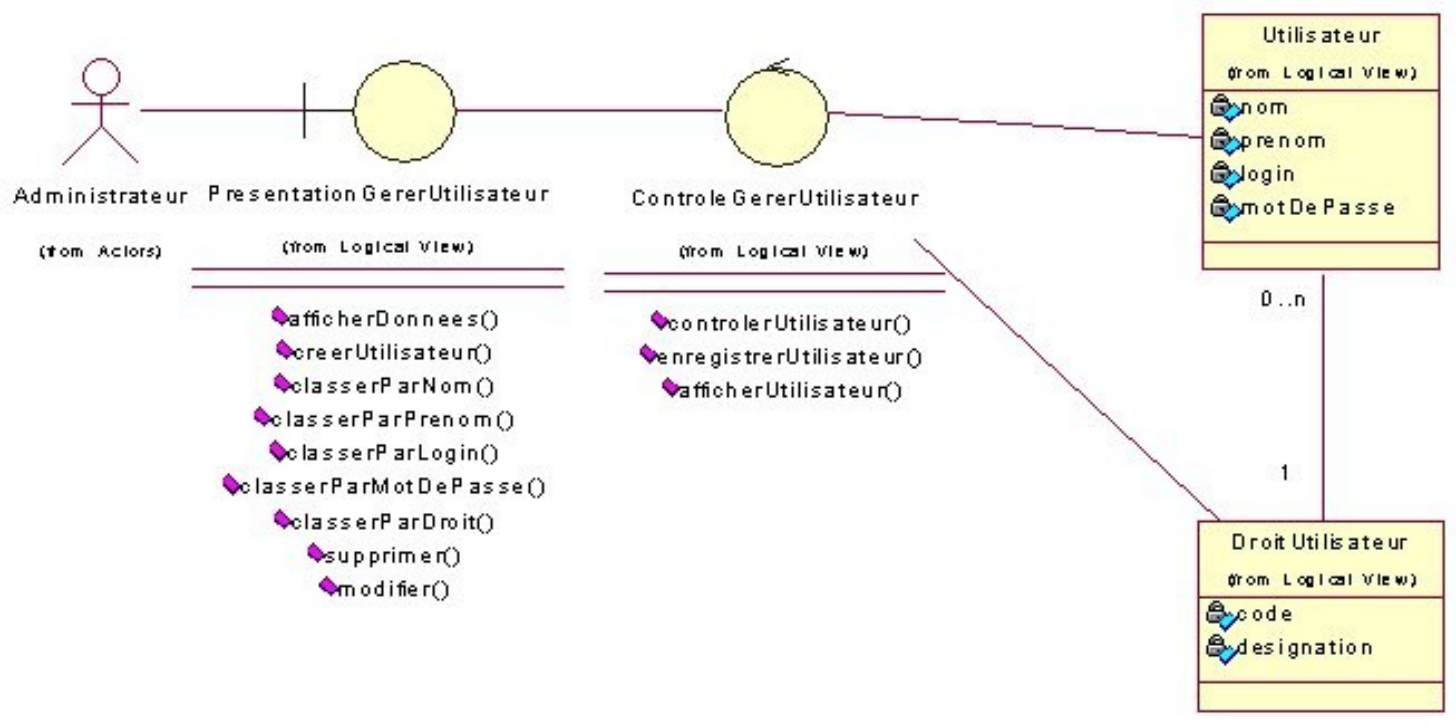
ANALYSE (4/8)

- L'analyse statique (suite)
  - Exemple de modèle du domaine pour le CU ***s'authentifier*** :



ANALYSE (5/8)

- L'analyse statique (suite)
  - Exemple de diagramme de classes participantes pour le CU ***s'authentifier*** :

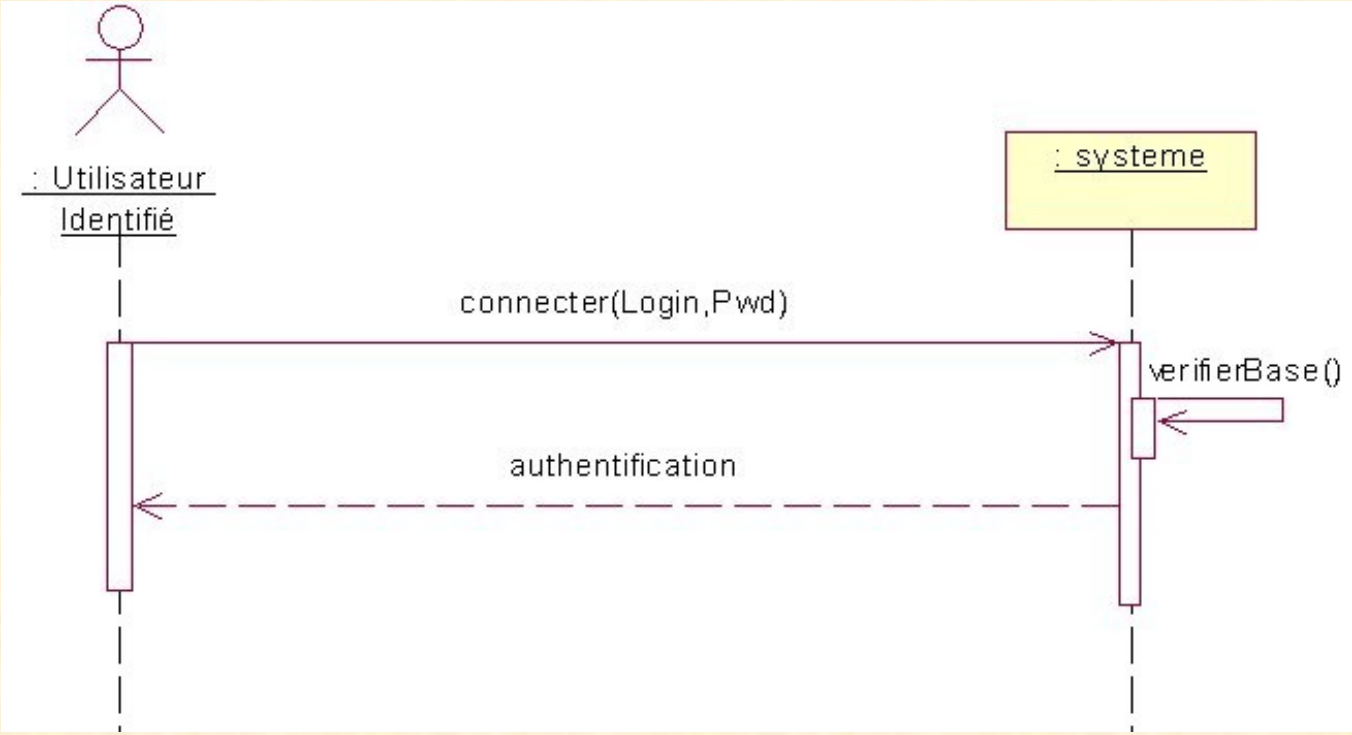


ANALYSE (6/8)

- L'analyse dynamique
  - Le modèle dynamique permet de décrire :
    - *Pour chaque cas d'utilisation, la **séquence des interactions entre les acteurs et le système** vue comme une boîte noire, représentée par les **diagrammes de séquence système**. Ce sont ces diagrammes qui feront le lien entre les cas d'utilisation et les diagrammes d'interaction du niveau conceptuel.*
    - *D'autre part, pour représenter de manière formelle l'ensemble des chemins possibles entre les principaux écrans proposés à l'utilisateur, et à partir des informations fournies par la maquette, il reste à détailler les **diagrammes d'activités de navigation**.*
    - *Si nécessaire, le cycle de vie commun aux objets d'une même classe, peut être explicité par les **diagrammes d'états.***

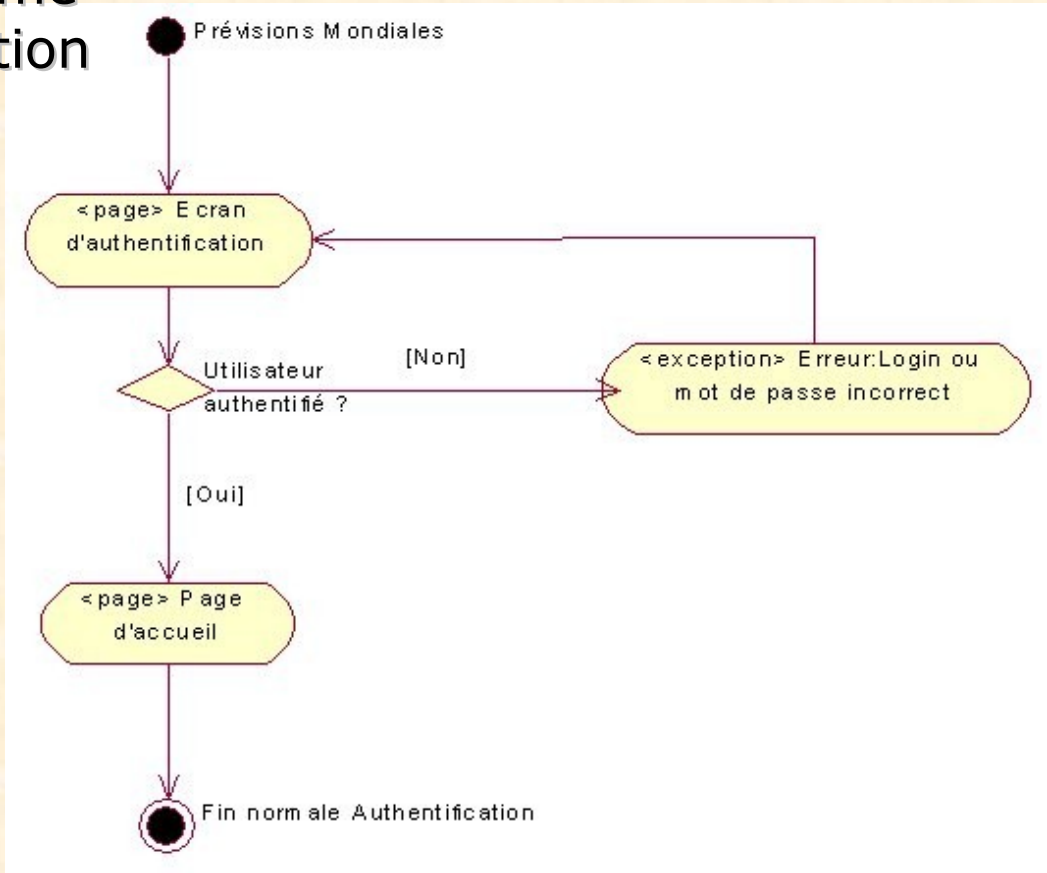
ANALYSE (7/8)

- L'analyse dynamique (suite)
  - Exemple de diagramme de séquence système pour le CU ***s'authentifier*** :



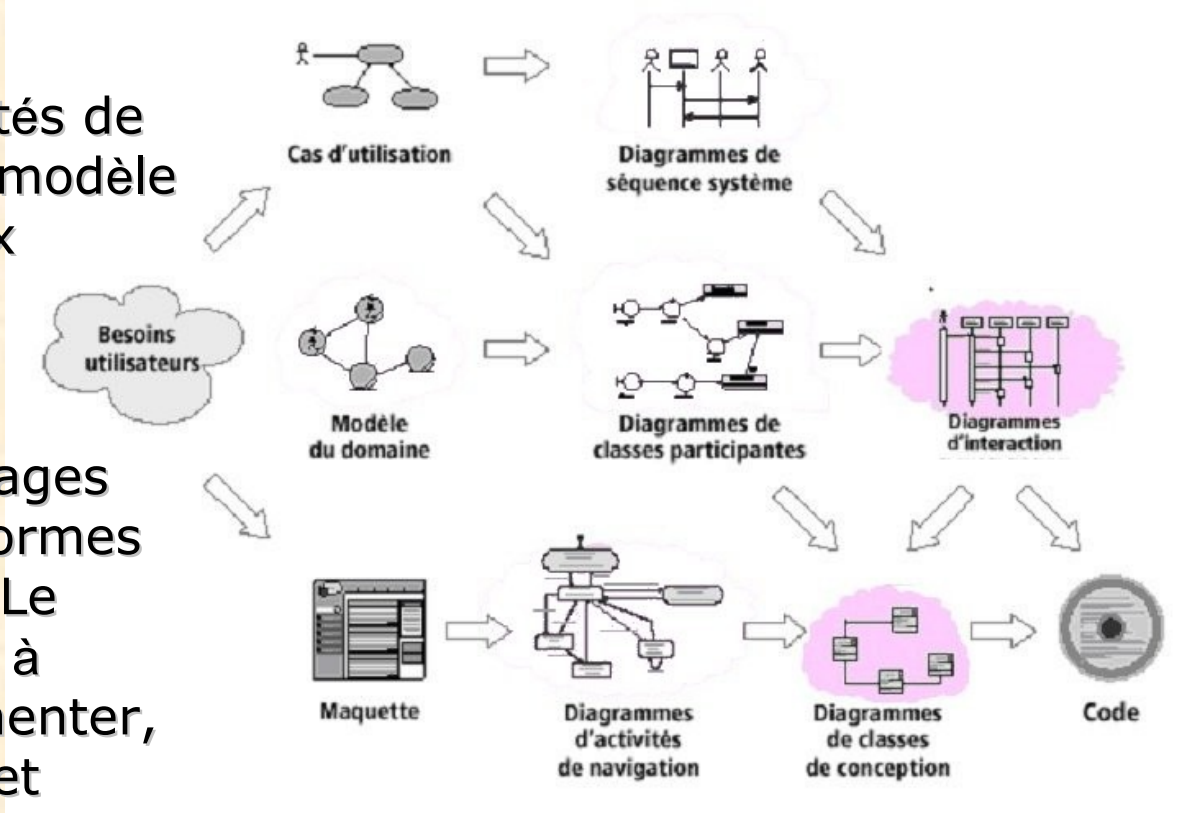
ANALYSE (8/8)

- L'analyse dynamique (suite)
  - Exemple de diagramme d'activités de navigation pour le CU ***s'authentifier*** :



CONCEPTION (1/3)

- La conception du système
  - **Cible**
  - Dans les activités de conception, le modèle correspond aux concepts informatiques utilisés par les outils, les langages ou les plates-formes de réalisation. Le modèle sert ici à étudier, documenter, communiquer et anticiper la solution logicielle.

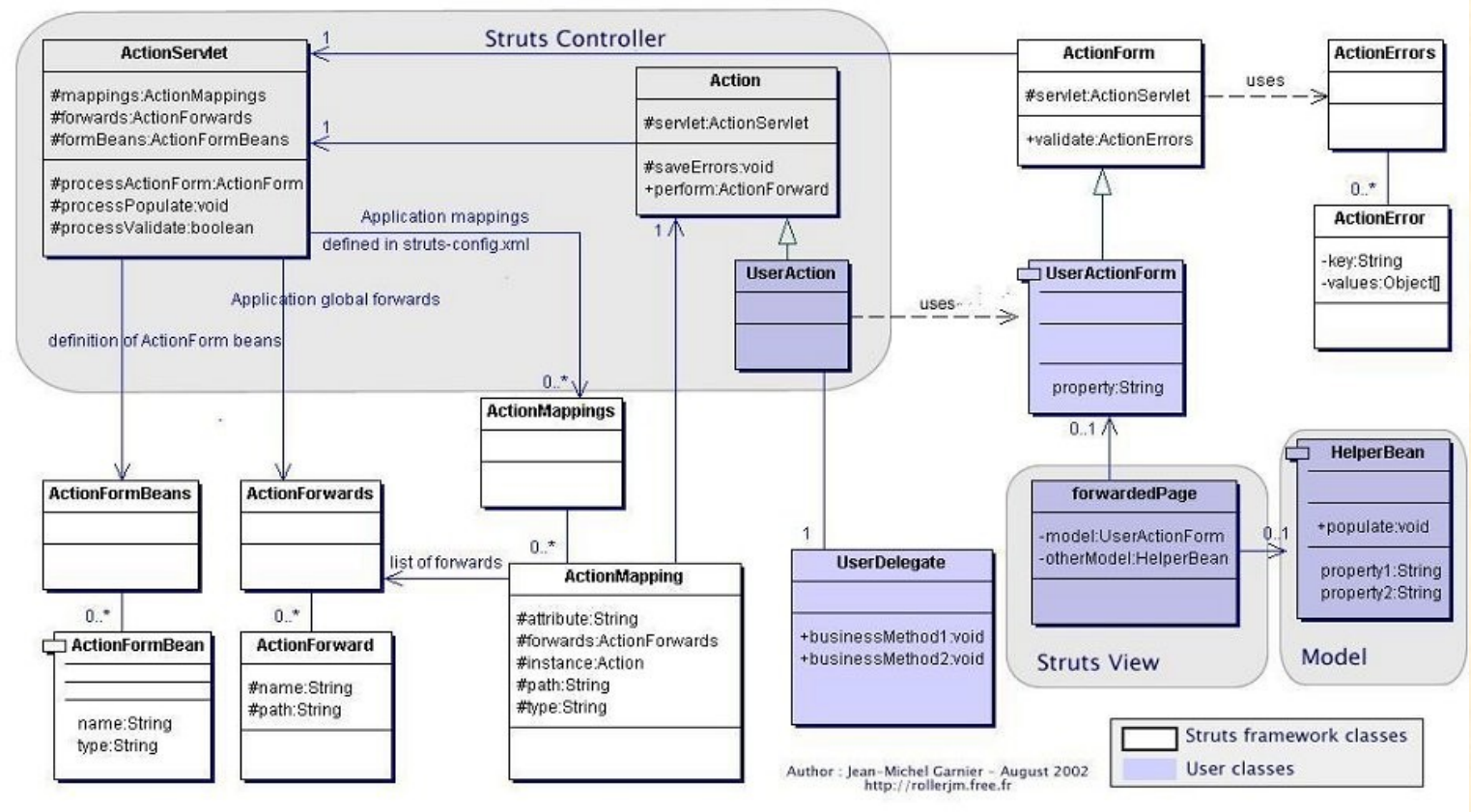


CONCEPTION (2/3)

- La conception du système (suite)
  - Dans le cadre de systèmes orientés objet, la structure du code est définie par des classes logicielles et leurs regroupements en ensemble (appelés **packages**). La conception représente deux points de vue : la **structure statique et le comportement dynamique**, deux perspectives qui complètent la compréhension du système à développer.
  - Le **modèle statique**, qui permet de décrire les différents composants logiciels structurant le système, est représenté à l'aide de **diagrammes de classes de conception.**
  - Le modèle dynamique, permet quant à lui de décrire l'attribution des bonnes responsabilités (services) aux bonnes classes. Tout le comportement du système est ainsi réparti entre les classes de conception. C'est le rôle des diagrammes d'interaction (**diagrammes de séquence ou de collaboration**) de représenter graphiquement ces décisions d'allocation de responsabilités ainsi que les collaborations induites.

CONCEPTION (3/3)

- La conception du système (suite) 1 - 2
  - Exemple de diagramme de classes de conception de Struts :



CONTENU DU DAF (1/2)

- Dossier d'Analyse Fonctionnelle
- Introduction
  - Objectifs du document
  - Cible du document
  - Terminologie et acronymes
  - Structure du document
  - Documents de référence
- Spécifications des besoins
  - Acteurs
  - Diagramme des cas d'utilisations
  - Pour chaque CU : description du cas d'utilisation
  - Contraintes fonctionnelles

CONTENU DU DAF (2/2)

- Analyse
  - Pour chaque CU :
    - *Modèle du domaine*
    - *Diagramme de classes participantes*
    - *Diagramme de séquence système*
    - *Diagramme d'activités de navigation*
    - *Diagramme d'états le cas échéant*
- Conception
  - Pour chaque CU :
    - *Diagramme de classes de conception*
    - *Diagramme de séquence ou de collaboration*
- Annexes
  - MCD et MPD (le cas échéant)

STRATEGIE DE TESTS (1/4)

- Enjeux de l'activité de tests
  - Enjeux métier :
    - ***Adéquation*** *aux besoins*
    - *Non régression*
  - Enjeux techniques :
    - *Niveau de service*
      - Performance
      - **Tolérance aux pannes**
      - Exploitabilité
    - *Sécurité*
    - *Pérennité /* ***évolutivité***
  - Enjeux économiques
    - *Réduction des* ***coûts de recette***
    - *Réduction de* ***coûts de maintenance***

STRATEGIE DE TESTS (2/4)

- Les 6 bonnes pratiques de l'activité de tests
  - L'activité de test doit être **anticipée**
  - Les **responsables** de l'activité doivent être **identifiés**
  - **Des ressources doivent être dédiées à l'activité**
  - L'utilisation **d'outils** doit être favorisé
  - **L'exhaustivité étant bien souvent utopique, il faut viser la représentativité des cas de tests**
  - Les tests doivent être formalisés dans un **dossier de test**. Celui-ci comportera 5 parties :
    - *Présentation du plan de test*
    - *Le cahier de test*
    - *Le référentiel des cas et scénarios de tests fonctionnels*
    - *Le référentiel des cas et scénarios de tests techniques*
    - *Le journal des tests*

STRATEGIE DE TESTS (3/4)

- Présentation du plan de tests
  - Objectif : détecter un **maximum d'anomalies** susceptibles de perturber le fonctionnement d'un logiciel. Les anomalies étant de nature diverse, il est nécessaire de définir une stratégie en fonction de chaque type de tests. La démarche consiste à :
    - *définir chaque type de test (fonctionnels en exécution **attendue**, fonctionnels en exécution **dégradée**, de performance…)*
    - *établir pour chacun les objectifs*
    - *assigner une priorité à ces objectifs*

Objectif	Fréquence	Impact	Catégorie	Profil	Risque
Fonctionnalité s'authentifier	Haute	Critique	Fonctionnalité	Utilisateur	Moyen
Fonctionnalité intégrer les expertises	Normale	Critique	Fonctionnalité	Utilisateur	Elevé
Fonctionnalité intégrer manuellement	Occasionnelle	Faible	Fonctionnalité	Exploitant	Elevé

    - *construire une matrice de couverture*
    - *estimer le plan de charges et planifier*

STRATEGIE DE TESTS (4/4)

- Cahier de test
  - Il a pour objectif de présenter les modèles de référence en terme de tests pour le projet. A ce titre, il contient tous les modèles de documents utilisés dans le cadre de l'activité de tests :
    - *cas de test*
    - *scénario de tests*
    - *fiche d'anomalie*
    - *tableau de bord des anomalies*
- Référentiel des cas et scénarios de tests fonctionnels
  - Il a pour objectif de rassembler les éléments suivants :
    - *Cas et scénarios de tests formalisés sur les fiches correspondantes*
    - *Présentation de la logique d'ordonnancement des tests et planning*

REUTILISABILITE (1/11)

- Définitions :
  - La réutilisabilité est l'aptitude d'un logiciel à être **réutilisé en tout ou en partie** pour de nouvelles applications
  - La réutilisabilité consiste à **se servir d'un composant** pour en créer un nouveau
- Objectifs :
  - Optimiser les **coûts de développement**
  - Optimiser les **coûts de maintenance**
  - **Fiabiliser** les développements
  - Favoriser **l'évolutivité**, **l'adaptabilité**, **l'utilisabilité**…

REUTILISABILITE (2/11)

- Techniques de réutilisabilité :
  - **La duplication** (inconvénient : la maintenabilité).
  - **Le pré-processing :** très similaire à la technique précédente, il intègre un outil spécialisé, le pré-processeur. Ainsi, les corrections d'erreurs comme **les évolutions apportées à la partie réutilisée profitent à l'application d'origine** (inconvénient : si des modifications sont faites sur le code réutilisable et si tous les utilisateurs ne sont pas informés de ces modifications, ceci peut altérer le fonctionnement perçu par les autres utilisateurs).
  - **Les bibliothèques de composants :** cette technique consiste à appeler un **composant exécutable** et d'en exiger un certain comportement à l'aide de paramètres.
  - **Les frameworks :** dans les bibliothèques de composants, c'est l'application qui invoque une routine de la bibliothèque et c'est l'application qui fait le contrôle. Dans les frameworks, la **situation est inversée**, c'est le **framework qui dirige les tâches et qui fait des appels à l'application** si nécessaire (Exemple : Struts).

REUTILISABILITE (3/11)

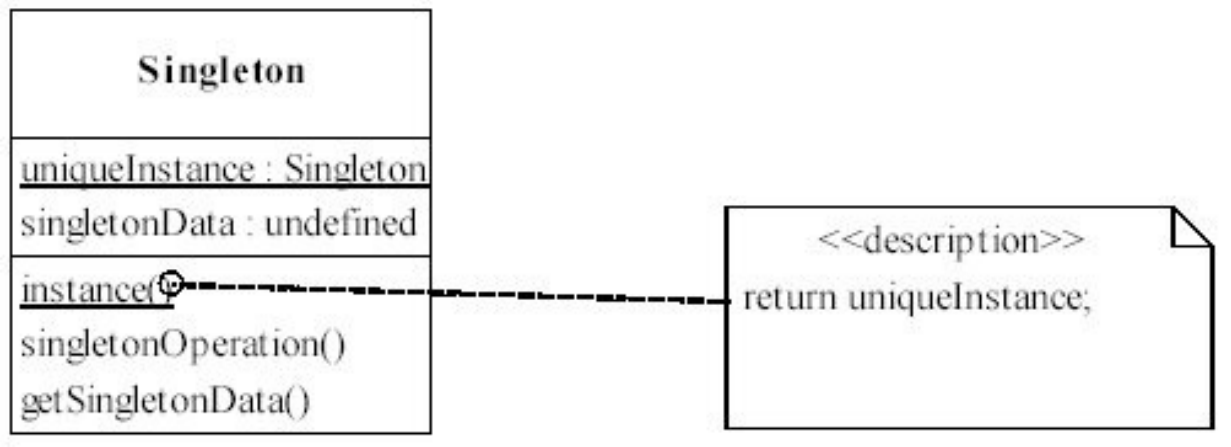
- Techniques de réutilisabilité (suite) :
  - **Les techniques citées précédemment ne s'appliquent qu'à la phase d'implémentation**
  - La réutilisabilité peut également être appliquée à **tous les niveaux** de développement d'un projet
  - **Les patterns de conception :** ils décrivent des **problèmes récurrents**, les solutions à ces problèmes et les contextes dans lesquels ces solutions sont considérées. Un pattern nomme une technique, **décrit ses coûts et ses avantages**. Il permet, à une équipe, de mettre un vocabulaire en commun pour décrire des modèles d'architecture (exemple : Model View Controler II)

REUTILISABILITE (4/11)

- Les trois grandes familles de design patterns :
  - **Les patterns de création :** ces patterns créent les objets pour vous, au lieu de les instancier directement :
    - *Le pattern **Factory Method** définit une **interface** pour créer un objet, mais **laisse les sous classes décider quelle classe instancier***
    - *Le pattern **Abstract Factory** fournit une **interface** pour créer des familles d'objets liés **sans spécifier leurs classes concrètes***
    - *Le pattern **Builder** **sépare la construction d'un objet complexe de sa représentation**, de telle sorte que plusieurs représentations différentes peuvent utiliser le même processus de construction*
    - *Le pattern **Prototype** spécifie le **type d'objet à créer** en utilisant une instance "prototype", et **crée un nouvel objet en effectuant une copie de cet objet***
    - *Le pattern **Singleton** assure qu'une classe a **uniquement une instance**, et fournit un accès à cette instance*

REUTILISABILITE (5/11)

- Diagramme de classe du pattern singleton :



REUTILISABILITE (6/11)

- Les trois grandes familles de design patterns :
  - **Les patterns structurels :** ces patterns composent des groupes d'objets en de larges structures, telles que des interfaces **graphiques complexes** :
    - *Le pattern **Adapter** convertit **l'interface d'une classe en une autre interface***
    - *Le pattern **Bridge** **découple une abstraction de sa représentation** pour que les deux puissent évoluer indépendamment*
    - *Le pattern **Composite** compose les objets dans une **structure arborescente** qui représente une hiérarchie "partie-de"*
    - *Le pattern **Decorator** attache de nouvelles **responsabilités à un objet dynamiquement***
    - *Le pattern **Facade** fournit une **interface unifiée** à un ensemble d'interfaces dans un sous-système*
    - *Le pattern **Proxy** fournit un suppléant pour un objet afin de **contrôler son accès***

REUTILISABILITE (7/11)

- Les trois grandes familles de design patterns :
  - **Les patterns comportementaux :** ces patterns définissent la **communication entre les objets du système et comment le flot de données est contrôlé dans un programme complexe** :
    - Le pattern **Chain of responsability** évite de coupler l'émetteur d'une requête et son receveur, en donnant la possibilité à plusieurs objets d'y répondre
    - Le pattern **Command** encapsule une requête dans un objet, pour permettre de paramétrer des clients avec différentes requêtes, de gérer des queues ou des traces de requêtes, ou de permettre des opérations annulables
    - Le pattern **Iterator** fournit un moyen **d'accéder séquentiellement** aux éléments d'une agrégation d'objets sans dévoiler sa représentation
    - Le pattern **Mediator** définit un **objet qui encapsule les interactions d'un ensemble d'objets**
    - Le pattern **Memento**, sans violer l'encapsulation, capture et externalise l'état interne d'un objet pour permettre de restaurer cet état plus tard
    - Le pattern **Observer** définit une relation **un-à-plusieurs telle que lorsqu'un objet change, tous les objets liés sont notifiés** de ce changement et mis à jour automatiquement…

REUTILISABILITE (8/11)

- Pour utiliser les patterns de conception :
  - Se **familiariser** avec l'utilisation des patterns de conception
  - Apprendre à identifier les **bons patterns de conception**
  - **Comprendre** leur applicabilité
  - Apprendre **à adapter un pattern** à ses besoins
  - Apprendre à **évaluer efficacement** les compromis durant la conception

REUTILISABILITE (9/11)

- Exemple d'utilisation des patterns de conception : MVC 1 - 2
  - L'idée de découpler modèle et vues permet de conserver une séparation nette entre le modèle et la façon de le représenter. Cette idée est applicable à un problème plus général : découpler des objets de façon à ce **qu'un changement d'un objet puisse affecter un nombre variable d'autres objets sans que l'objet qui a changé ait à connaître en détail les autres objets**.
    - *Ce problème plus général est décrit par le patron de conception **Observer**.*
  - Une autre caractéristique du modèle MVC est de permettre l'imbrication de vues les unes dans les autres, afin de permettre la construction de vues plus complexes. MVC supporte les vues imbriquées à l'aide de la classe **CompositeView**, une sous-classe de View. Un objet de la classe **CompositeView** se comporte exactement comme un objet de la classe **View**, et peut être utilisé partout où un objet de la classe **View** est utilisé, mais en plus, un objet de la classe **CompositeView** permet de **gérer des vues imbriquées**.

REUTILISABILITE (10/11)

- Exemple d'utilisation des patterns de conception :
  - Encore une fois, l'idée d'imbriquer des objets et de traiter l'objet résultant comme un seul objet individuel correspond à un problème de conception plus général.
    - *Ce problème plus général est décrit par le patron de conception **Composite**.*
  - Le modèle MVC permet également de modifier la façon dont **réagit une vue à une entrée faite par l'usager** sans changer son apparence visuelle. MVC encapsule le mécanisme de réponse dans un objet **Controler**. Les différents types de **Controler** sont organisés dans une hiérarchie de classe, et une vue utilise une instance d'une des sous-classes de **Controler**.
    - *La relation View-Controler est décrite par le patron de conception Strategy.*

REUTILISABILITE (11/11)

- Pour décrire un pattern de conception :
  - Le nom du patron
  - Son intention
  - Une motivation
  - Son applicabilité
  - Sa structure
  - Les classes ou objets participants
  - Les collaborations entre les participants
  - De même que des commentaires sur les conséquences d'utilisation, sur certains compromis possibles, sur l'implémentation et sur les patrons reliés.

REUTILISABILITE

- Principes de réutilisabilité (suite)
  - **Les patterns de conception :**
    - *MVC pour la couche présentation*
    - *DAO pour la couche accès aux données…*
  - **Les frameworks techniques :**
    - *Struts, Barracuda, Expresso, Turbine, Avalon, Spring*
    - *log4J pour la gestion de log*
    - *Castor, hibernate, pour le mapping relationnel…*
  - **Les bibliothèques de composants techniques :**
    - *taglibs*
    - *javamail*
    - *jaxp...*

Modèle Vue Contrôleur (1/2)

- MVC (Model - View - Controller) est un pattern destiné à la conception d'applications **GUI**. Son principe de base est la séparation des composants suivants :
  - **Le modèle** : il conserve toutes les **données relatives à l'application** (sous quelque forme que ce soit : base de données, fichiers…) et contient la **logique métier de l'application**.
  - **La vue** : elle a pour rôle d'offrir **une présentation du modèle** (IHM). On peut avoir de **nombreuses vues pour un même modèle**, chacune présentant les informations de manière différente (on pourrait ainsi imaginer une liste d'articles présentable à la fois sur l'écran d'un navigateur web, sur un PALM, sur un minitel ou sur une imprimante)
  - **Le contrôleur** : c'est le composant qui répond **aux actions de l'utilisateur**. Il traduit les évènements de l'IHM en modifications du modèle et définit également la manière dont l'IHM doit réagir face aux interactions de l'utilisateur.

Modèle Vue Contrôleur (2/2)

- Les avantages du modèle MVC sont les suivants :
  - **Les 3 parties de l'application** – logique de présentation, logique métier et logique applicative – sont parfaitement **indépendantes**. Ainsi, la programmation de chacune peut être distribuée à des équipes de développement différentes.
  - La **maintenance** de l'application est plus **souple**. Ce découpage permet par exemple de modifier la présentation sans toucher à la structure du site ni à la logique métier.
- L'inconvénient du modèle MVC est le suivant :
  - Une architecture de type MVC nécessite un temps d'apprentissage non négligeable.
- Au final, cet inconvénient est largement contrebalancé par le temps gagné lors des évolutions et de la maintenance.