Politehnica University of Bucharest
Faculty of Engineering in Foreign Languages

Méthodes du Développement du Logiciel

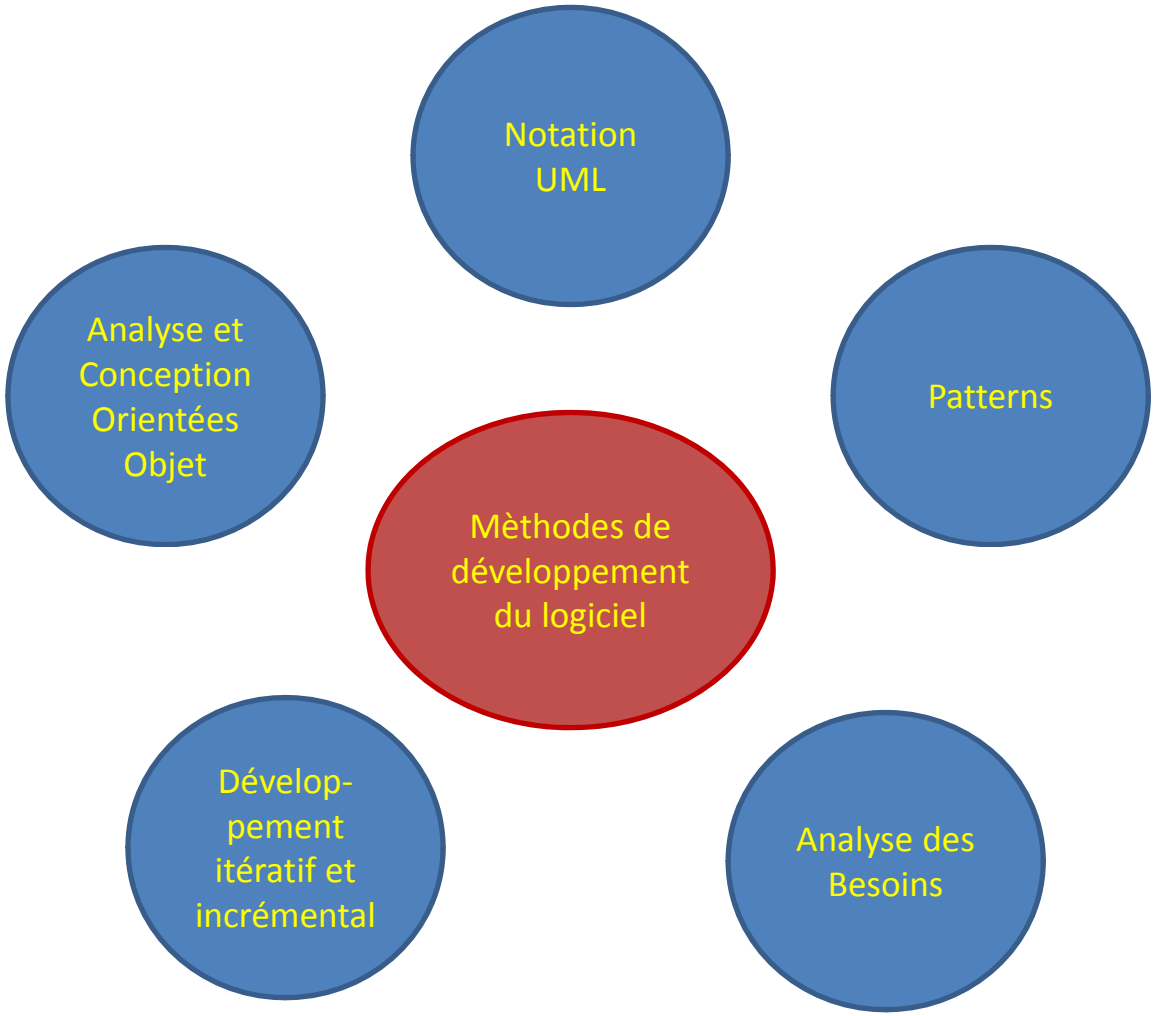
Luca Dan Serbanati

Copyright 2010

Introduction au course

- La programmation est une activité plaisante, mais la production de logiciels de qualité est un exercice difficile avec plusieurs étapes: analyse et conception, définition des solutions, identification des éléments à programmer, transformation de la conception en processus faciles a communiquer, à réviser, à implémenter et à faire évoluer.
- Une approche systématique de l'analyse et de la conception orientées objet (A/COO), domaines aujourd'hui matures, aide l'ingénieur de logiciel à la gestion de la complexité du logiciel.
- UML (Unified Modeling Language) est devenu le langage de représentation graphique standard pour les projets de conception logicielle. Il est employé pour véhiculer les notions tout au long du cycle de vie du logiciel.
- Réutilisation tout au long du processus de fabrication du logiciel est fondamentale pour la qualité du logiciel produit et pour la productivité de ce processus. Heuristiques abstraites, règles et bonnes pratiques des techniques orientées objet sont synthétisées en patterns.
- Le Processus Unifié est un processus itératif and incrémental, centré sur l'architecture et piloté par les risques. Il est un authentique guide pour développer logiciels.

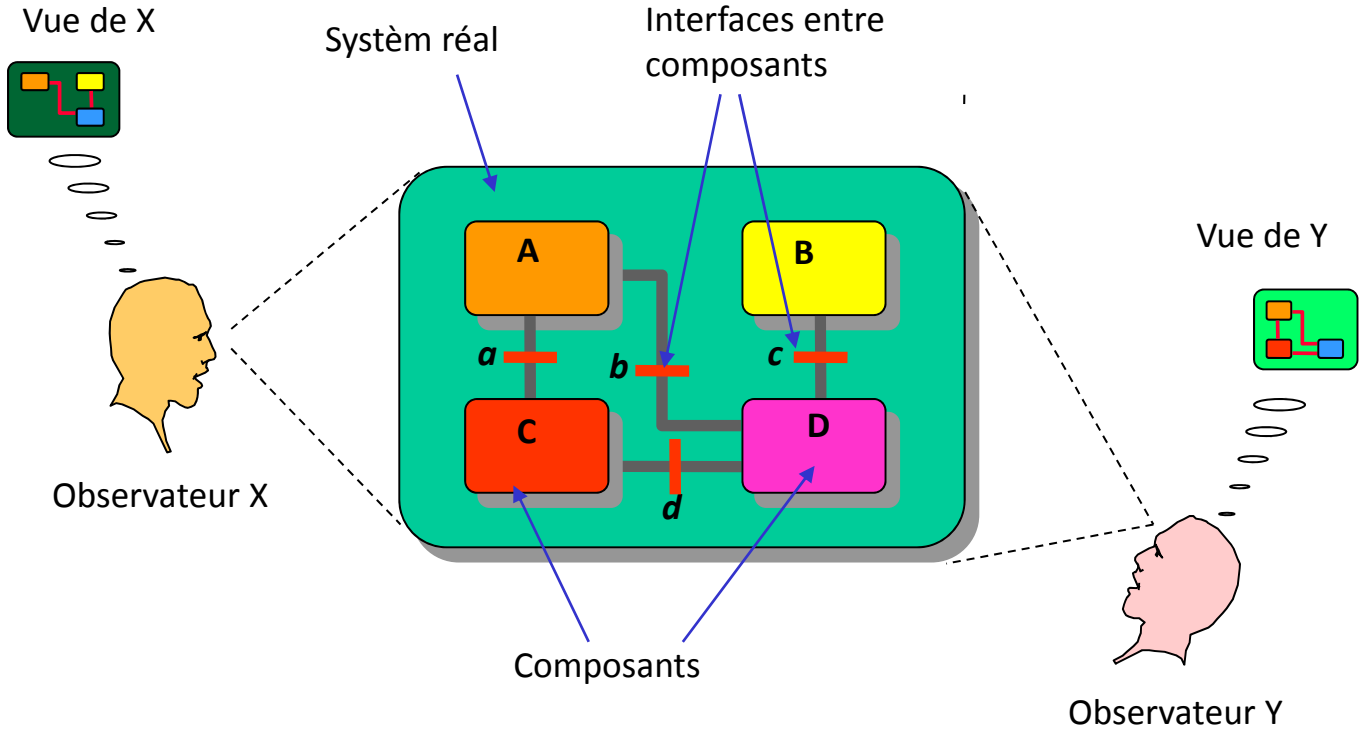
Techniques et sujets abordés



Partie 1.
Introduction à l'Ingénierie du Logiciel

Système

- Système =une collection de composants que interagissent et que paraissent come une entité unitaire quand sont regardés par un observateur externe.
- État de système = une abstraction d'une situation stable d'un système.
- Comportement = une séquence d'états de système (l'évolution du système)
  Une classification possibles des systèmes : naturel et artificiel.



Un modèle informationnel est fondamentalement une vue abstraite simplifiée de la réalité complexe traduite dans les informations. Il peut se fixer sur les vues particulières, en appliquant le principe ‹ diviser et conquérir › utilisé pour un problème composé.

Systèmes d'Information

Système d'Information (Informationnel)= un ensemble organisé de ressources permettant d'acquérir, traiter, stocker, communiquer des informations (sous forme de données, textes, images, sons, etc…) dans des organisations. Ces ressources sont interreliées . Les composants principaux :

- ordinateurs
- données, compris bases de données
- systèmes de télécommunications (l'infrastructure de communication)
- personnel entraîné
- procédures manuelle et automatisé.

- Le système d'information est donc par nature hétérogène, complexe, evolutif
- Les organisations et les individus comptent sur les systèmes d'information pour gérer leurs opérations (planification, contrôle, coordination et aide à la décision), concourir dans le marché, fourniture des services, et enrichir la vies des personnes.
- Un système d'information peut être manuel ou même informel.

Evolution:

- Années 60 : automatisation des processus administratifs
- Années 70 : automatisation des processus de production
- Années 80 : automatisation du travail individuel de bureau
- Années 90 : la mise en réseaux des micro-ordinateurs professionnels
- Années 2000 : automatisation des échanges

Systèmes Informatiques

Système informatique = un système d'information informatisé. Il contient :
- matériel
- Logiciel
- Utilisateurs de logiciel
- bases de données
- Documentation

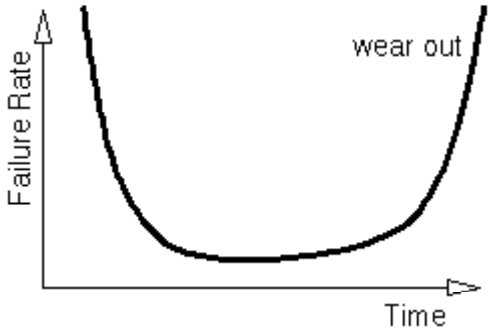
Système de logiciel = un composant d'un système informatique. Il contient:
- programmes
- structures de données
- documentation.

- Une application est un ensemble de programmes du type logiciels, articulés entre eux, utilisés pour automatiser ou assister des tâches de traitement de l'information dans un domaine particulier.

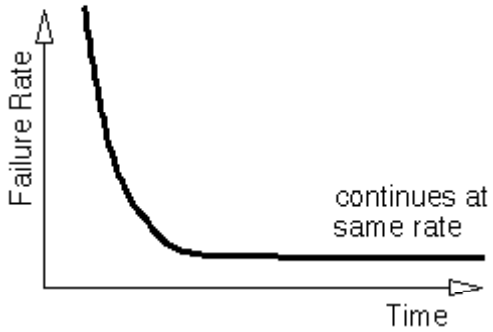
Characteristiques du Logiciel

1. Le logiciel n'est pas fabriqué dans le sens classique, mais développé ou ingénieurisé.
   - Pour le matériel la fabrication peut introduire problèmes à la qualité qui ne sont pas existant pour le logiciel.
   - De plus, un processus séparé de fabrication doit est développé.
   - Les coûts de logiciel de développement sont concentrés dans l'ingénierie.
   - Le logiciel souffre seulement des erreurs de conception.

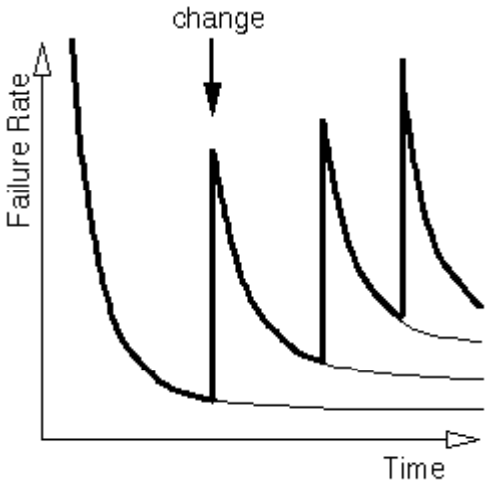
2. Le logiciel ne s'use pas :



Hardware



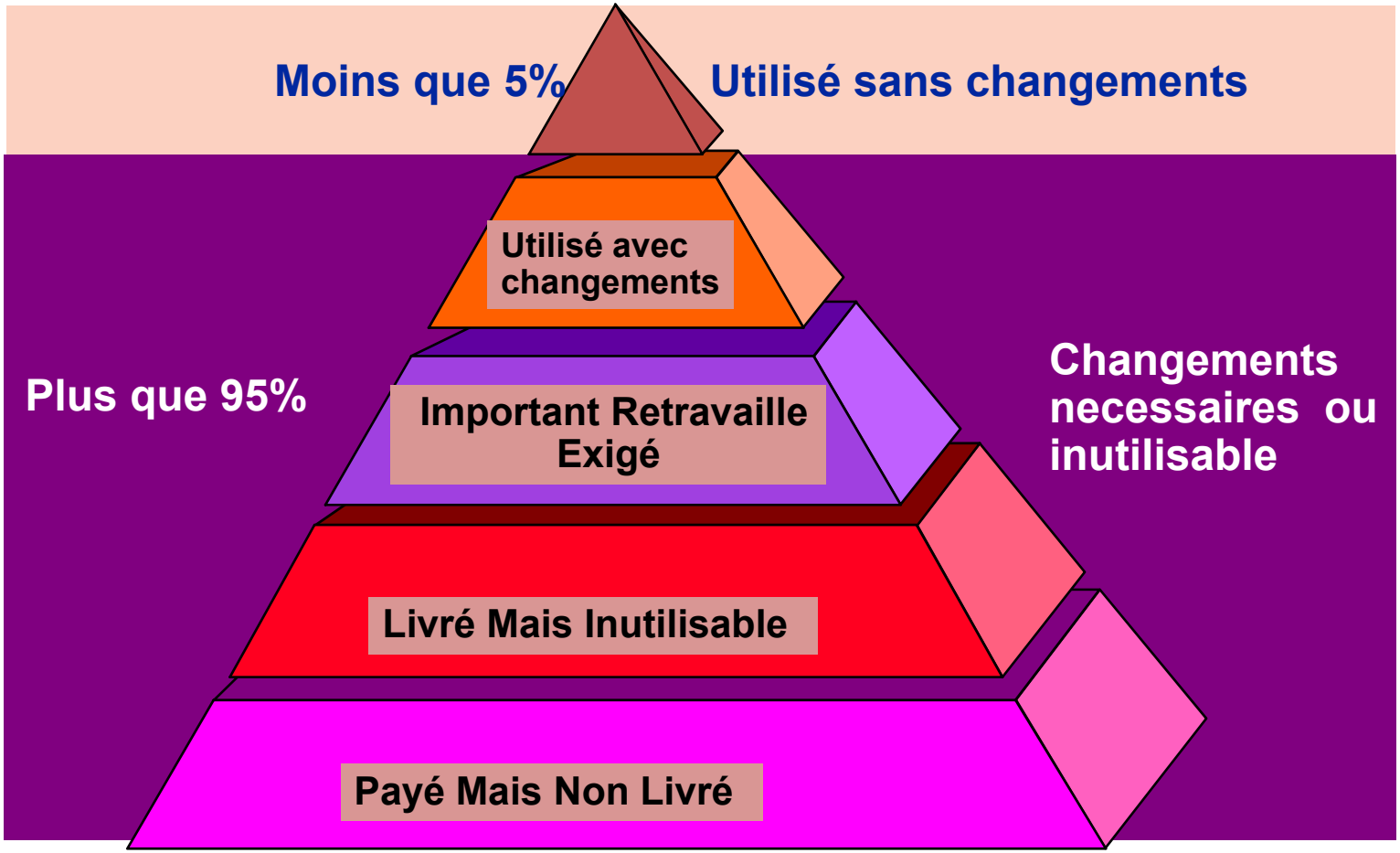
Software –Courbe idéal



Software – Courbe rèale

3. La plupart de logiciel est coutume-construit, au lieu d'est assemblé des composants actuels.

La Crise de logiciel



Les symptômes de problèmes de développement logiciel et leurs causes premières

Les symptômes des Problèmes du Développement de logiciel :

- Implémentation imprécise des besoins d'utilisateur final
- L'incapacité de traiter avec des exigences(des conditions) changeantes
- Modules qui n'ajustent pas ensemble
- Le logiciel c'est dur de maintenir(d'entretenir) ou s'étendre
- Tardive découverte de défauts sérieux de projet
- Faible qualité du logiciel
- Inacceptable performance du logiciel

Les causes premières de Problèmes de Développement Logiciels :

- Gestion d'exigences insuffisante
- Ambiguës et imprécises communications
- Écrasante complexité
- Inconsistances non détectées parmi exigences, conceptions and mises en œuvre
- Test insuffisant
- Aucun effort d'équipe coordonné
- Un processus construire-et-livrer incertain
- Évaluation subjective de statut de projet
- Réduction de risque retardé en raison du développement de type de chute d'eau
- Propagation de changement non contrôlée
- Automation Insuffisante

Logiciel

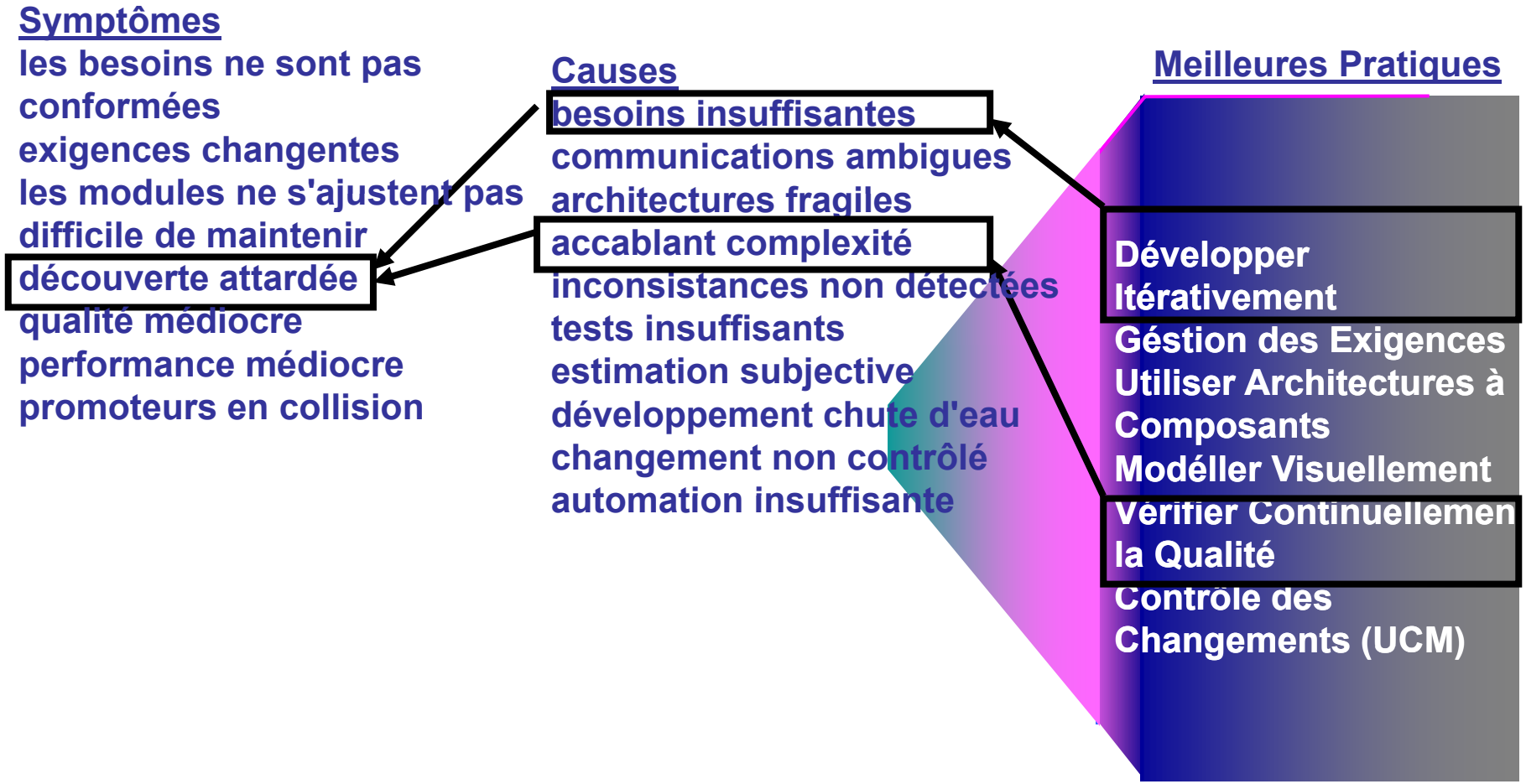
- **Logiciel** est un ensemble d'informations relatives à des traitements effectués automatiquement par un appareil informatique
- Y sont inclus les instructions de traitement, regroupées sous forme de programmes, des données et de la documentation. Le tout est stocké sous forme d'un ensemble de fichiers dans une mémoire
- Le **développement de logiciel** comprend l'ensemble des étapes et processus qui permettent de passer de l'expression d'un besoin informatique à un logiciel fonctionnel et fiable.

Activités necéssaires:

  - Gestion des exigences : le cahier des charges, (spécifications) ;
  - Planification de projet ;
  - Supervision et suivi de projet ;
  - Gestion des achats ;
  - Assurance qualité ;
  - Gestion de la configuration;
  - Mesures et analyses.

- « Programming-in-the-large » = les projets sont généralement de grande envergure et dépassent souvent les 10 000 lignes de code. Ces projets nécessitent une équipe de développement bien structurée.

Adressant les causes élimine les problèmes



Les meilleures pratiques = Une série organisée et documenté de principes, méthodes, et processus que augmente la qualité et la productivité de développement de logiciel.

Liste de Controle dei Facteurs de La Qualité de logiciel

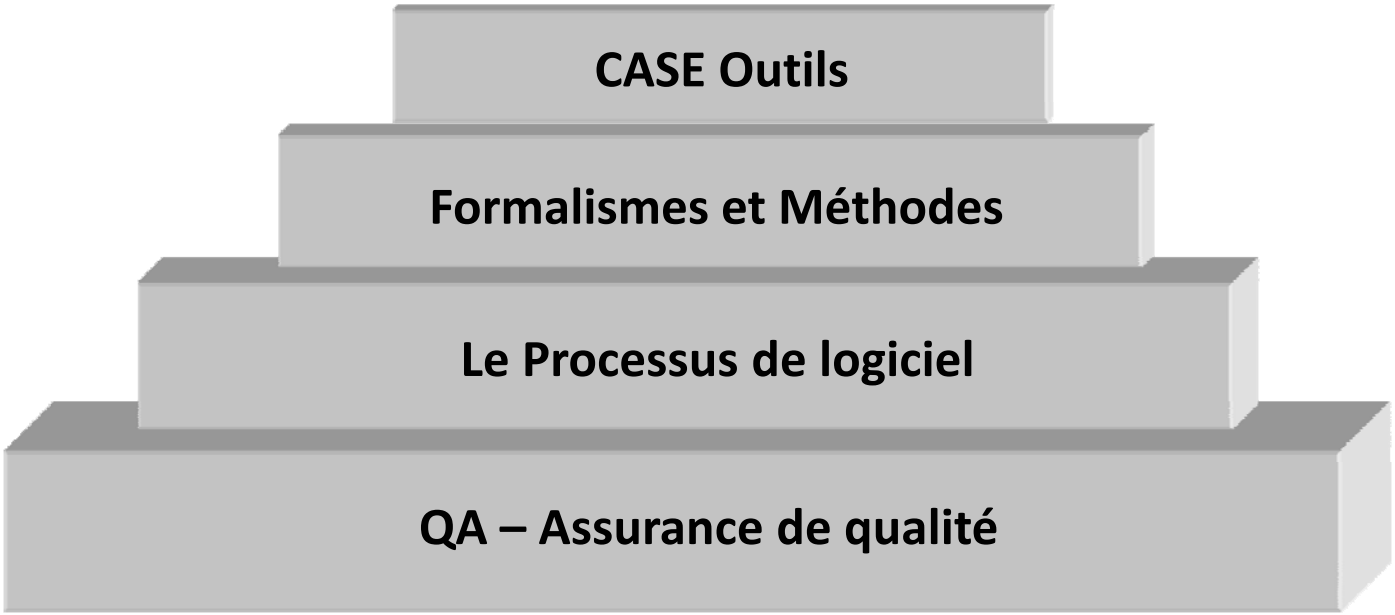
■ Facteur externe ■ Facteur interne

Exactitude	La mesure à laquelle le logiciel satisfait son cahier des charges(spécifications) et accomplit les objectifs de l'utilisateur. Aussi la mesure à laquelle le logiciel est la faute libren(c'est-à-dire, libre de défauts de conception et de coder des défauts).
Efficacité	La mesure à laquelle le logiciel exécute ses fonctions destinées avec une consommation minimale de ressources de calcul.
Extensibilité	La facilité avec lequel le logiciel peut être modifié pour ajouter la fonctionnalité.
Intégrité	La mesure à laquelle l'accès non autorisé ou à la modification du logiciel ou des données peut être contrôlé dans un système informatique (manuellement ou avec des outils automatisés).
Interopérabilité	La capacité de deux ou plus systèmes pour échanger des informations et mutuellement utiliser les informations qui ont été échangées.
Maintenabilité	La facilité avec laquelle les composants de logiciel peuvent être maintenus au cours de leur vie utile prévu.
Modularité	La mesure à laquelle le logiciel est structuré dans des unités peu couplées et très cohésives (des modules)
Portabilité	La facilité avec lequel le logiciel peut être transféré aux nouveaux environnements d'exploitation, des plates-formes de matériel et des systèmes d'exploitation.
Réutilisabilité	L'étendue à laquelle les modules peuvent être utilisés dans les applications multiples.
Fiabilité	La capacité du logiciel pour exécuter une fonction exigée sous les conditions affirmées pour une période affirmée de temps.
Robustesse	La capacité du système pour contrôler des situations anormales.
Sécurité	La capacité du logiciel pour garantir la confidence, l'intégrité, et la disponibilité d'ordinateurs.
Testabilité	La mesure à laquelle le logiciel est facile de tester pour assurer qu'il exécute sa fonction destinée.
Possibilité de comprendre	La mesure à laquelle le logiciel est facile être compris.
Utilisabilité	La mesure à laquelle le logiciel est facile à apprendre et fonctionner.

Génie Logiciel

Le Génie Logiciel est l'application d'Ingénierie au logiciel, c'est-à-dire :

- L'établissement et l'usage de solides principes d'ingénierie pour obtenir économiquement de logiciel qui est fiable et s'exécute efficacement sur des vraies réelles, ou
- L'application d'une stratégie systématique, discipliné et mesurable au développement, l'exercice et la maintenance du logiciel, ou
- L'ensemble des méthodes, des techniques et des outils concourant à la production d'un logiciel, au-delà de la seule activité de programmation, ou
- **Procédures**, **méthodes**, **langages**, **ateliers**, imposés ou préconisés par les **normes** adaptées à l'environnement d'utilisation afin de favoriser la **production et la maintenance de composants logiciels de qualité.**



Le Processus de Logiciel

Le processus de Software est une structure imposée sur le développement d'un produit logiciel, un cadre où toutes les activités nécessaires devraient être effectuées pour obtenir un logiciel de qualité dans le temps et les coûts estimés.

.



Capability Maturity Model (CMM)

- La CMM a été initialement conçu comme un outil permettant d'évaluer objectivement la capacité des processus logiciel des entrepreneurs du gouvernement pour effectuer un projet de contrat logiciel.
- Bien que cela provient de la zone de développement logiciel, il peut être (et a été et est encore l'être) appliquée comme un modèle généralement applicable pour l'aider à comprendre la maturité de la capacité des processus logiciel des organisations dans divers domaines.
- Un tel modèle de maturité peut être utilisé comme référence pour évaluer les différentes organisations avec une comparaison équivalente. Le modèle décrit la maturité de la société fondée sur le manière l'entreprise gére le projet et les clients connexes.

Les aspects du CMM

Le CMM comporte les aspects suivants:

- Niveaux de maturité: Il s'agit d'un cadre en couches fournissant une progression à la discipline nécessaire pour s'engager dans l'amélioration continue.
- Key Process Areas: Une Région de Processus Clé (KPA) identifie un group d'activités connexes qui, s'ils sont effectués collectivement, réalisent une série d'objectifs considérés comme importants.
- Objectifs: Les objectifs d'une KPA de résumer les états qui doivent exister pour que cette KPA d'avoir été mis en œuvre de manière efficace et durable. Les objectifs signifient l'ampleur, les limites, et l'intention de chaque KPA.
- Caractéristiques communes: Les caractéristiques communes comprennent les pratiques qui mettent en œuvre et d'institutionnaliser une KPA. Cinq types de caractéristiques communes: Engagement à réussir, Capacité à accomplir, Activités effectuées, Mesure et Analyse, et mise en œuvre de la Vérification.
- Key Practices: Les pratiques clés décrivent les éléments de l'infrastructure et les pratiques qui contribuent le plus efficacement à la mise en œuvre et l'institutionnalisation des KPA.

Niveaux du CCM

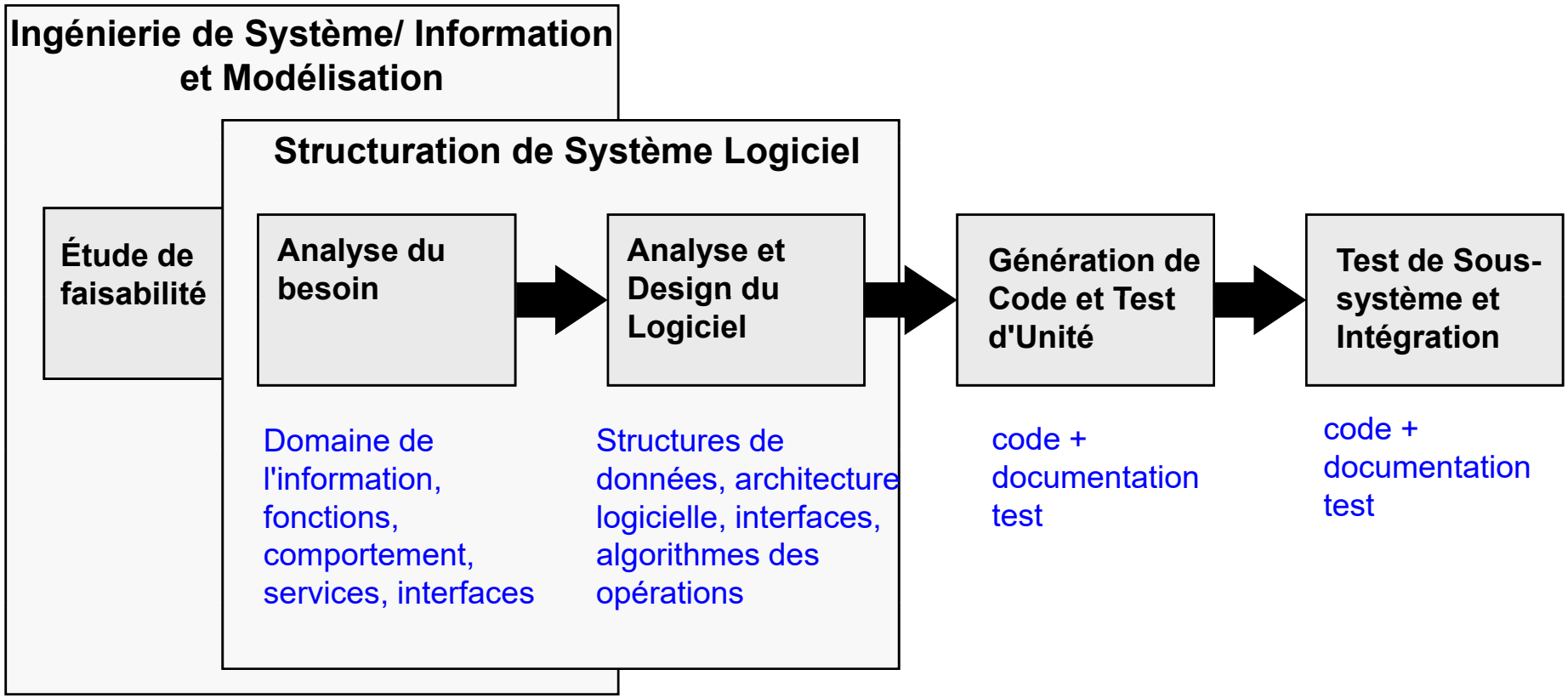
- Niveau 1 – Initial. Le processus de développement est chaotique et les résultats sont imprévisibles (la qualité, le temps pour dépenser, ...). Les processus ne sont pas d'ordinaire documentés et changent fondé sur l'utilisateur ou l'événement. Le succès dans ces organisations dépend de la connaissance, la compétence et la héroïque des gens dans l'organisation, et le niveau d'effort dépensé par l'équipe.
- Niveau 2 – Répétable. Les procédures sont documentées (le coût, le plan, la fonctionnalité est suivie à la trace). Les succès donc précédents sont susceptibles d'être répétés pour les applications similaires. Quelques processus de logiciel de développement sont susceptibles d'être répétés. L'état du projet et la livraison de services sont visibles à la gestion aux points bien-définis. Il y a toujours un risque significatif de dépasser les estimations de coût et temps.
- Niveau 3 –Défini. Les processus de logiciel standard de l'organisation, qui sont la base pour ce niveau, sont établi et exposés à quelque amélioration progressivement. Ces processus standard sont utilisés pour établir l'homogénéité à travers l'organisation. Les projets établissent leurs processus en appliquant l'ensemble des processus standard de l'organisation, adaptés, si nécessaires, en accord avec des directives aussi normalisées.
- Niveau 4 –Géré. Le processus de logiciel est instrumenté. Les mesures (métriques) du processus sont analysées pour ajuster le processus. Utilisant ces métriques, la direction peut contrôler efficacement le processus. Les organisations à ce niveau fixe objectifs de qualité quantitatifs pour le processus et l'entretien du logiciel.
- Niveau 5 – Optimisé. Il se concentre sur d'amélioration continue de l'exécution de processus par des améliorations tant technologiques progressives qu'innovatrices. Des objectifs d'amélioration de processus quantitatifs sont établis pour l'organisation. Ils sont révisés continuellement pour refléter des objectifs d'affaire changeants et utilisé comme des critères dans la direction de l'amélioration de processus.

La méthodologie de développement de logiciel

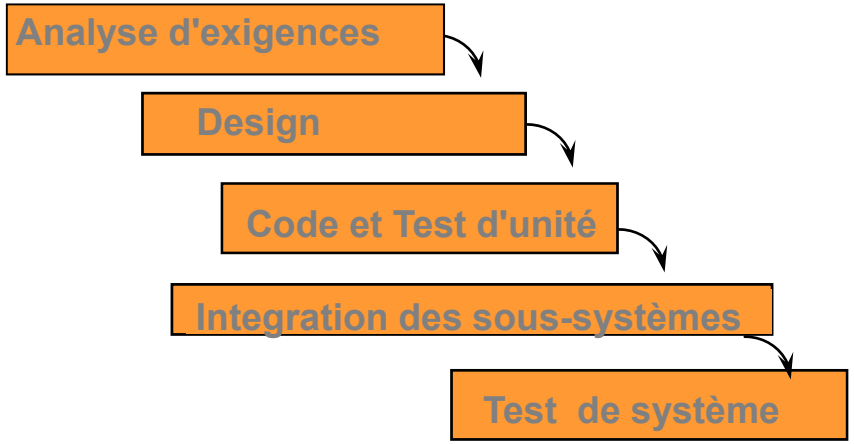
- Une méthodologie de développement de logiciel consiste en deux composants complémentaires :
  - Un *processus standard* = la séquence de tâches et de décisions qui se déplace le projet du début à l'achèvement ;
  - Une *notation standard* = la forme de communication qui est utilisée pour documenter la connaissance et les décisions ont accumulé comme les produits du processus général.

- Il n'y a pas de processus standard de développement de logiciel. Toutes les méthodologies sont centrées d'un propre modèle du cycle de vie de logiciel qui inclut le projet et l'entretien du produit.
- Toutes les méthodologies ont au moins un concept dans commun : leur processus est itératif parce qu'il n'y a pas une solution algorithmique pour le développement de logiciel.

Cycle de vie en « Cascade » (Waterfall LifeCycle)



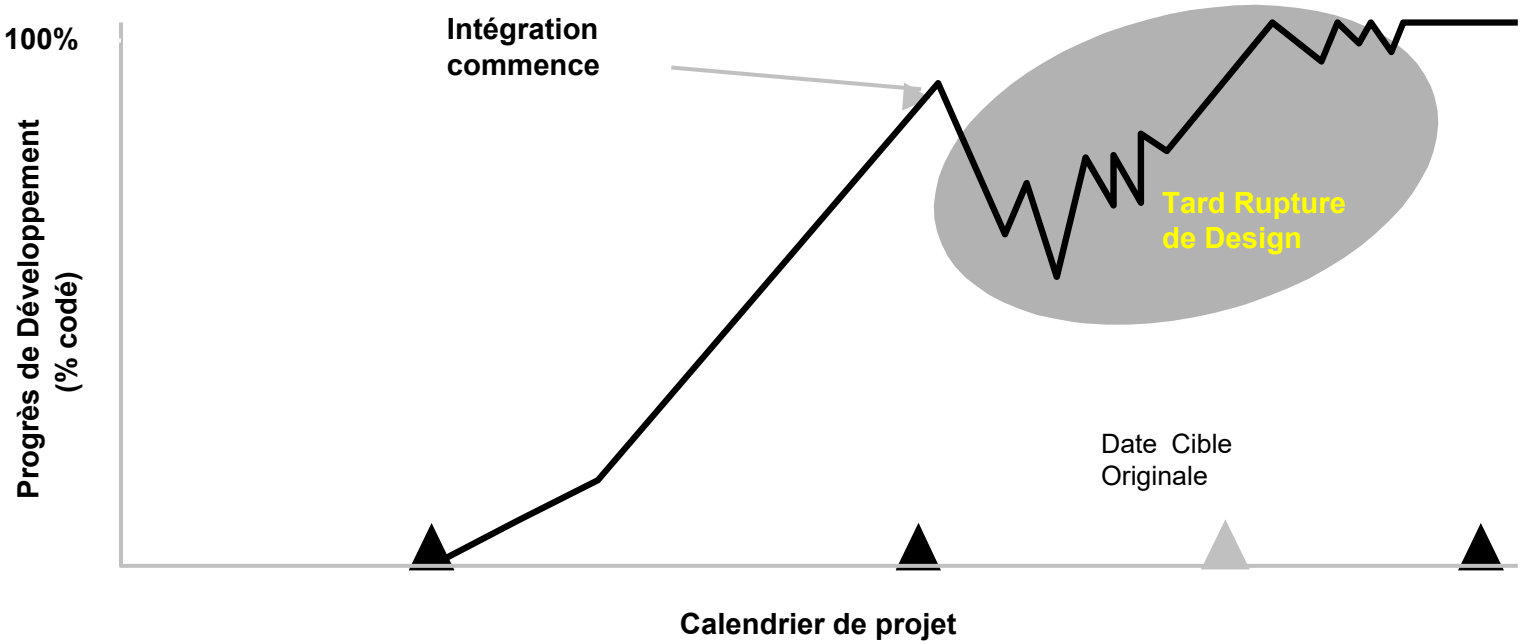
Inconvénients de Processus en Cascade



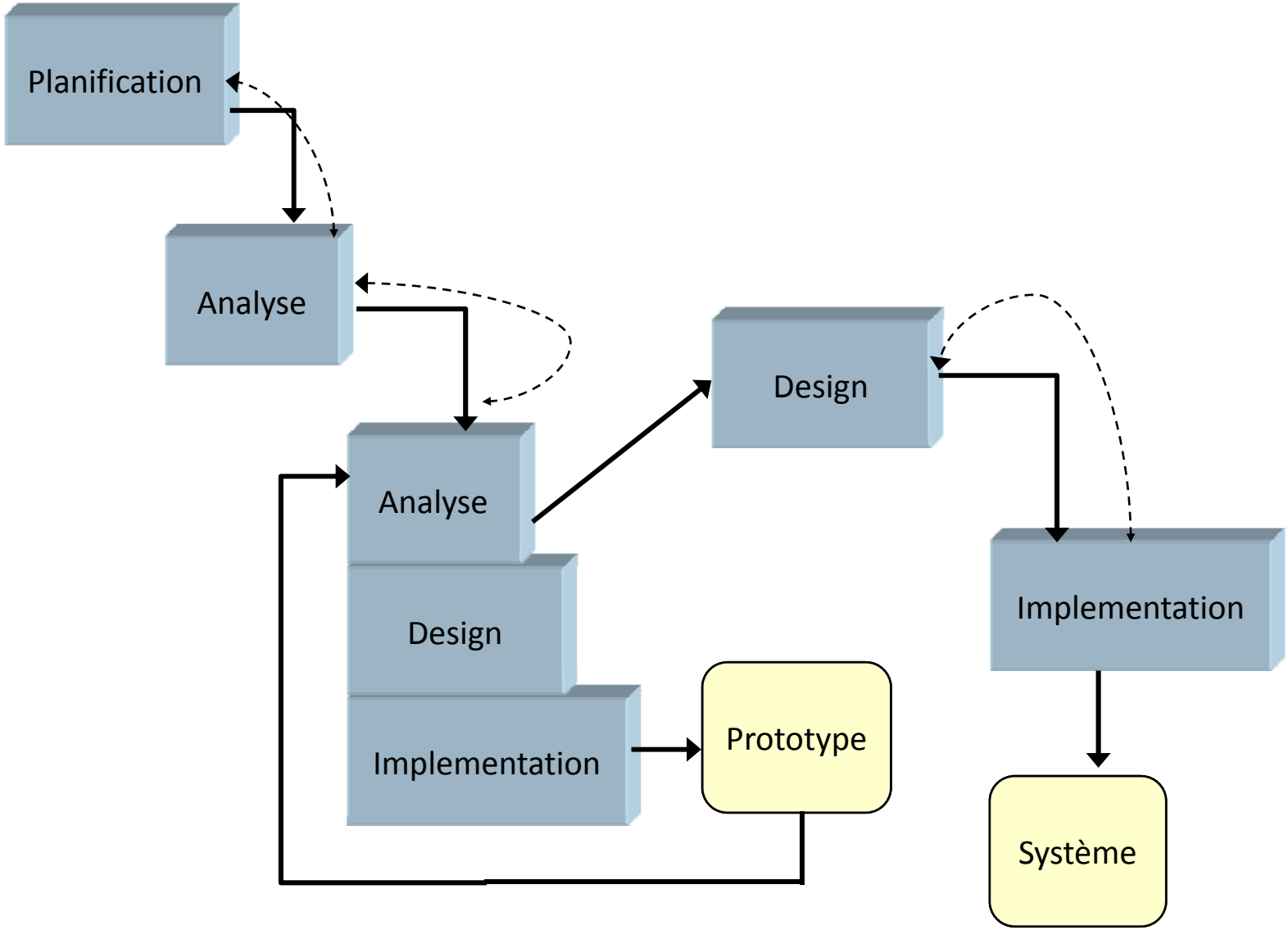
- Retard la confirmation de la résolution des risques critiques
- Mesure le progrès en évaluant les produits intermédiaires qui procurent faibles prédiction de temps-à-achèvement
- Retarde et agrège l'intégration et le test
- Écarte le premier déploiement
- Fréquemment aboutit aux itérations non planifiées

Activités séquentielles :

Exigences ➡ Design ➡ Code ➡ Integration ➡ Test

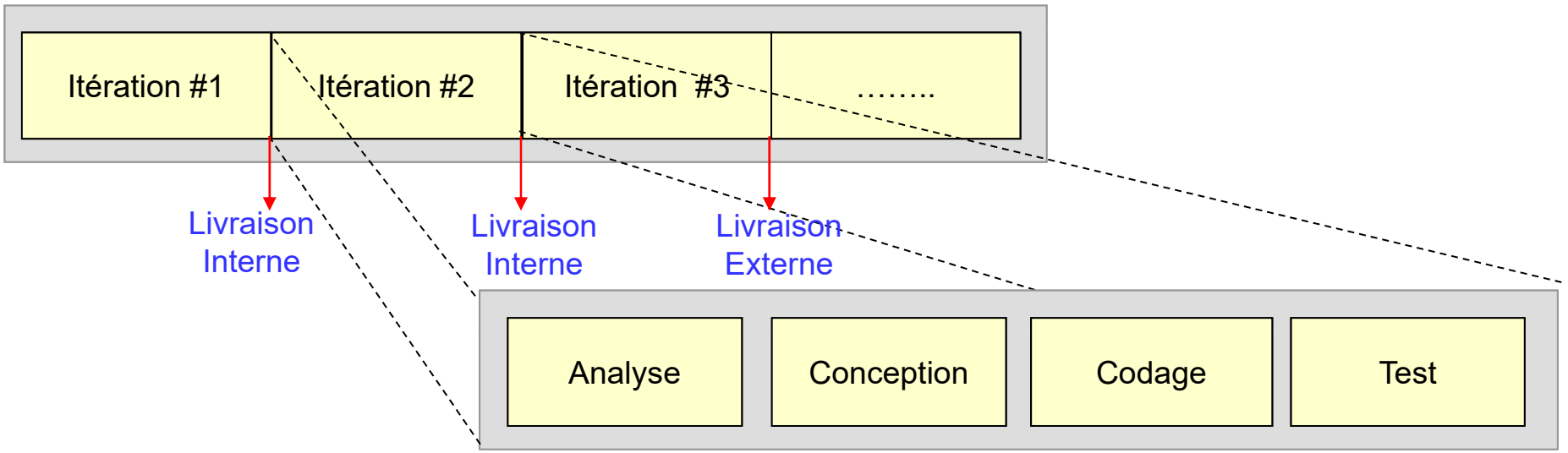


Prototyping



Développement Itératif

- Le développement itératif est une stratégie de planification des modifications du logiciel dans laquelle le temps est utilisé pour réviser et améliorer les parties du système. La production d'une itération est examinée pour la modification et particulièrement pour réexaminé les objectifs des itérations successives.
- Le développement s'organise en une série de projets très courts, de durée fixe (2-3 semaines) nommées itérations. Le résultat de chaque itération est un système partiel exécutable, testé et intégré. Chaque itération comprend ses propres activités: analyse des besoins, conception, implémentation et tests.

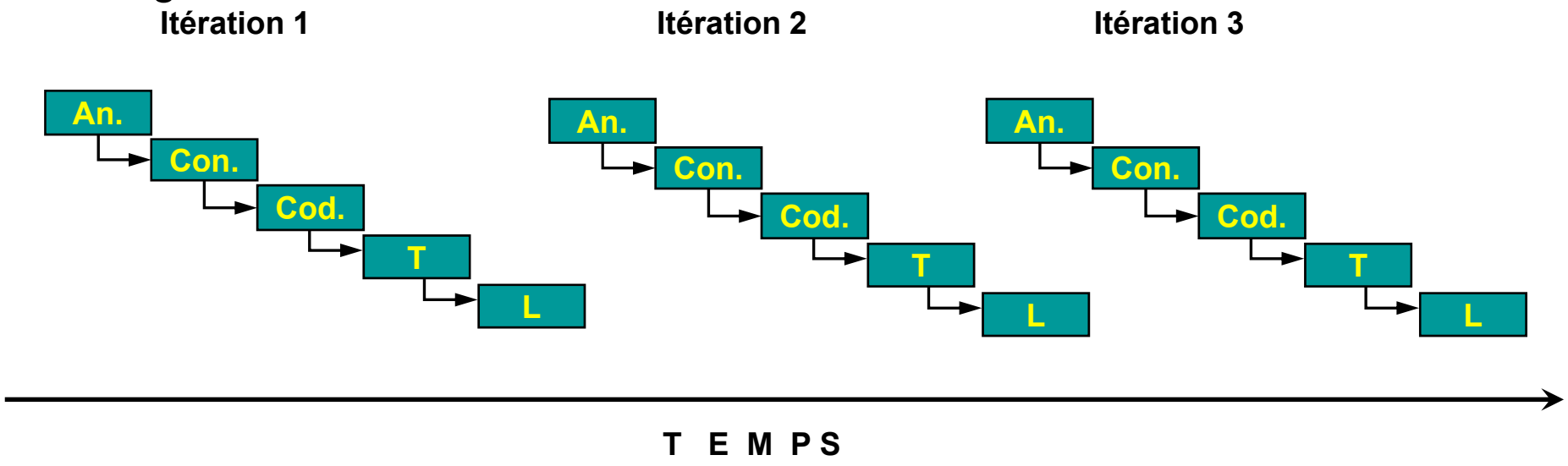


Avantages du développement itératif

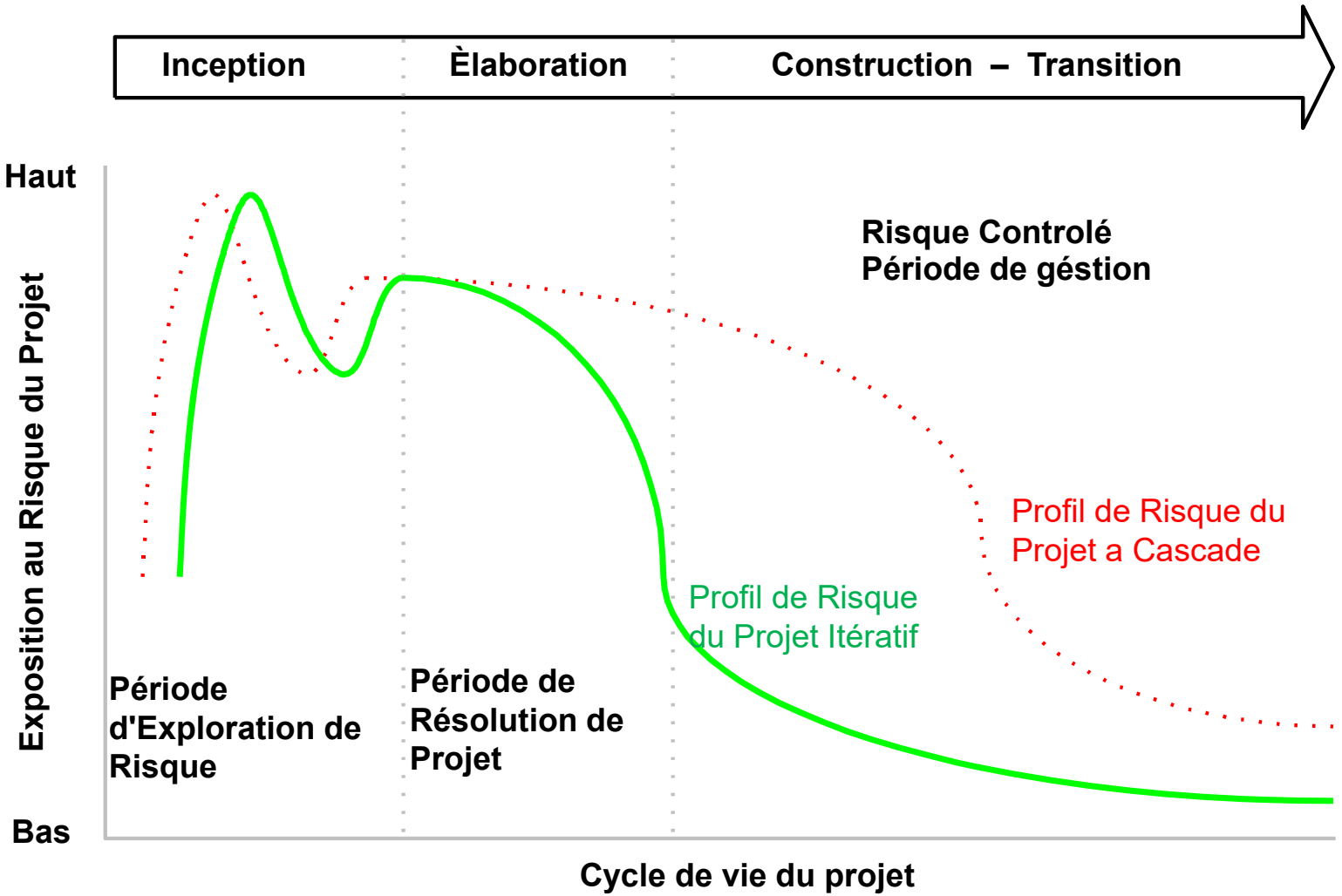
Le cycle de vie itératif est fondé sur la croissance et l'affinement successifs d'un système par le biais d'itérations multiples, le feed-back e l'adaptation cycliques étant les moteurs principaux qui permettent de converger vers un systèmes satisfaisant.

Avantages:

- Diminution des échecs, amélioration de la productivité et de la qualité
- Gestion précoce des risques élevés avant la décision de grands investissements
- Feed-back, implication des utilisateurs et adaptation précoces
- Complexité gérée
- Possibilité d'exploiter méthodiquement les leçons tirées d'une itération
- Progrès immédiatement visibles.

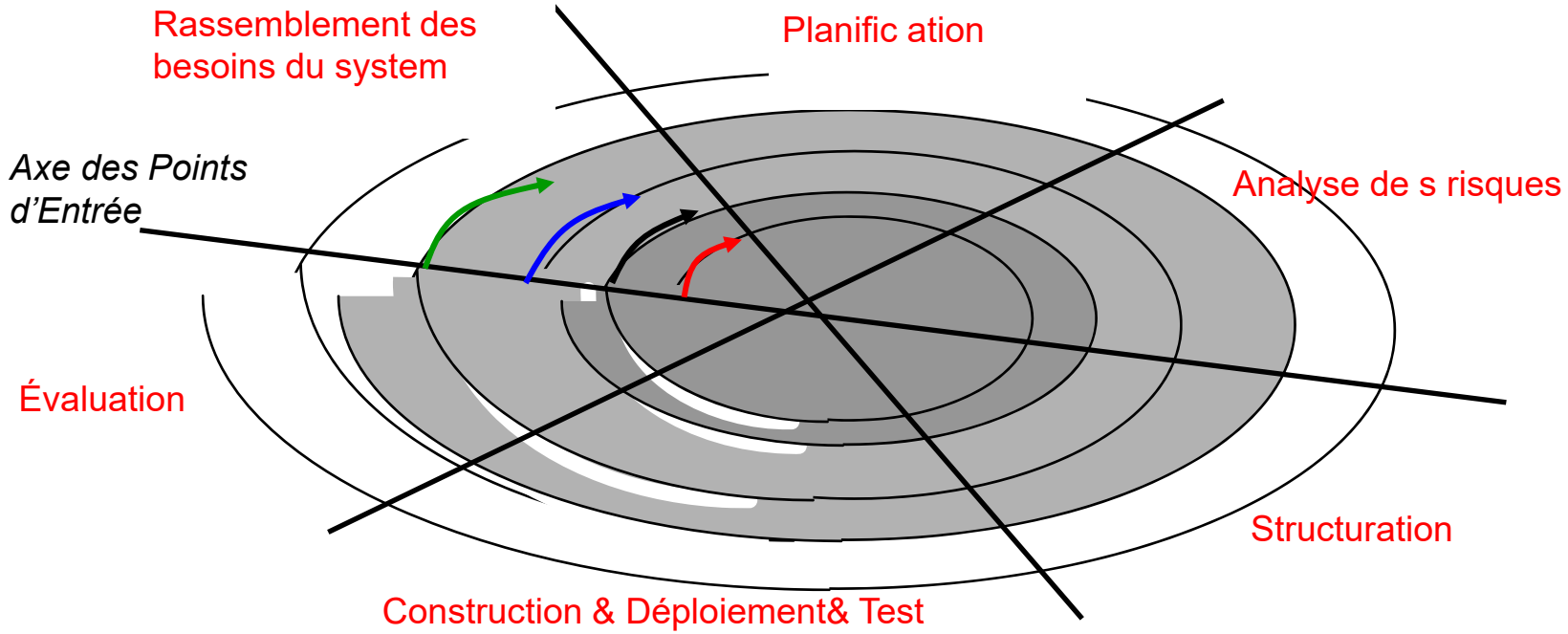


Itératif Vs. Cascade: Profils de Risque

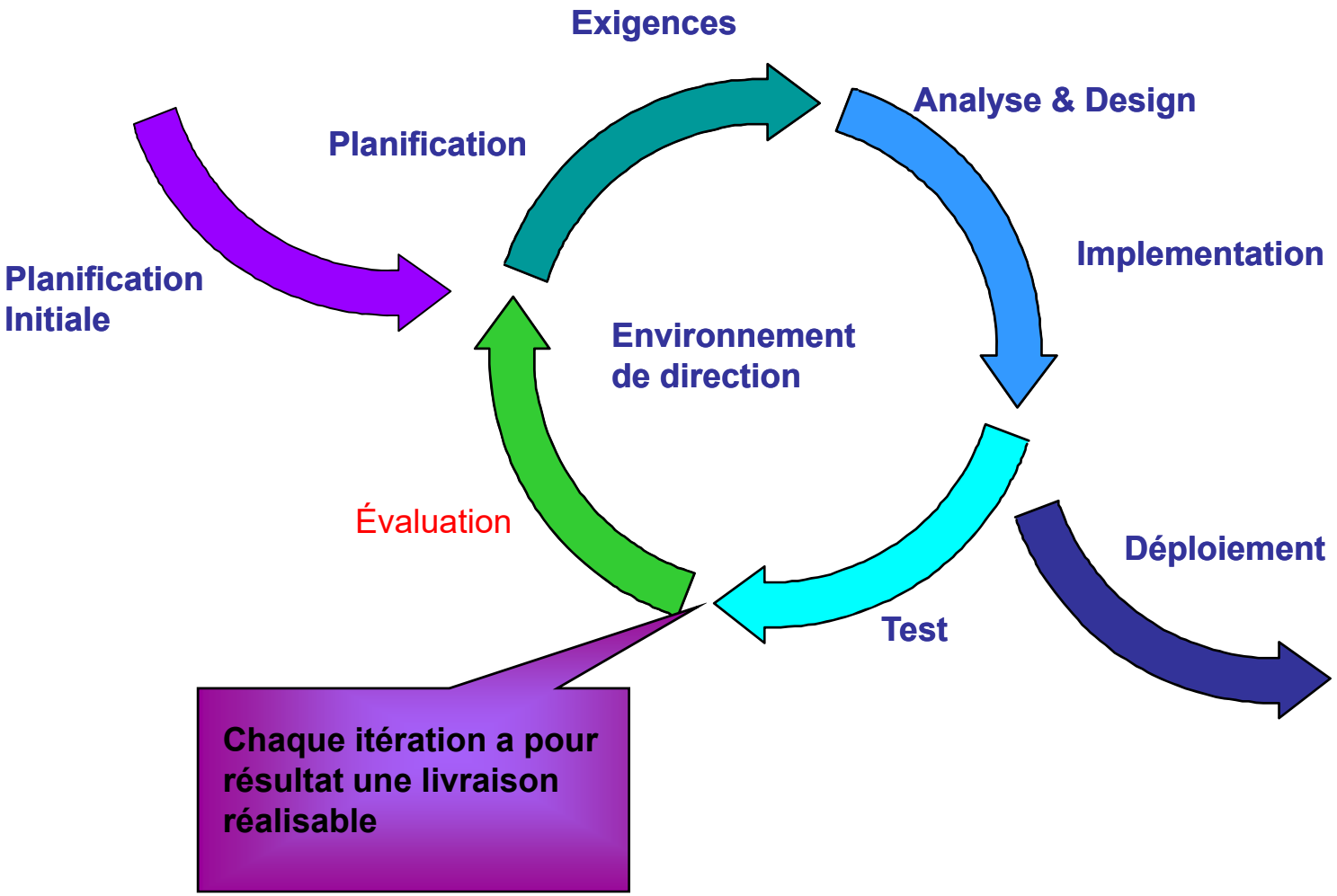


Le modèle « en spirale »

- Le modèle « en spirale », aussi connu comme le modèle de cycle de vie en spirale, est une méthode de développement de systèmes utilisée dans la technologie d'informations. Ce modèle combine les caractéristiques du modèle de prototyping et du modèle en cascade.
- Le modèle en spirale est voulu pour les projets grands, chers et compliqués.

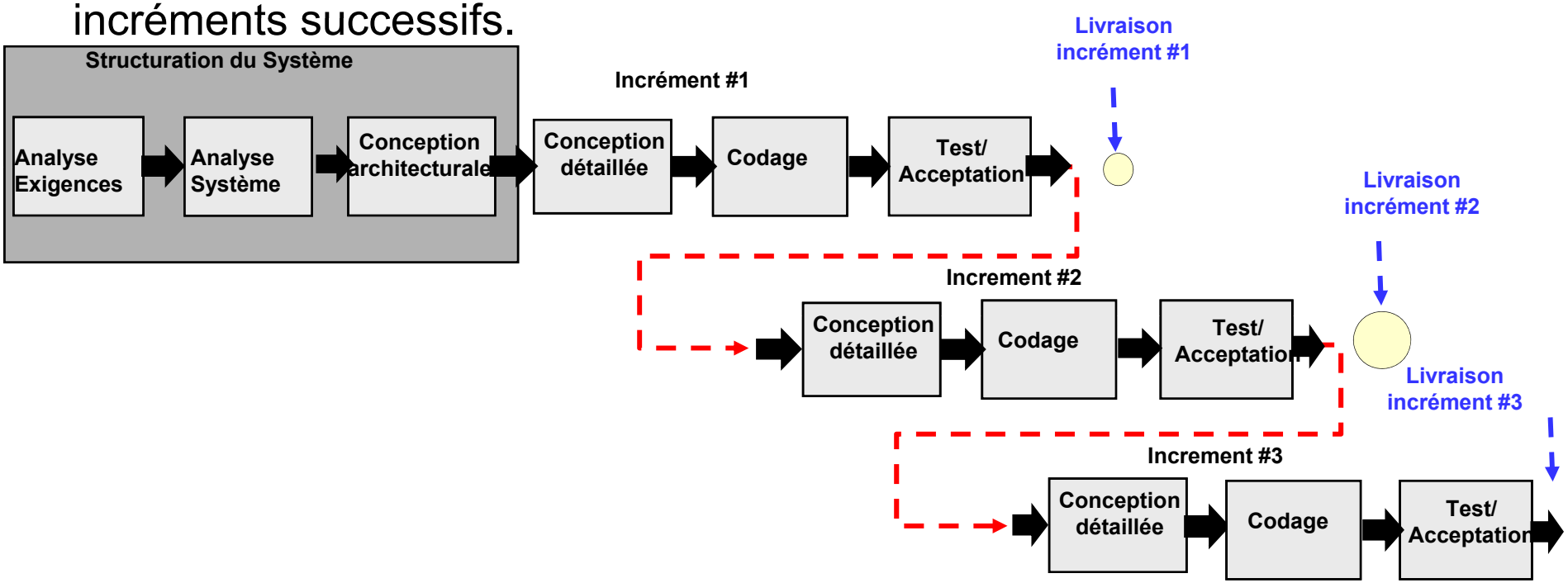


Développement Incrémental (I)



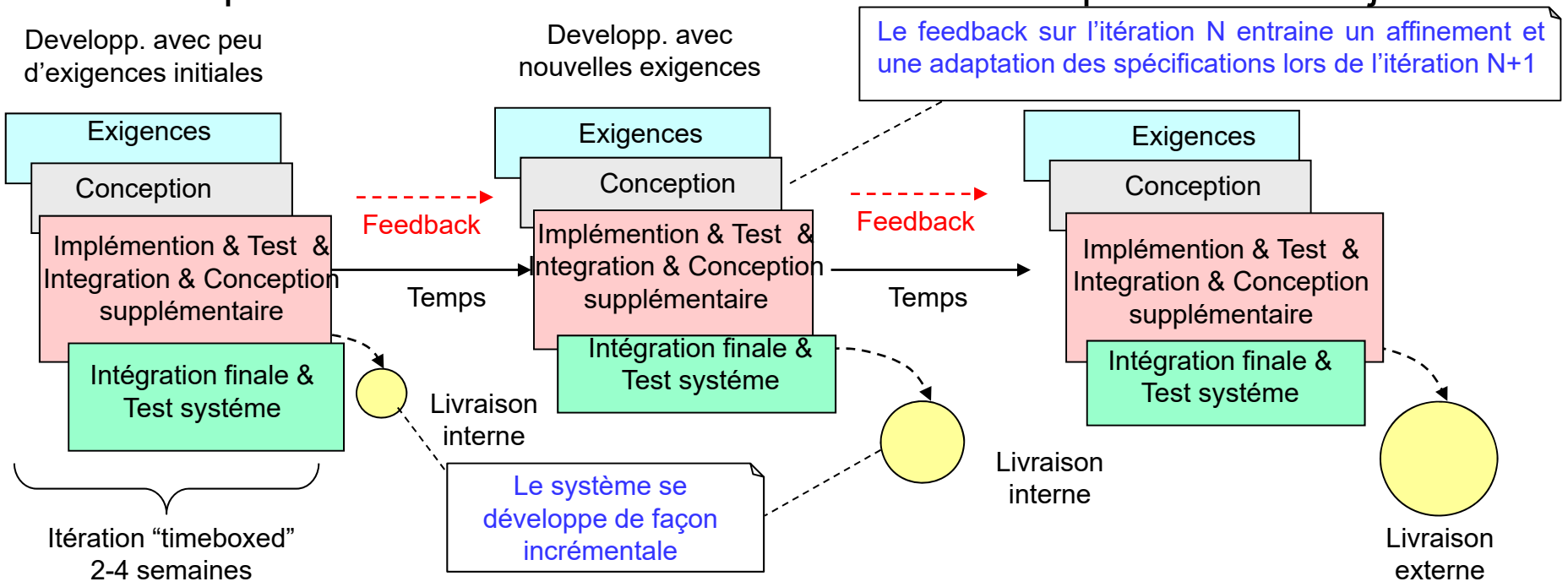
Développement Incrémental (II)

- Le développement incrémental est une stratégie de planification et mise en scène dans laquelle les diverses parties du système sont développées aux temps ou aux taux différents, et intégrées quand elles sont achevées. L'alternative au développement cumulatif est de développer le système entier avec une intégration de « big bang ».
- Une différence typique avec le développement itératif est que pendant que l'output d'une itération est examinée pour la modification, et surtout pour réviser les objectifs des itérations successives, l'output d'un incrément est soumis au nouveau raffinement, et son test ou la réaction d'utilisateur sont utilisées comme un input pour réviser les plans ou les spécifications des incréments successifs.

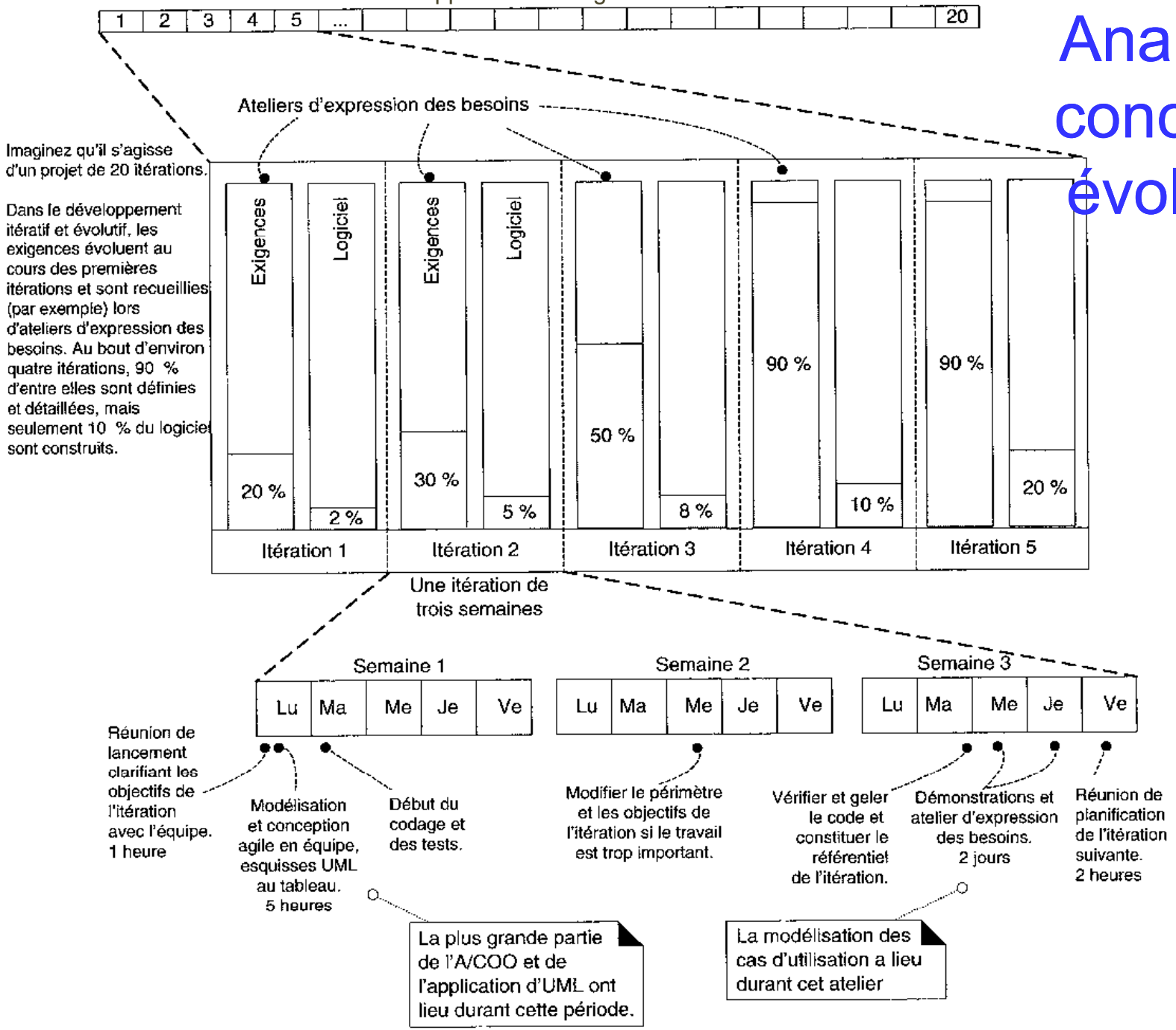


Cycle de vie itératif et incrémental

- L'idée de base derrière la accroissement itérative est de développer à un système logiciel en manière incrémental, permettant au développeur de profiter de ce qu'était appris pendant le développement de versions précédentes, incrémentales et livrables du système.
- Les étapes clés dans le processus devraient commencer avec une implémentation simple d'un sous-ensemble des exigences de logiciel et améliorent itérativement la séquence en évolution de versions jusqu'à ce que le système entier soit mis en oeuvre. À chaque itération, modifications de conception sont faites et des nouvelles fonctions peuvent être ajoutées.

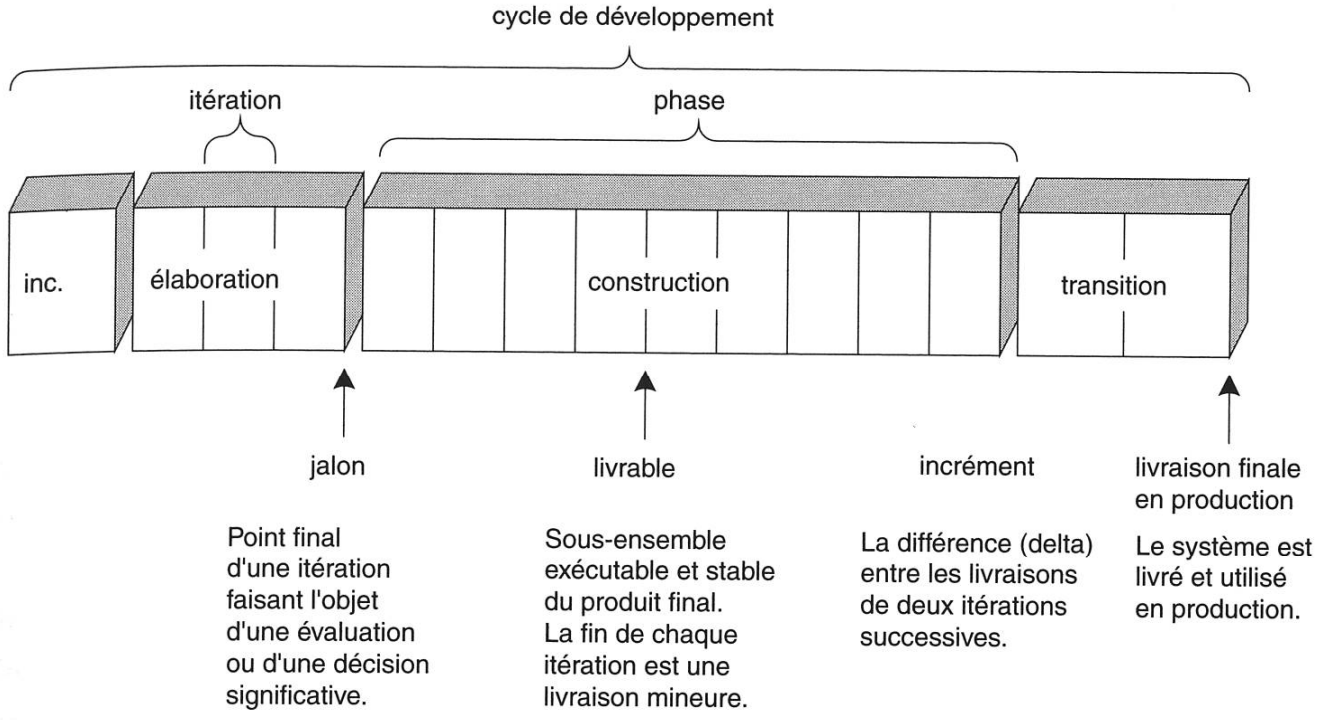


Analyse et conception évolutives



Le Processus Unifié (I)

- Le Processus Unifié (UP) n'est pas simplement un processus, mais plutôt un cadre extensible qui devrait être personnalisé pour les organisations ou les projets spécifiques.
- Il décrit le processus générique, y compris ces éléments qui sont communs à la plupart des raffinements.
- L'Élaboration, la Construction et des phases de Transition sont divisés dans une série d'itérations de durée fixée. (Chaque itération aboutit à un incrément, qui est une livraison d'un système qui contient la fonctionnalité supplémentaire ou améliorée comparée avec la sortie précédente.



Le Processus Unifié (II)

- UP est un processus de développement itératif et incrémental, dans que le logiciel est développé et est relâché dans les morceaux.
- Pendant l'**inception**, le argumentation pour le projet est établi, une vision approximative de la finalité du projet, une étude d'opportunité, une définition du périmètre et des estimations globales.
- L'**élaboration** construit une vue plus élaborée avec l'implémentation itérative de l'architecture du noyau, la résolution des risques élevés. La plupart des besoins sont recueillies et l'identification du périmètre réel et des estimations plus réalistes est réalisée.
- La phase de **construction** consiste en une implémentation itérative avec beaucoup de **itérations**, dans lequel chaque **itération** construit un logiciel de qualité, essayé et intégré, qui présentent des risques et une complexité moindres, et satisfait un sous-ensemble des conditions du projet. La livraison peut être externe ou purement interne. Chaque itération contient toutes les phases d'un normal cycle de vie : analyse, conception, implémentation et test. Le déploiement est préparé.
- La phase de **transition** include bêta tests, réglage de performance d'exécution et formation d'utilisateur.

Caractéristiques de l'UP

- **Piloté par cas d'utilisation (par le client).** Dans l'UP les cas d'utilisation sont utilisé pour capturer les besoins fonctionnelles et définir les contenus des répétitions. Chaque répétition prend une série de cas d'usage ou les scénarios des conditions complètement par l'implémentation, le test et le déploiement.
- **Piloté par le client.** Cela signifie que les objectifs des premières itérations sont choisis afin de construire les fonctionnalités visibles qui comptent le plus pour le client.
- **Piloté par les risques et par le client.** L'UP éxige que les objectifs des premières itérations sont choisis afin de identifier et atténuer les risques les plus importants.
- **L'architecture d'abord**. L'UP conseille que les premières itérations soient dédiées à construire, tester et stabiliser l'architecture noyau, car l'absence d'une architecture robuste constitue couramment un risque élevé.

Inception (initialisation, étude préalable).

L'inception est la plus petite phase dans le projet, et idéalement le ce devrait être tout à fait court. Elle consiste en une vision approximative de la finalité du projet, une étude d'opportunité, une définition du périmètre et des estimations globales.

Il faut répondre à certaines questions :

- Quelle est la vision du projet ? Quelle est son opportunité ?
- Le projet est il réalisable ?
- Faut-il construire et/ou acheter ?
- Quelle est l'estimation grossière du cout : doit-on envisager des dizaines de milliers, des centaines de milliers ou des millions d'euros ?
- Faut il poursuivre ou renoncer ?

- Pour définir une vision d'ensemble et obtenir un ordre de grandeur (peu fiable), il faut étudier quelque peu les besoins. Mais l'étape d'inception n'a pas pour but de définir tous les besoins, de parvenir à une estimation crédible ni d'élaborer un plan de projet.
- Objectifs typiques pour la phase de Inception.
  - Etablir une raison ou les affaires reconnaissent pour le projet
  - Etablir l'étendue de projet et les conditions de frontière
  - Esquisse les cas d'utilisation et les besoins clés qui conduiront les décisions de conception
  - Esquisse une ou plus architectures candidates.
  - Identifie les risques principaux.
  - Prépare un plan de projet préliminaire et l'estimation de coût

Elaboration Phase

- Elaboration débouche sur une vue plus élaborée, avec l'implémentation itérative de l'architecture du noyau, la résolution des risques élèves, l'identification de la plupart des besoins et du périmètre réel et des estimations plus réalistes.
- Pendant cette phase l'équipe de projet devrait capturer la majorité des besoins de système.
- Cependant, les objectifs primaires d'Elaboration sont d'adresser les facteurs de risque connu et établir et valider l'architecture de système.
- Les processus communs entrepris dans cette phase incluent la création de diagrammes de cas d'usage, les diagrammes conceptuels (les diagrammes de classe avec la notation seulement fondamentale) et les diagrammes de packet (les diagrammes architecturaux).
- The phase doit livrer:
  - Implémentation d'un brouillon d'Architecture (Architecture Draft): une implémentation partielle du système qui inclut le noyau, le plus du point de vue de l'architecture significatif, les composants.
  - Un plan (y compris les estimations de coût et plan) pour la phase de Construction. Le plan devrait être précis et croyable, puisque le ce devrait être fondé sur l'expérience de phase d'Elaboration et puisque les facteurs de risque significatif auraient dû être adressés pendant la phase d'Elaboration.

Construction Phase

- La construction est la plus grande phase dans le projet. Il s'agit de l'implémentation itérative des éléments qui présentent des risques et une complexité moindres, et de la préparation du déploiement.
- Dans cette phase les caractéristiques de système sont exécutées dans une collection de court, les itérations timeboxed. Chaque itération a pour résultat une livraison du logiciel exécutable .

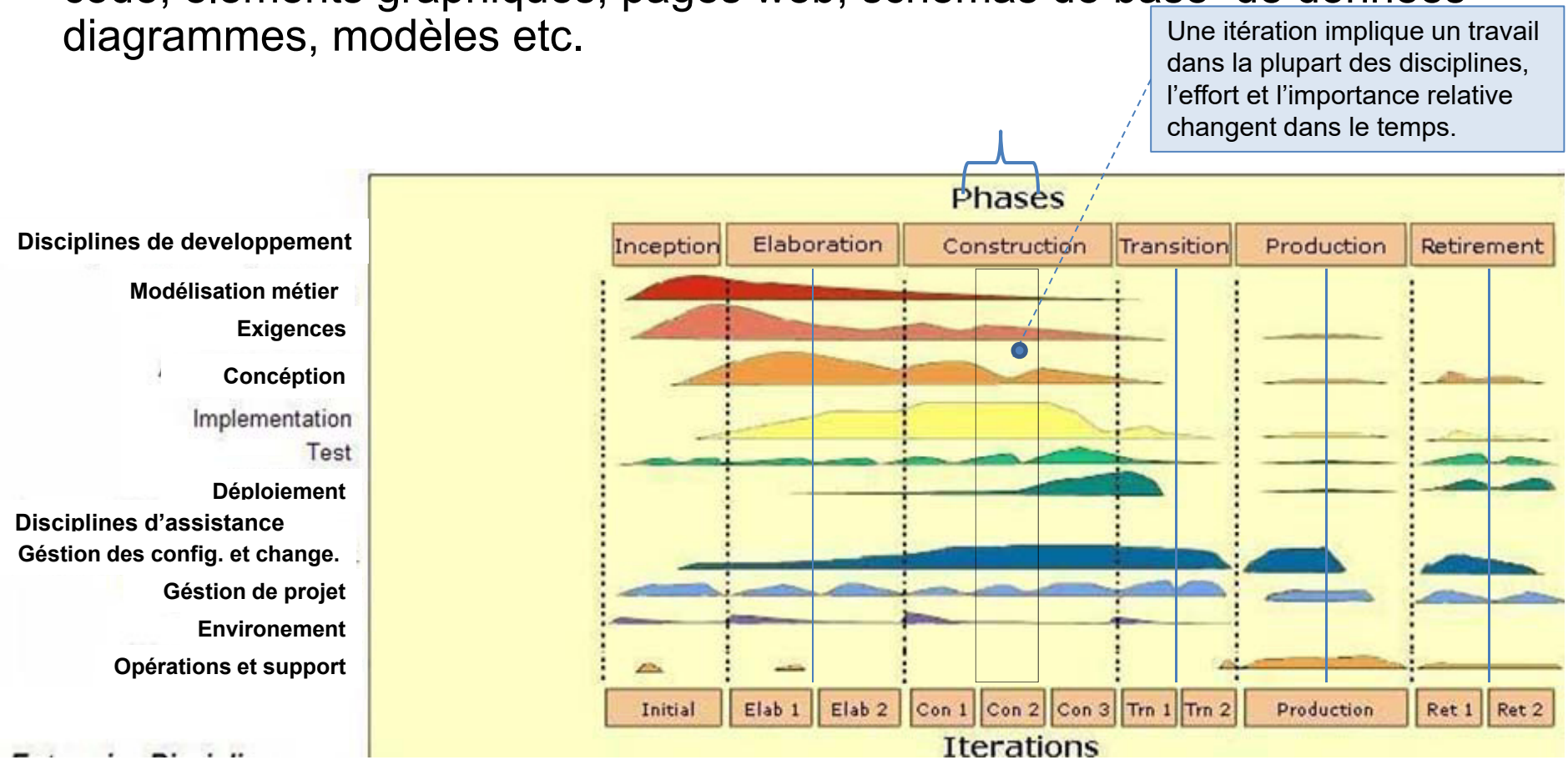
- Pendant la phase de construction c'est coutumier écrire les cas d'utilisation complets et chaque l'un devient le début d'une nouvelle itération. Diagrammes UML sont utilisés pendant cette phase : diagrammes de activité, séquence, collaboration, etc.

Transition Phase

- C’est le moment des bêta tests et du déploiement.
- Dans cette phase finale du projet le système est déployé aux utilisateurs finals. Les feed-backs reçus après le livraisons initiales peuvent avoir pour résultat de plus amples raffinements que doivent être incorporés au cours de plusieurs itérations dans la phase de Transition. La phase de Transition inclut aussi les conversions du l’ancien système et l'entraînement des utilisateurs.

Les disciplines et phases

- L'UP décrit les activités qui ont lieu dans chaque discipline. Une itération fait appel à un certain nombre de taches dans la plupart des disciplines. Mais la répartition des taches en fonction des disciplines change avec le temps.
- Une *discipline* est un ensemble d'activités et d'actefacts dans un domain d'activités donné.
- Artefact est le terme générique qui désigne n'importe quel produit du travail: code, éléments graphiques, pages web, schémas de base de données diagrammes, modèles etc.



Modélisation métier

- Modélisation métier explique comment décrire une vision de l'organisation dans laquelle le système sera déployé et comment alors utiliser cette vision comme une base pour esquisser le processus, les rôles et les responsabilités.
- Le but de cette discipline est de modeler le contexte d'affaires, l'étendue, de votre système. Les activités communes de modelage incluent le développement de :
  - Un modèle de contexte (souvent un organigramme de flux) montrant comment votre système entre dans son environnement complet
  - Un modèle d'exigences d'affaires de haut niveau (souvent un modèle de cas d'utilisation essentiels)
  - Un glossaire définissant termes critiques du business
  - Un modèle de domaine (souvent un diagramme de classes ou de données) pour les classes métier majeures ou les objets métier
  - Un modèle de processus métier (souvent un diagramme de flux de données ou d'activité) montrant une vue d'ensemble de haut niveau du processus d’affaires à être supporté par votre système. Ce diagramme est un niveau de détail plus grand que le diagramme de contexte
- L’artefact correspondant est le Modèle du Domain pour visualiser les concepts du domaine d’application les plus important.

Exigences

- Le but de la discipline d'exigences est de décrire ce que le système devrait faire et de permettre aux développeurs et au client de convenir de cette description. Pour réaliser cela, les analystes mettent à jour, organisent et documentent la fonctionnalité exigée.
- The purpose of this discipline is to engineer the requirements for your project, including the identification, modeling, and documentation of those requirements.
- La discipline réalise les exigences pour le projet, y compris l'identification, la modélisation et la documentation de ces exigences.
- Les artefacts correspondants sont le Modèle de Cas d’Utilisation et les Spécifications Supplémentaires, pour capturer les besoins fonctionnels et non fonctionnels. Le principal artefact de cette discipline est la Spécification d'Exigences Logicielle (SRS), aussi mentionnés comme le Modèle d'Exigences, qui englobe les besoins capturées.

Conception

- Le but de cette discipline est de:
  - évoluer une architecture robuste pour le système fondé sur les besoins,
  - transformer les besoins dans une conception, et
  - garantir que les problèmes d'environnement d'implémentation sont reflétés dans le conception.
- L'objectif d'analyse et de conception est de montrer comment le système sera rendu compte dans la phase d'implémentation.
- L'objectif de la conception est de construire un système qui :
  - Exécute - dans un environnement spécifique d'implémentation - les tâches et les fonctions spécifiées dans les descriptions des cas d'utilisation;
  - Satisfait toutes ses exigences;
  - Est facile à changer quand les conditions fonctionnelles changent.
- Pour concevoir les objets logiciels la conception a pour résultat un Modèle de Conception et facultativement un modèle d'analyse.
- Le Modèle de Conception sert comme une abstraction du source code; c'est-à-dire, le modèle de conception sert comme un « idée » de comment le source code sera structuré et écrit. Il consiste en classes de conception structurées dans des packets et des sous-systèmes avec interfaces bien définies, représentant ce que deviendra les composants dans le code de l'implémentation. Il contient aussi des descriptions de comment les objets de ces classes de conception collaborent pour exécuter.

Implementation

- Les buts d'implémentation sont :
  - Définir l'organisation du code, avec des composants organisés dans une architecture.
  - Codifier des classes et des objets en composants (fichiers source, binaire, exécutables, et d'autres).
  - Essayer les composants développés comme unités.
  - Intégrer les résultats produits par individuel programmateurs (ou des équipes), dans un système exécutable.
- Les systèmes sont réalisés par l'implémentation des composants. Le processus décrit comment vous réutilisez des composants actuels, ou exécuter de nouveaux composants avec responsabilité bien définie, faisant le système plus facile à maintenir, et augmenter les possibilités pour réutilisation

Test

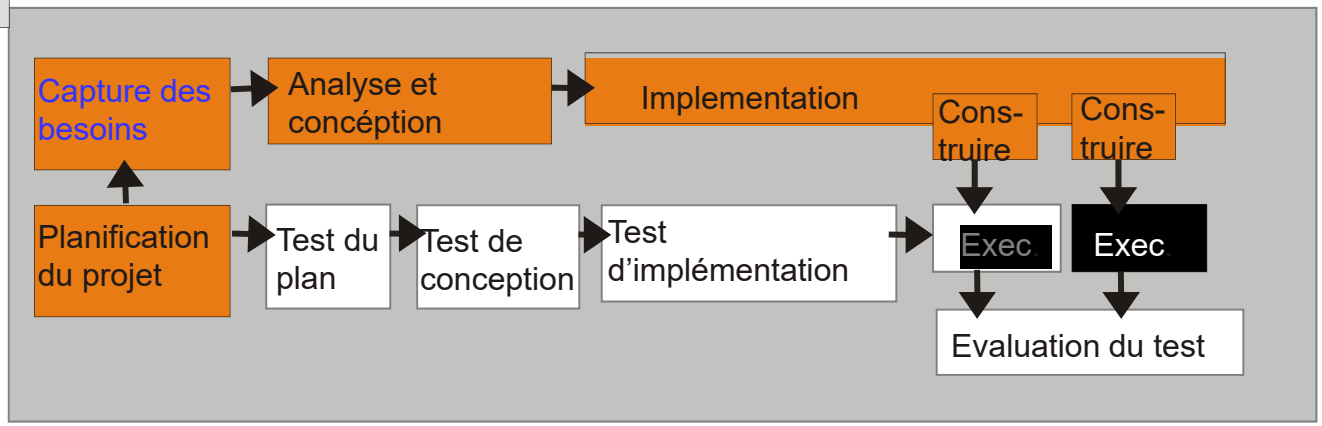
- Les buts de la discipline de Test sont :
  - Vérifier l'interaction entre les objets.
  - Vérifier l'intégration correcte de tous composants du logiciel.
  - Vérifier que toutes besoins ont été correctement exécutées.
  - Identifier et garantir que les défauts sont adressés avant le déploiement du logiciel.
  - Garantir que tous les défauts sont réparés, tester à nouveau et fermé.
- Le Processus Unifié propose une approche itérative qui signifie que vous testez toujours pendant le projet. Ceci vous permet de trouver des défauts tôt, qui réduit radicalement le coût de réparation du défaut.
- Les tests sont exécuté le long de quatre dimensions de la qualité : la fiabilité, fonctionnalité, l'exécution d'application, et l'exécution du système. Pour chacune de ces dimensions de la qualité, le processus décrit comment vous traversez le cycle de vie de test de la planification, de la conception, de l'implémentation, de l'exécution et de l'évaluation.

Le développement itératif bénéficie le test

Itération X

Test:
- commence plus tôt
- est continu

Itération X + 1



Itération X + 2

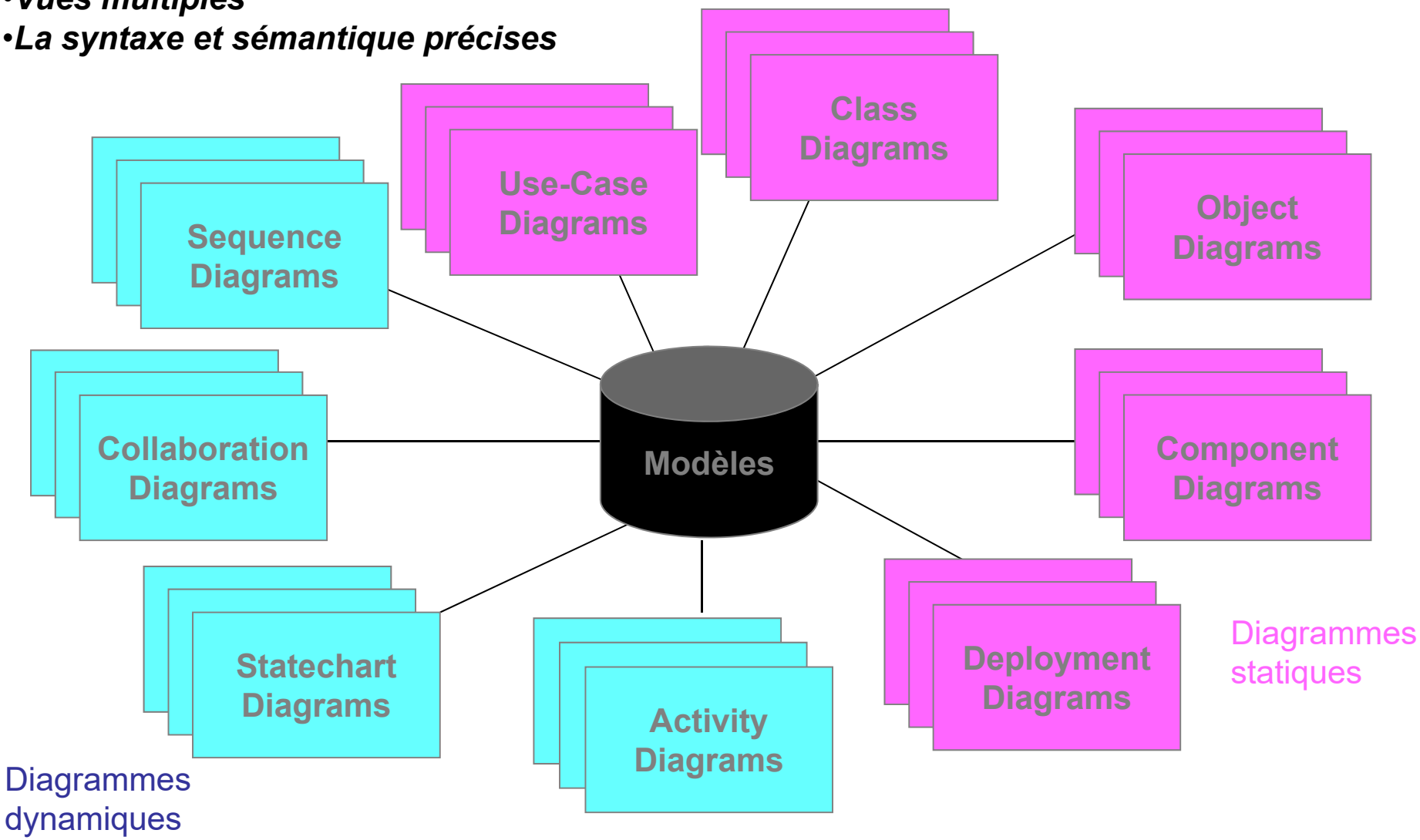
Résultat :
Plus haute qualité
Abaisse le risque

Discipline de déploiement

- Le but de déploiement est de produire avec succès les livraisons du produit et livrer le logiciel à ses utilisateurs. Il comprend une grande variété d'activités y compris :
- Produire les livraisons externes du logiciel
- Emballer l'application
- Distribution du logiciel
- Installation du logiciel
- Fournir l'aide et l'assistance aux utilisateurs
- Bien que les activités de déploiement sont surtout concentré sur la phase de transition, beaucoup des activités ont besoin d'être inclus dans les phases précédentes pour préparer au déploiement à la fin de la phase de construction.

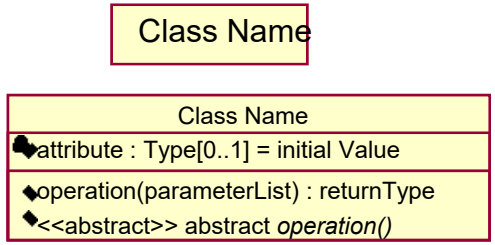
Modélisation visuelle avec UML

- ***Vues multiples***
- ***La syntaxe et sémantique précises***

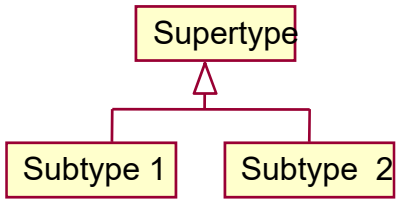


Symboles UML (I)

Classes



Generalization



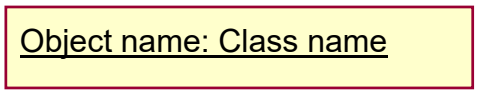
Constraint {nome: descrizione}

Stereotype <<stereotipo>>

Comment



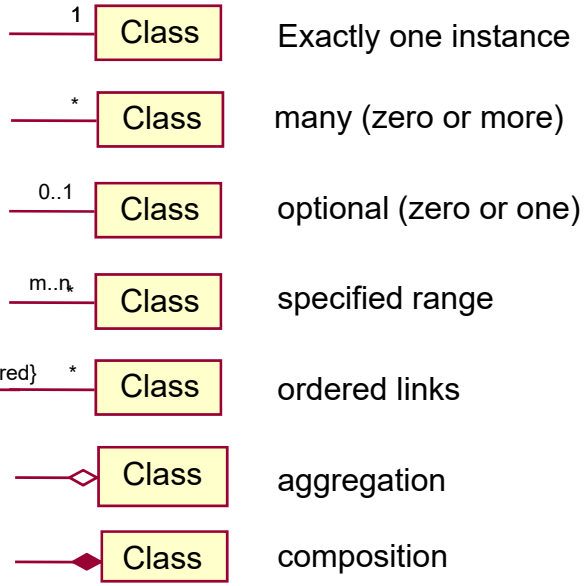
Object/Instance Specification



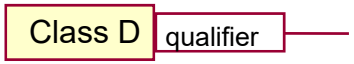
Association



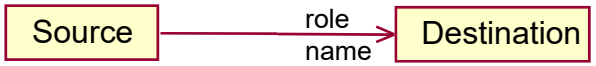
Multiplicity



Qualified Association



Navigability

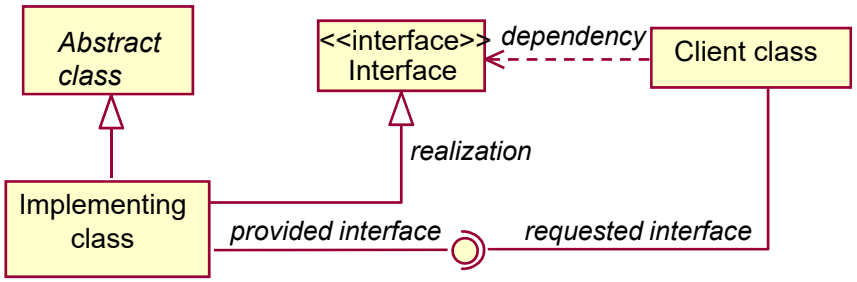


Dependency

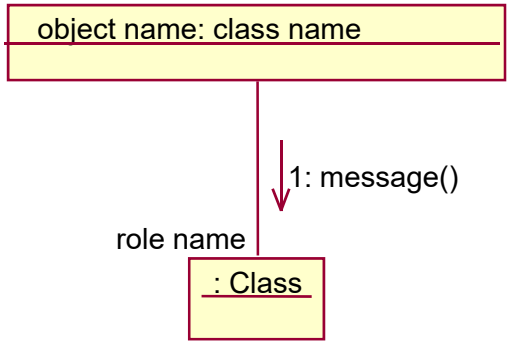


Symboles UML (II)

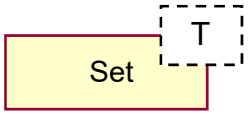
Class Diagram



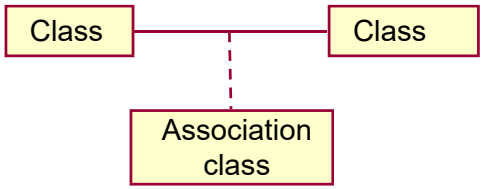
Communication Diagram



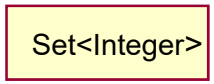
Template Class



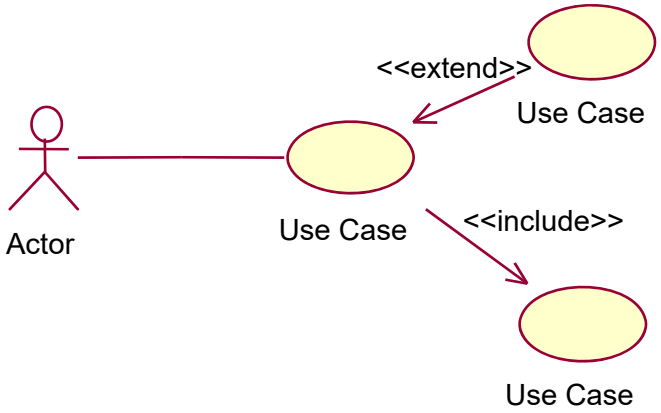
Association Class



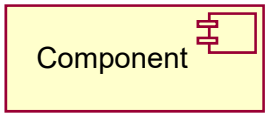
Bound Class

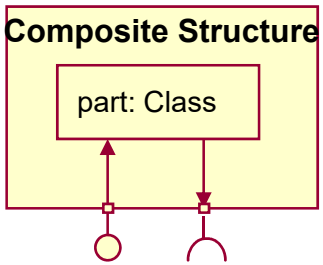


Use Case Diagram



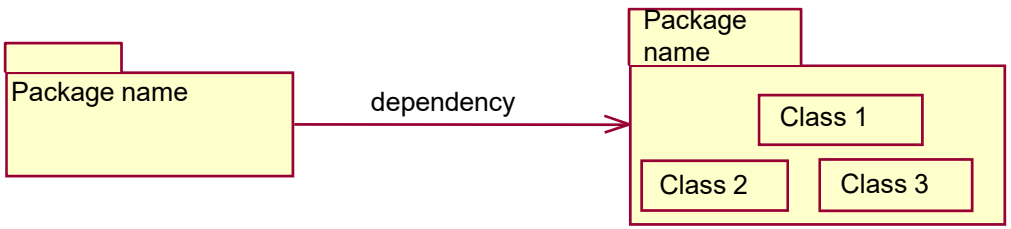
Component



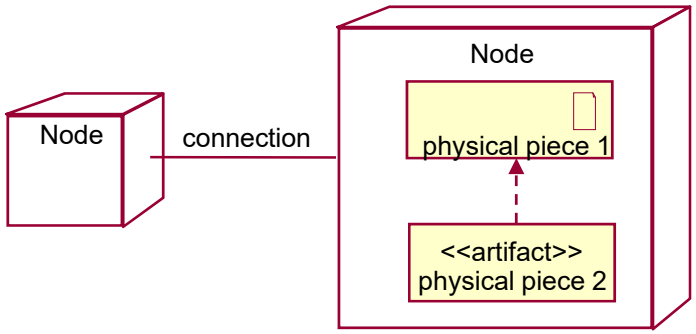


Symboles UML (III)

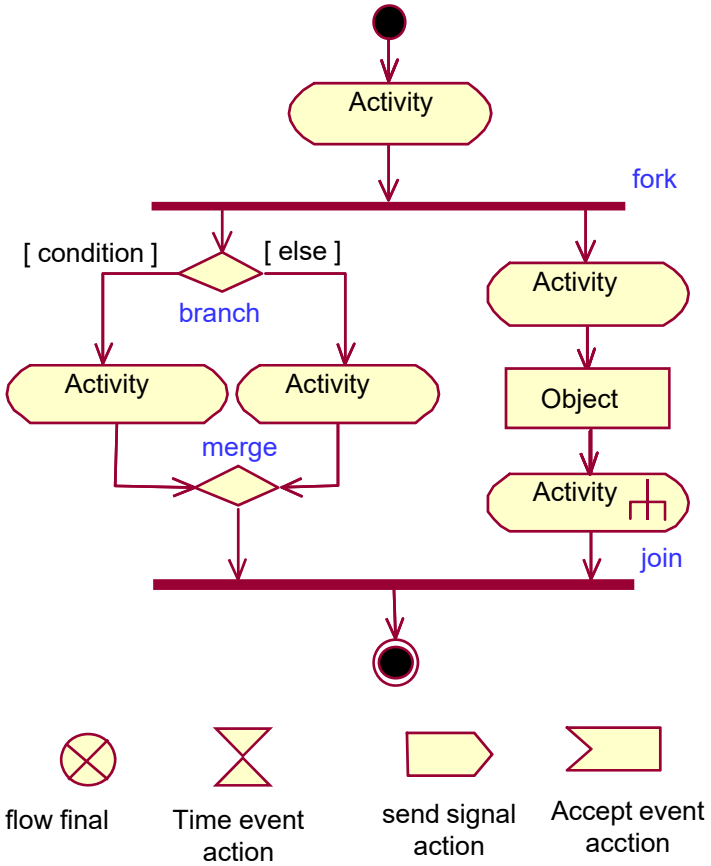
Package Diagram



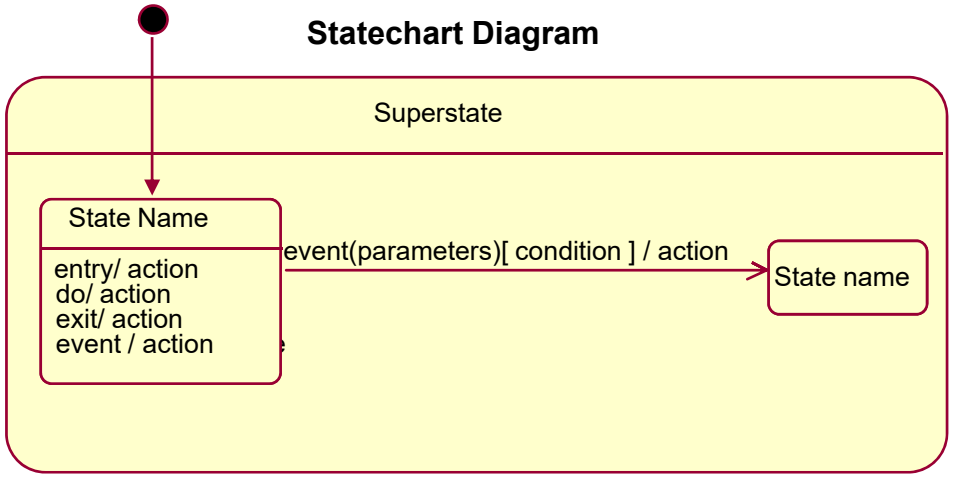
Deployment Diagram



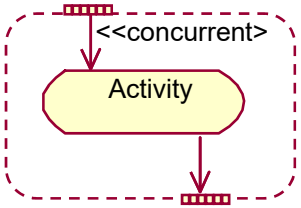
Activity Diagram



Statechart Diagram

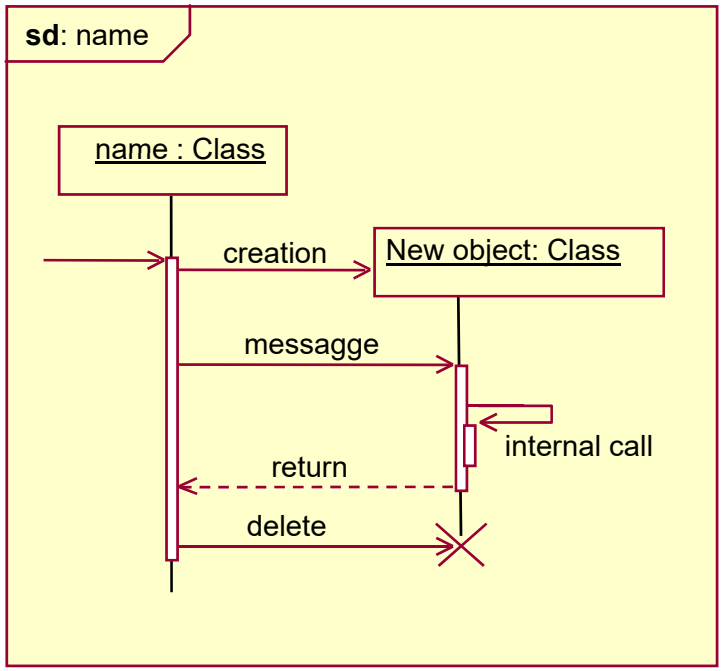


Expansion Region

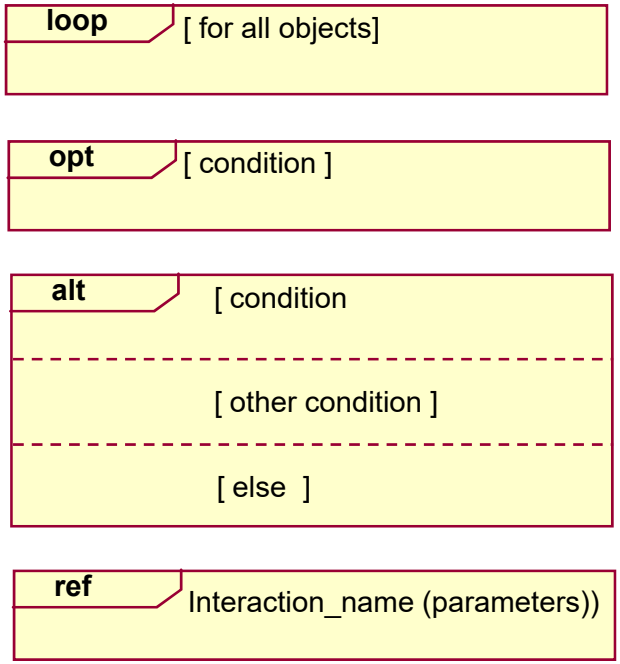


Symboles UML (IV)

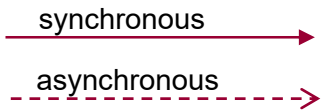
Sequence Diagram



Overview Diagram

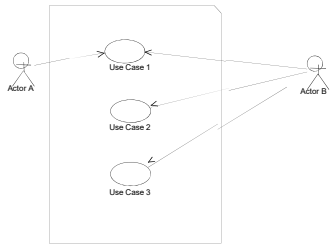


Messages

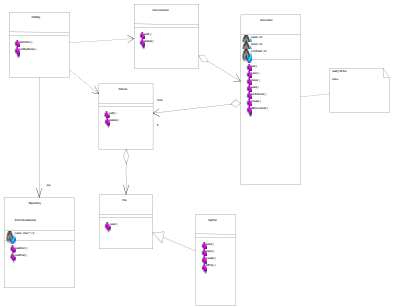


Diagrammes UML

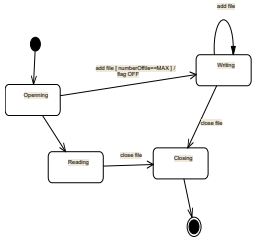
Use-case diagram



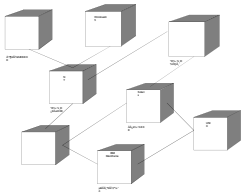
Class diagram



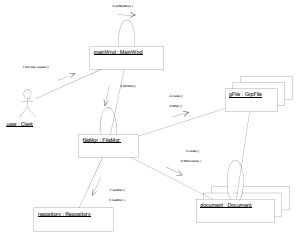
Statechart diagram



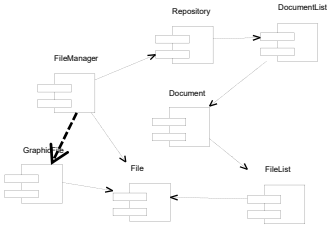
Deployment diagram



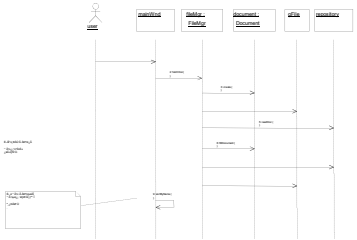
Collaboration diagram



Component diagram

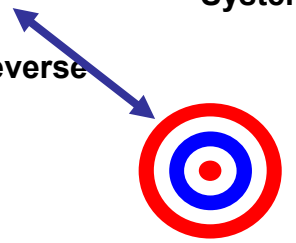


Sequence diagram



Target System

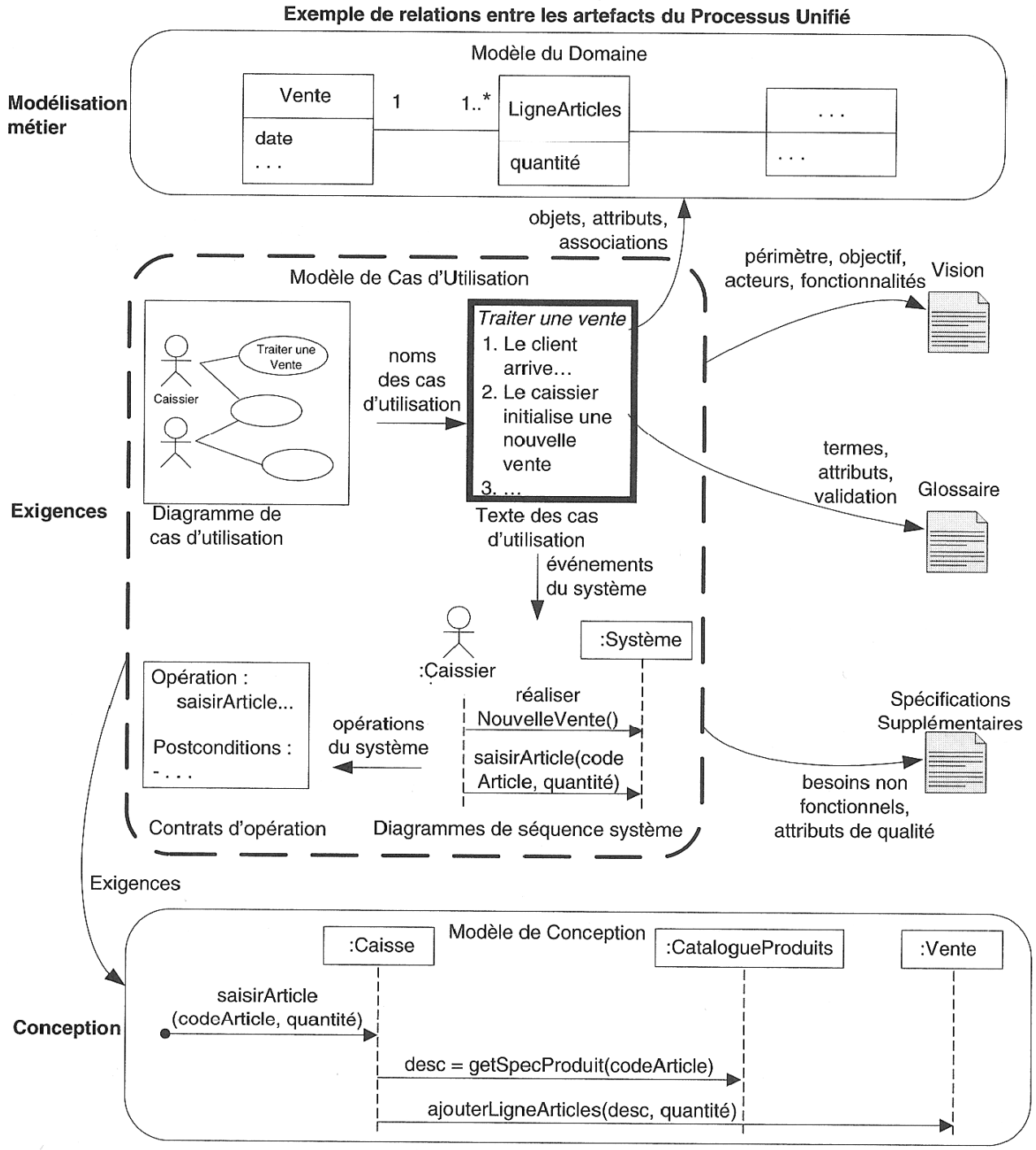
Forward and Reverse Engineering



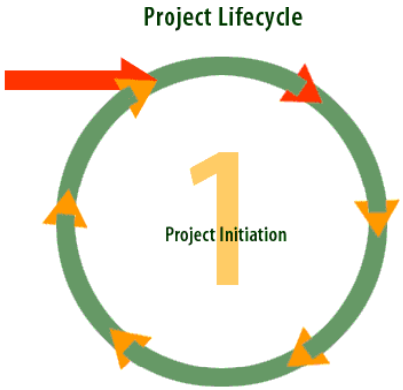
Disciplines vs. Diagrammes

Discipline	Pratique	Artefact / Itération →	Inc. I1	Élab. E1..En	Const. C1..Cn	Trans. T1..T2
Modélisation métier	Modélisation agile Ateliers d'expression des besoins	Modèle du Domaine		d		
Exigences	Ateliers d'expression des besoins Méthodes de travail en groupe	Modèle de Cas d'Utilisation	d	a		
		Vision	d	a		
		Spécifications Supplémentaires	d	a		
		Glossaire	d	a		
Conception	Modélisation agile Développement piloté par les tests	Modèle de Conception		d	a	
		Document d'Architecture du Logiciel		d		
		Modèle de Données		d	a	
Implémentation	Développement piloté par les tests Programmation en binômes Intégration continue Standards de codage	…				
Gestion de projet	Conduite de projet agile Réunion Scrum quotidienne	…				
…						

d = début, a = affinement.

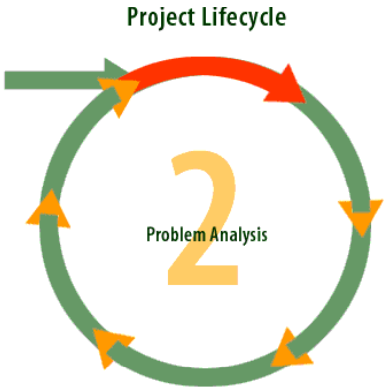


Les relations entre les artéfacts dans le modèle UP



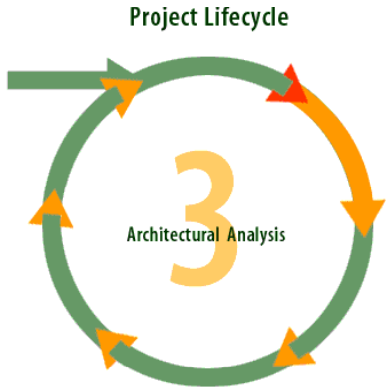
Project Initiation:
Document the users' expectations

L'étendue : établir une définition de problème qui dirigera le reste du projet.



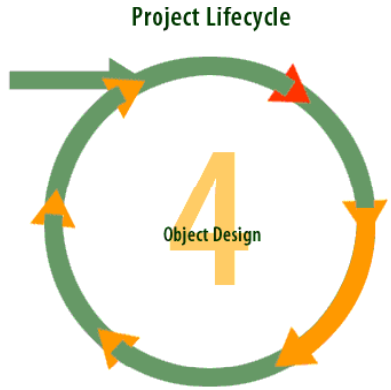
Problem Analysis:
Define the resources of the problem domain

L'étendue : découvrir les ressources exigés par le domaine de problème. Pour chaque ressource, définir son but et l'interface qu'il soutient
.



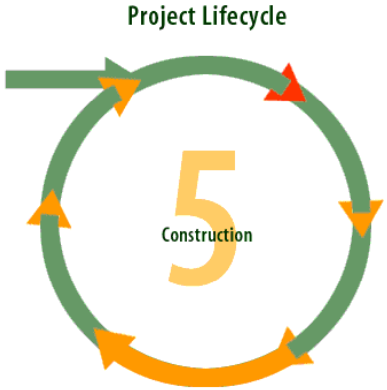
Architectural Analysis:
Select the architectural approach for the solution

L'étendue : identifier une architecture où les exigences des utilisateurs (la sécurité, le nombre d'utilisateurs, la distribution de données et du traitement, les ressources partagées, le temps de réponse) pourraient être placées.



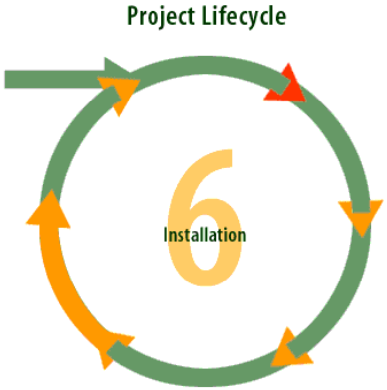
Object Design:
Select and define the implementation for the software solution

L'étendue : 1. définir l'implémentation interne de chaque ressource du domaine du problème.
2. Ajouter les classes qui rendent capable le logiciel soutenir l'usage des ressources.
3. Itérer par le modèle pour le améliorer et raffiner.



Construction:
Build, buy, integrate code to satisfy the design

L'étendue : créer la solution spécifié par la conception du logiciel.



Installation:
Put the application into production

L'étendue : 1. valider l'application finie; 2. planifier le déploiement de système complété ; 3. coordonner les portions manuelles et automatisées du système ; 4. instruire les utilisateurs sur les caractéristiques et la mécanique du système .

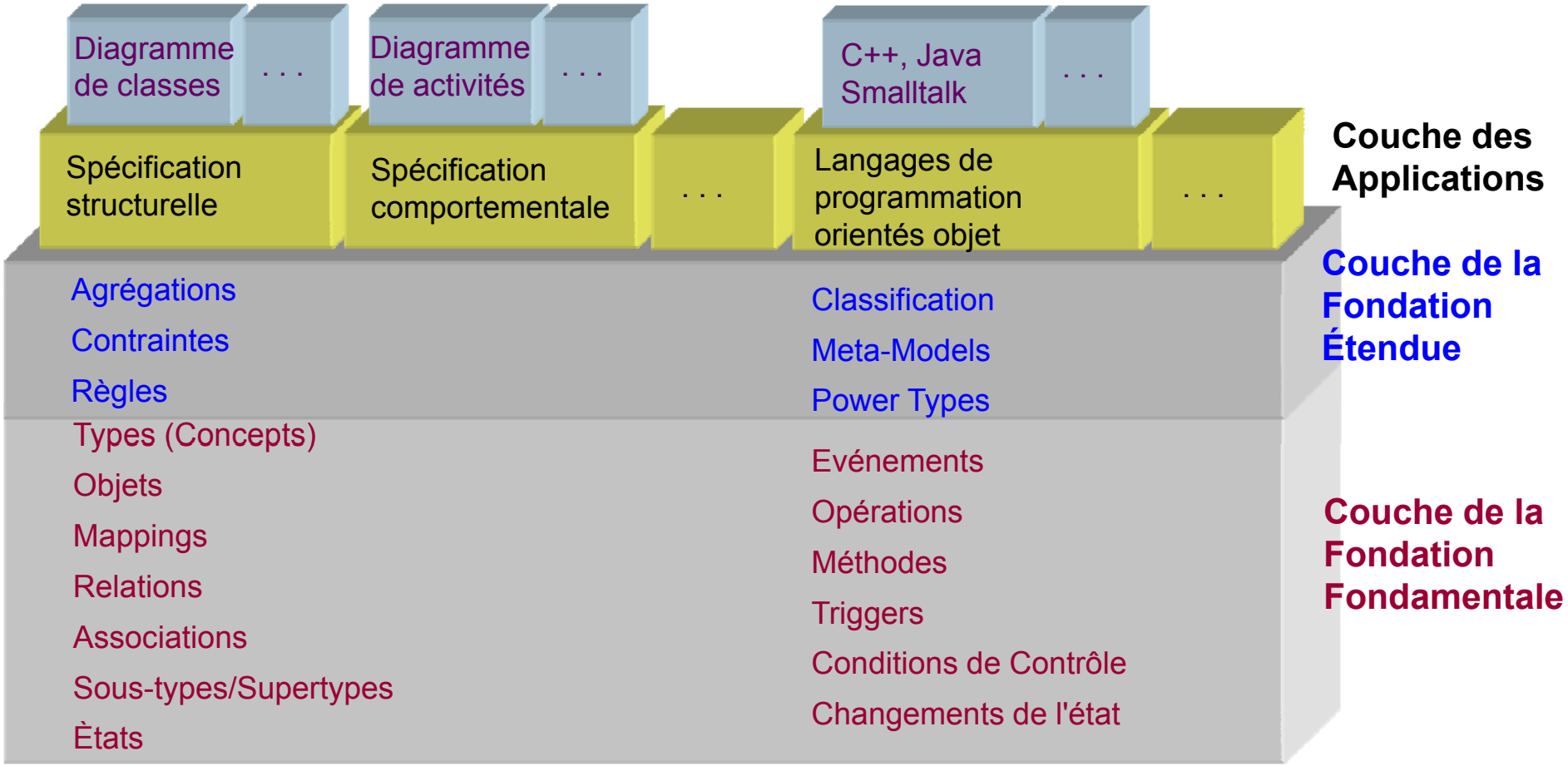
Le cycle de vie du projet

FIN DU PREMIÈRE CHAPITRE

- Découpage de l’entreprise en trois catégories encastrées : Département - Fonction – Tâches
- But : développer et faciliter la transmission des connaissances, des savoirs au sein de l'entreprise et vers ses partenaires privilégiés, tout en dématérialisant ce processus.
- Deux dynamiques majeures concernant les systèmes d'information :
- - Une généralisation du traitement en temps réel
  - Une forte interconnexion des réseaux
  Modification de la perspective spatio-temporelle de l’entreprise.
- Les SI vont aider à la prise de décision de l'organisation (avant, pendant, après)
- En fonction de la situation dans l'organisation, les caractéristiques des SI seront différentes :
  - Si la décision est stratégique,
  - Si la décision est opérationnelle.
- Aussi, les SI auront un impact sur la décision pour plusieurs raisons: un plus grand nombre de participants va intervenir dans la prise de décision, l’intelligence de l’organisation s’accroît.
- Les technologies de l’information entraînent (…) une gestion nouvelle de l’information. Dans la mesure ou le management repose en grande partie sur la gestion de l’information, il semble inéluctable que les pratiques managériales soient appelées à évoluer et donnent naissance à un management électronique, appelé e-management

Partie 2. Fondements de l'orientement objet

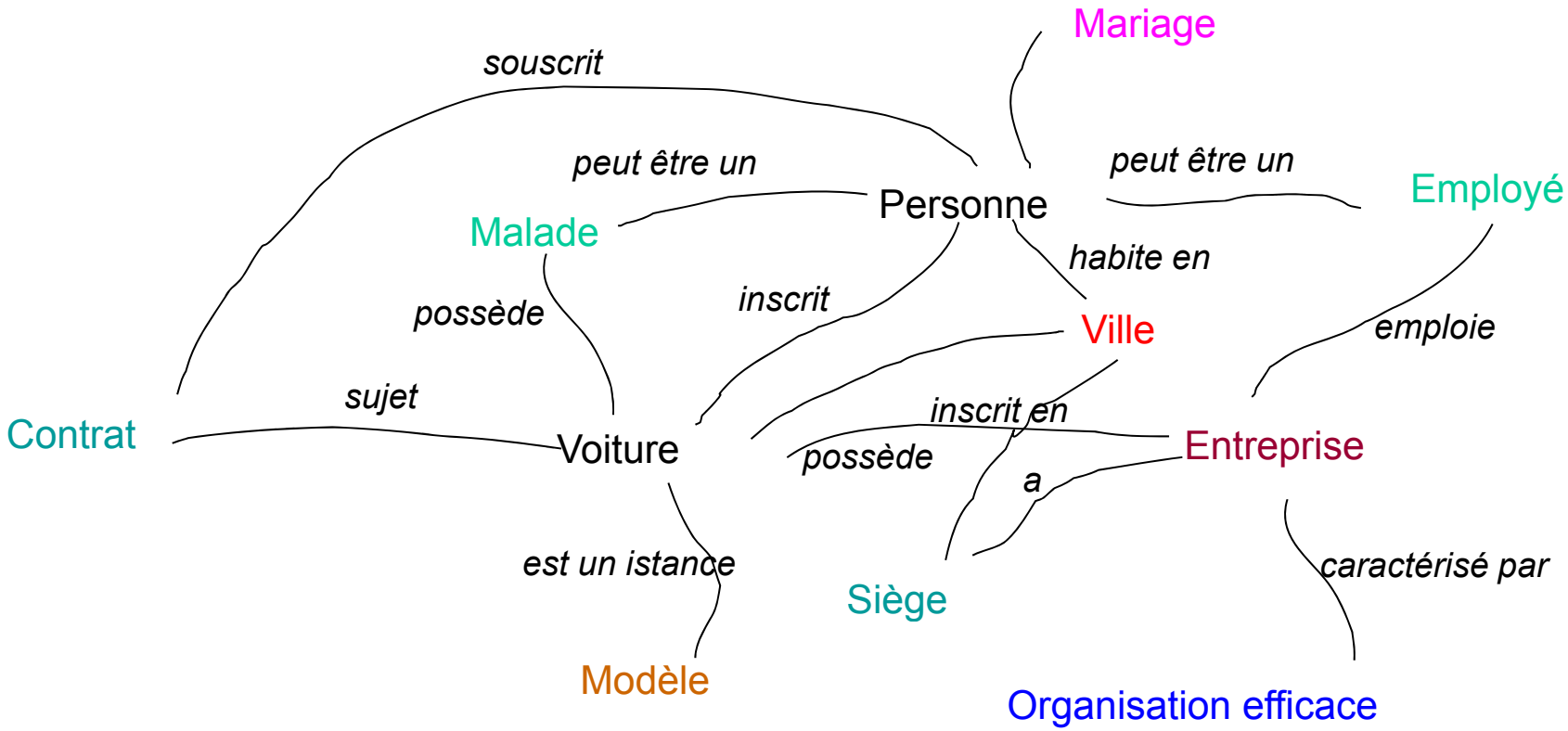
Fondements de l'orientement objet



Concepts

Les humains ont la capacité à former et utiliser des concepts. Ils peuvent étiqueter leurs concepts avec des symboles et peuvent manipuler ces symboles pour définir et communiquer leur sens.

Un concept est une idée ou une notion que nous appliquons aux choses, ou aux objets, dans notre connaissance du mode réel. Un exemple d'un concept est n'importe quoi dans la réalité à lequel ce concept s'applique.



Utilisation des concepts

- Pour les humains les concepts sont des appareils de reconnaissance en réalité que entoure nous. Ils peuvent être :
  - Tangible (personne, voiture, table) ou Intangible (temps, qualité, entreprise)
  - Rôles (médecin, malade, propriétaire),
  - Jugements (haut salaire, bon exemple, travail productif)
  - Relationnel (mariage, partenariat, possession)
  - Evénements (vente, achat, accident de marché)
  - Autres types de concepts (nombre, icône, image, signal, processus)
- Les concepts que les humains utilisent sont aussi des tests pour la réalité. Nous pouvons appliquer des concepts aux objets dans notre réalité et rejeter les concepts qui non plus long s'applique. Chaque concept est fondé sur les tests qui détermine quand il s'applique.

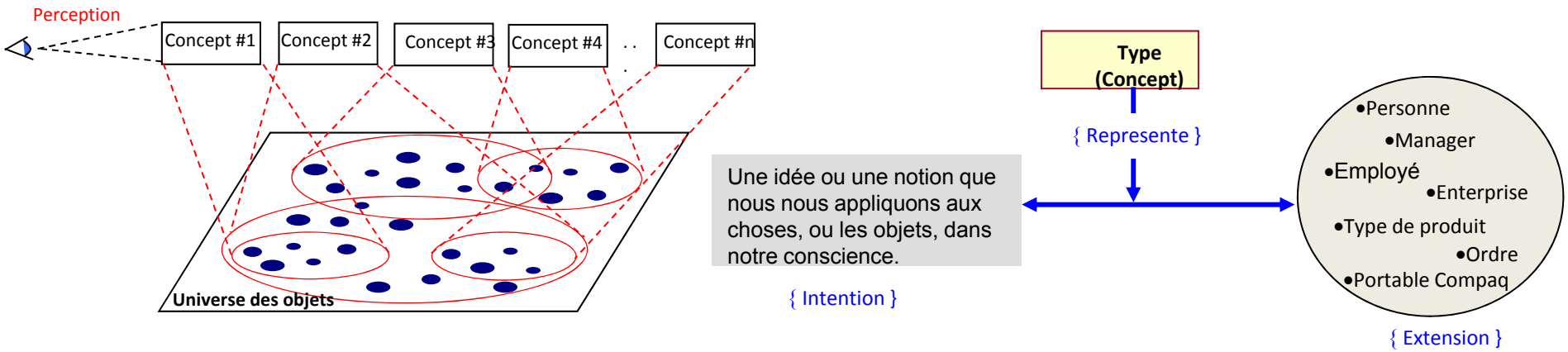
Definition du Concept (Ontology)

Nous utilisons des concepts comme des unités de connaissance pour laquelle chaque concept a un nom ou symbole, un intension et une extension. Intension et extension sont deux côtés de la même monnaie :

- Intension (définition, idée, compréhension) : la définition complète du concept et le test qui détermine si le concept s'applique à un objet.
- Extension : l'ensemble de tous instances à lesquels le concept s'applique (les exemples de concept).

Domaines et instances des concepts

Domaine = une collection d'entités dans un secteur choisi d'intérêt. Les entités dans un domaine sont des instances de quelques concepts.
Spécification du domaine = la collection de concepts qui s'appliquent à un domaine.
Note. Le domaine d'un système est le vrai monde que l'analyste analyse ; la spécification de domaine est un produit de l'analyse de système.
Les concepts peuvent être eux-mêmes des concepts: Compaq Portable est un concept qui s'applique à beaucoup d'exemples produits par Compaq. Compaq Portable est aussi un exemple du Type de Produit de concept qui est un exemple lui-même du concept appelé Concept.

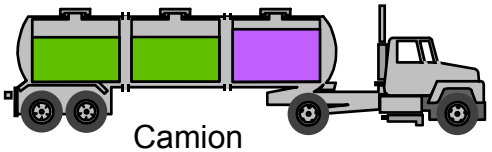


Objet = Une abstraction ou une représentation abstraite d'un élément matériel ou immatériel dans la réalité étudiée avec une identité et des frontières clairement définies dont l'état et le comportement sont encapsulés . Le terme instance est fréquemment utilisé dans le logiciel pour décrire un objet particulier.

Que est un objet?

Un objet représente une entité physique, conceptuel, ou de logiciel :

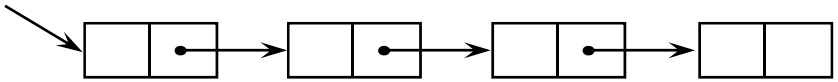
- Entité physique



- Entité conceptuelle :



- Entité de logiciel :



Important. Il y a une quantité presque infinie d'informations de un entité réele. Mais combien de ces informations sont vraiment nécessaires pour les enfermer dans un objet?

Réponse : Un objet est un modèle ou une abstraction d'une entité du monde réel avec des frontières tranchantes et le sens bien défini pour l'application qui a besoin de lui. Introduire un nouveau objet signifie de représenter une vraie entité dans une manière utile pour votre problème.

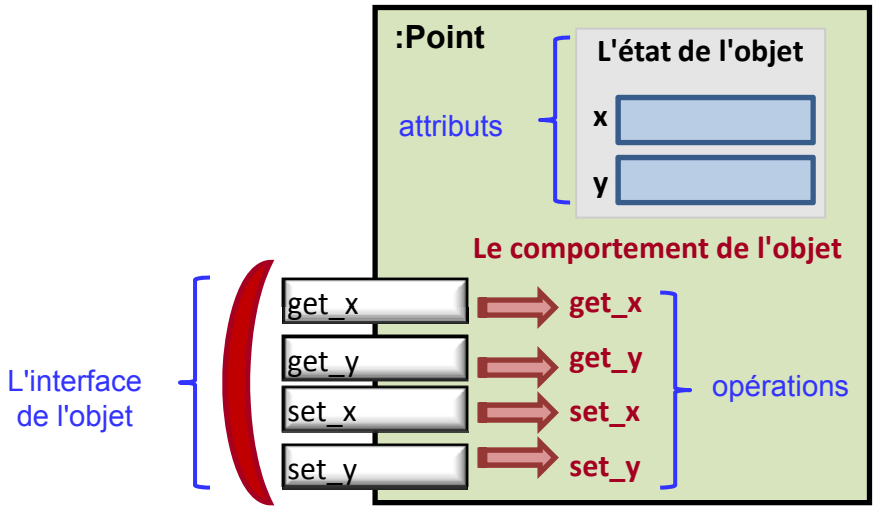
Questions sur contexte, le niveau de détail et le temps détermine aident pour décider si la représentation que vous créez est utile pour le problème que vous résolvez :

- Stakeholders (parties prenantes): “Qui utilisera l'objet ?” “Pourquoi ont-ils besoin de l'objet ? “
- Niveau d'abstraction : “De combien détails ai-t-je besoin ?”
- Cycle de vie : “Combien de temps ai-t-je besoin de traquer l'objet ? ”

Encapsulation

Encapsulation = La capacité à fournir les utilisateurs de l'objet avec une interface bien définie à une série de fonctions dans une façon qui cache leurs fonctionnements internes. C'est la technique de monter ensemble les structures de données de l'objet et les méthodes (les procédures) qui utilisent ou modifient ces données.

- Pour utiliser un objet nous avons besoin de savoir deux choses de lui: son but et son interface (l'interface est la partie visible de l'objet).



- Conception fondée sur l'encapsulation est une clé importante pour obtenir des objets réutilisables et interchangeables.
- L'encapsulation est parfois appelé à tort «cacher l'information", car l'implémentation d'un objet est cachée. C'est ignorer l'aspect le plus important de l'encapsulation: ce qui un objet expose.

Cohésion

Cohésion = Une mesure du degré à lequel toutes les parties d'un objet soutiennent un but seul.

- *Haute cohésion* = tous les éléments dans l'objet soutiennent le même but;
- *Cohésion basse* = les éléments différents soutiennent des buts différents.

1. Combien informations et comportement devraient être inclus dans un objet?

Réponse: Seulement les informations nécessaires pour soutenir l'unique but de l'objet.

2. Quand devrions-nous combiner deux petits objets dans l'un?

Réponse : Quand les objets deviennent fragmentés et ne peuvent pas remplir un but sans demander constamment l'aide de l'autre objet.

3. Quand devrions-nous fractionner un grand objet dans plus petits objets

Réponse : Quand l'objet est responsable de plus qu'un but.

Accouplement

Accouplement = une mesure du degré de dépendance entre les objets.
La dépendance signifie qu'un objet exige les données ou le caractère fonctionnel possédé par un autre objet.

- L'accouplement détaché signifie un degré bas de dépendance
- L'accouplement Tendu signifie un haut degré de dépendance.
  L'accouplement a un effet direct sur l'entretien de système.

L'accouplement tendu a pour résultat un effet de propagation, c'est-à-dire, un changement dans un objet exige des changements (, ou essayant tout au moins) de tous les objets associés.

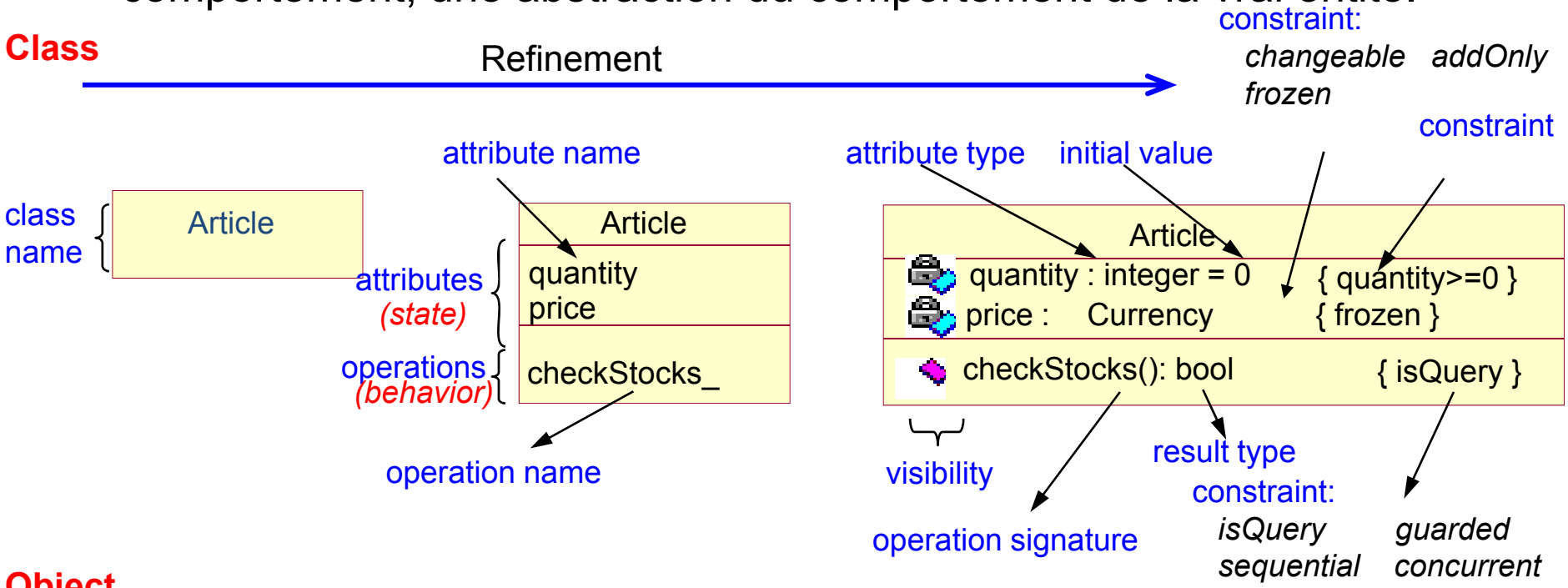
L'accouplement détaché peut être atteint en assignant à un objet seulement les comportements qui sont liées au but de l'objet (haute cohésion).

La cohésion et l'accouplement toujours devraient être ensemble évalués et devraient être équilibrés parce que souvent ils sont dans le conflit.

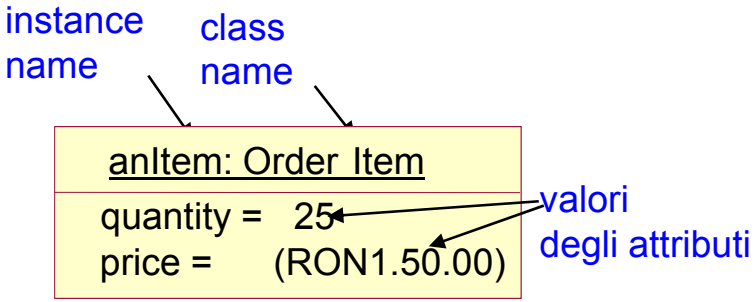
2.2 Le modéle structurel des objets

Classes et objets en UML

- Un objet est quelque chose qui a :
  - identité, il est unique comme la vraie entité qui il représent
  - état, une abstraction des propriétés de la vrai entité
  - comportement, une abstraction du comportement de la vrai entité.

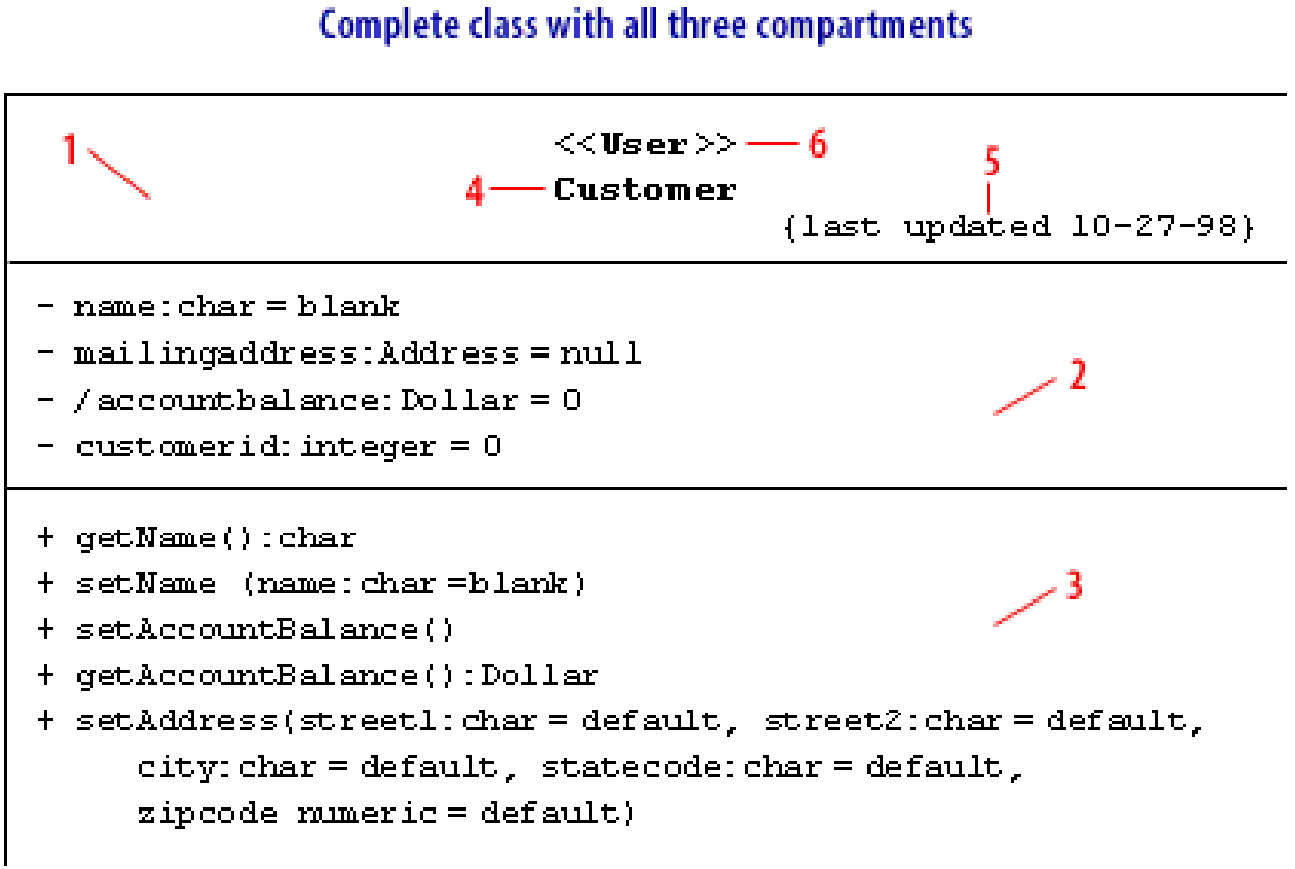


Object



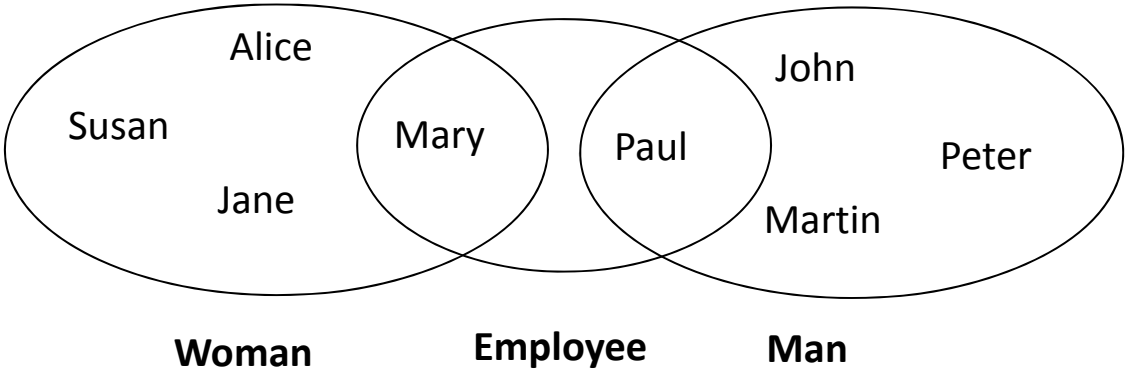
Une classe complète

1. La section du nom
2. La section des attributs
3. La section des opérations
4. Le nom de la classe
5. La contrainte
6. Stéréotype



Le cycle de la vie d'un objet

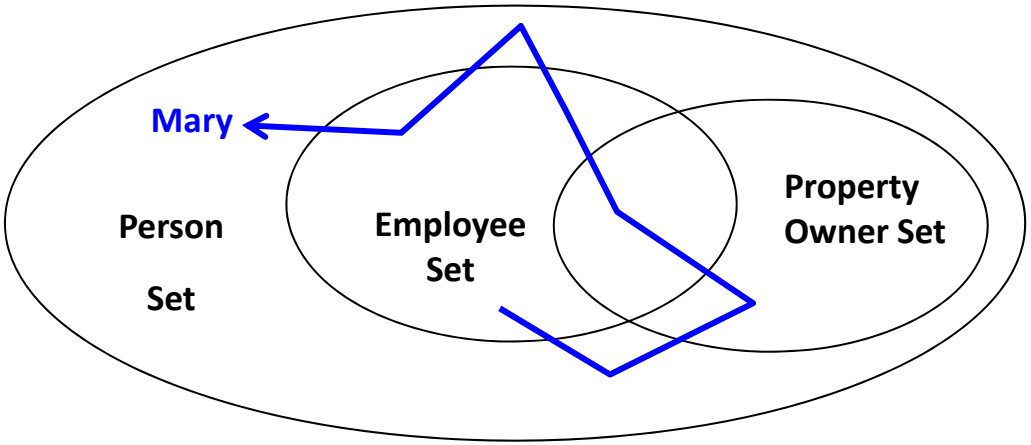
- La plupart des objets ont des périodes d'existence ou cycles de la vie. Pour ces objets les points de apparition dans le monde et de conclusion de leur vie peuvent être bien définis: pour nous les objets apparaissent quand nous appliquons un concept à eux pour la première fois, et ils disparaissent quand les concepts non plus long s'appliquent. Pendent leur vie, plusieurs concepts peuvent être appliquées. Pour décrire ces changements, nous pouvons employer des ensembles.



- Ensemble = une collection particulière d'objets .
- L' appartenance d'objet à un ensemble est déterminé par le fait que la définition complète du concept s'applique à l'objet. Un ensemble peut comprendre beaucoup d'objets et chaque objet peut être membre de beaucoup de ensembles.
- Quand nous déterminons qu'un concept s'applique à un objet spécifique, l'objet est classifié comme un membre d'un ensemble spécifique. Un objet peut être déclassifié quand il est enlevé d'un ensemble particulière.

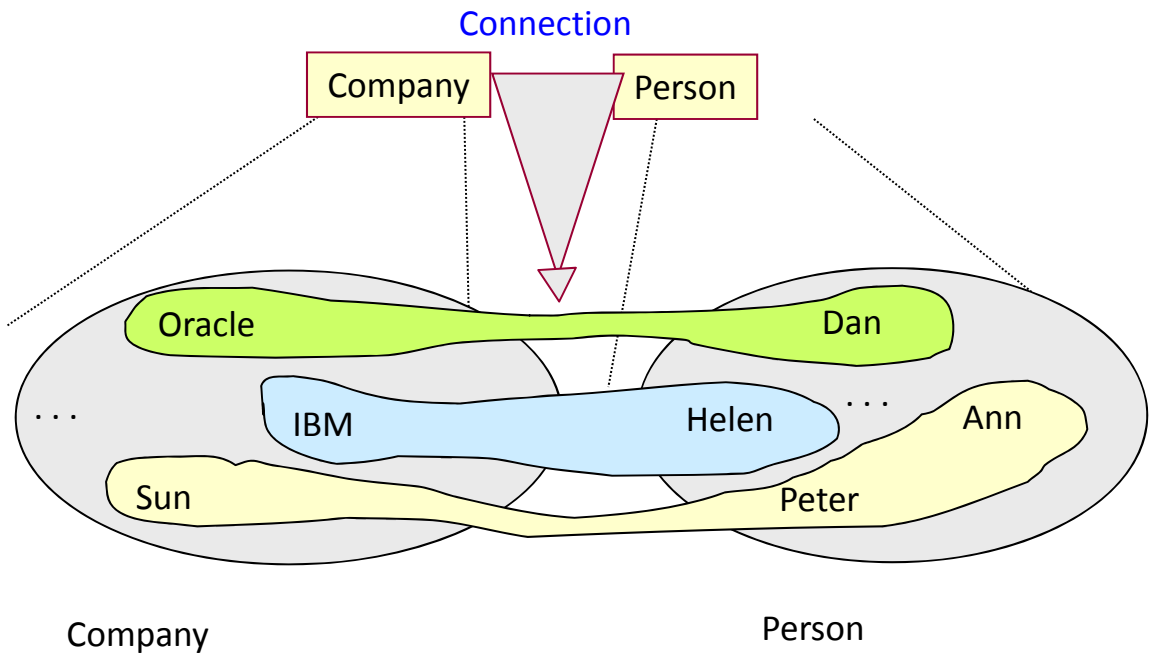
Classification multiple et classification dynamique

- Spécifier comment ces changements de classification ont lieu est une technique au coeur même de la spécification OO.
- Dans sa vie un objet peut être membre de plusieurs ensembles et, en beaucoup d'occasions, il peut changer son appartenance à un ensemble.
- En autres termes :
  - classification multiple : un objet peut avoir des concepts multiples qui lui s'applique à n'importe quel un moment.
- Attention ! La plupart de langages de programmation OO sont restrictives avec leurs objets : un objet peut être instance de seulement une classe du langage pour la vie. Pour changer sa classe, un nouvel objet doit être créé pour remplacer l'ancien objet.



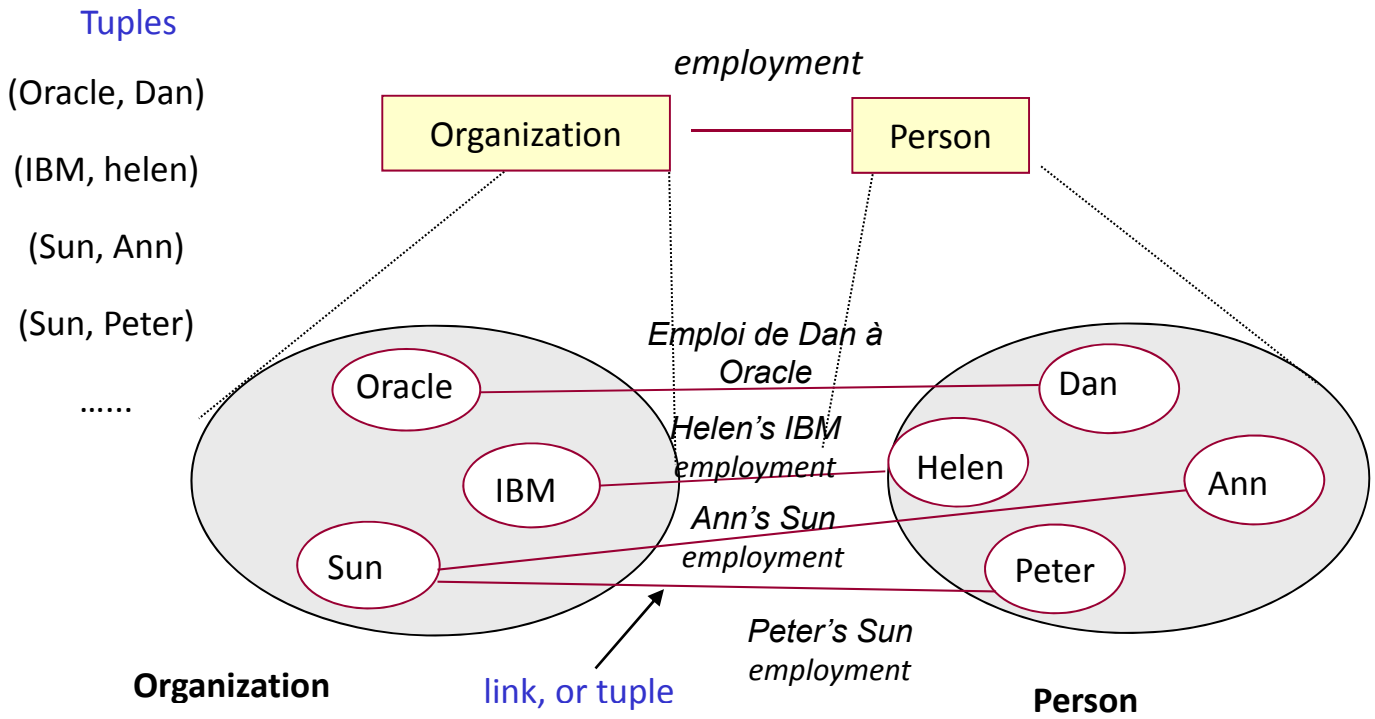
Raccorder Objets

- Utilisant de plus ces tuples nous rend capable de faire la application des objets d'un type aux objets d'un autre et de retour.
- Les connexions entre objets fournissent un moyens pour lier des objets de divers types dans une façon significative.
- Les objets peuvent être connectés dans deux façons différentes : avec relations et avec applications.
- Les relations nous rendent capable de joindre des objets dans unités appelées tuples ou liens.
- Utilisant ces tuples nous rend capable de faire correspondre des objets d'un type aux objets d'un autre type et de revenir s'il est nécessaire.



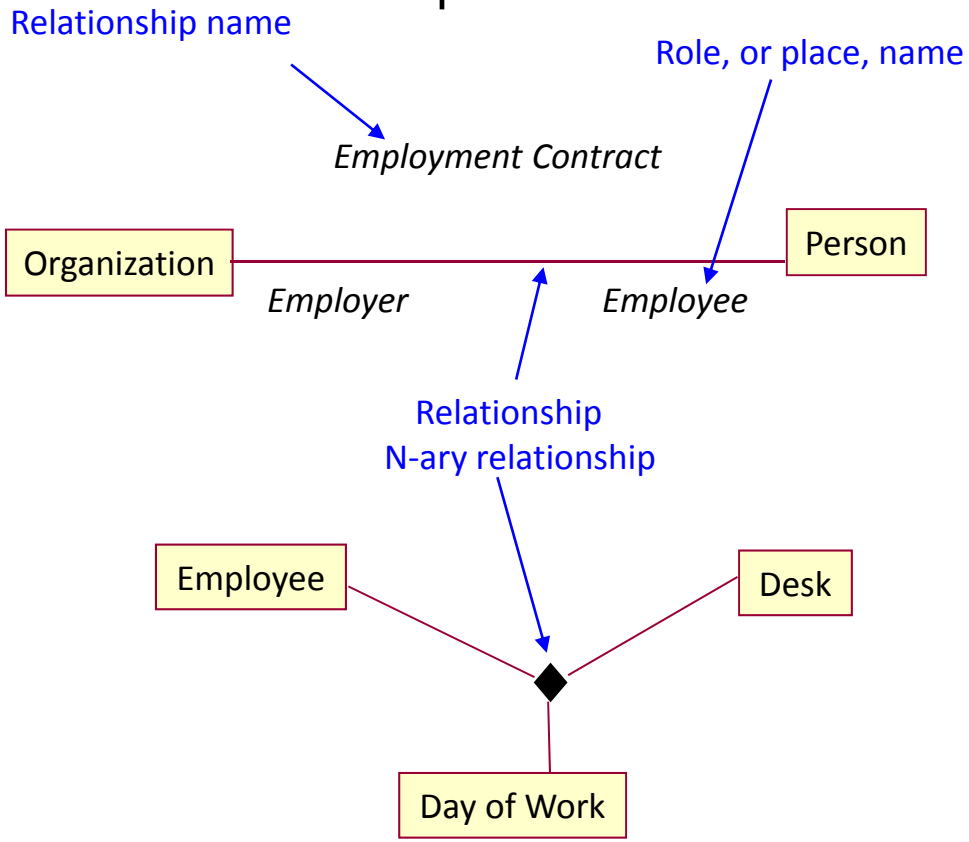
Relations (I)

- Un tuple (ou lien) est un accouplement immuable d'objets. Il peut être traité comme un objet lui même.
- Une relation est un type dont les instances sont tuples.



Relations (II)

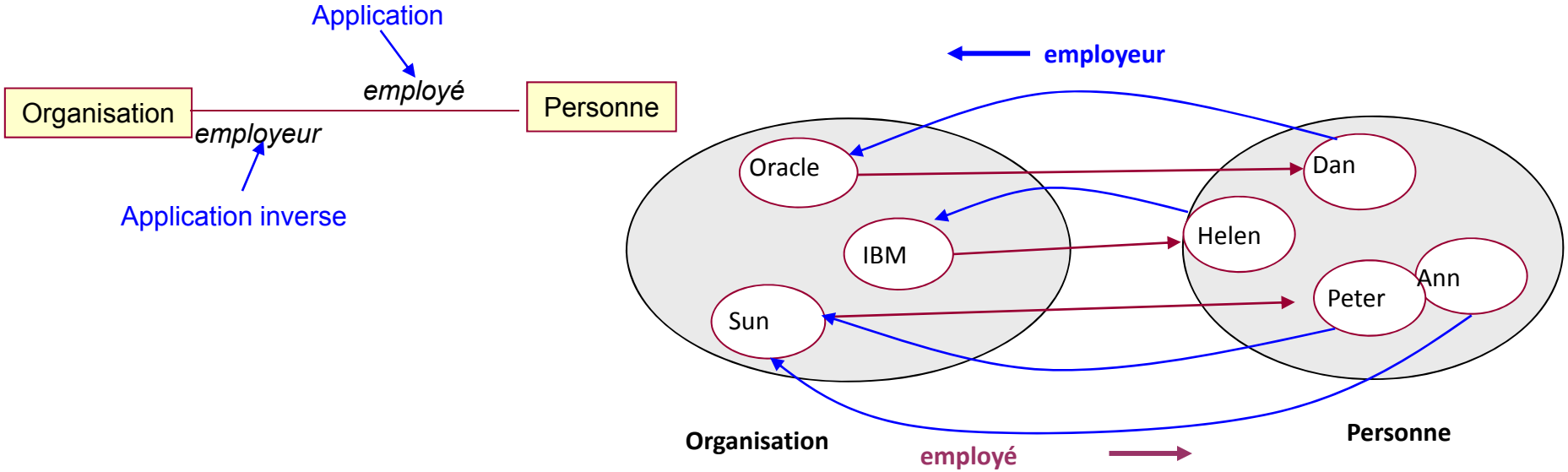
- Les relations nous permettent de traiter les connexions comme des ensembles liès. Une relation a deux places l'une pour chacun de ses composants liés. Dans UML ces places sont référés comme rôles.



- Il y a plusieurs formes communes de relations utilisées dans le modelage. Ceux-ci incluent la classification, la généralisation, l'agrégation, et l'association.

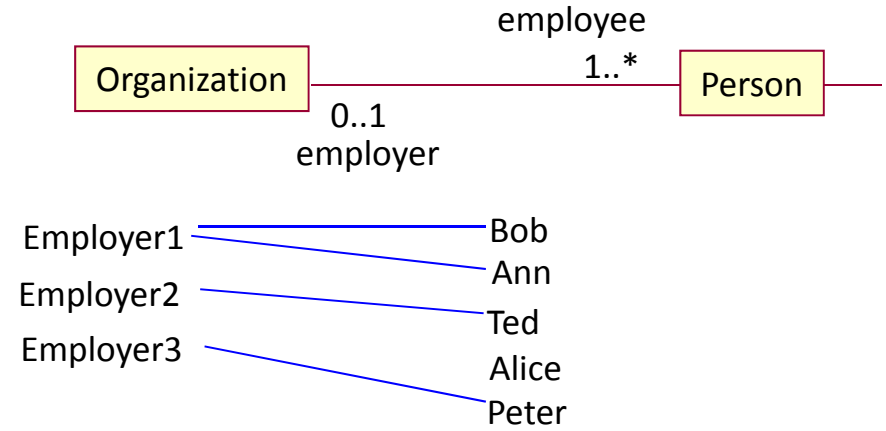
Applications

- Les applications ne regardent pas l'ensemble entier. Ils nous permettent de naviguer commençant avec un objet d'un type et traverser aux objets du type lié. Le nom de rôle peut être aussi utilisé comme une étiquette pour exprimer une façon directionnelle de penser à une connexion.
- In mathématiques une application définie par un ensemble de départ *E*, un ensemble d'arrivée *F* et une relation de *E* vers *F* dans laquelle chaque élément de *E* possède une image et une seule
- Dans OO une application assigne les objets d'un type aux objets d'un autre type.
- La capacité à appliquer telles applications dans le monde dynamique d'objets exige un processus.



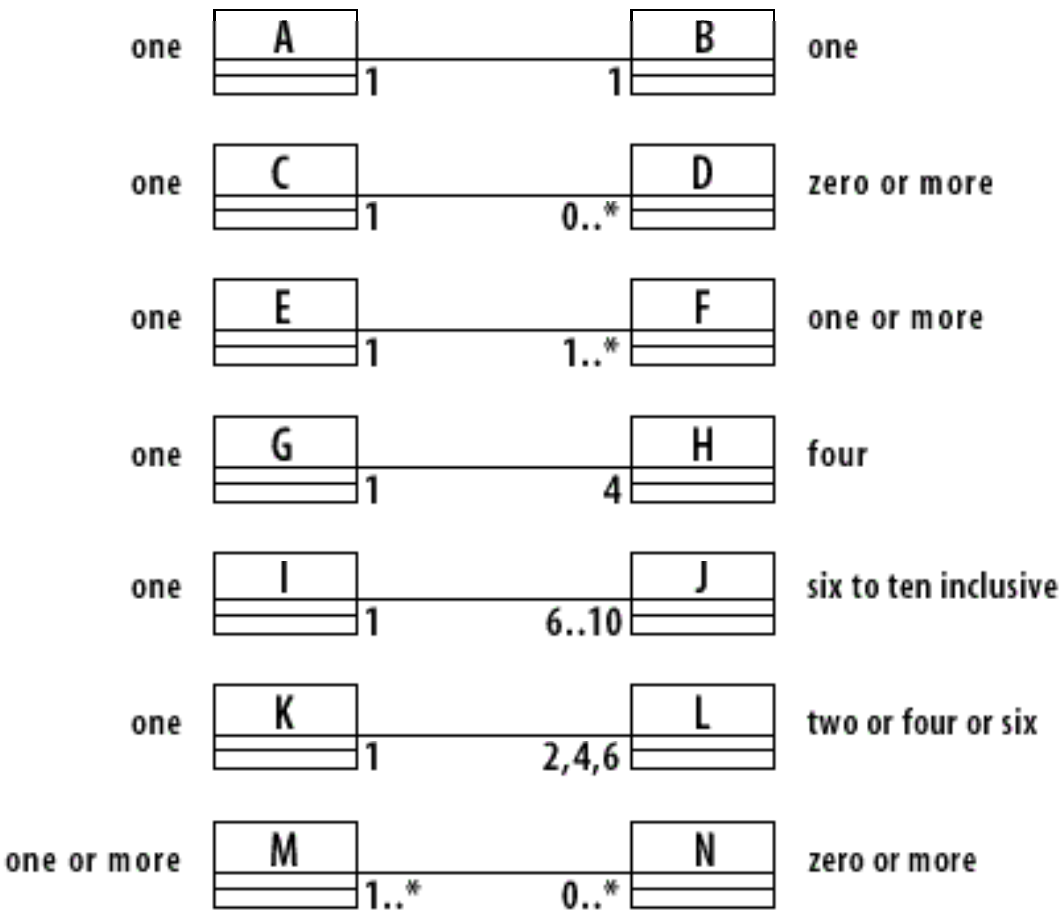
Contraintes de la Cardinalité

- Pour placer des contraintes sur les applications sont souvent utiles. A sens unique de contraignant une cartographie est en limitant le nombre d'objets à lesquels chaque carte d'objet. Une telle contrainte est appelée la contrainte de cardinality. Dans UML, il est référé à comme la multiplicité.



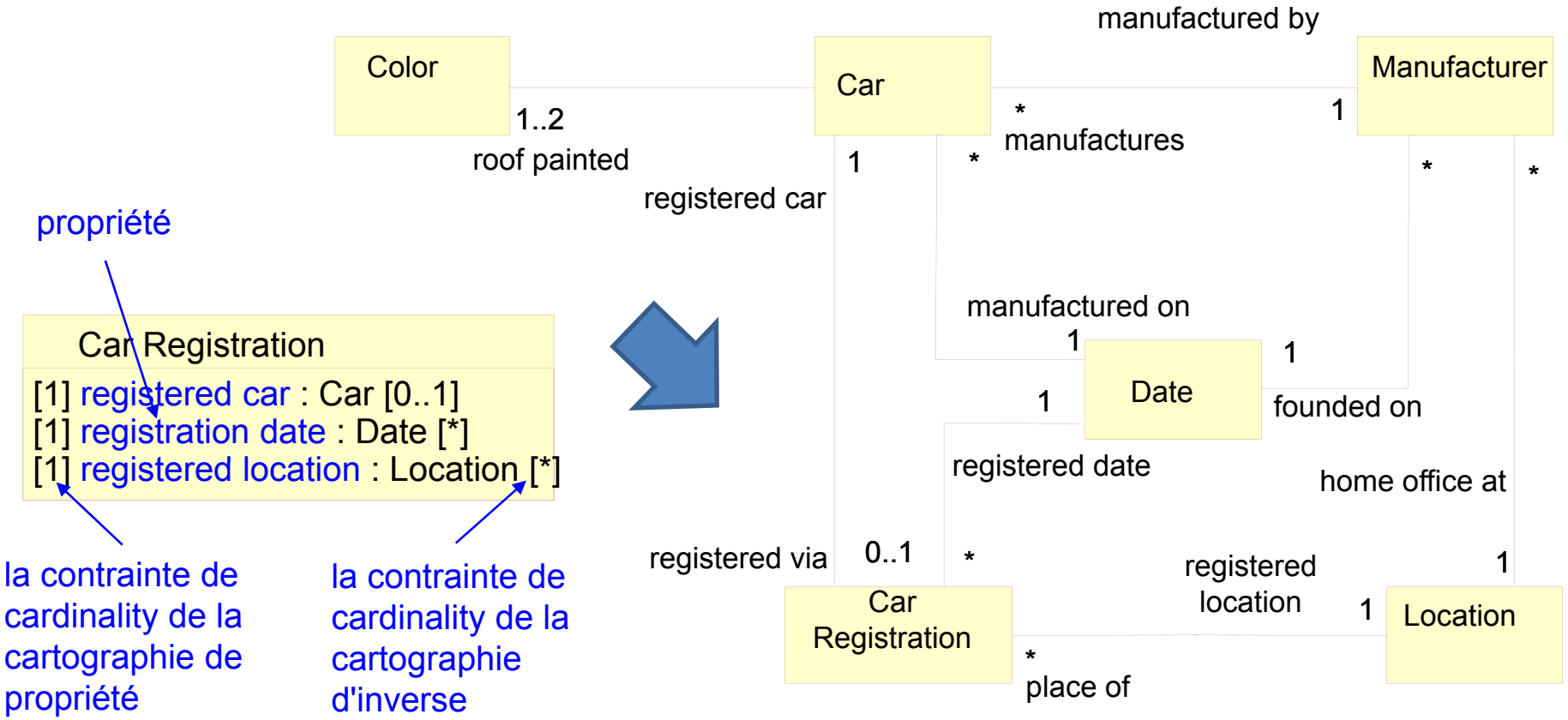
- Cardinalité indique qu'un nombre entier est utilisé pour représenter le nombre d'objets. Les contraintes de Cardinalité peuvent être : zéro, l'un, ou l'autre nombre entier positif, mais aussi un domaine de nombre entier : 0 .. 1 (à plus l'un), * (beaucoup ou 0..*), 1..* (au moins l'un) etc. Quelquefois nous utilisons « pouvons » ou « devons » en exprimant des contraintes. Donc nous avons des contraintes facultatives et obligatoires.

Exemples de cardinalité



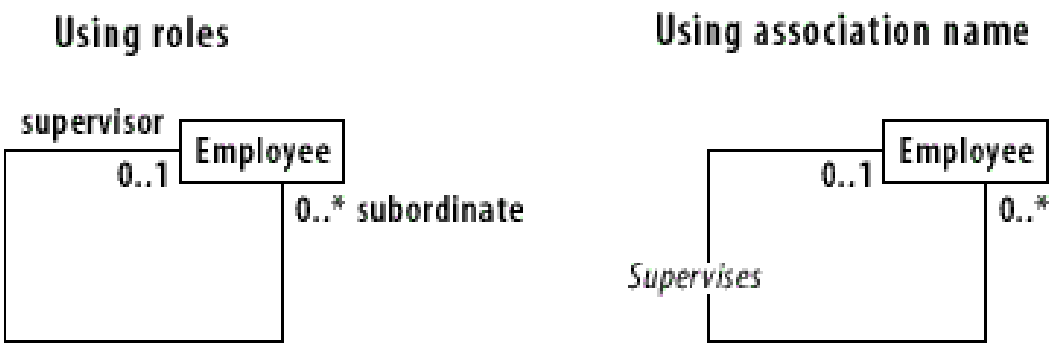
Propriétés des objets

- En référant aux diverses applications que un type particulier a, la propriété de terme est souvent utilisée plutôt. Les propriétés de terme se réfèrent à toutes cartographies qu'un type donné d'objet a avec les autres types d'objets. Les valeurs de propriété d'un objet se réfèrent à tous exemples de cartographie que l'objet a avec les autres objets.

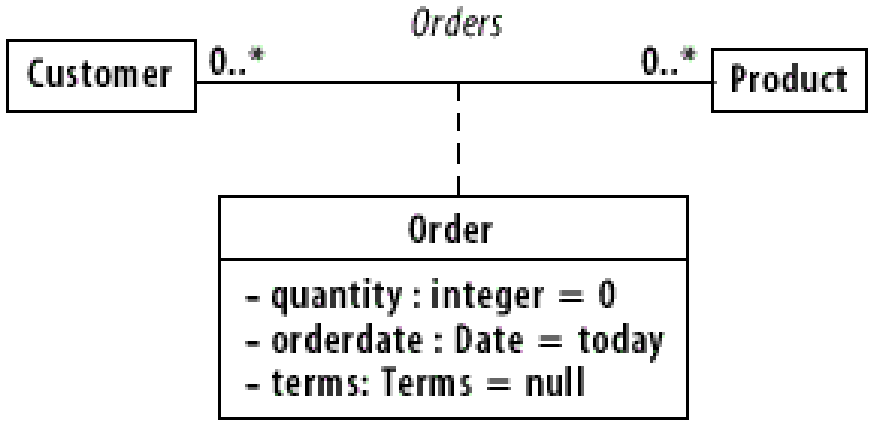


Associations avancées

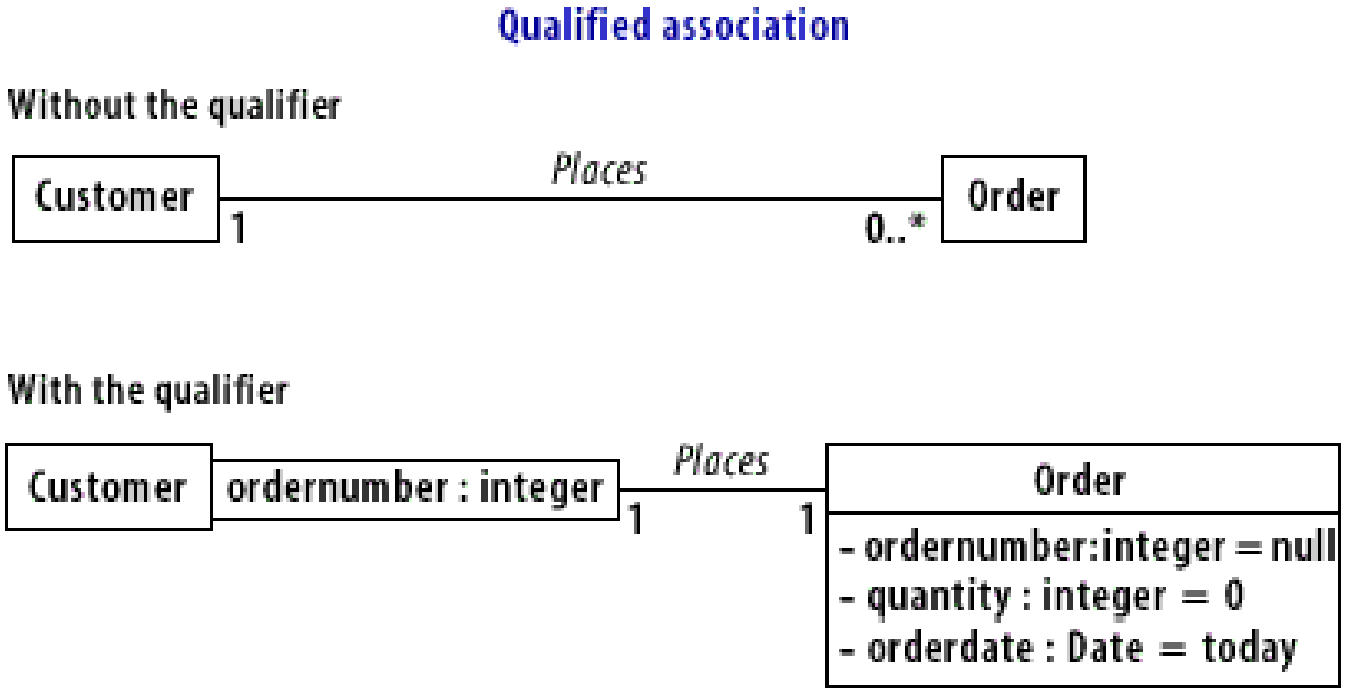
Reflexive associations



Association class



Association qualifiée



Applications de base et dérivées

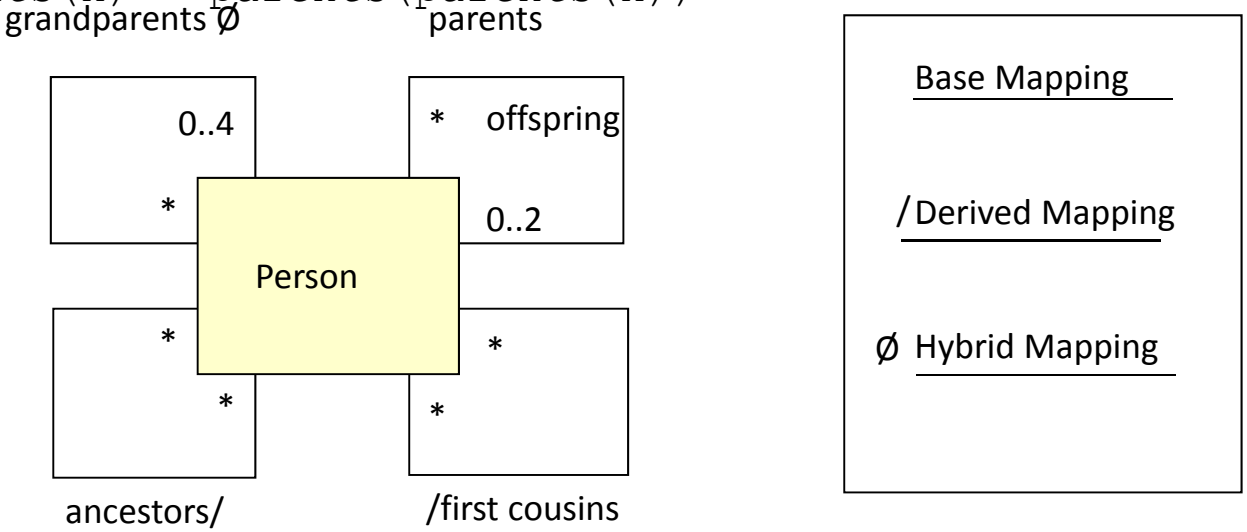
Les applications de base impliquent des associations qui sont déclaré, ou est affirmé, être vrai. Les applications dérivées ceux-là dont les résultats de cartographie peuvent être dérivés par le calcul ou l'inférence. Applications dérivées :

1. `first_cousins(x) = offspring(offspring(parents(parents(x)))-parents(x))`
2. `ancestors(x) = applied recursively (parents(x), Person(x))`

   ou

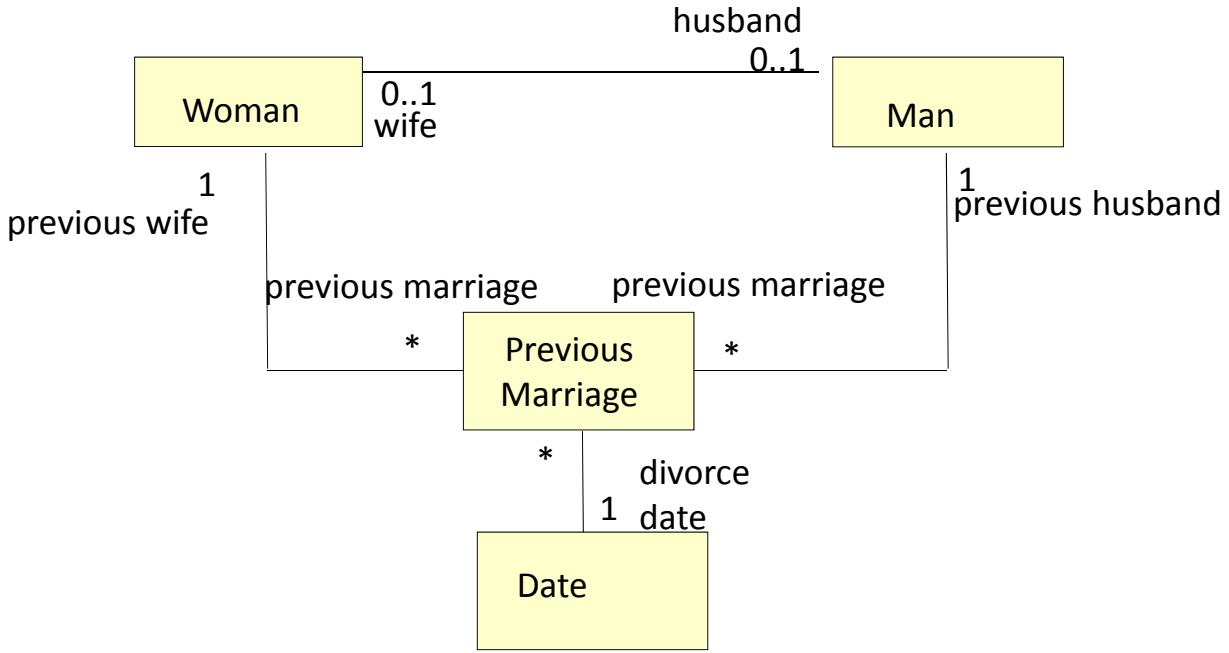
   `ancestors(x) = parents(x) + ancestors(parents(x))`

- Quelquefois, une application peut être de base et dérivé. Ce type de cartographie est appelé une cartographie d'hybride. Les cartographies hybrides nous permettent de choisir si une cartographie devrait être dérivée. Quand dérivé, les grands-parents font la carte de peut être exprimé comme :
- `grandparents(x) = parents(parents(x))`



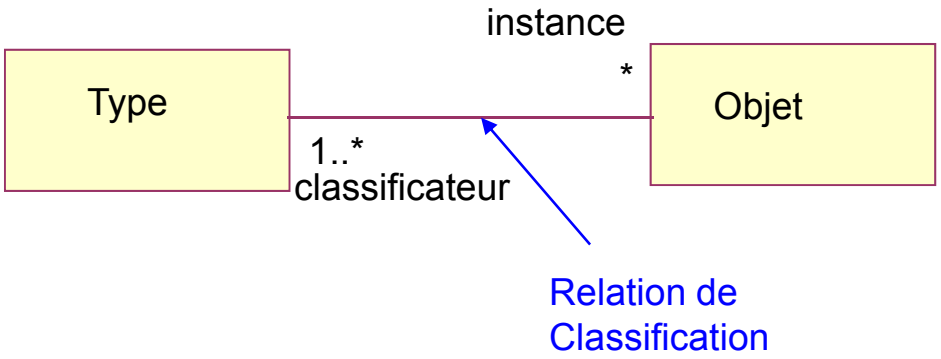
Associations

- Une classe peut communiquer seulement avec un autre classe avec laquelle il est associé. Pour associer deux classes, dessiner une ligne entre les classes parées avec de qualifier des éléments.
- L'association = Une relation établie par l'utilisateur.
- Chacun du tuples d'association peut être pensé comme un objet dans sa propre droite. Donc, une association peut être considérée un type dans sa propre droite. Les associations peuvent avoir des propriétés.



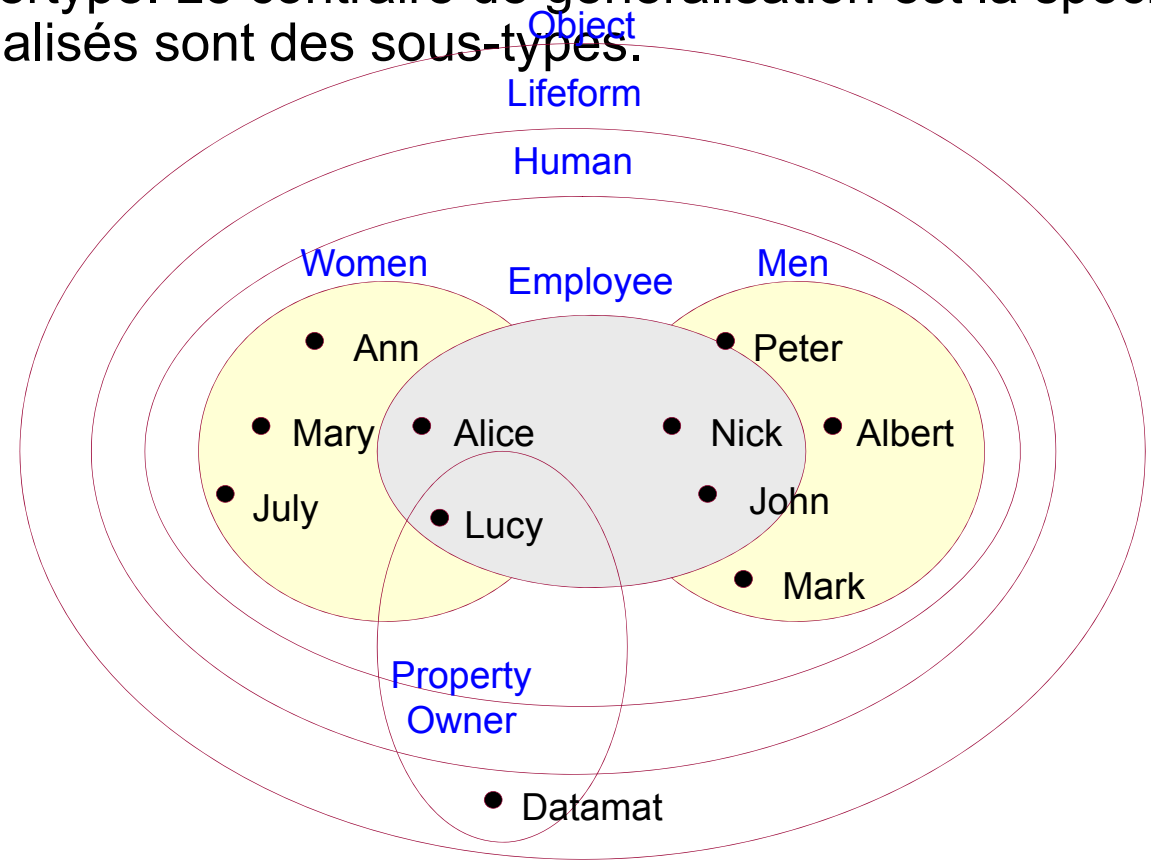
Classification

- La *classification* est l'acte ou le résultat d'appliquer un concept (c.-à-d. le type) à un objet. La classification est un mécanisme que nous utilisons gérer la complexité des objets dans le monde : il nous permet de traiter beaucoup de phénomènes individuels (les objets) avec une notion seule (le type).



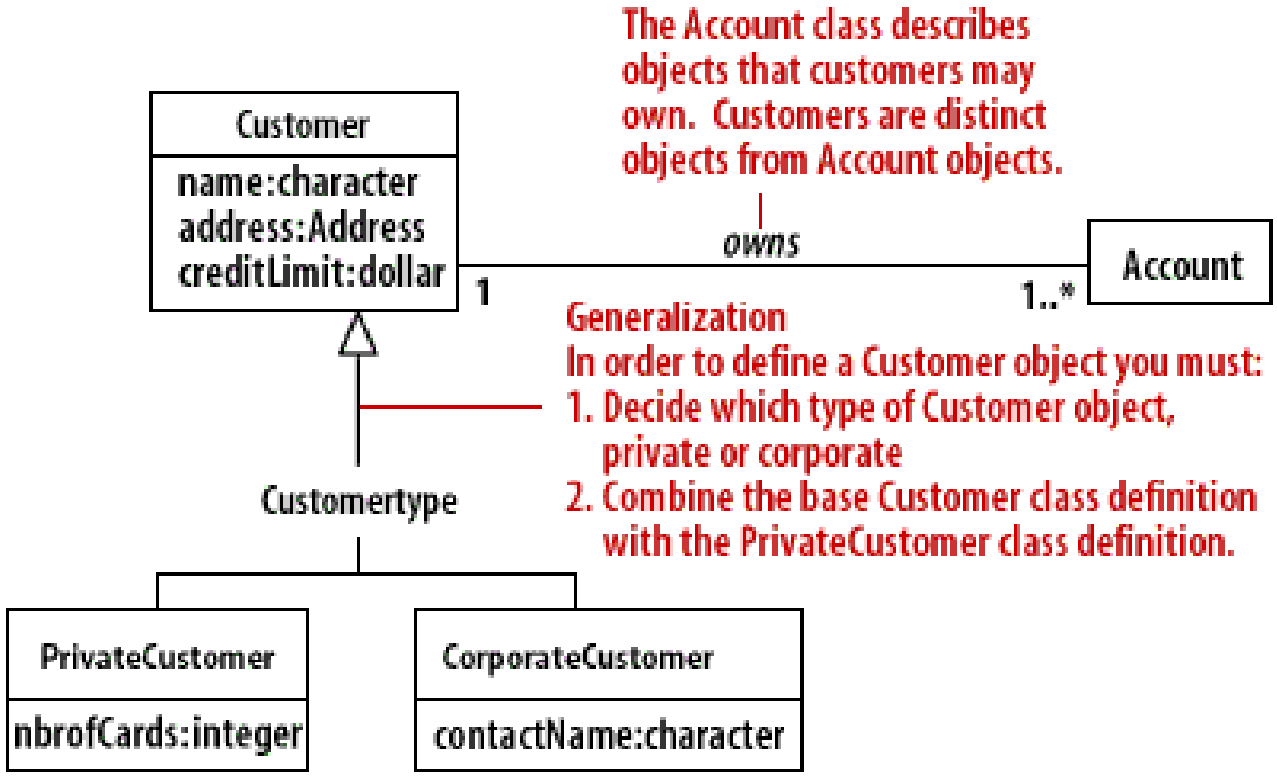
Généralisation

- La généralisation est l'acte ou le résultat de distinguer un concept (c.-à-d. le type) qui inclut complètement ou entoure un autre. La généralisation nous rend capable de dire que tous exemples d'un type spécifique sont aussi des exemples de type plus général – mais pas nécessairement l'autre façon environ. Avec la généralisation nous pouvons définir des hiérarchies de types, formant de plus en plus de types généraux. Le type plus général est appelé supertype. Le contraire de généralisation est la spécialisation. Les types spécialisés sont des sous-types.



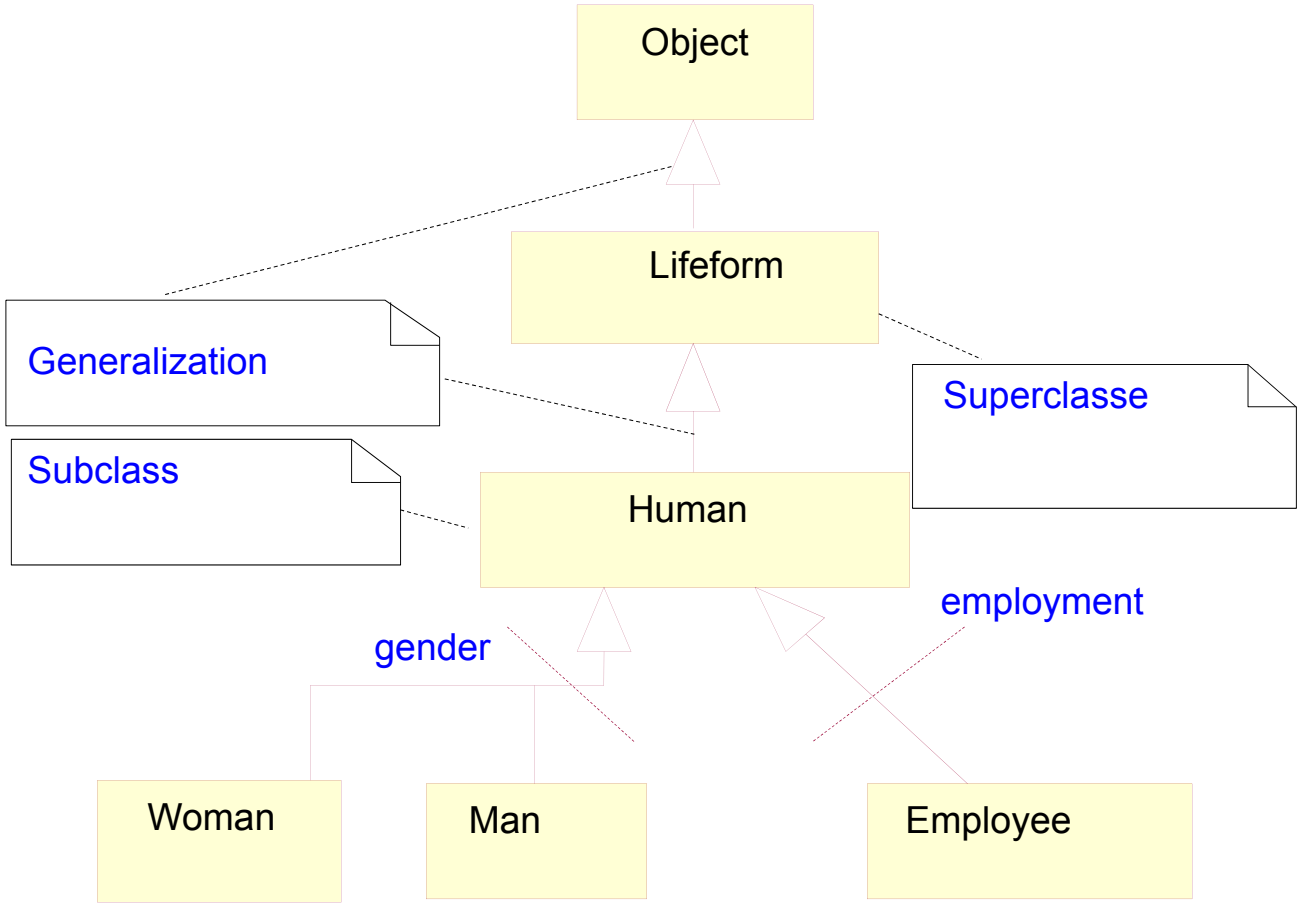
Modélisation de la Généralisation

- La généralisation est une technique pour organiser des informations des objets selon les similarités et les distinctions dans la série d'objets. La généralisation est contrairement à aucun autre type d'association. Les associations décrivent comment les objets se sont relatés. La généralisation lie ensemble des morceaux de définitions de classe pour créer une définition de classe complète.



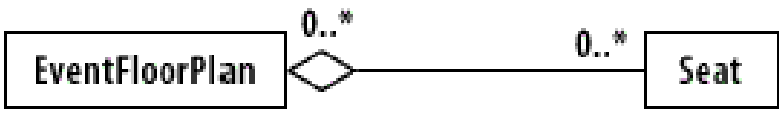
Discriminants de la généralisation

- La généralisation est l'acte ou le résultat de distinguer un type qui inclut complètement, ou entoure, un autre type. La spécialisation est l'acte ou le résultat de distinguer un type qui est inclus tout à fait dans, ou le com-pletely a entouré par, un autre type.

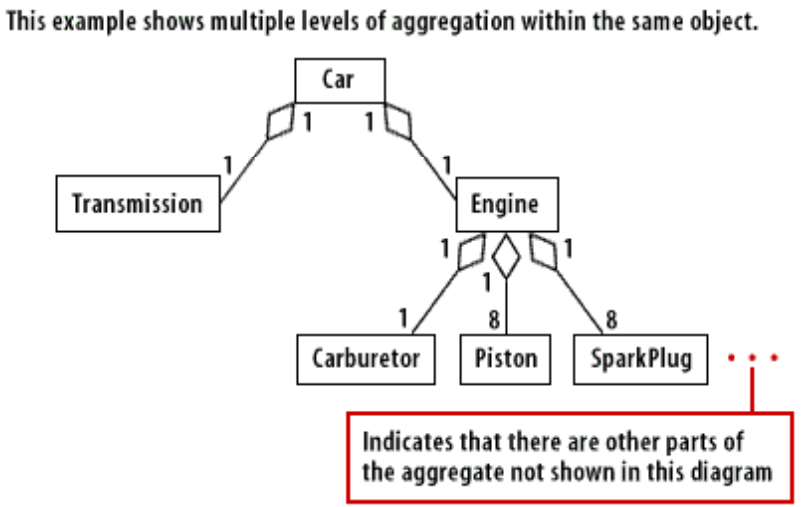


Aggrégation

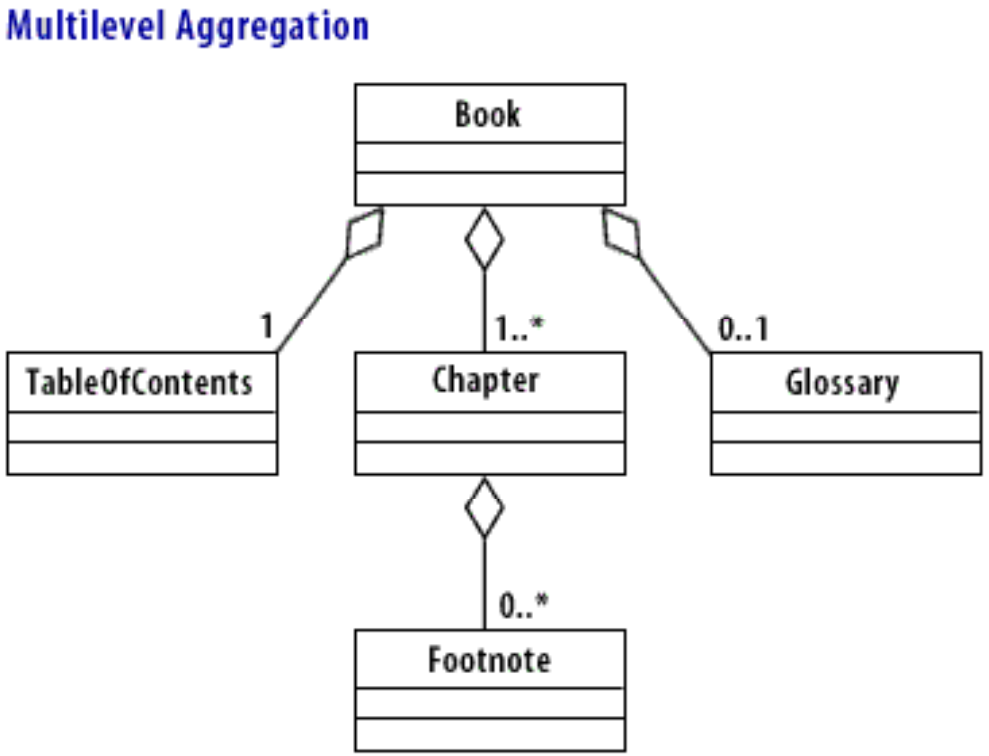
- L'agrégation est un mécanisme pour former un ensemble des parties de composant. Le c'est l'acte ou le résultat de former un ensemble d'objet utilisant d'autres objets comme ses parties. L'agrégation peut définir des hiérarchies de configurations de partie-ensemble. Les exemples du type agrégé sont composés d'exemples d'autres types. L'objet collectif est plus grand que la somme de ses parties. Toutes interactions avec cette assemblée d'objets sont exécutées par l'interface d'objet collectif. Les objets de membre sc grégat.



- L'agrégation est utilisée pour la modélisation des assemblées logiques et physique. Par exemple, une équipe de baseball est une assemblée logique. Une voiture est cependant une assemblée physique avec un nombre de composant parties travaillant Ensemble comme une unité. Pour déplacer la voiture exige en avant que les parties travaillent ensemble dans une manière prescrite et comme une unité.

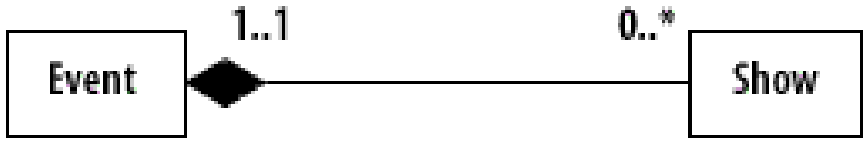


Agrégation à plusieurs niveaux



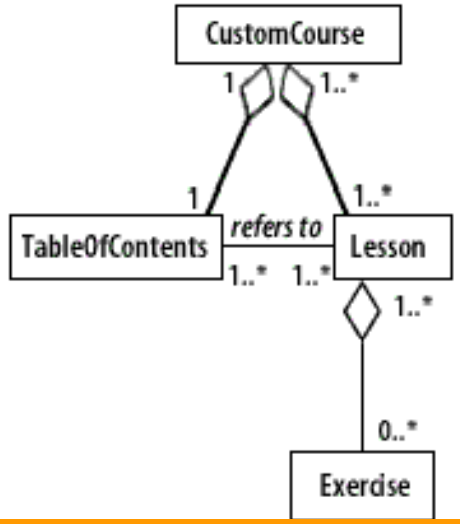
Composition

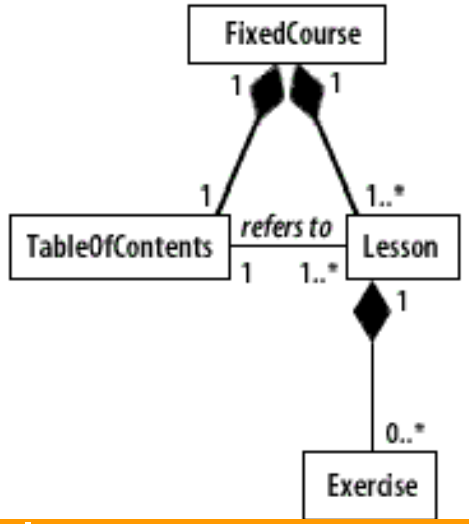
- La composition est une forte forme d'agrégation. Par fort, l'UML signifie que la portée de vie des parties dépend de la portée de vie de l'agrégat. Les parties n'existent pas en dehors de leur participation dans l'agrégat. Cette forte affiliation implique des objets physiques qui sont ensemble assemblés former le composite. Pour cette raison, la multiplicité sur la fin de composé/diamant de l'association est toujours l'un.



- La multiplicité de l'un est impliqué à la fin de composé/diamant de l'association. La multiplicité doit être assignée à la fin de « partie » de l'association. Les parties non plus long existent si le composite est dissimulé ou est détruit. Les parties ne pourraient pas être déplacées d'un composite à un autre.

Comparaison entre agrégation et composition





Course Materials as an aggregate	Course Materials as a composite
Definition: Custom courses may be developed from existing courses by combining existing lessons and exercises and creating a new table of contents	**Definition: Fixed courses are developed and delivered as is. To create a new course all of the material is created from scratch**
• The table of contents is unique for each course • The table of contents refers to every lesson in the course. The lesson is listed in the table of contents for every course it is used in. • The lessons are initially developed for one course but may be used to make other customer courses • The exercises are initially developed for one course but may be used with other lessons for other courses	• The table of contents is unique for each course • The table of contents refers to every lesson in the course. Each lesson is listed in only the table of contents for the course for which it was developed • The lesson is only used in the course it was developed for • The exercises are only used in the lessons for which they are developed

Le contexte de la composition

- L'agrégation pose un problème en interprétant certaines des associations liées. Le contexte de composition aide à rectifier ce problème. Dans alors calculer nous avons des compositions, parce que les chapitres et les autres parties ne peuvent pas exister en dehors du livre. Regarder le diagramme de classe de Livre, nous ne pouvons pas déterminer si une note de bas de page se réfère aux chapitres dans le même livre ou les seulement chapitres dans les autres livres, ou si les notes de bas de page se réfèrent au livre dans lequel ils résident ou seulement aux autres livres.

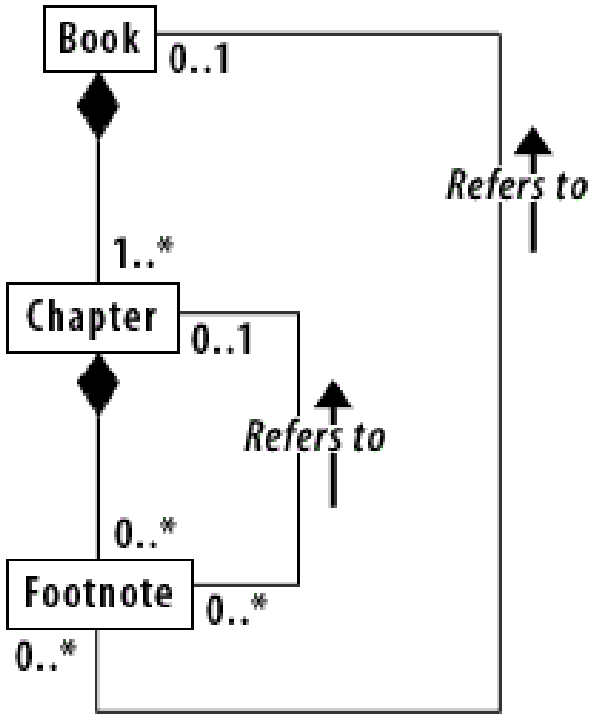
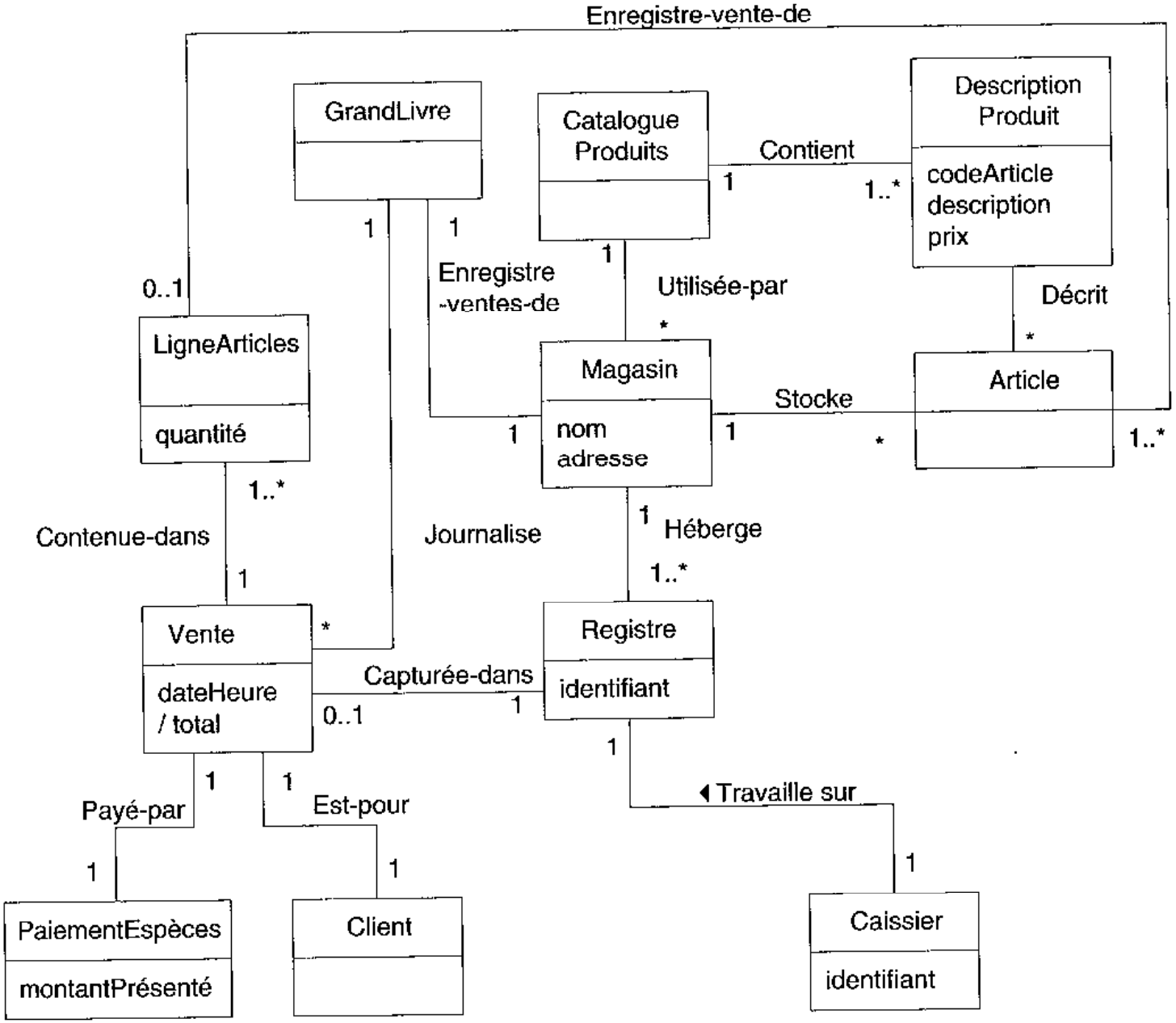
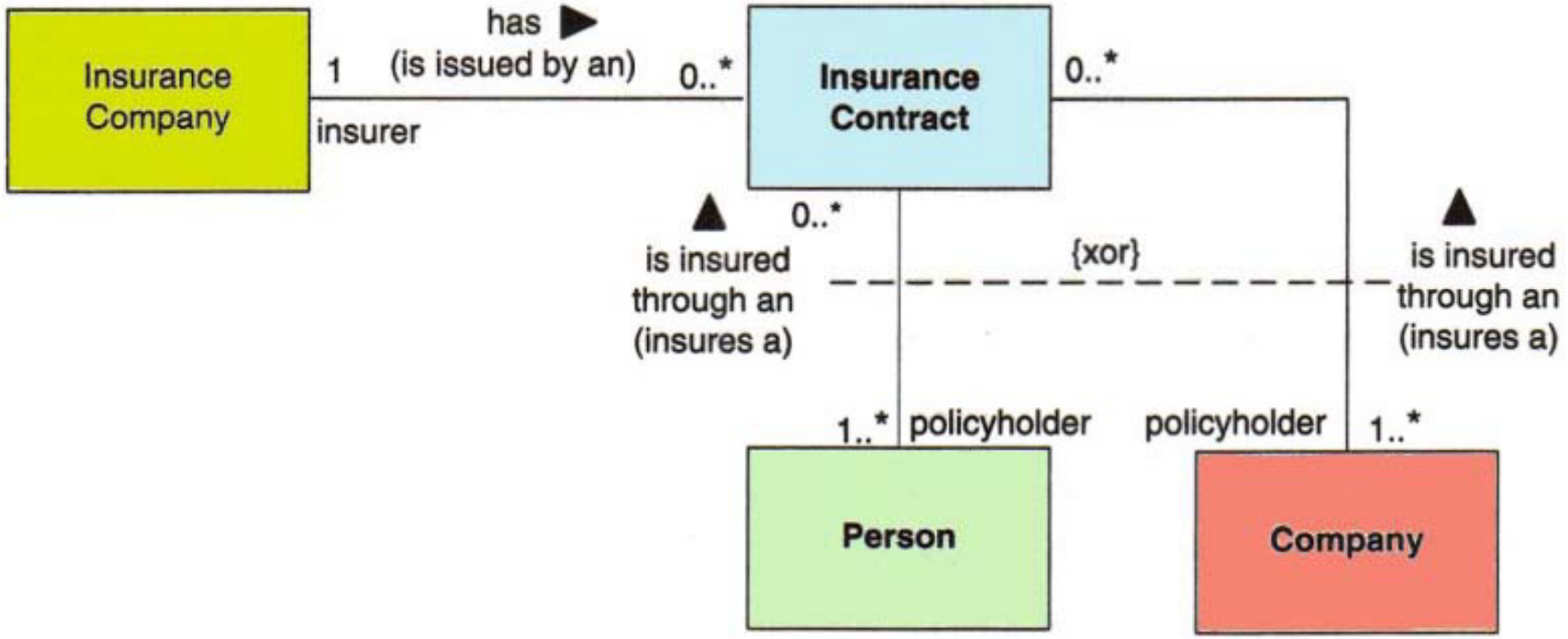


Diagramme de classes

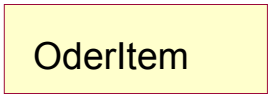


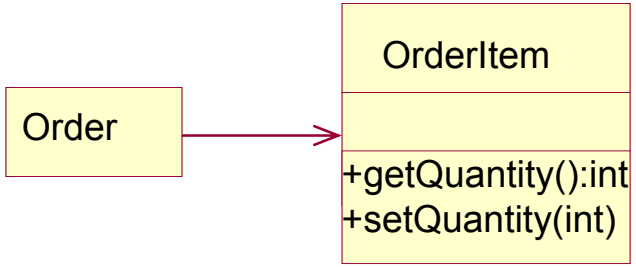
Diagrammes de classes

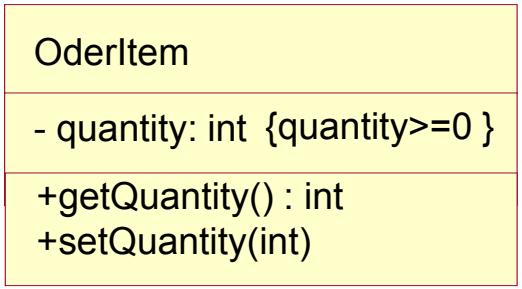


Perspectives on Classes

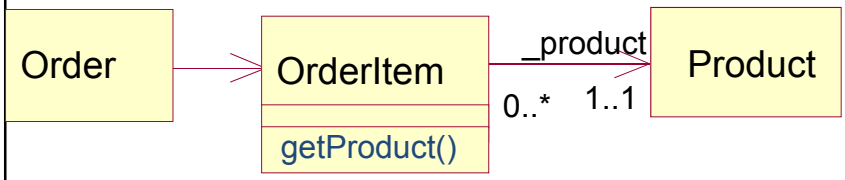
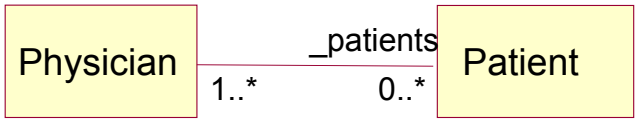
	UML	Java implementation

Conceptual perspective		class OrderItem {
		
		}

Specification perspective		class OrderItem { public int getQuantity(){} public void setQuantity(int q){}

Implemen-tation perspective		class OrderItem {
		private int quantity;
		public void setQuantity(int q){
		if (q>=0)
		quantity = q;
		}
		}

Perspectives on associations

Perspective	Case 1	Case 2
Conceptual perspective		
Specification perspective	class OrderItem { public Product getProduct();	class Physician { public Iterator patients();
Implementation perspective	class OrderItem { private Product _product;	class Physician { private ArrayList _patients;

Quand vous devriez utiliser diagrammes de classes ? (I)

- Le diagramme de classe est le produit central d'analyse et la conception objet-orienté. Toutes informations qui ont été découvertes et ont été documentées dans autres modèles doit être représenté dans le diagramme de classe.
- Diagrammes de classes sont d'abord utilisés dans la phase de modélisation métier. Le *domaine de business* est représenté par un diagramme de classes contenant tous les concepts principaux dans le business et leurs relations sémantiques.
- L'objectif de la phase d'*analyse du problème* appartenant au cycle de vie d'un projet de logiciel est de définir seulement ces classes d'objets dans le domaine de business qui feront partie du domaine du problème même si vous n'avez pas créé encore du logiciel. L'étendue des définitions de ces classe sont aussi limitées au but des objets et les interfaces ils ont besoin pour les utiliser. Le diagramme de classes résultant est le modèle du problème ou du domaine et fait partie du modèle conceptuel du futur produit de logiciel.

Quand vous devriez utiliser diagrammes de classes ? (II)

- Le diagramme de classes résultant est le modèle du problème ou du domaine et fait partie du modèle conceptuel du futur produit de logiciel. Un modèle de domaine est composé de classes que représentent concepts dans le monde réel, pas de composants de logiciel. Il n'est pas une série de diagrammes décrivant des classes de logiciel, ou des objets de logiciel avec leur responsabilités.
- Tout vous avez défini dans l'analyse doit rester intact comme vous transférez à la conception. En fait, le domaine conceptuel sera la base pour votre conception de données. Peu de, si n'importe quel, des nouveaux objets supplémentaires pendant la conception sera ajouté au modèle conceptuel et devient la partie de la base de données. Enfin un diagramme de classe résulte représentant la conception (le domaine) le modèle.

2.3. Object Behavior

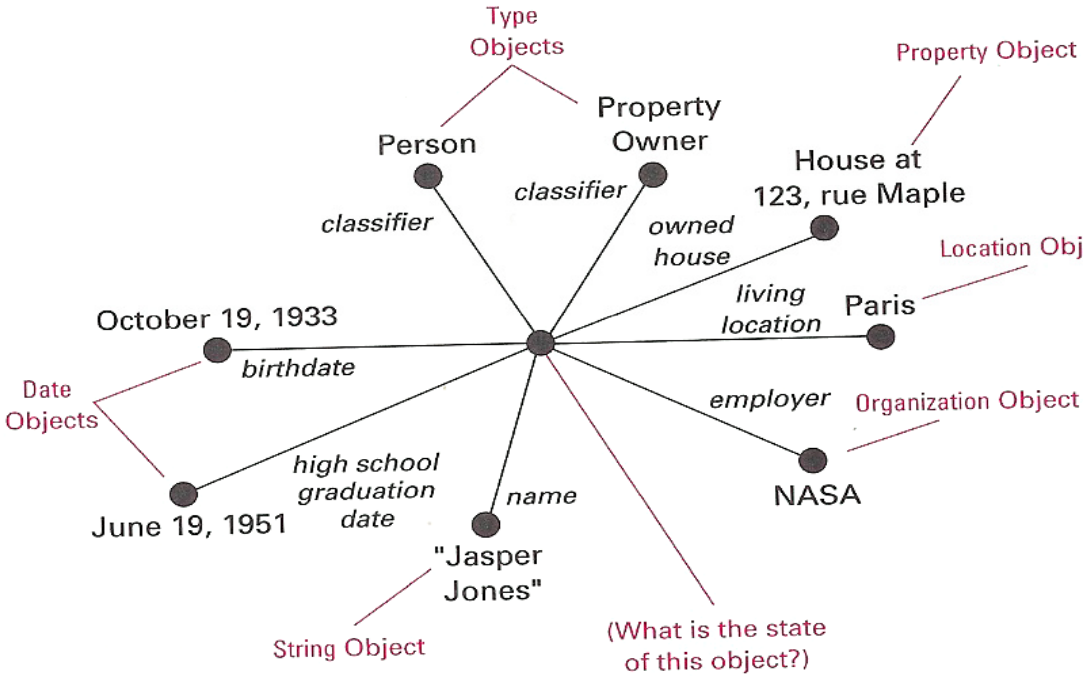
Temporal relationships are difficult to understand. A system can best be understood by first examining its static structure, that is, the structure of its objects and their relationships to each other at a single moment in time.

Then we examine changes to the objects and their relationships over time. Those aspects of a system that are concerned with time and changes are the dynamic model, in contrast with the static, or structural model.

Finally, how do the objects communicate with each other is concerned with time, too, and communication scenarios are the collaborative model, in contrast with the above two models.

Object States

- Object state is a collection of relationships an object has with other objects and types.
- In defining states, we ignore those relationships that do not affect the object's behavior that is interesting for us, and we lump together in a single state all combinations of relationship links that have the same response to events. Technically, the state is an abstraction that represents a combination of possible values of the attributes, combination that is meaningful for an external observer.



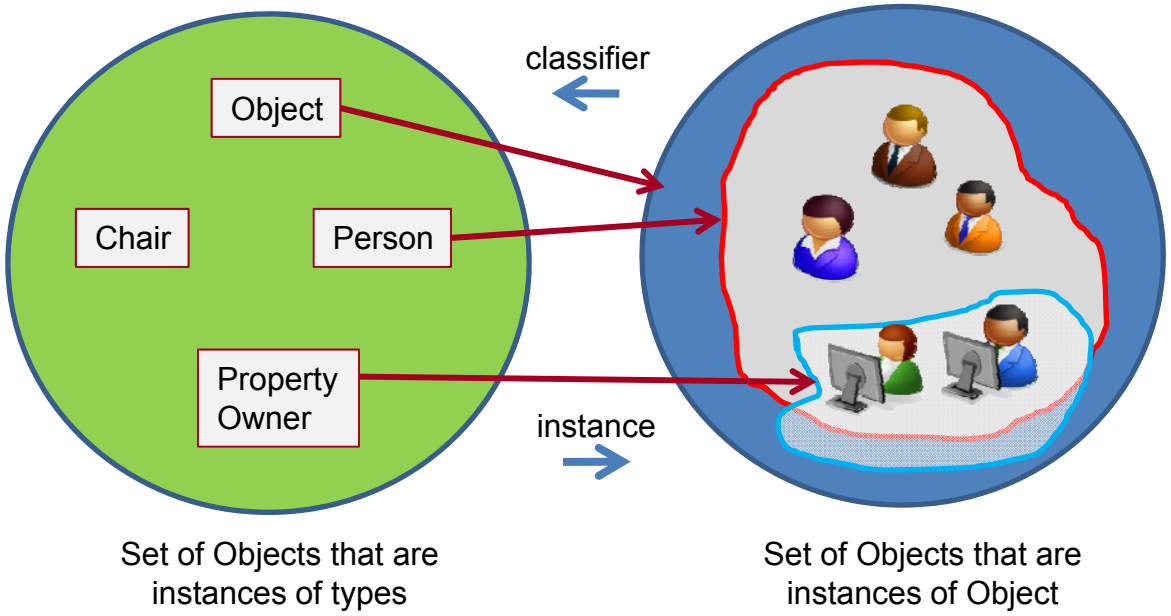
- Point states are states at a particular point in time
- Period states are states whose durations can be measured in time periods such as nanoseconds or milliseconds.

State and Mappings

- The classification of an object is just as much a part of the state as any other association.
- The objects on the left are instances of Type and are mapped to another set of objects that are instances of Object.
- Attributes and relationships in class diagrams are in reality alternate representations of mappings.
- Since all relationships determine an object's state, an equivalent definition of state can be used:

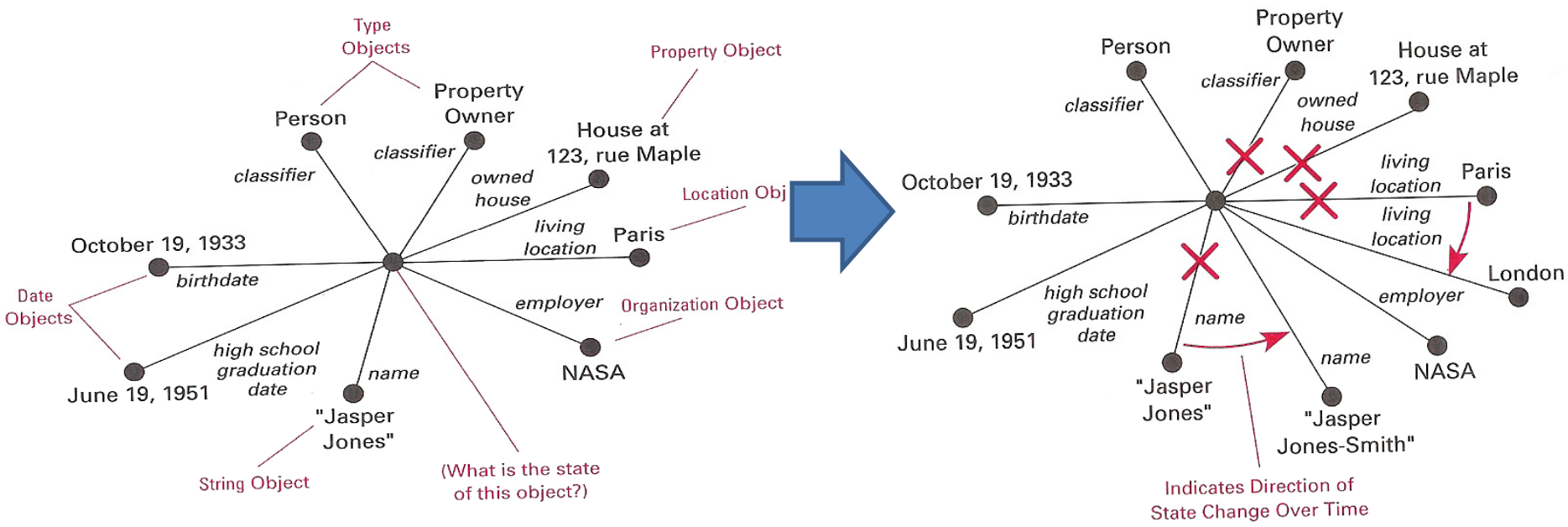
  Object Stare is a collection of mapping instances for an object.
- is equivalent with:

  Object State is a collection of attributes and relationships for an object.



Object State Changes (II)

- *An object state change is the transition of an object from one state to another.*
- An object's state changes when its relationships are created or terminated.
- An object state change is the transition of the object from one state to another. It is thus a change in an object's attribute(s) and/or relationship(s).

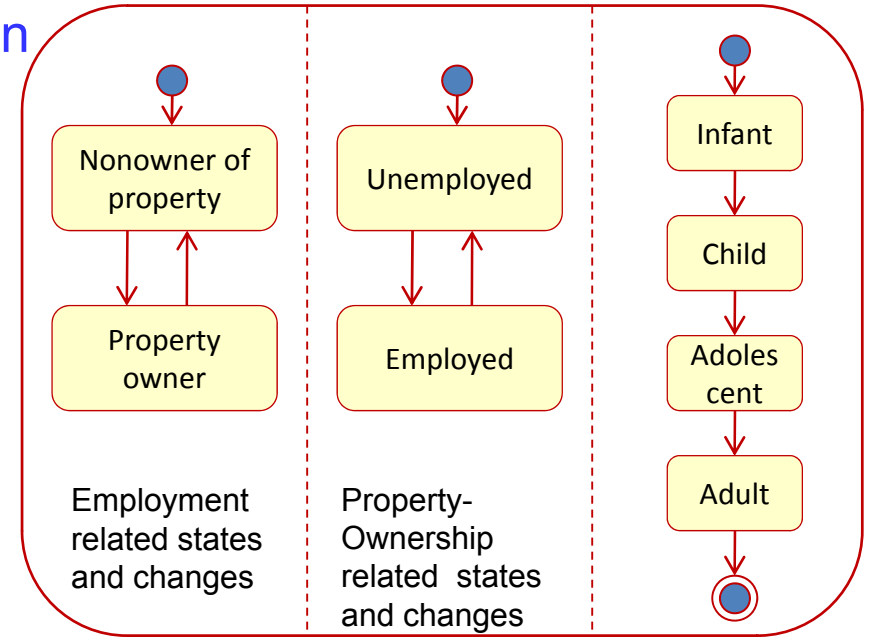


Object State Changes (II)

- More lifecycle views for an object:
  - A Person object can be Unemployed and make a state change to be Employed. Independently of the employment - related states and changes, the same object can participate in the property-ownership - related states and changes - as well as the person/age-related states and changes.
- State change means that one or more of an object's relationships change:

  An object state change is a change in an object's relationship(s).

Person has multiple, concurrent life cycles, that were obtained from multiple views.
All the orthogonal views react to the same events and answer simultaneously to them.



Events

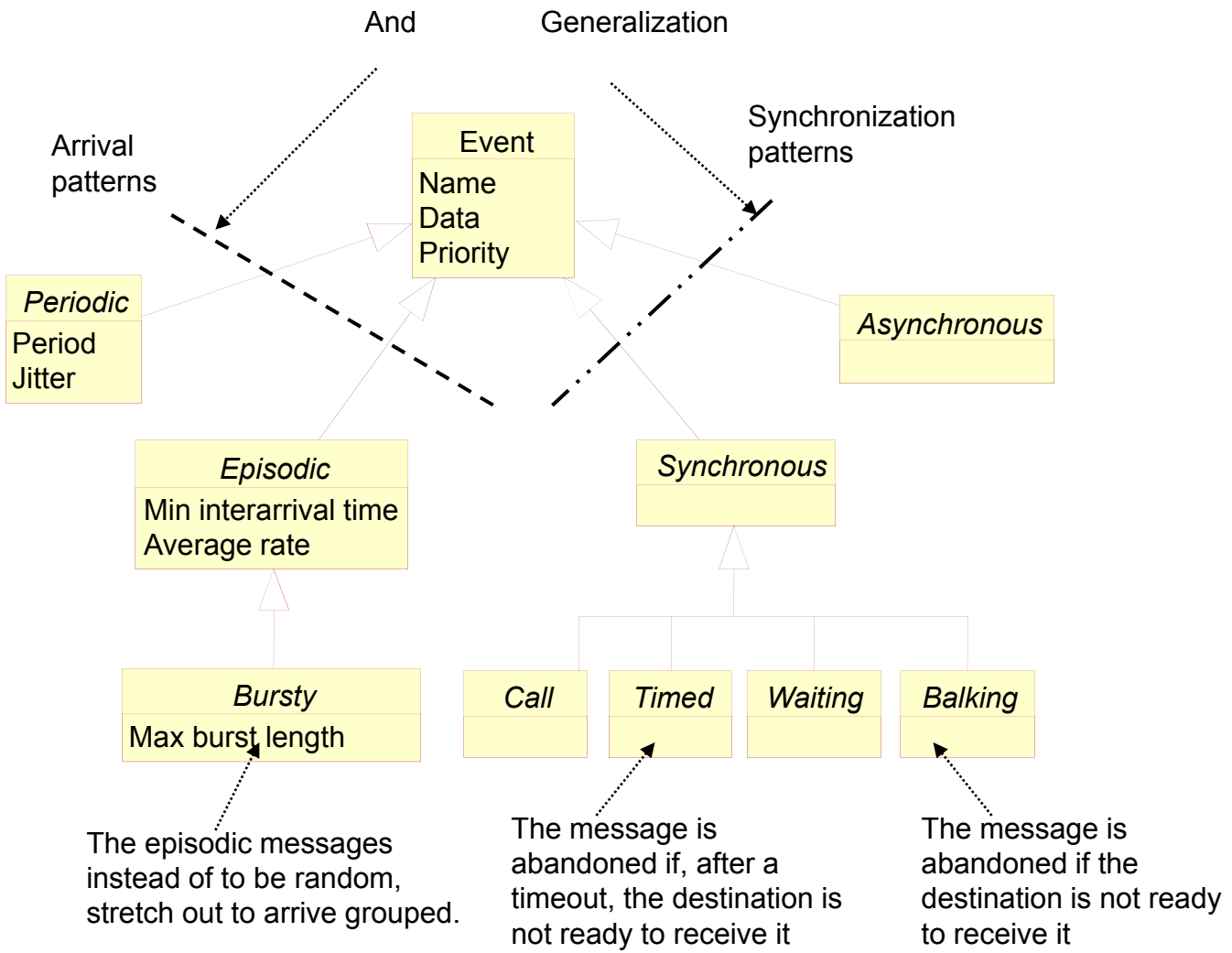
- An event is something that happens at a point in time.
- An event has no duration, that is an event is simply an occurrence that is fast compared to the granularity of the time scale of a given abstraction.
- When state changes are important enough for us to acknowledge, we model them as events:

  Every occurrences of an event is a noteworthy change in state.
- Several basic kinds of state changes and consequently event types:
  - Creation event: an entirely new object appears.
  - Termination event: an existing object is removed from our awareness.
  - Classification event: the new classification of an existing object.
  - Declassification event: the declassification of an existing object.
  - A connection event adds an entirely new relationship between 2 objects.
  - A disconnection event removes an existing relationship between 2 objects.
- Compound events:
  - A reclassification event is the simultaneous declassification of an object as one type and classification of it as another type.
  - A reconnection event is the simultaneous termination of one tuple and creation of another of the same relationship.
  - A coalesce event: a set of objects-previously recognized as distinct-becomes the same object.
  - A decoalesce event: one object becomes a set of objects.

Event Characteristics

- One event may logically precede or follow another because they are causally related, or the two events may be unrelated.
- Two events that are causally unrelated are said to be concurrent; they have no effect on each other.
- An event prestate is a state that must apply to an object before the event occurs.
- An event poststate is a state that must apply to an object after the event occurs.
- Event occurrences are not spontaneously generated: they have a cause.
- The cause is due to some process that makes the state change occur. Events, then, result from the completion of an operation.
- Events can also be thought of as being:
  - Internal (a result of an operation that is within the analyst's domain);
  - External (the result of an operation that is external to the analyst's domain);
  - Temporally caused (the result of a clock operation: End-of-year is reached, April 15th has occurred, or My Birthday has arrived).
- An event occurrence is a noteworthy change in state.
- An event is a type, or classification, of an event occurrence.

Event Class Model in UML

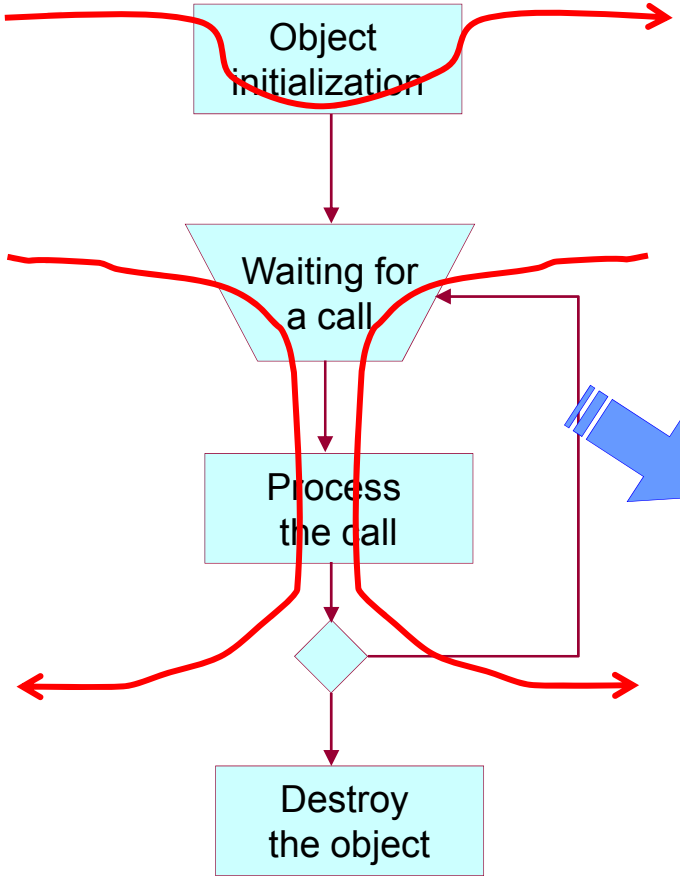


Events and States Are Duals of One Another

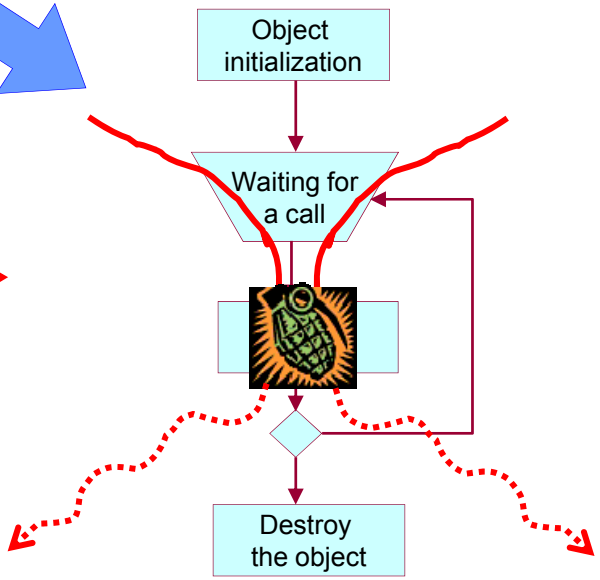
- A state corresponds to the interval between two events received by an object. Events represent points in time; states represent intervals of time.
- The state of an object depends on the past sequence of events it has received, but in most cases past events are eventually hidden by subsequent events. For example, events that happened before the phone is hung up have no effect on future behavior; the Idle state "forgets" events received prior to the hang up event.
- A state has duration; it occupies an interval of time. A state is often associated with a continuous activity, such as the ringing of a telephone, or an activity that takes time to complete, such as flying from Paris to London.
- Both events and states depend on the level of abstraction we use. For example, a travel agent planning an itinerary would treat each segment of a journey as a single event; a flight status board in an airport would distinguish departures and arrivals; an air traffic control system would break each flight into many geographical legs.

Objects and threads

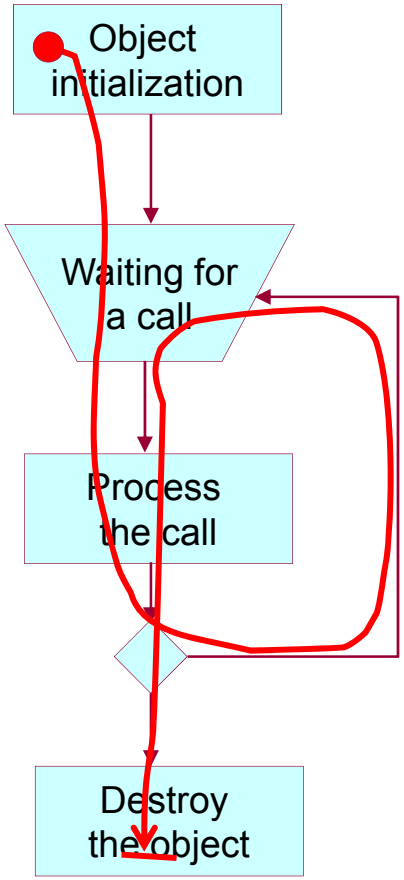
Passive objects are involved in a control thread initiated in outside objects



- Encapsulation does not protect the object from coming conflicts due to concurrency.
- Explicit synchronization is needed.

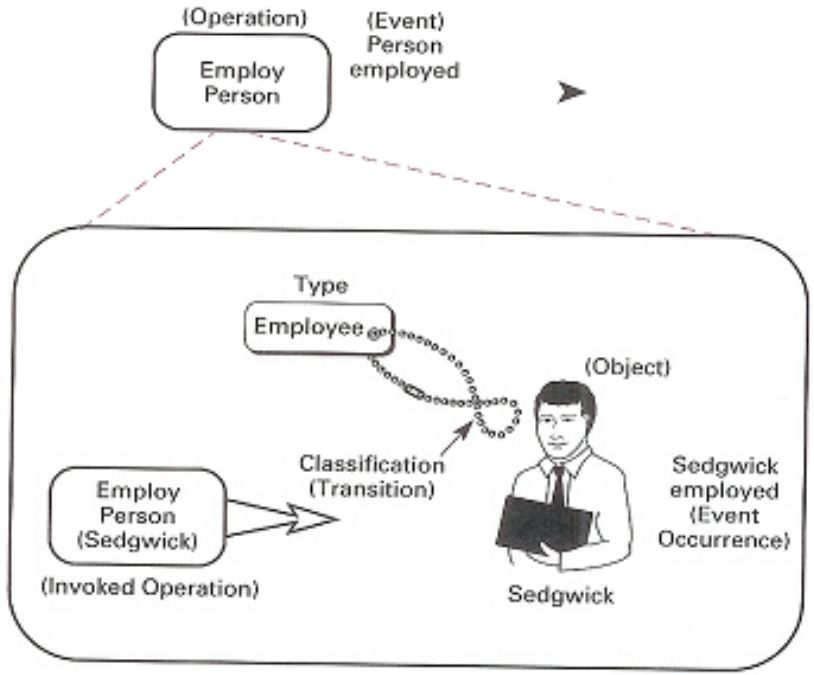


Active objects embody their own threads



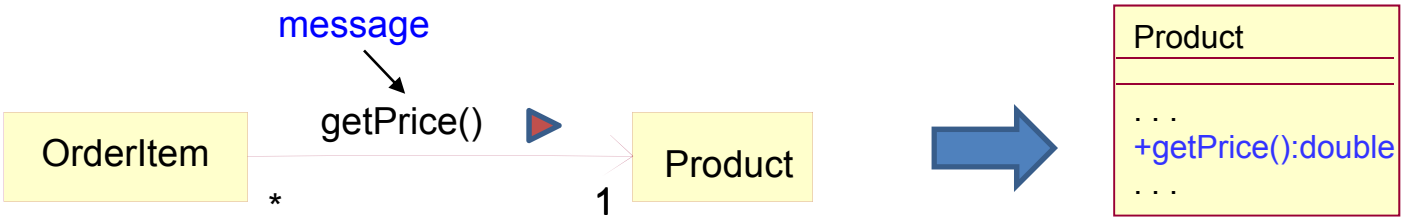
Object Operations (I)

- An operation is a process that can be requested as a unit.
- Operation and event are two different views of the same thing. When the OO analyst wishes to focus on the mechanism for change, she uses the operation view. When focusing on its effect, she uses the event view.
- Events and operations are also types of things. The instances of operations are invoked operations. The instances of events are event occurrences.



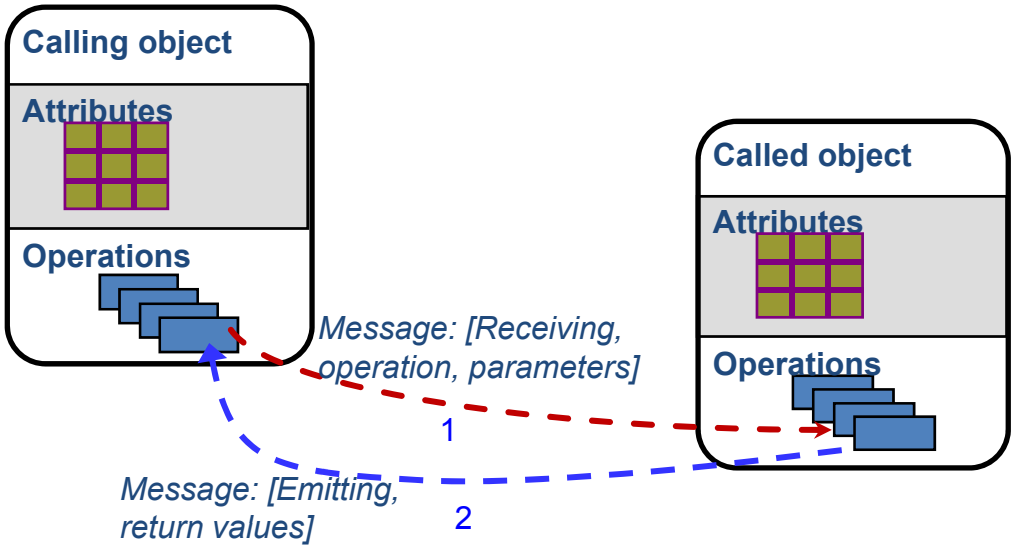
Object Operations (II)

- Each class contains those operations necessary to mediate access to its type of object. Operations perform two basic functions—to query and to change the state of the objects.
- A public operation in a class is an indication that the objects of the class are planned to receive a stimulus or a message from other objects that can ask for execution of the operation.



Messages

- A method is a processing specification for an operation.
- Message = The mean with which the objects act in the environment. It is the specification of a communication among objects that transports an information with the intent to trigger an activity in the receiving object.
- In the object-orientation the messages exchanged among objects are of type synchronous request-replay.



- A message stimulates the activation of a behavior in the receiving object. The behavior ends through the execution of a method that implements the message. After the execution the called object returns the control to the calling one.
- The messages are the true glue of a system OO.

FIN DE LA DEUXIÉME PARTIE

Partie 3. Modélisation métier

Le processus de la modélisation métier est employé pour créer une abstraction d'un business autrement complexe. Ceci rendra capables les parties intéressées (les propriétaires, les clients, la direction, etc.) gagner une meilleure compréhension des fonctions du business et promeut aussi l’innovation et/ou améliorations du business.

Introduction à la modélisation métier

- Un **métier** est une activité humaine, le plus souvent à but lucratif. C'est aussi un savoir-faire acquis et, lorsque le métier le permet, une amélioration de ce savoir.
- Le processus métier est employé pour créer une abstraction d'une activité autrement complexe. Ceci rendra capable les parties intéressées (les propriétaires, les clients, la direction etc.) gagner une meilleure compréhension des fonctions de l'entreprise et promeut aussi de l'innovation et/ou d'améliorations dans l'entreprise.
- La modélisation métier nous permet de représenter deux dimensions de la problématique de l'élaboration des systèmes d'information informatisés (SII):
  - l'expression des besoins sous forme de cas d'utilisation;
  - la définition du domaine et des objets métier.

Modélisation métier vs. Modélisation logiciel

- Pour survivre dans le monde compétitif d'aujourd'hui, les entreprises doivent continuellement réexaminer leurs produits, leurs services, et leurs relations avec l'environnement (les fournisseurs, les concurrents, les clients, les lois, etc). Pour évaluer la qualité de leurs produits et l'efficacité de leurs services, les entreprises comptent sur les systèmes informatiques. Au début seulement un composant de soutien, les systèmes informatiques sont devenus maintenant une partie intégrale des entreprises.
- C'est l'environnement de l'entreprise (c'est-à-dire le « business ») que doit définir les conditions pour le système informatique.
- Malheureusement, très souvent le système de logiciel ne soutient pas convenablement les affaires. Les causes peuvent être : le manque de définition des besoins précis, les manques dans la compréhension correcte des affaires par l'équipe de conception de logiciel, ou même la nature du business (qui peut changer si souvent que le logiciel ne le peut pas suivre).
- Le modélisation de logiciel est une façon admise de concevoir les systèmes de logiciel. Appliquant un approche similaire à la modélisation du business lui-même (la modélisation du métier!), besoins plus précis pourraient être atteintes pour la conception subséquente de logiciel. En plus, on peut utiliser le même langage de modélisation pour le logiciel et pour le business.

Models métier

1. Les organigrammes ont été la façon traditionnelle pour documenter un business. Les modèles métier améliorent cette méthode par fournissant aussi: visions, processus métier, agents métier, ressources métier, objectifs, règles d'exécution, efficacité et autres informations supplémentaires.
2. Les parties intéressées (stakeholders) peuvent avoir vues légèrement différentes sur le business. Donc, un accord sur le modèle métier rendrait capable toutes le parties intéressées pour travailler à une solution commune et comprendre mutuellement leur concepts .

- Un modèle métier devrait servir comme une base pour le modèle des systèmes informatiques, garantissant l'homogénéité et des besoins précises quand on pas à la conception de logiciel. En utilisant un modèle, les entrepreneurs peuvent améliorer leur compréhension des affaires et les possibilités d'amélioration du business.
- Un modèle doit avoir un **but** et la forme et les détails du modèle dépendront principalement du but Pour un business, le but peut être:
  1) compréhension de la structure du business;
  2) l'amélioration du business ou
  3) la re-ingénierie du business.

Types de modélisation métier

- Un modèle métier peut exprimer non seulement un business existent mais aussi la vue future d'un business. Dans ce cas, il y a toujours un degré d'incertitude, et donc un risque, quant à ce que peut arriver pendant l'implémentation du modèle.
- Il y a deux approches principales vers le changer un business:
  - amélioration des processus métier (business process improvement-BPI)
  - innovation des processus métier (business innovation - BI), avec sa forme « radicale », re-ingénieuriser les processus métier (business process reengineering - BPR).

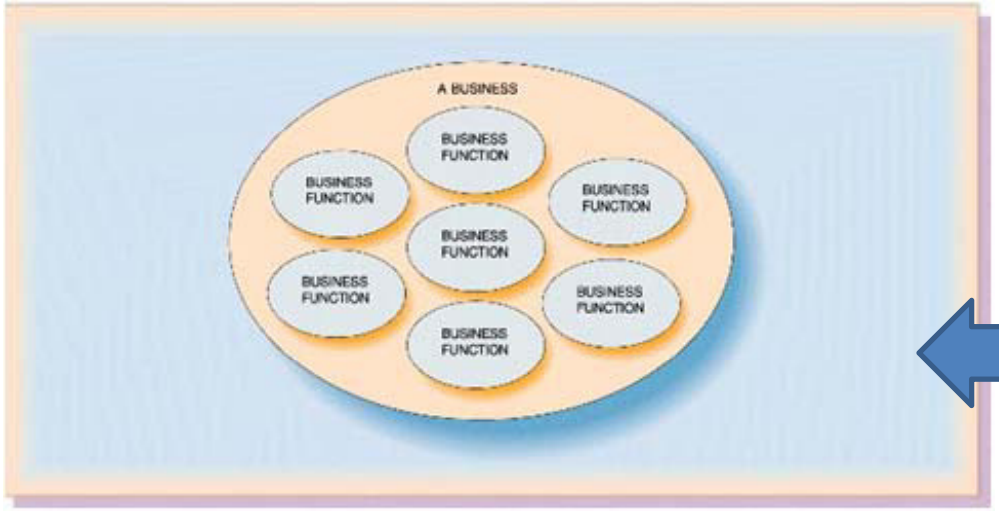
- Quelques caractéristiques pourraient être identifiées dans la modélisation du processus métier.

1. Les objectifs d'affaires et l'allocation des ressources sont imposés par le *propriétaire* ;
2. La structure du business et ses processus sont créés par un *modeleur* ;
3. La conception des systèmes informatiques convenables pour soutenir le business est réalisée par le *développeur des système*.

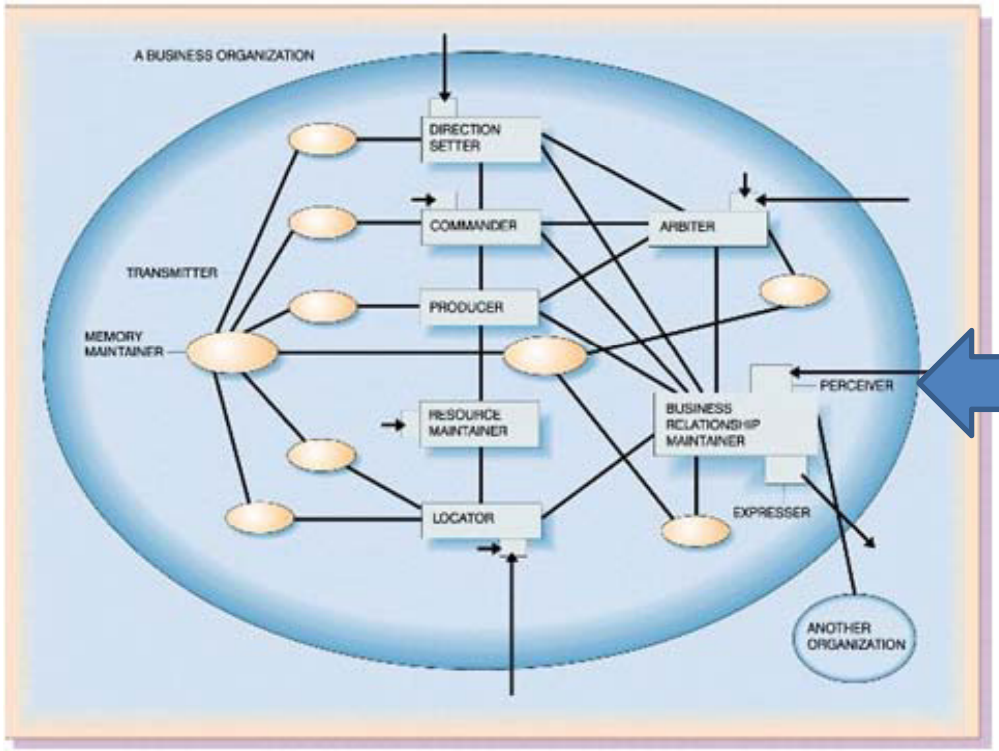
BPI vs. BPR

- L'**approche BPI** est le changement cumulatif fondé sur un modèle métier. Les changements sont appliqués dans des petites étapes continues pour que l'impact (possiblement négatif) sur les affaires soit minimisé.
- L'**approche BI** est plus radicale – le modèle et les processus métier sont fortement changés - et peut amener des améliorations substantielles mais implique aussi un plus haut risque d'échec.
  - L' approche BPR prend l'approche BI même plus ample, visant à doubler/tripler l'efficacité du business. En questionnant chaque aspect du business actuel, BPR implique un haut risque et une difficulté pour réaliser les tâches. BPR peut rencontrer une forte résistance des ressources humaines et échouer même à cause de cette raison (ou d'autres, comme par exemple l'inopportune conception des fonctions du business).
- Le système informatique est un élément clé indépendamment de l'approche utilisée, mais a un spécial signifiant dans le contexte de BPR où il peut aider réduire le risque d'échec de re-ingénieurisation.
- Pour soutenir l'amélioration/l'innovation des processus métier, le système informatique doit refléter et doit soutenir le modèle métier envisagé, et pas le modèle courant que est désuet.

Fonctions de l'organisations



Une organisation peut être conçu comme un ensemble de fonctions de l'organisation. Les fonctions de l'organisation devraient constituer en ensemble une série de fonctions qui sont typiquement exigé pour n'importe quel vivre ou système viable.



Voici un exemple d'un système vivant de comment une série complètement générique de fonctions peut être construite.

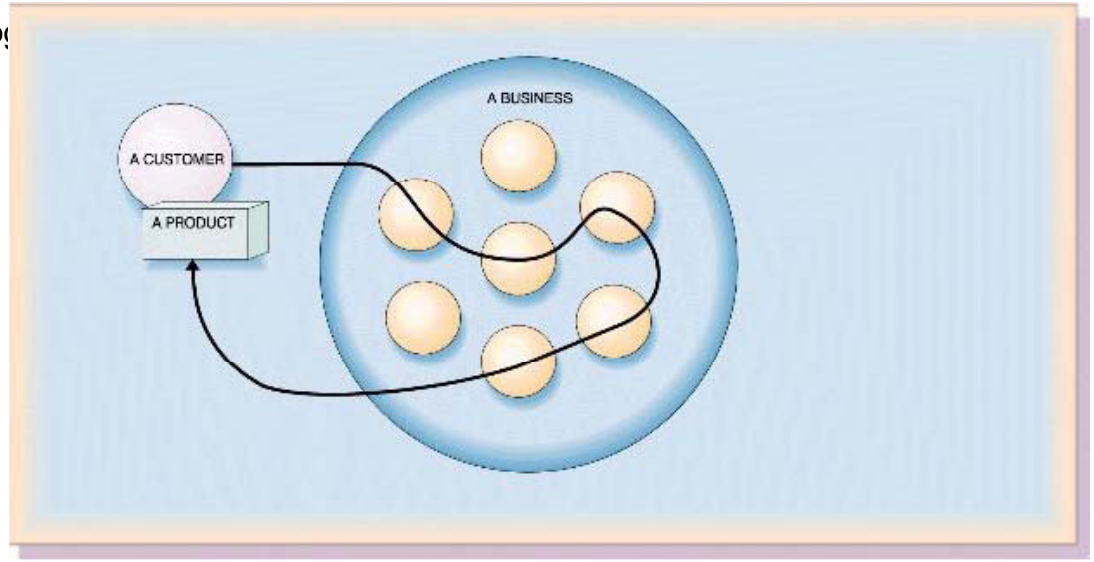
Fonctions de l'organisation

- Fonction de l'organisation est un concept utilisé dans le domaine de l'architecture d'organisation et représente le travail que on fait dans une organisation, une unité d'organisation ou un rôle dans le business. (L'architecture d'organisation définit la manière dont sont répartis les rôles des individus qui la composent et les moyens mis en place pour assurer les relations entre ces derniers.)
- Le concept de fonction de l'organisation peut être pensé comme une organisation virtuelle ou idéalisée à l'intérieur du business.
- Les fonctions de l'organisations devraient être aussi indépendantes que possibles des structures typiques de l'organisation.
- Les fonctions de l'organisations sont plus stables que la structure d'organisation elle-même et souvent une unité d'organisation ou un rôle métier peut être responsable de fonctions de l'organisations multiples. Une fonction de l'organisation jamais est seulement exécuté dans une organisation par un unique rôle métier/unité d'organisation.
- Exemples de fonction de l'organisations: Achats, Fabrication, Technique, Marketing, Gestion Financière, CRM, etc.

Comportement métier

- Le comportement métier est un générateur et contrôleur de tâches ou d'activités qui réalise les objectifs du business et satisfait des engagements du business. Il peut inclure le manuel ou les opérations automatisées qui complètent des unités de travail (units of work).
- Le comportement métier peut être déclenché par:
  - événements dans l'environnement, ou
  - initiatives ou conditions internes.
- Il est justifié parce qu'il ou produit valeur pour le business ou adoucit les coûts au business.
- Le comportement du business :
  - produit les issues qui remplissent le but des affaires;
  - est gouverné par des engagements,
  - est exécuté par les joueurs de rôle.
- Comme une de-bout-en-bout série d'activité, le comportement peut invoquer diverses fonctions de business. Le comportement manipule diverses ressources du business pour produire le résultat désiré.
- Le processus métier est à la base du comportement des systèmes d'information.

Processus métier



- Le processus métier = une séquence complète de comportement métier que consiste en un ensemble d'activités que s'enchaînent de manière chronologique pour atteindre un objectif dans le contexte d'une organisation. Une interaction complète entre le business et un de ses clients est un exemple de processus métier.
- Un processus métier est déclenché par un événement et continue jusqu' produire à un résultat significatif pour business: une valeur observable, généralement la livraison d'un produit ou d'un service, à un acteur métier, au client qui a déclenché le processus.
- In the course of this string of activity, various areas of the business must get involved. These can be interpreted as organizations or as business functions. Processes cut across, or invoke, organizations and functions in the business.
- Au cours de cette séquence de activités, divers secteurs du business seront impliqués. Ceux-ci peuvent être interprétés comme sous-organisations ou comme les fonctions de l'entreprise. Un processus métier traverse le business en invoquant organisations et fonctions de l'entreprise.

Types de processus métier

- Les sous-classes majeures de processus métier sont :
  - transactions,
  - transformations,
  - services,
  - maintenance.
- Les transactions peuvent être principalement dirigées vers l'intérieur, amenant des choses dans le business (argent, informations ou autres ressources) ou dirigées en dehors, faisant sortir des choses (factures, produits, produits dérivés et pertes).
- Les processus de transformation (conversion) prennent des ressources comme des données (matériel/énergie ou informations) et les transforme dans autre (la valeur ajoutée) les états.
- Dans un business que offre services, chaque service est notamment défini par son processus, qui, exécuté, est un produit du business. Ces processus de service peuvent être ou passif du point de vue du cliente ou éxécuté en collaboration avec le client.

Événements

- Les événements business initient les activités dans un business. Ils sont des stimuli qui incitent le business agir. Beaucoup d'événements arrivent au point d'interface entre le business et une des entités externes avec lesquelles il réagit réciproquement. Par exemple, une enquête peut déclencher l'activité qui mène à un ordre, ou un appel pour un défaut peut déclencher l'activité de réparation.
- Les autres événements sont des détentes internes fondées sur une condition spécifique ou sur des intervalles de temps prédéfini. Par exemple, la quantité d'un article dans un inventaire peut déclencher une commande, ou le quinzième jour du mois peut déclencher automatiquement un cycle facturant.
- Les événements produits en extérieur peuvent être sollicités, et donc prévus, ou presque prévisible (par exemple, les ventes, stimulées par des campagnes sur la marché). Les événements imprévus sont les choses catastrophes, telles naturelles, les mouvements de bourse, ou l'apparition d'une nouvelle technologie. Bien qu'imprévu dans le sens de n'être pas sous le contrôle du business, ils peuvent être prévus dans beaucoup de façons et fournis avec les plans d'urgence.
- L'événement business et la sortie représentent le stimulus ou réponse du business. Initiée par le stimulus jusqu'au ultime réponse nous avons la structure d'un « processus métier ».

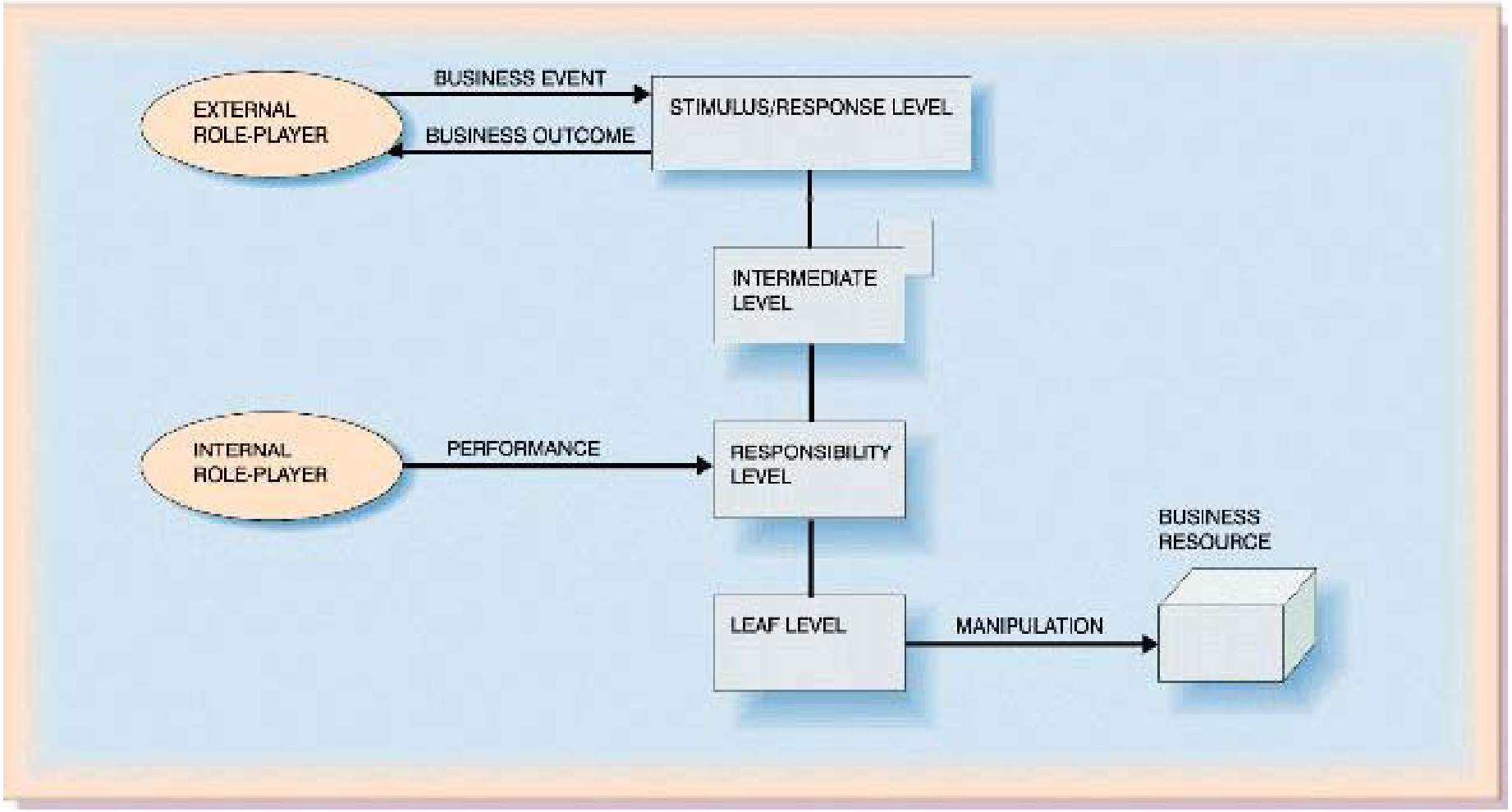
Composants métier

Dans la modélisation métier nous devons considéré les composants :

1. Objectif métier
2. Agents métier
3. Objets (ressources) métier
4. Processus métier
5. Règles métier

1. Internement, le comportement métier est assigné à et est exécuté par un rôle-joueur interne responsable (agent métier).
2. Plus au fond, quelque comportement de niveau-feuille manipule une ressource métier discrètes dans quelque façon significative.
3. A un niveau intermédiaire, il peut y avoir quelque nombre de groupements identifiables de comportement, appelés sous-processus métier, sont appelé pour le bien de convenance d'analyse et de compréhension.
4. A n'importe quel de ces niveaux, ces comportement métier sont gouverné par les règles métier qui dicte comment les flux devraient être inaugurés et devraient être dirigés.

Dans le processus métier



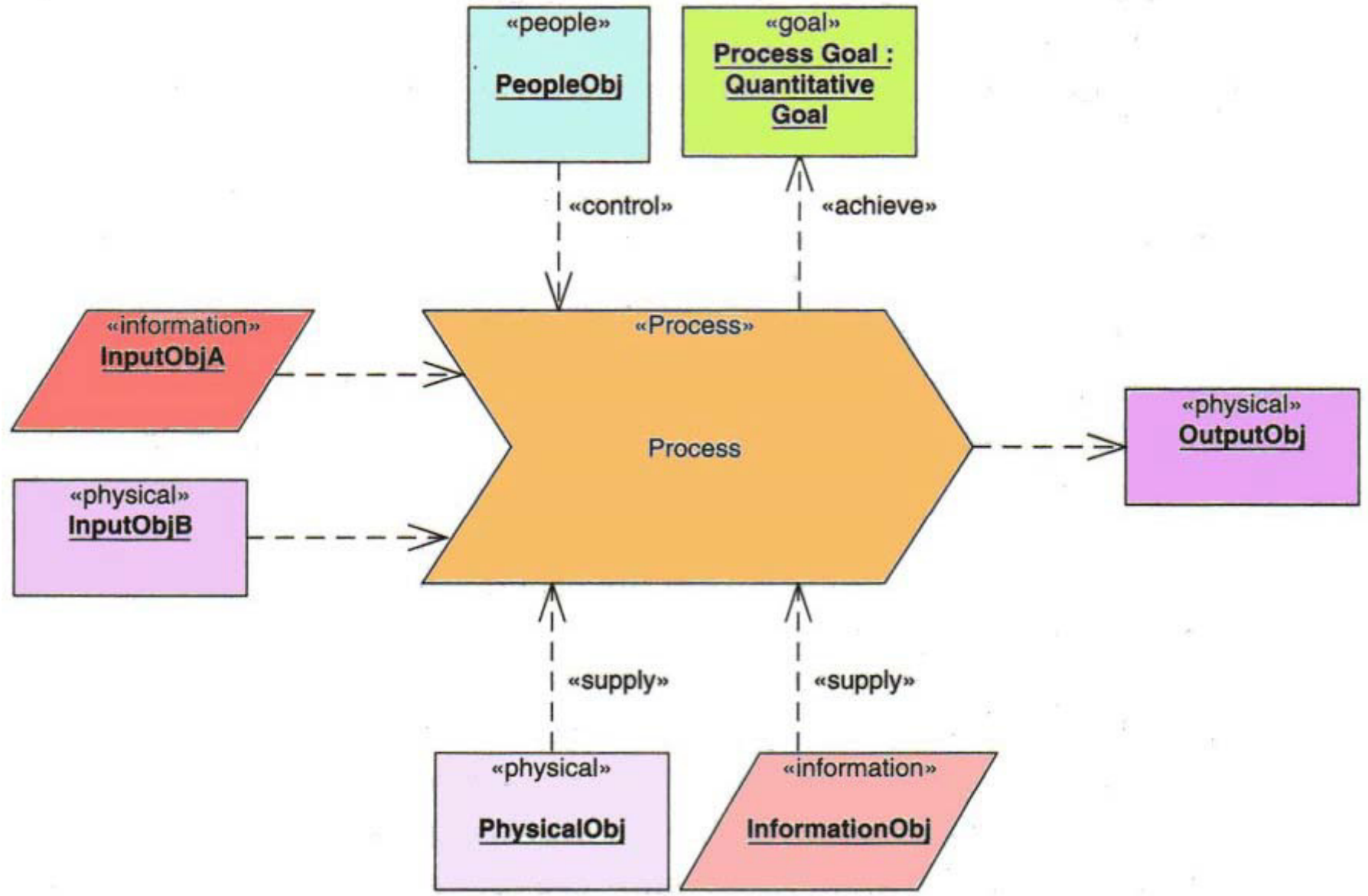
Objet (ressource) métier

Exemple de ressources: ressources physique, énergie, argent, capacité productive (une ressource majeure de toutes entreprises, des compétences, la connaissance, les attitudes, et l'expérience d'humains) et ressources d'information (ils représentent les types de choses qui peuvent être sus par les individus et les organisations dans les affaires).

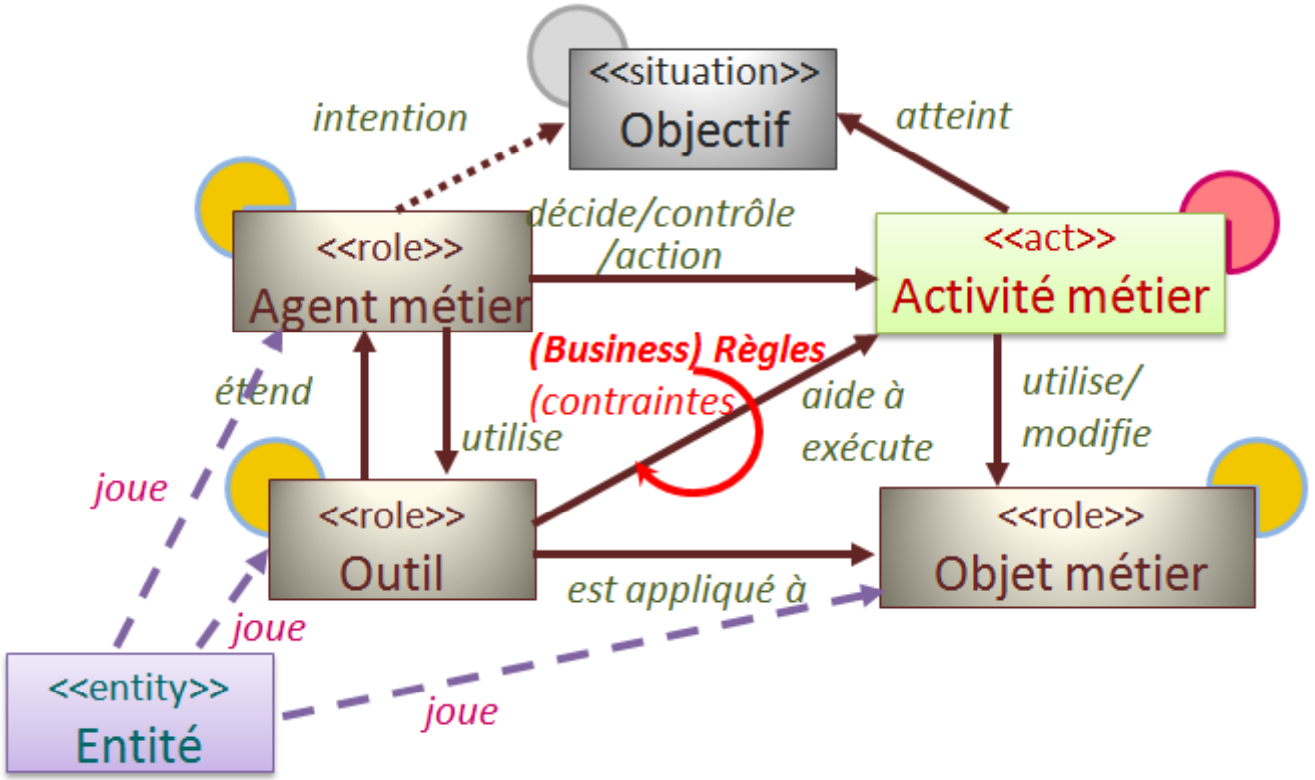
Un objet (ressource) métier se caractérise par un état représenté par une identité, un but (ou comportement), des attributs et des relations qu'il entretient avec les autres objets :

- L'identité ou identifiant. Cette identité le distingue des autres objets métiers qui peuvent être égaux sans toutefois être identiques. L'identité de l'objet métier est indépendante de son état.
- Le but ou le comportement de l'objet métier décrit les actions et les réactions lorsqu'il est soumis à une opération. Il désigne la manière dont ses relations et ses attributs changent lors des interactions avec les autres objets métier au sein, par exemple, de Processus. Par exemple, l'attribut Vitesse de l'objet Voiture augmente lorsque l'on accélère.
- Les attributs sont des propriétés fixes ou variables portées par l'objet. Ces propriétés peuvent être représentées par des valeurs dont la définition est intentionnelle ou extensionnelle en lien avec une table de valeurs possibles ou une table de nomenclature. Par exemple, l'objet Voiture possède l'attribut Couleur qui définit la couleur la carrosserie.
- Les relations avec les autres objets métiers portent une information complémentaire à l'objet métier. Par exemple, l'objet Voiture est caractérisé par une relation d'agrégation avec les objets Roues et Moteur.
- Le dictionnaire des objets métier de l'entreprise centralise toute cette terminologie métier.

UML extension for business

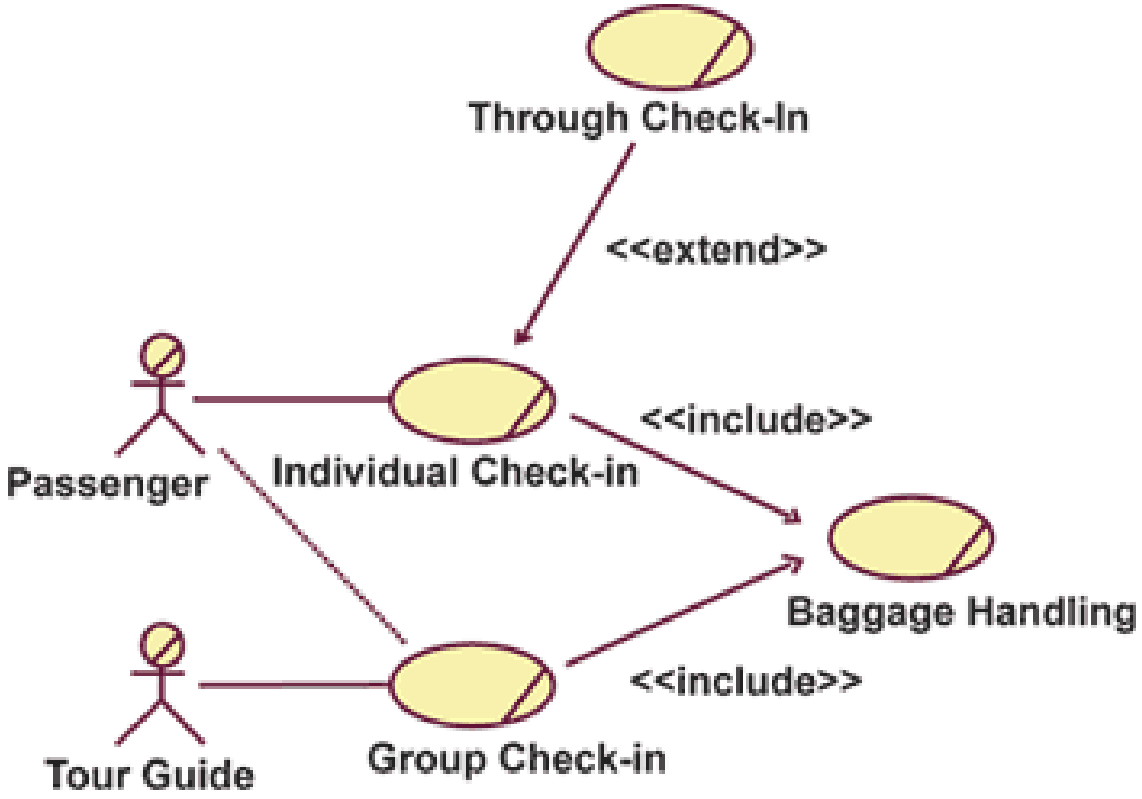


Documenter le métier



Cas d'utilisation métier

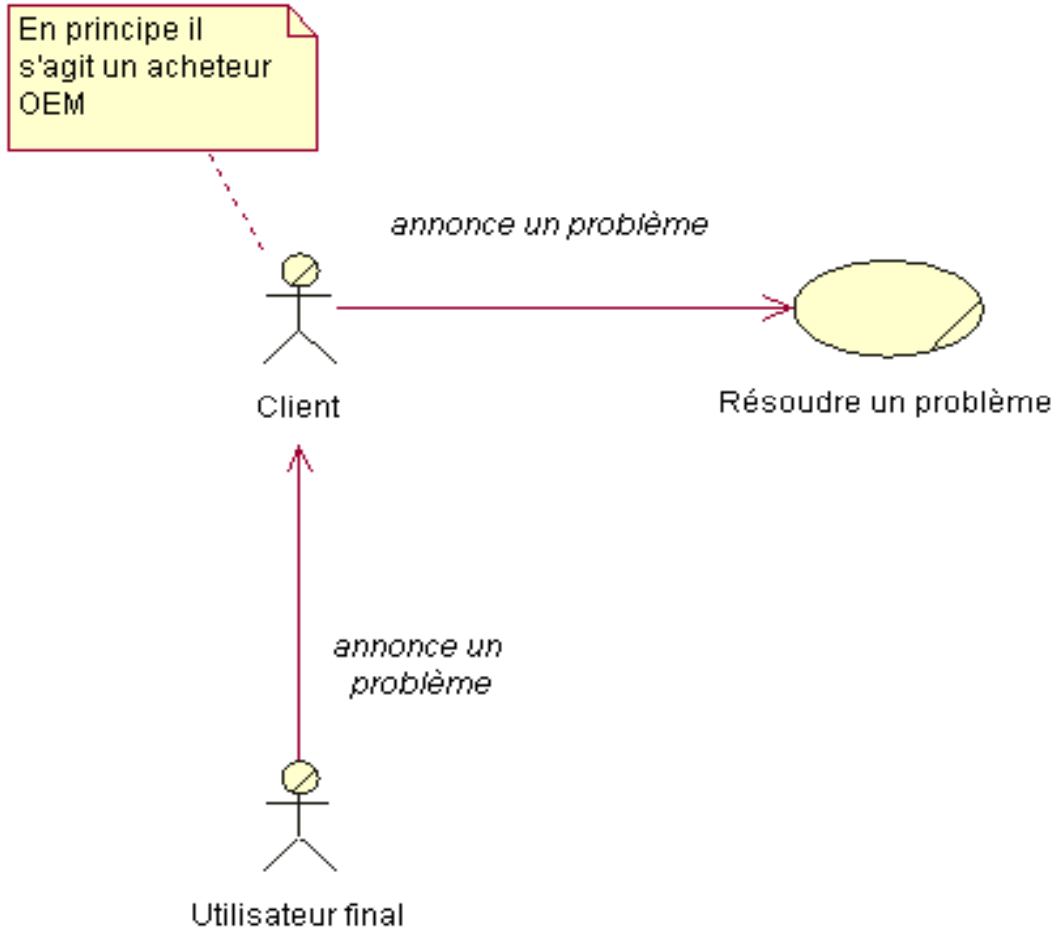
- Les cas d'utilisation métier décrivent des processus métier, c'est-à-dire une séquence d'actions qui fournissent une valeur observable à un acteur métier.
- Un cas d'utilisation documente une séquence particulière de travails métier – workflow ("individual check-in" or "group check-in").



- Le diagramme montre aussi « les acteurs métier » ("passenger" et "tour guide"). Ils « interactionnent » avec le business : principalement les clients, mais ils pourraient inclure aussi des fournisseurs, les partenaires, les clients potentiels (le "market place"), les autorités locales, ou les collègues en parties du business non modelées.

Acteurs métiers

- Deux acteurs métiers:
  - le **client** de notre fabricant de cartes de téléphonie, en principe, un acheteur OEM, comme DELL, HP ou IBM, qui installera les cartes sur les ordinateurs qu'il construit;
  - l'**utilisateur final** qui connaîtra effectivement un problème; il annoncera le problème à son fournisseur.

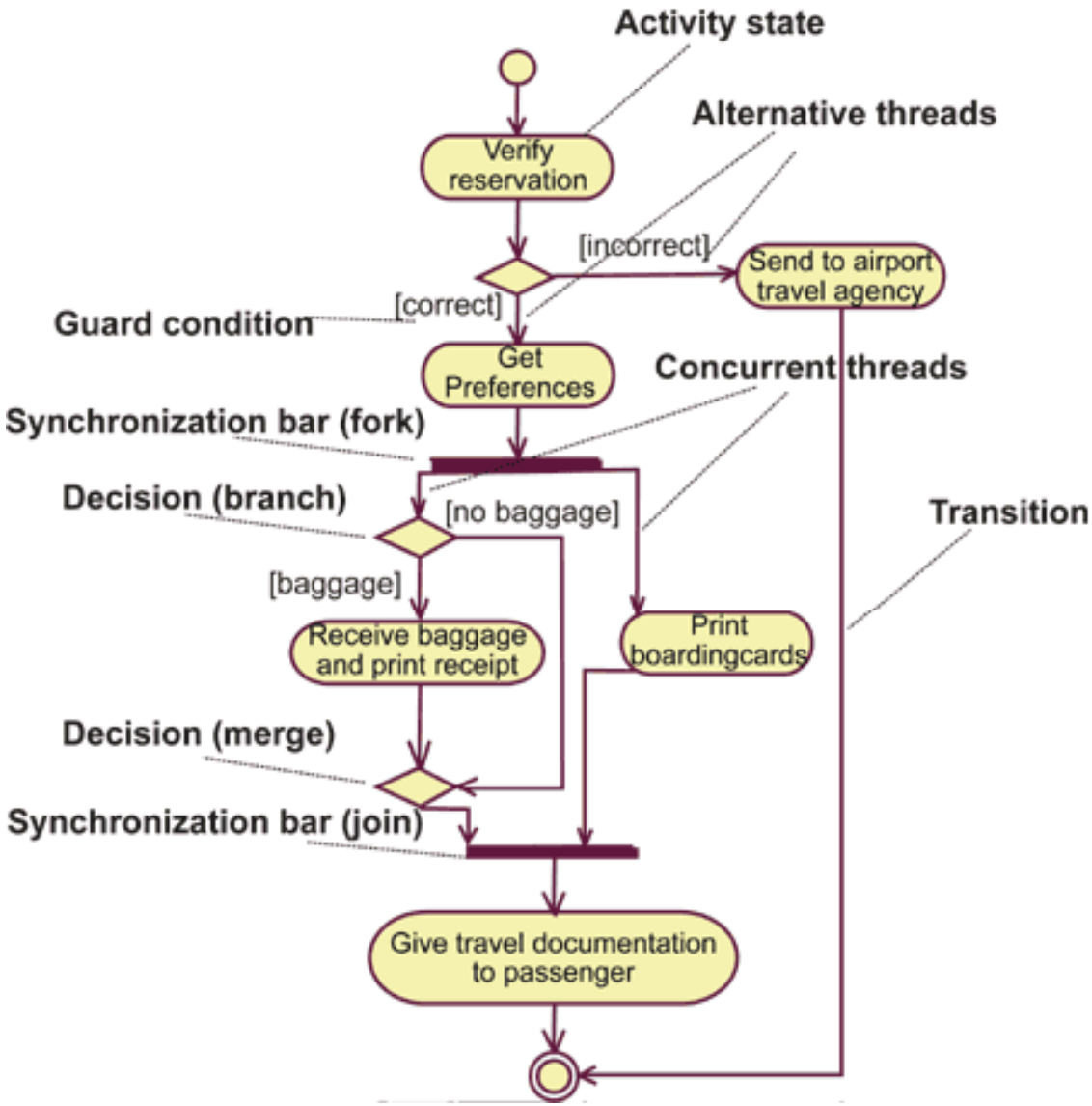


Comment documenter les cas d'utilisation métier?

- Le détail associé avec un cas d'utilisation métier est documenté dans une spécification de cas d'utilisation métier. Ceci inclura:
  - un texte ou des diagrammes d'activité UML et
  - les diagrammes des cas d'utilisation du système.
- Les articles suivants peuvent être inclus dans une spécification de cas d'utilisation métier:
  - Nom
  - Brève description
  - Objectifs d'exécution
  - Profit/Valeur
  - Séquence de travail / Flux d'événements
  - Besoins spéciaux,
  - Points d'extension
  - Relations

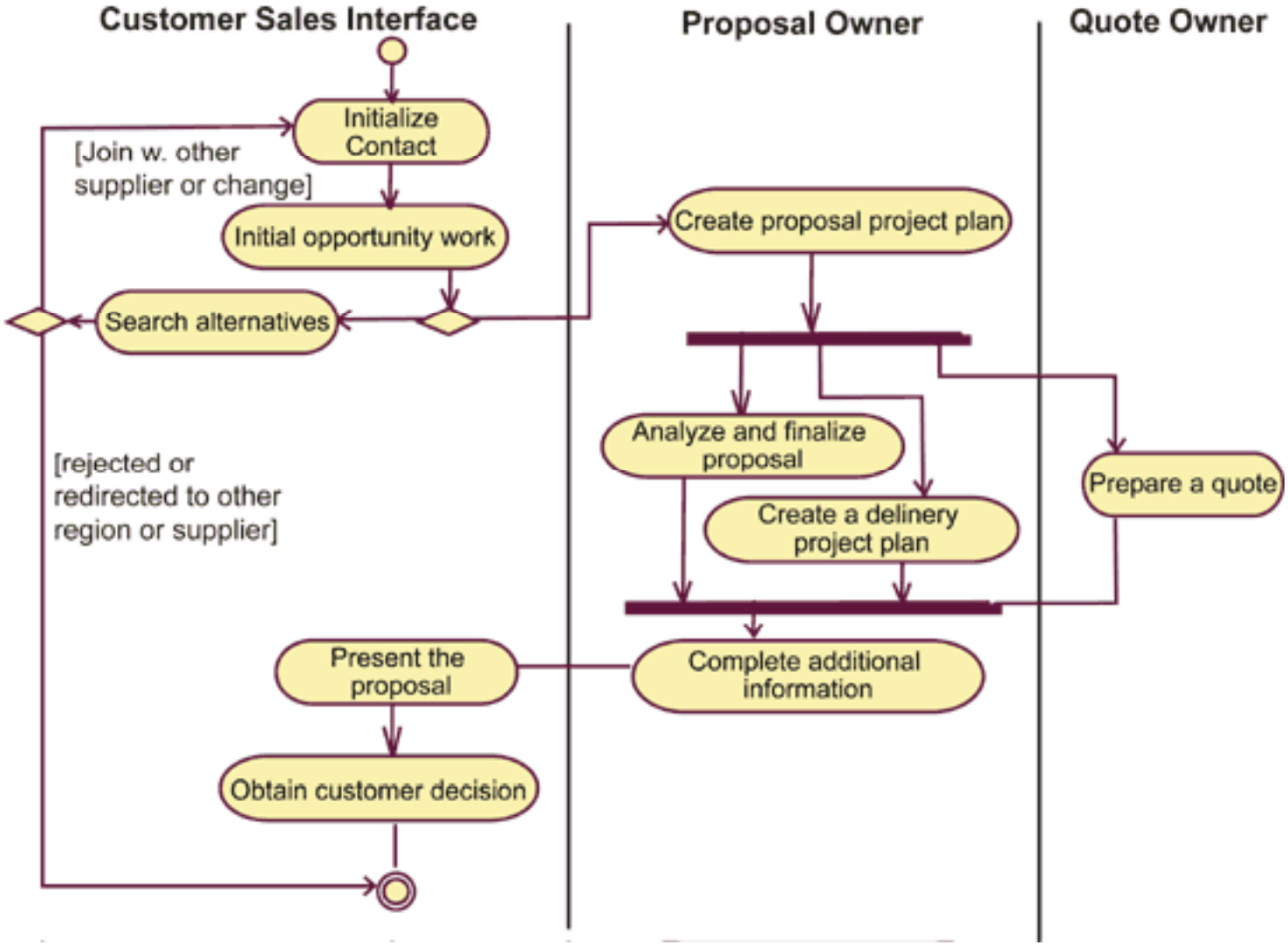
  The diagrams used in business use case documentation are *Use-Case Diagrams* and *Activity Diagrams*. The business object model (or business domain) can be added as a *Class Diagram*.
- Les diagrammes utilisés dans la documentation des cas d'utilisation métier sont: diagrammes des cas d'utilisation et diagrammes d'activité. Le modèle des objets métier (le domaine métier) peut être ajouté comme un Diagramme de Classes.

Diagramme d'activité décrivant une structure di Workflow



Swim Lanes

- Une variation du diagramme d'activité qui contient «swim lanes» ou colonnes, que décrivent les travailleurs métier qui exécute chaque activité.



Le modèle des objets métier (I)

Le modèle objet métier = Un diagramme de classe qui contient les acteurs métier, les travailleurs métier et les entités métier.

Le travailleur métier (business worker) = Acteur appartenant à l'entreprise dont le travail participe à la réalisation d'un cas d'utilisation
Exemples : commercial, responsable des ressources humaines.

L'entité (objet) métier (business entity) = un concept ou une abstraction ayant un sens pour des acteurs (partie prenante interne) d'une organisation (p.ex. entreprise). L'objet métier permet de décrire les entités manipulées par les acteurs dans le cadre de la description du métier.
Exemples : dossier client, catalogue fournisseur.

Acteur métier (business actor) = Acteur externe à l'entreprise ou au champ de l'étude. Personne ou système externe qui représente une interface avec le système.
Exemples : client, fournisseur, détecteur de présence, base de données de consolidation des projets européens.

Associations

- *entre un travailleur métier et une entité métier*: une consultation ou une mise à jour de l'entité métier par le travailleur métier.
- *entre entités métiers* : un lien sémantique sur lequel on peut fixer des multiplicités, ce lien peut être une généralisation, une agrégation, une composition ou une simple association.
- *entre un acteur métier et un travailleur métier, de même qu'entre les travailleurs métier* : une interaction ou une demande de service.

Le modèle des objets métier (II)

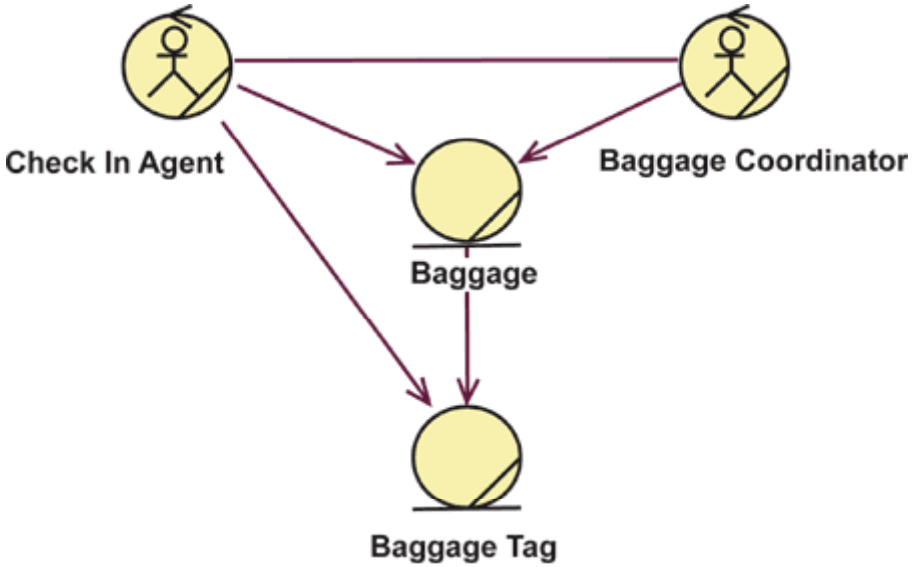
Composants

- Agents
- Fournisseurs et matérials
- Produits (intermediaires and finales)
- Rapports de transactions pour acquisition, ventes, maintenance, et livraison.

- Un modèle de domaine métier est un modèle d'objet métier qui se fixe sur « produit, livraisons, ou événements qui sont importants au domaine du business ». Le but de la modélisation de domaine est de fournir « l'image grossière » des rapports parmi les entités métier dans une organisation complexe.
- Le modèle de domaine montre typiquement les entités métier majeures, leurs responsabilités fonctionnelles et les relations entre les entités. Il fournit aussi une haute description des données que chaque entité fournit.

Le modèle des objets métier (III)

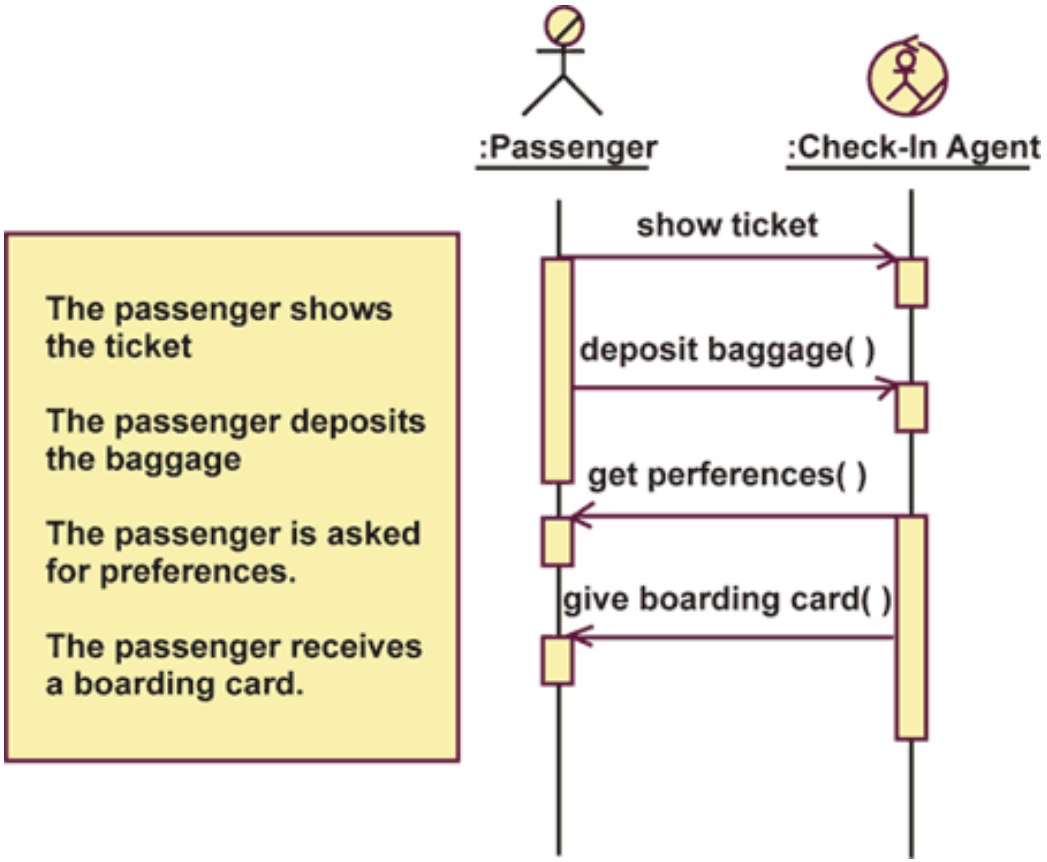
- L'objet métier permet de décrire les entités manipulées par les acteurs dans le cadre de la description du métier.
- The second part of a business model is the business object model.
- Tandis qu'un modèle de cas d'utilisation métier dit que un processus métier fera, un modèle des objets métiers dit comment il sera fait. Il sert come une abstraction de comment les travailleurs métiers et les entités métiers ont besoin d'être relaté et collaborer pour exécuter leurs business.
- Le modèle des objets métiers est un diagramme de classe métier montrant des travailleurs aux affaires et les entités d'affaires.



- Il indique comment les travailleurs relatent à chaque autre et aux « choses » dans le business.

Diagramme de séquence métier

- Un diagramme de séquence décrit graphiquement des détails de l'interaction parmi les travailleurs métier (business workers) et les acteurs métier, et aussi montre comment les entités d'affaires sont accédées, pendant l'exécution d'un cas d'utilisation métier. Dans un tel diagramme les entités (objets) métier échangent des messages.
- Diagramme de séquence métier= diagramme de séquence qui est utilisé pour montrer l'interaction d'un acteur métier avec le business .



Diagrammes d’activité

Il y a au moins trois lieux dans l'UP où les diagrammes d'activité fournissent un outil valable :

- cas d’utilisation métier,
- cas d‘utilisation de logiciel, et
- opérations des classes.

Diagrammes d'activité pour les cas d'utilisation

- Un cas d'utilisation modélise un objectif que le système doit atteindre pour être considéré réussi. Avec ce modèle il est plus facile révéler les éléments essentiels du domaine de problème dans une façon familière à l'utilisateur.
- C'est très utile à modeler les cas d'utilisation dans le contexte où ils sont ou seront utilisés (dans le business).
- En définissant des opérations, vous avez besoin de modeler la séquence de manipulation de données, un cycle, et la logique de décision pour la fonction.
- Utilisez le diagramme d'activité pour mener l'utilisateur par un workflow (déroulement des opérations) ou une procédure, notant les décisions faites et processus exécutés à chaque étape. La procédure peut incorporer beaucoup de cas d'utilisation ou seulement la partie d'un cas d'utilisation comme vous les avez modelés.

1. Un cas seul d'utilisation, ou seulement une partie d'un cas d'utilisation,
   Un diagramme d'Activité pour un cas d'utilisation explique comment l'acteur réagit réciproquement avec le système pour accomplir l'objectif du cas d'utilisation, y compris les règles, les informations échangé, les décisions prises, et les produits de travail créés.
2. Beaucoup de cas d'usage liés ensemble pour créer un workflow.
   Un diagramme d'activité au niveau d'un workflow représente l'ordre et les conditions sous lesquelles les cas d'utilisation sont exécutés en série.

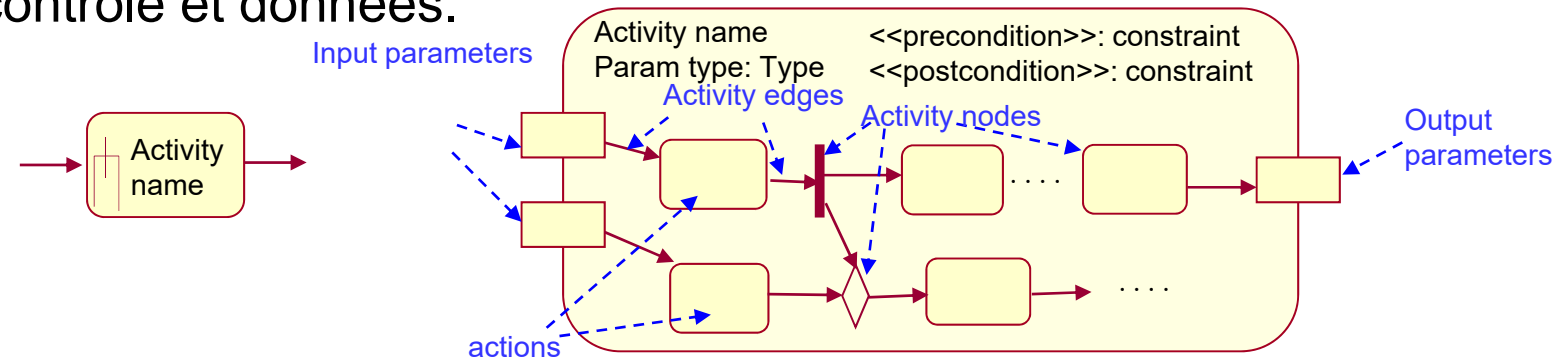
- Créant un tel modèle révélera probablement les éléments essentiels du domaine de problème dans une façon familière aux utilisateurs.

Diagrammes d'activité pour définir une méthode

- Le diagramme d‘activité peut être aussi utilisé pour modeler l'implémentation de méthodes complexes.
- En spécifiant l'implémentation pour une opération, vous avez besoin de modeler la séquence de manipulation et transformation des données, cycles et décisions. La modélisation des fonctions complexes peuvent empêcher apparition des problèmes plus tard quand vous écrierez le code en révélant toutes les besoins explicitement dans le diagramme.
- Pas toutes les méthodes sono assez compliquées pour mériter le dessin un diagramme d‘activité.
- Utiliser les diagrammes d‘activité seulement pour méthodes non insignifiantes!

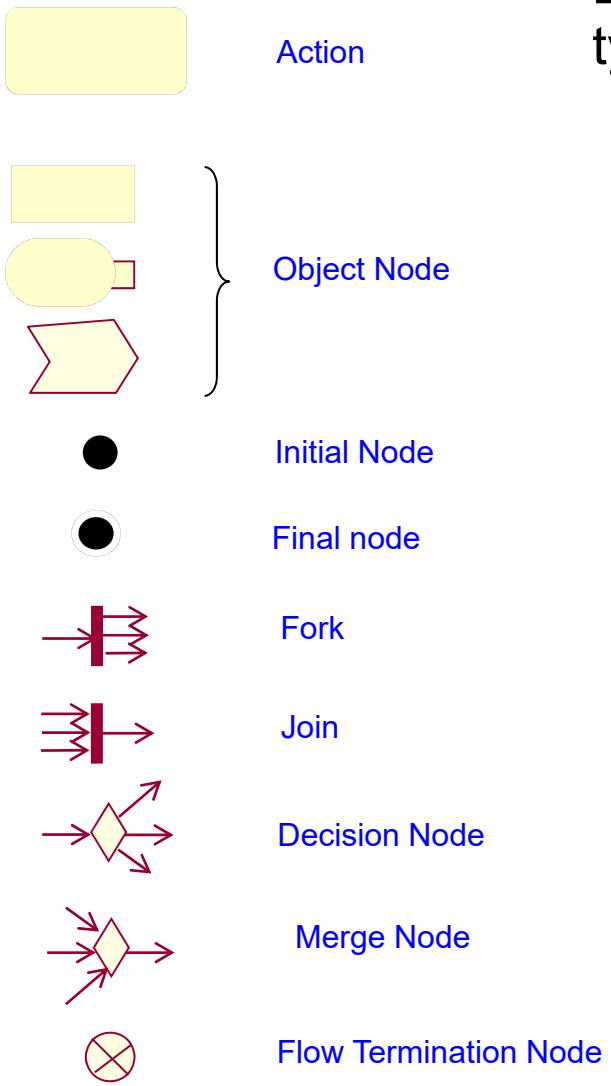
Activité UML

- Activité = Un pas dans un processus métier où nous devons fair quelque travail. Ce pourrait être un calcul, trouver quelques données, manipuler des informations, ou vérifier les données.
- Une activité dans UML est la spécification de comportement paramétrée comme le séquence coordonné d'unités subordonnées dont les éléments individuels sont des actions. Une activité spécifie la coordination d'exécutions de comportements subordonnés, utilisant un modèle de flux de contrôle et données.

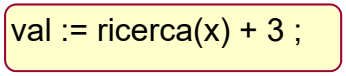


- Le flux de contrôle est modelé comme un graph ayant comme nœuds d'activité connectés par les arches d'activité représentant des transitions. Un nœud d'action peut représenter l'exécution d'un comportement subordonné :
  - un calcul arithmétique,
  - un appel à une opération,
  - une manipulation d'oppose des contenus.
- Les nœuds d'activité incluent aussi opérateurs de flux de contrôle, comme synchronisation, décision, et contrôle de la simultanéité.

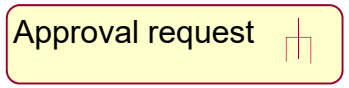
Nœuds d'activité



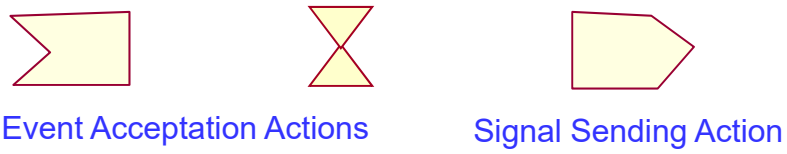
- Les activités peuvent contenir des actions de divers types :
  - fonctions primitives, par éxemple fonctions arithmétiques.



  - Les appels de comportement, comme activités.



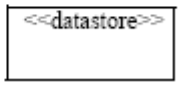
  - Les actions de communication, comme l'envoi de signaux.

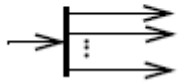
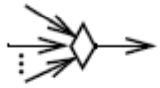
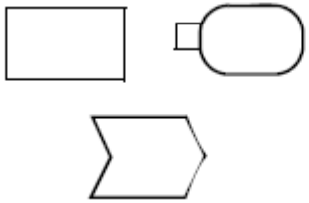
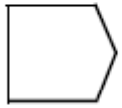


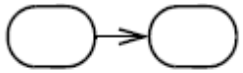
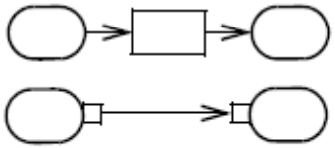
  - Manipulations d'objets : lecture ou d'écriture des attributs ou les associations.



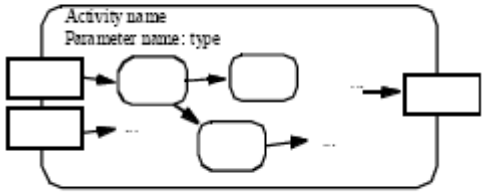
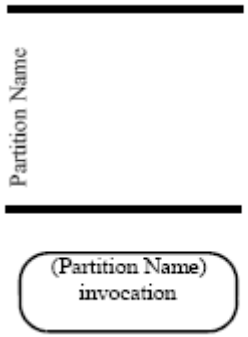
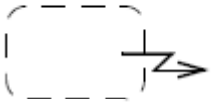
Nœuds graphiques et routes en diagrammes d'activité

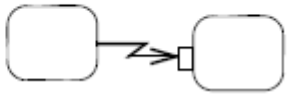
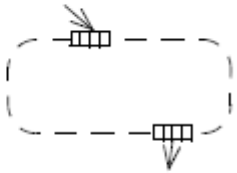
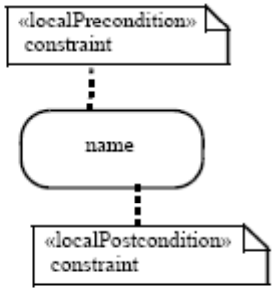
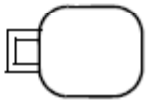
Node Type	Notation
AcceptEventAction	
Action	
ActivityFinal	
ActivityNode	*See ExecutableNode, ControlNode, and ObjectNode.*
ControlNode	*See DecisionNode, FinalNode, ForkNode, InitialNode, JoinNode, and MergeNode.*
DataStore	
DecisionNode	
FinalNode	*See ActivityFinal and FlowFinal.*
FlowFinal	

Node Type	Notation
ForkNode	
InitialNode	
JoinNode	
MergeNode	
ObjectNode	
SendSignalAction	

Path Type	
ControlFlow	
ObjectFlow	

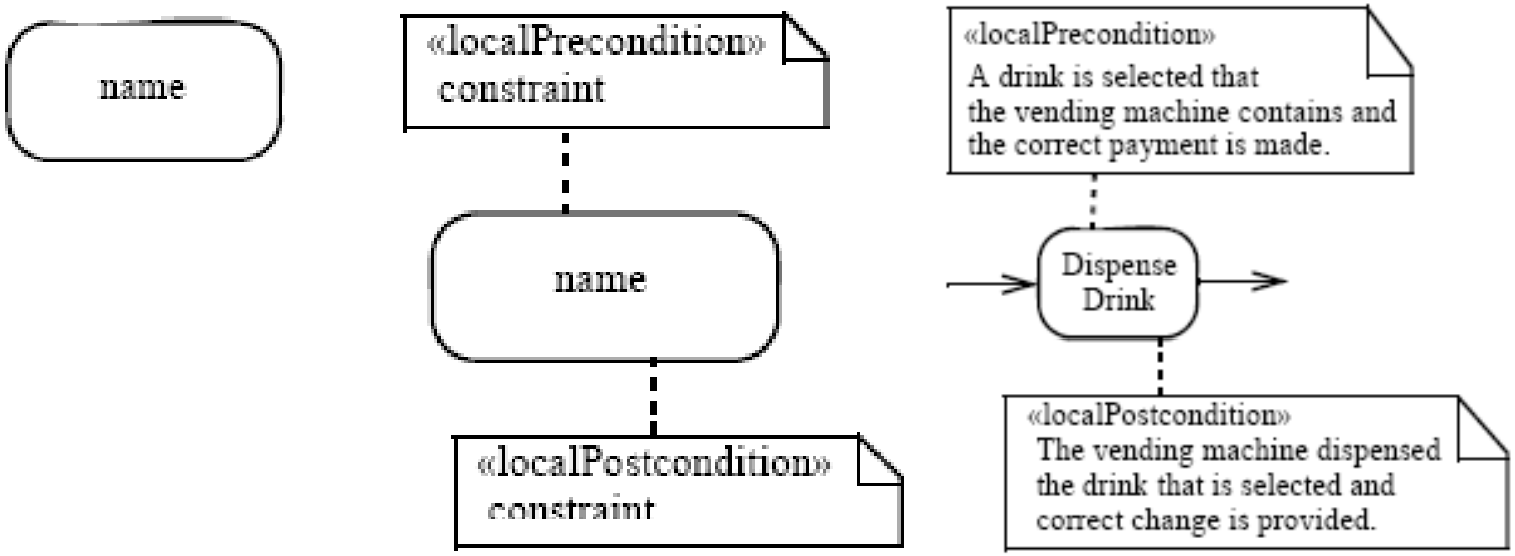
Eléments graphiques comme récipients en diagrammes d'activité

Type	Notation
Activity	
ActivityPartition	
InterruptibleActivityRegion	

Type	Notation
ExceptionHandler	
ExpansionRegion	
Local pre- and postconditions.	
ParameterSet	

Actions

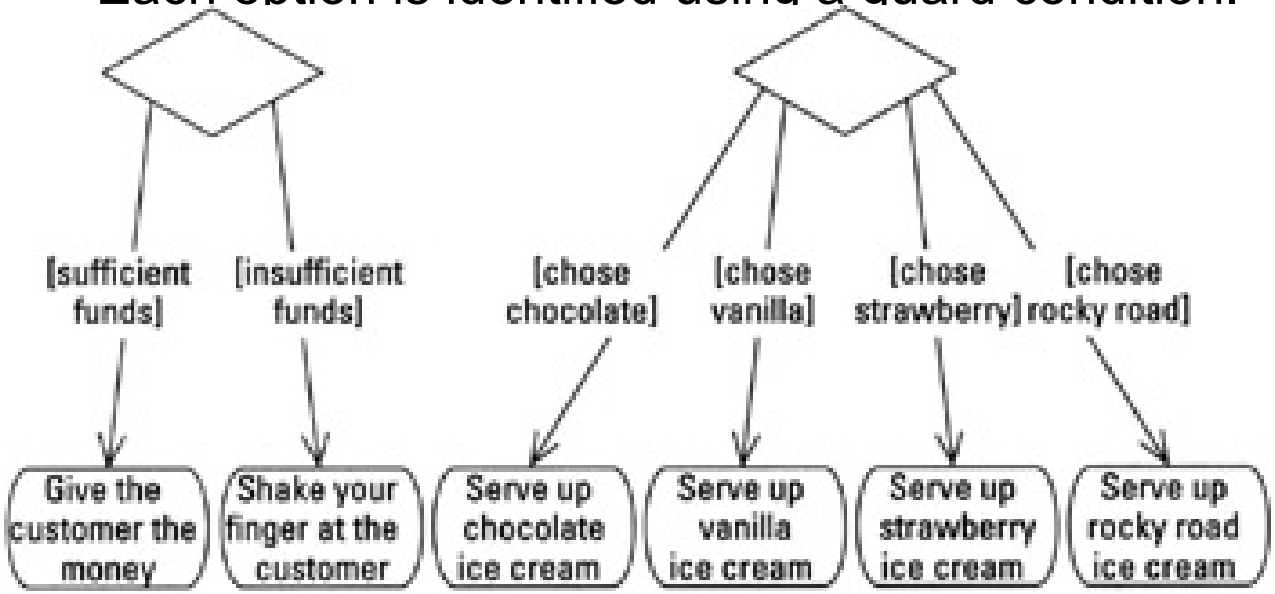
- Une action représente un pas seule dans une activité, c'est-à-dire, un opération qui n'est pas plus décomposé dans l'activité.
- Une action est donc simple du point de vue de l'activité le contenant, mais peut être complexe dans son effet et peut être non atomique.

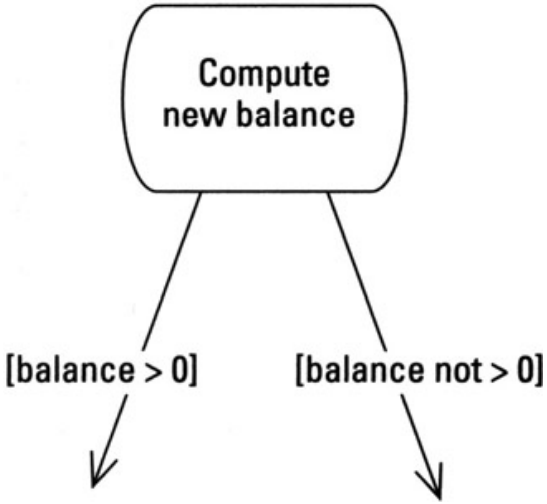


- Une action est un seul élément discret dans un diagrammes d'activité; comme une spécification d'un comportement que doit être exécutée, il peut invoquer un comportement qui est arbitrairement complexe.
- En conséquence :
  - une activité définit un comportement sera employé dans beaucoup de lieux,
  - une instance d'une action est utilisé seulement une fois à un point particulier dans une activité.

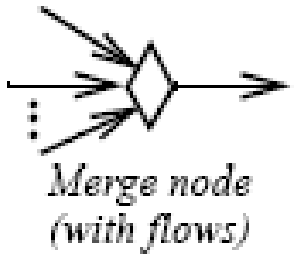
Decisions

- The Activity diagram diamond is a decision icon, just as it is in flowcharts The decision may be as simple as a true/false test. In other cases, the decision may involve a choice among a set of options.
- Each option is identified using a guard condition.

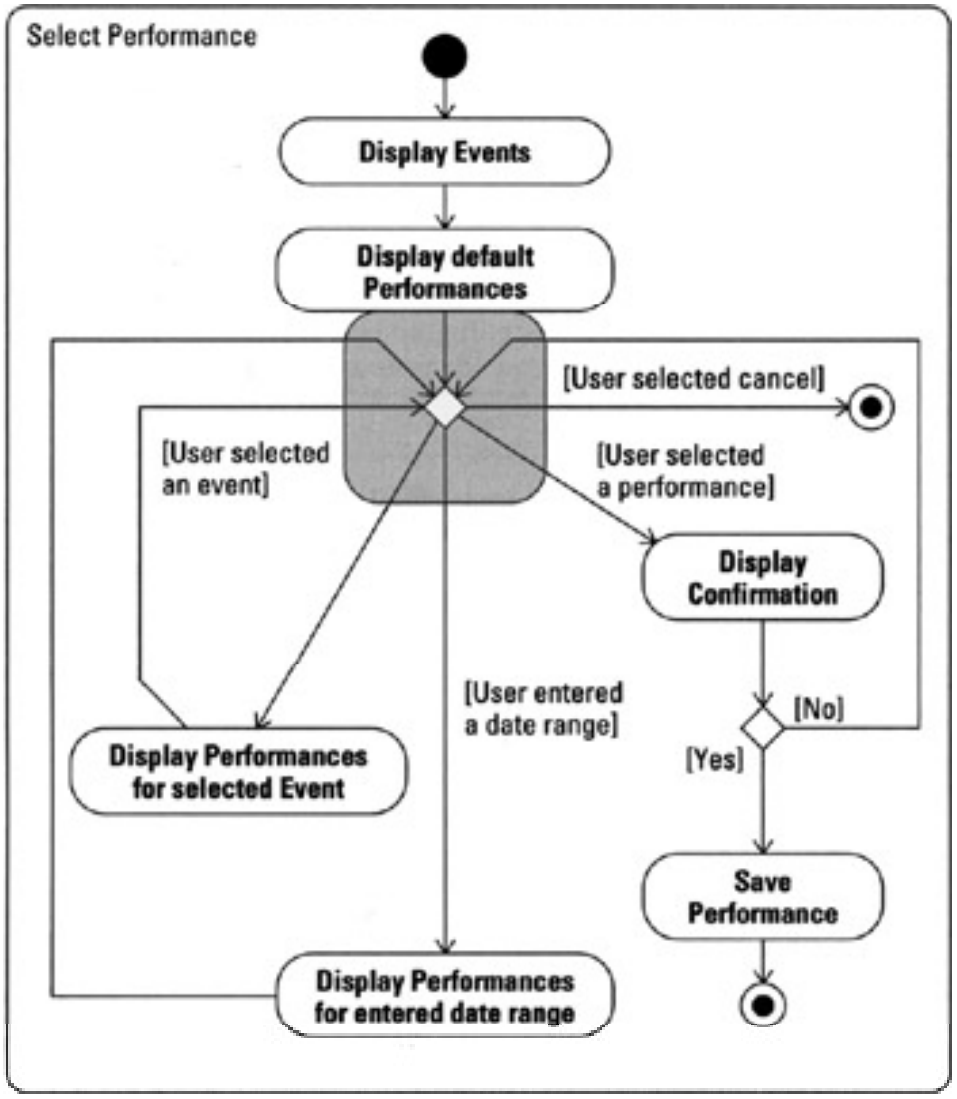


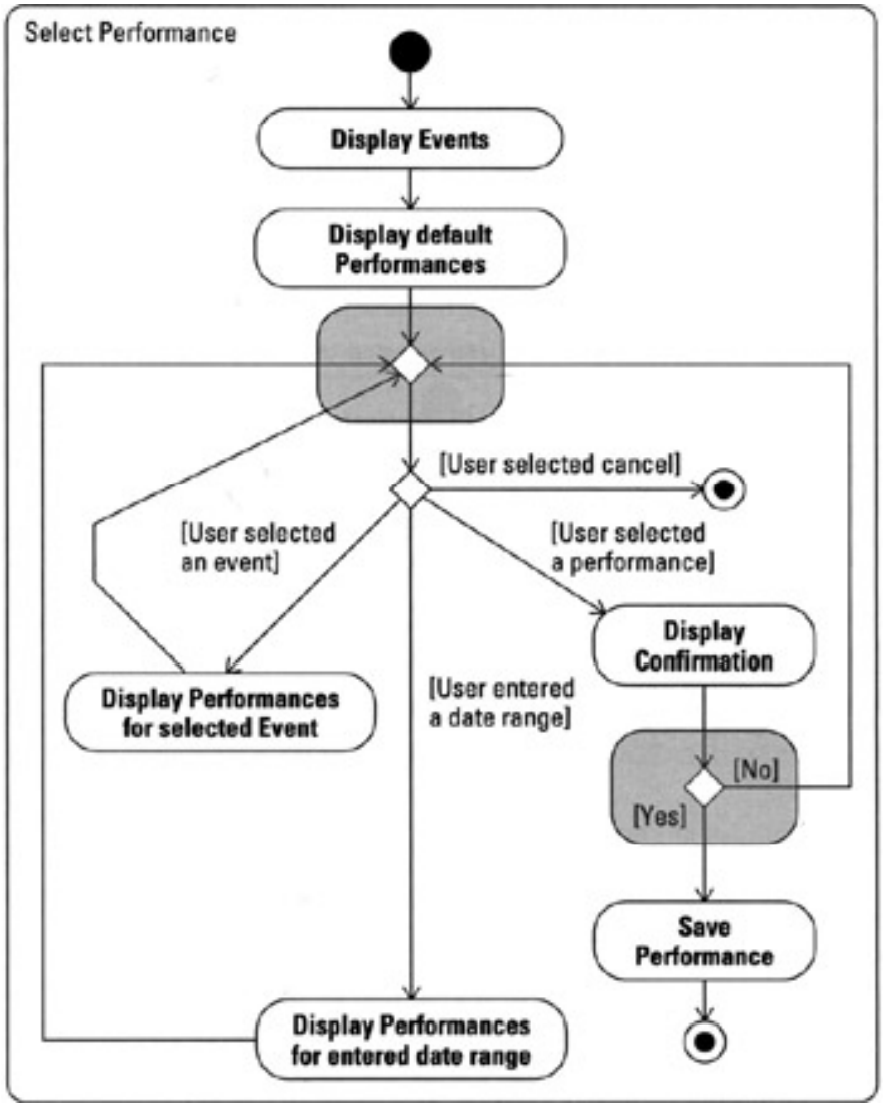


- A merge node is a control node that brings together multiple alternate flows. It is not used to synchronize concurrent flows but to accept one among several alternate flows.



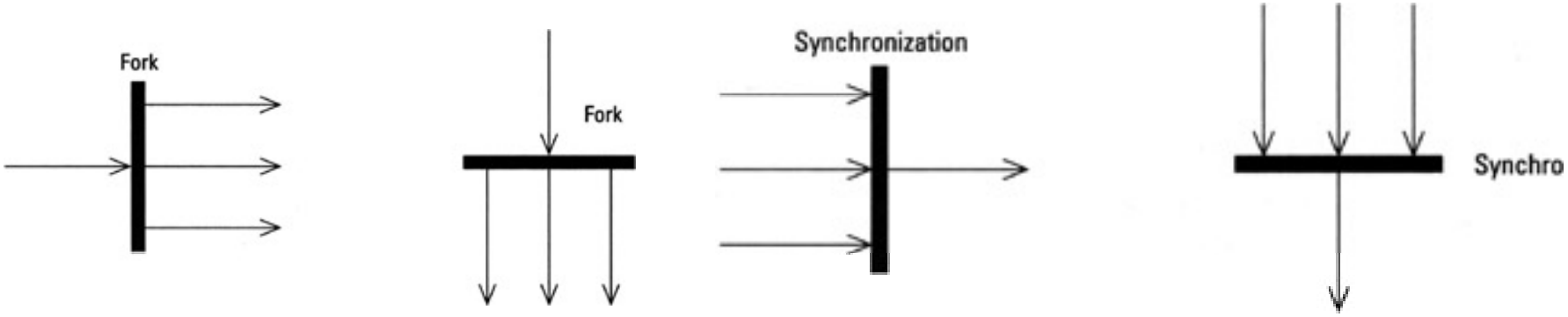
Decision and Merge points



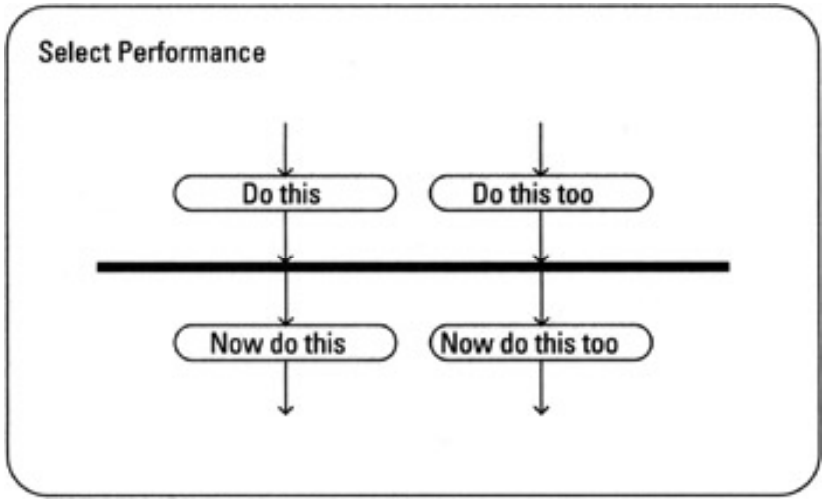


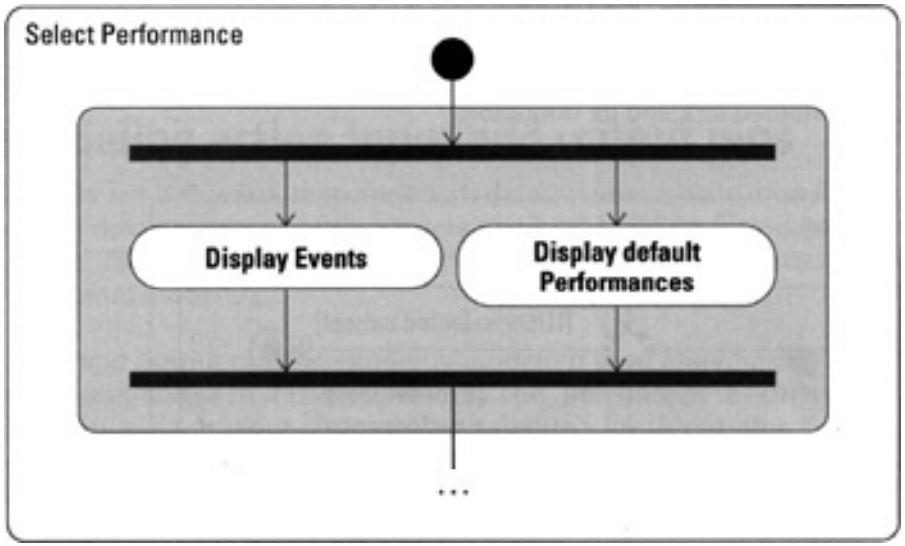
Concurrency

- Multiple concurrent threads or processes with *fork*, or *split of control.*



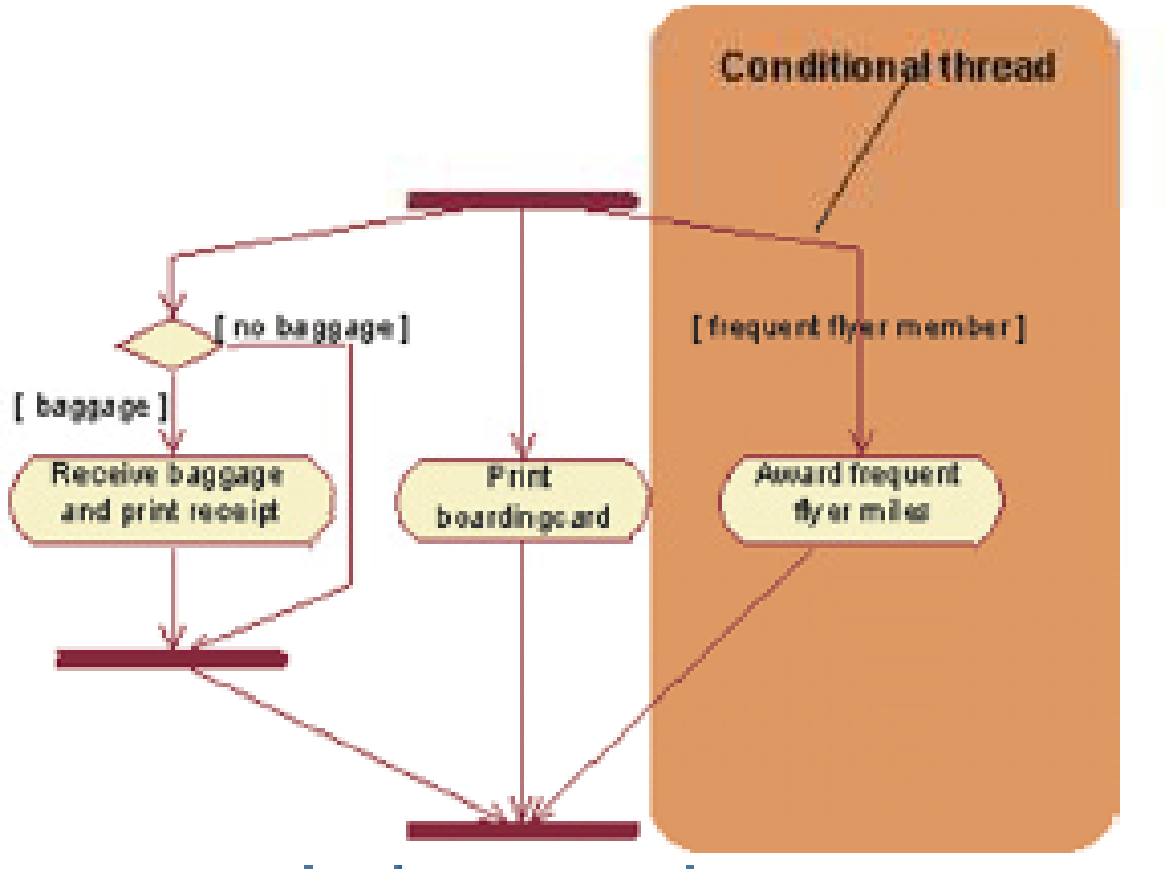
- Synchronization or merging of the concurrent threads or processes : *synchronization bar* .



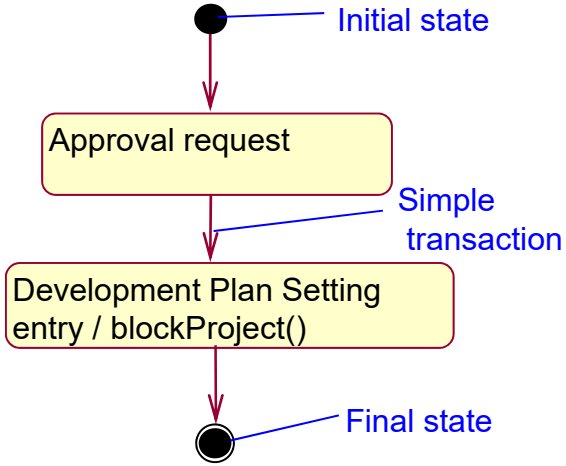


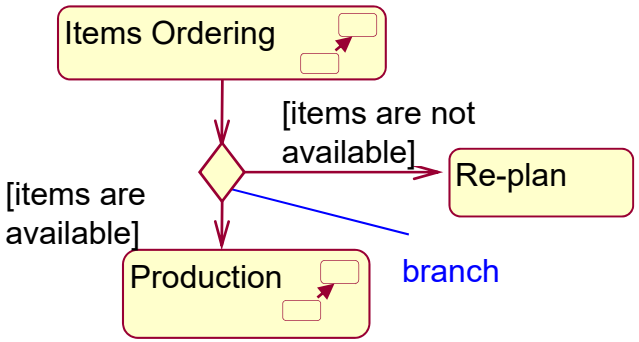
The individual check-in example

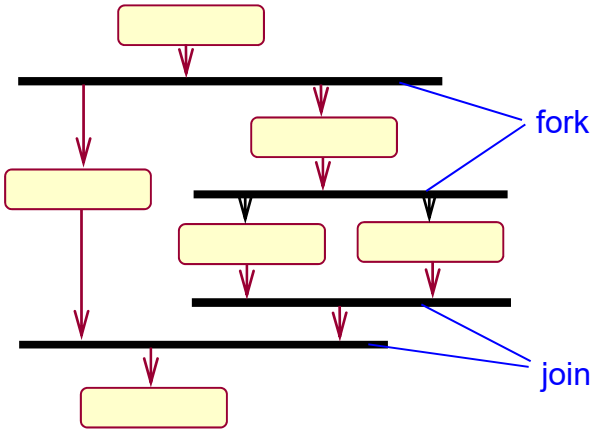
- The passenger checking in might be a frequent flyer member. In that case, you need to award the passenger frequent flyer miles.



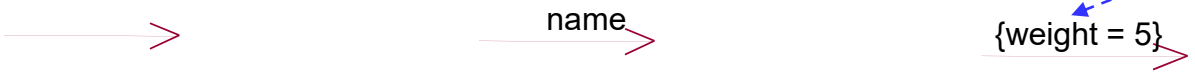
Activity Edges







Connections between two activity nodes.



The transactions are fired by occurrence of some events:

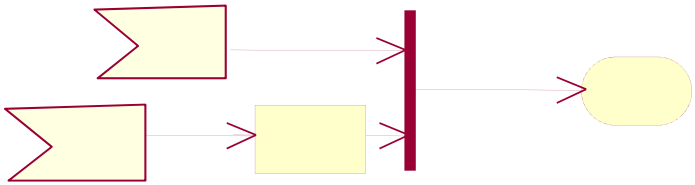
- previous action achievement:



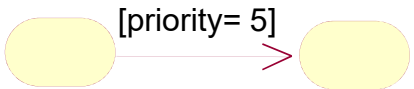
- availability of an object in a given state:



- signal occurrence:

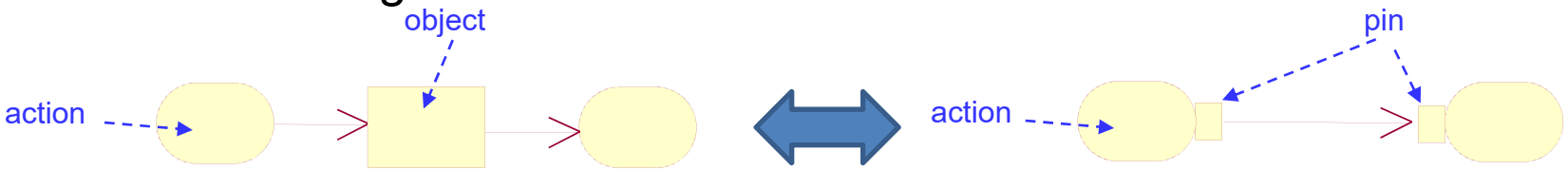


- checked condition:

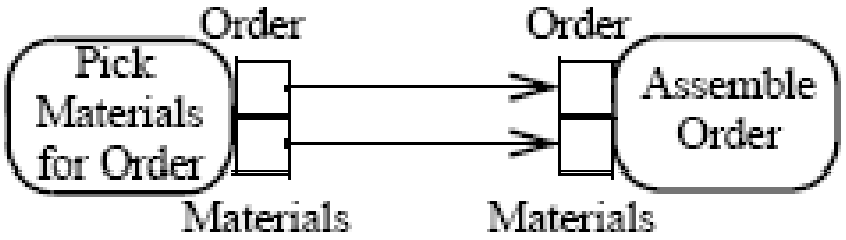


Modeling action input and output pins

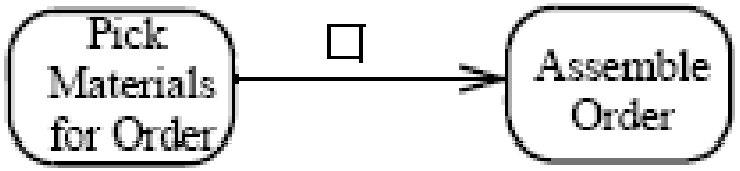
- Input and output for data processing may be explicitly modeled using *Input pins* and *Output pins.* Pins are object nodes that contain values. Each input pin receives a single value.



- Input pins correspond to the parameters passed to an invocation action.
- Output pins represent values produced as the result of an action. The values are passed along an object flow to subsequent actions.
- An object flow is an activity edge that can have objects or data passing along it. An object flow models the flow of values to or from object nodes.
- Object flows add support for multicast/receive, token selection from object nodes, and transformation of tokens. The pins can be elided, too.



With explicit pins



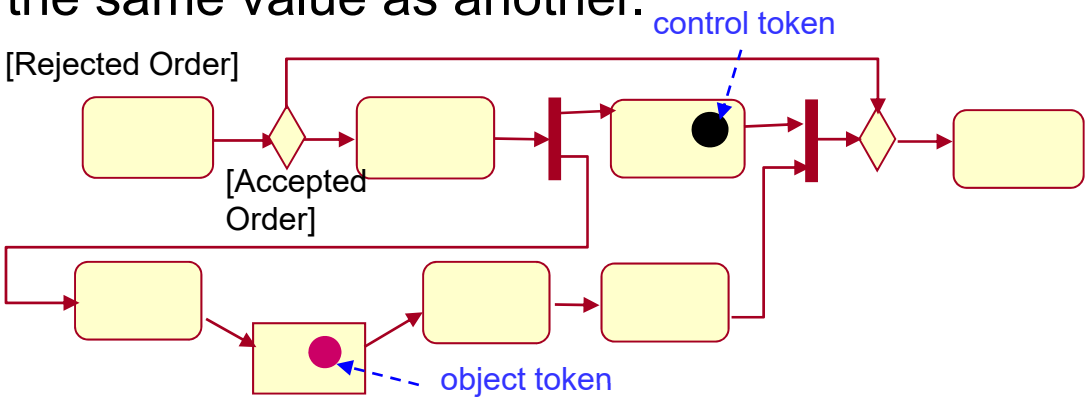
With elided pins

Action Semantics

- The semantics of activities is based on token flow.
- Flow = the execution of one node affects, and is affected by, the execution of other nodes.
- A token contains an object, datum, or a locus of control, and is present in the activity diagram at a particular node. Each token is distinct from any other, even if it contains the same value as another.

Two token types:

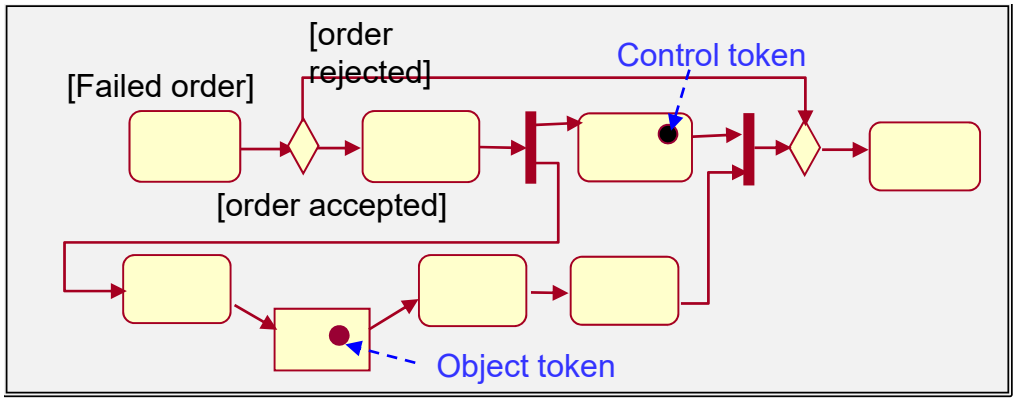
1. Control token
2. Object token



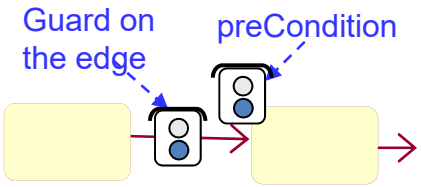
Nodes and edges have token flow rules:

- Nodes control when tokens enter or leave them. Source node can only offer tokens to the outgoing edges, rather than force them along the edge. Multiple tokens offered to an edge at once is the same as if they were offered one at a time. The same token can be offered to multiple targets.
- Edges have rules about when a token may be taken from the source node and moved to the target node. A token traverses an edge when it satisfies the rules for target node, edge, and source node all at once.

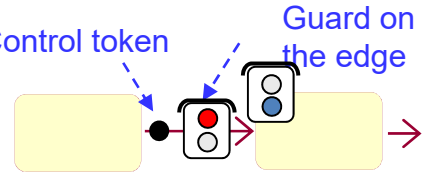
Semantics of activities



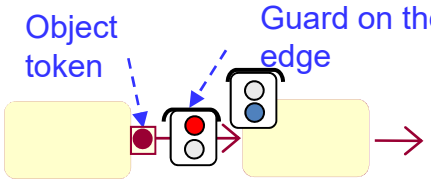
The tokens are blocked:



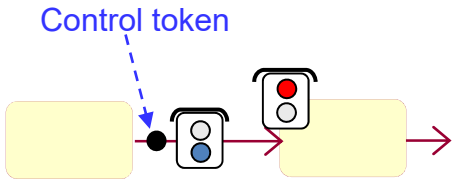
1)



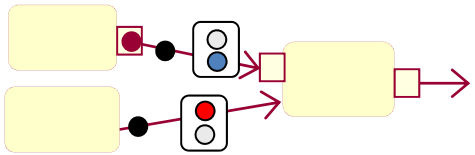
2)



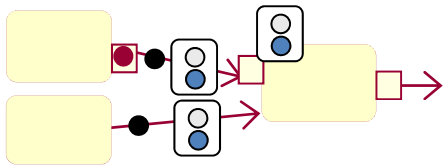
3)



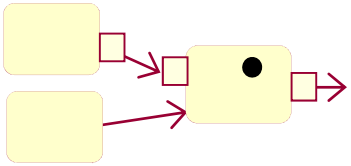
4)



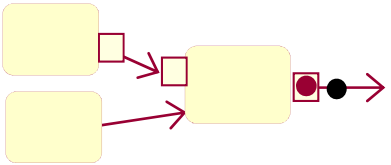
Only one condition is verified



2a. The control and object tokens contained in the previous actions, both input conditions are true, the precondition is also true



2b. The control token flows in the next action and remains there during the action execution



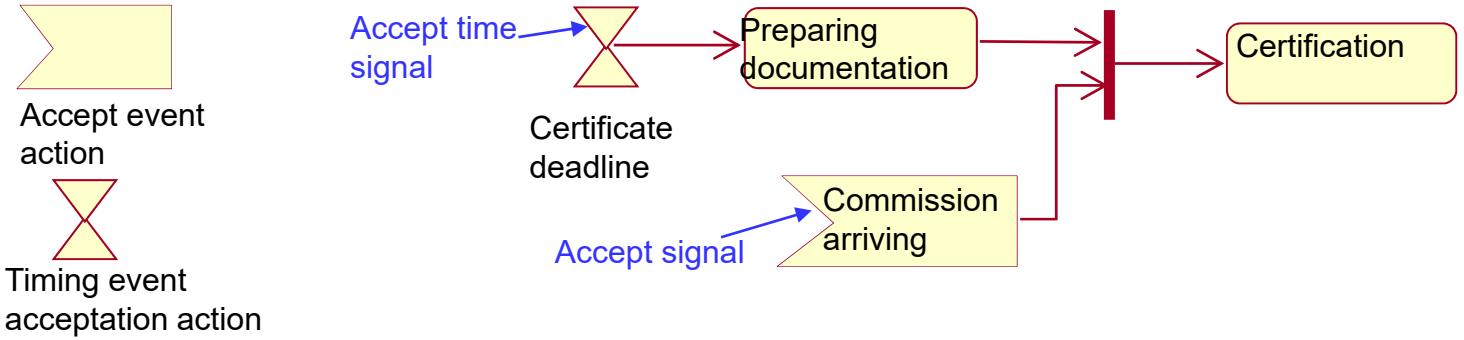
2c.The action was executed and the tokens are moving towards output

1. The tokens are blocked!

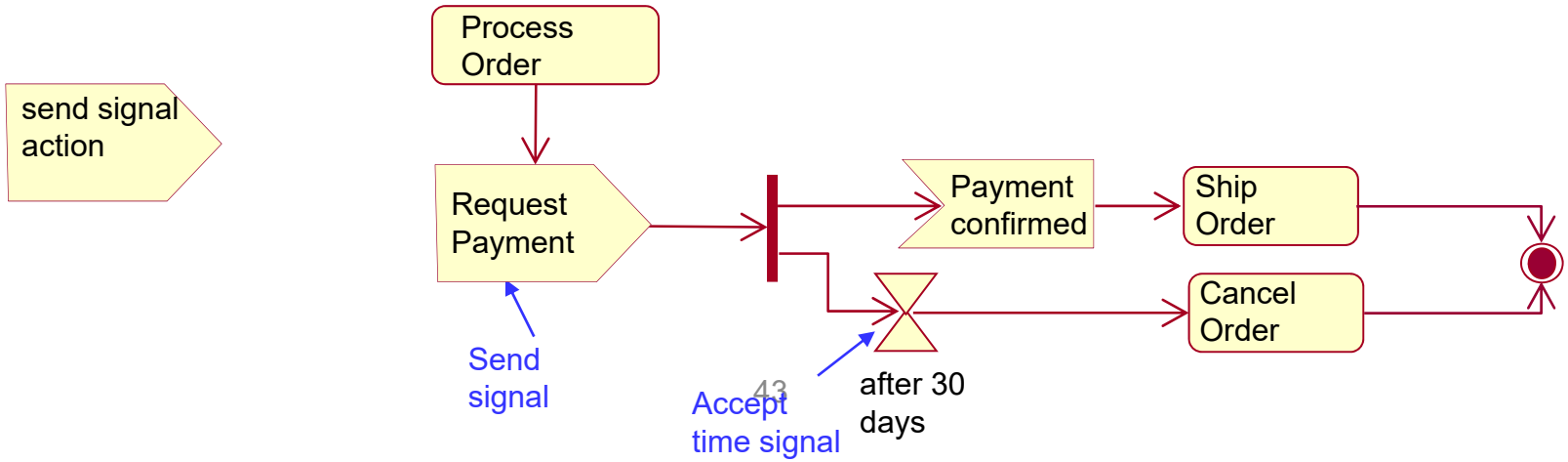
2. The tokens flow!

Event-Based Actions

Accept event actions

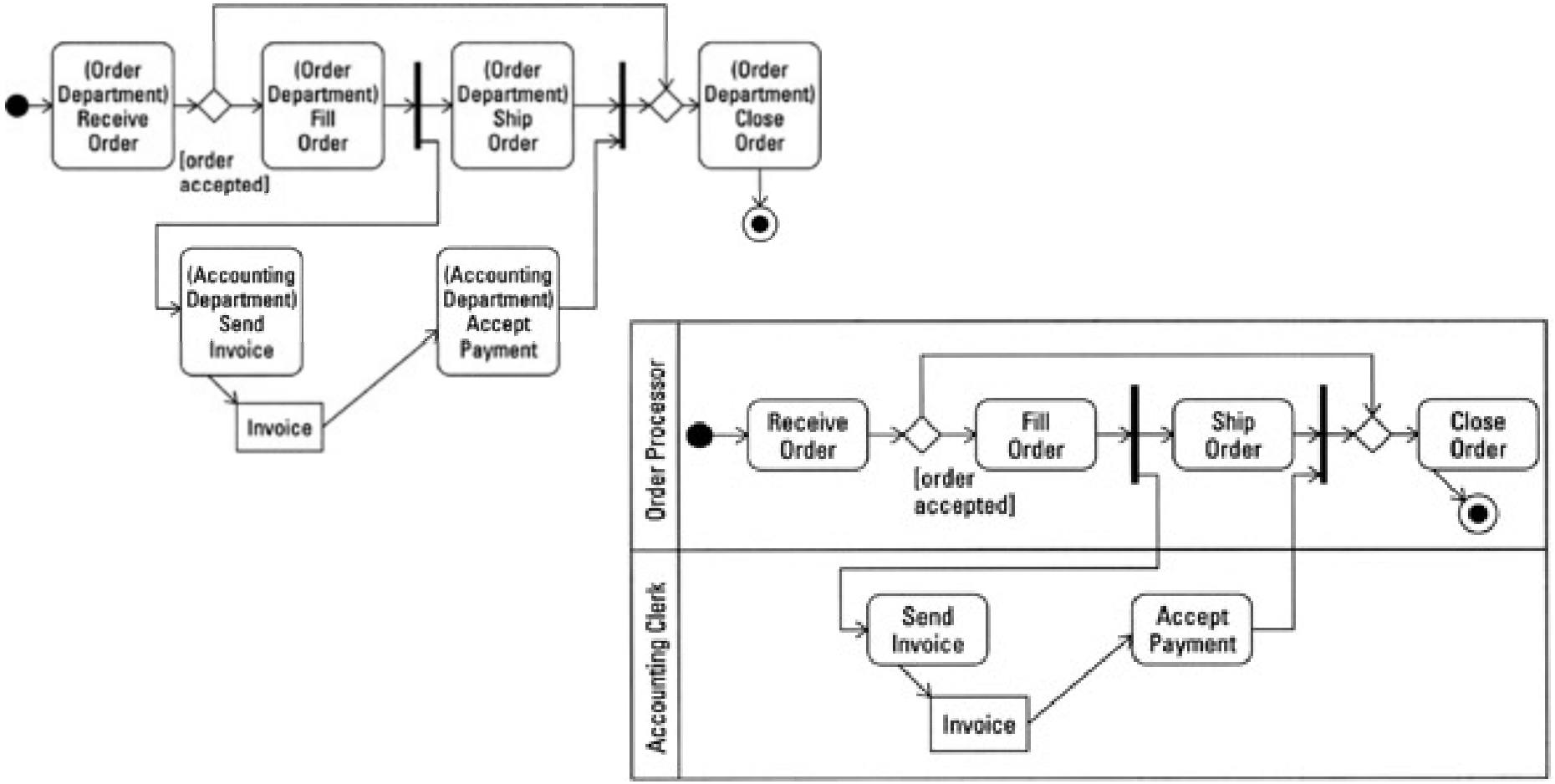


Send signal actions



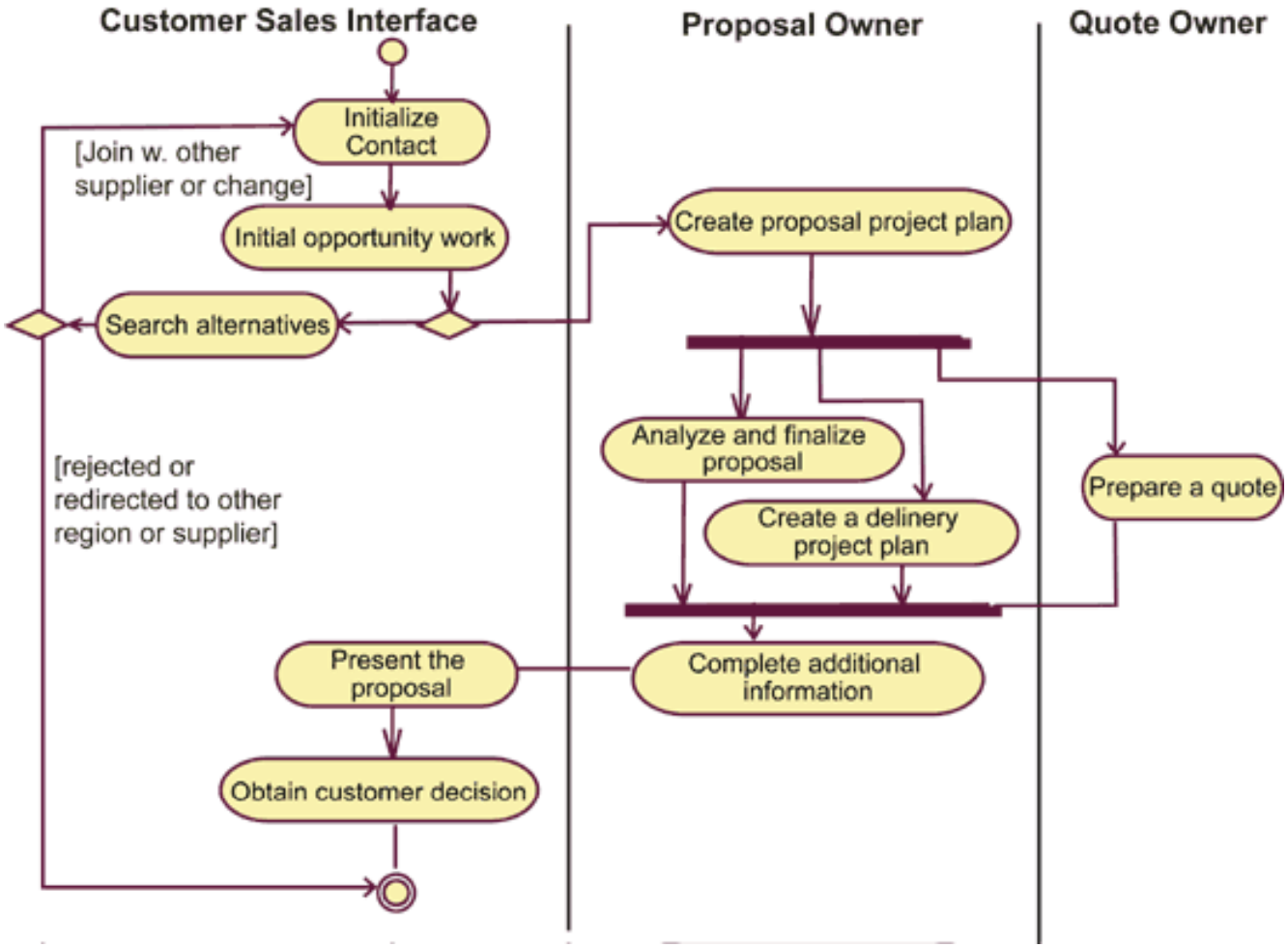
Activity Partition

- An ActivityPartition provides an organizational view of nodes and edges within an activity, much like packages provide an organizational view of classes and associations within a system.
- Each activity partition models part of an activity and *represents* the participation of some classifier.

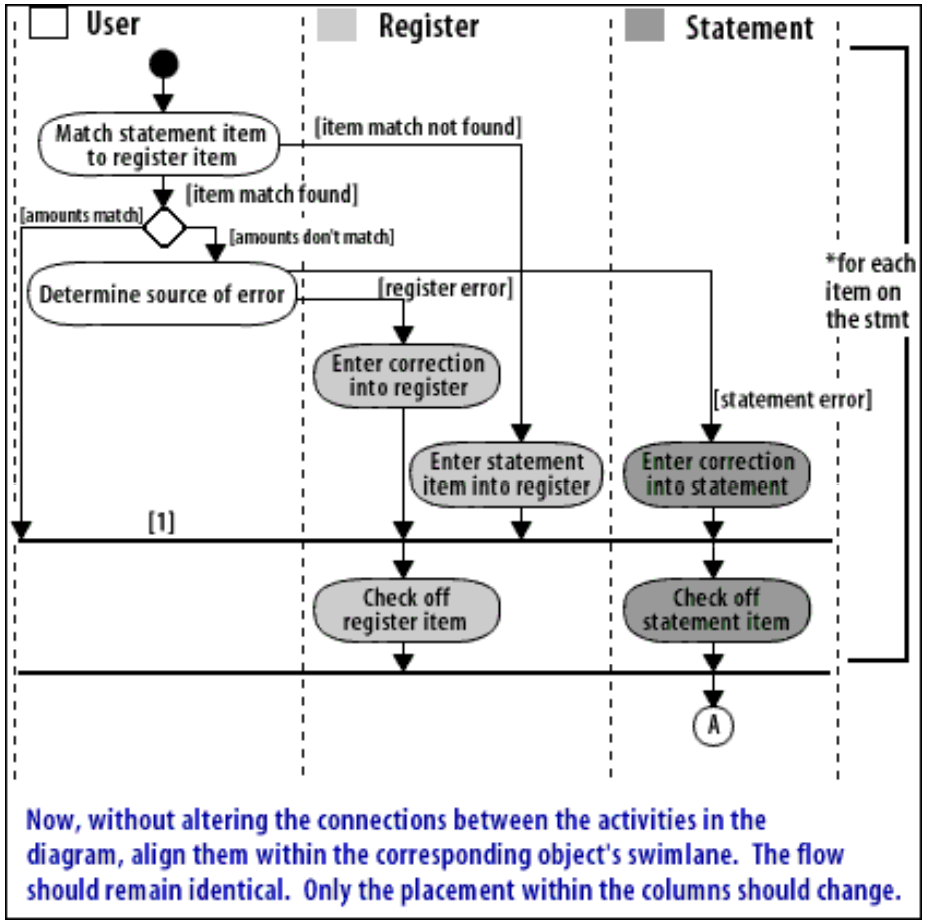


Swim Lanes

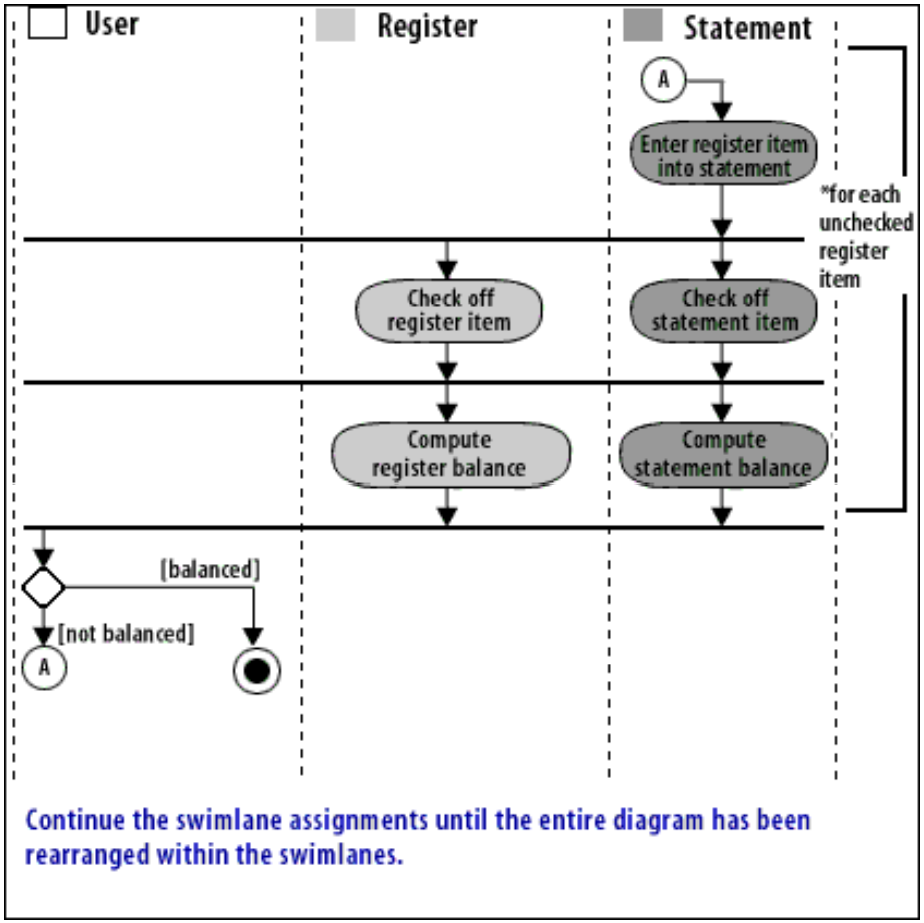
- Another type of diagram used in a business object model is a variation of the activity diagram that contains "swim lanes," or columns, that show which business worker is performing each activity.



Move activities in swimlanes



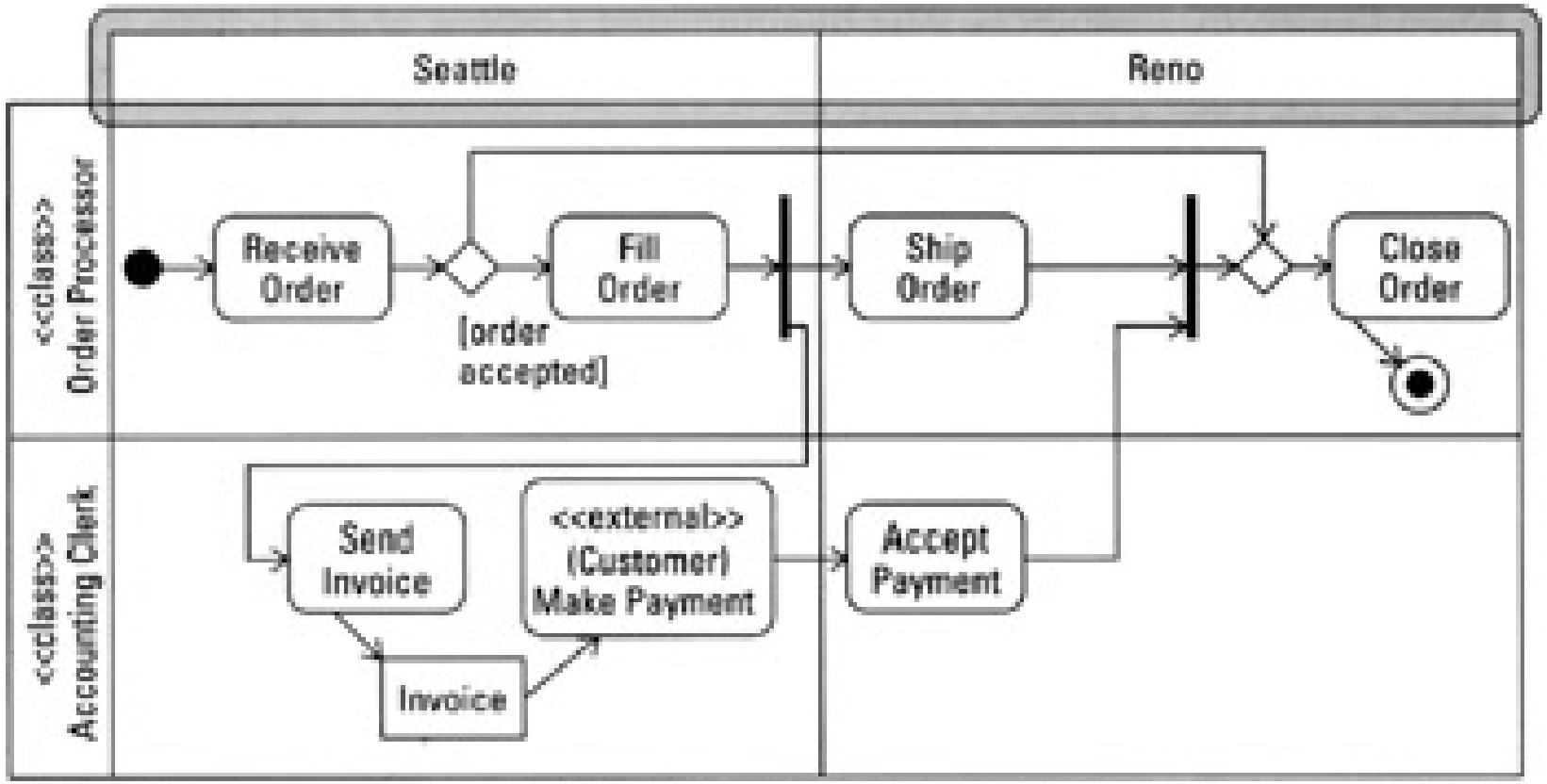
Now, without altering the connections between the activities in the diagram, align them within the corresponding object's swimlane. The flow should remain identical. Only the placement within the columns should change.



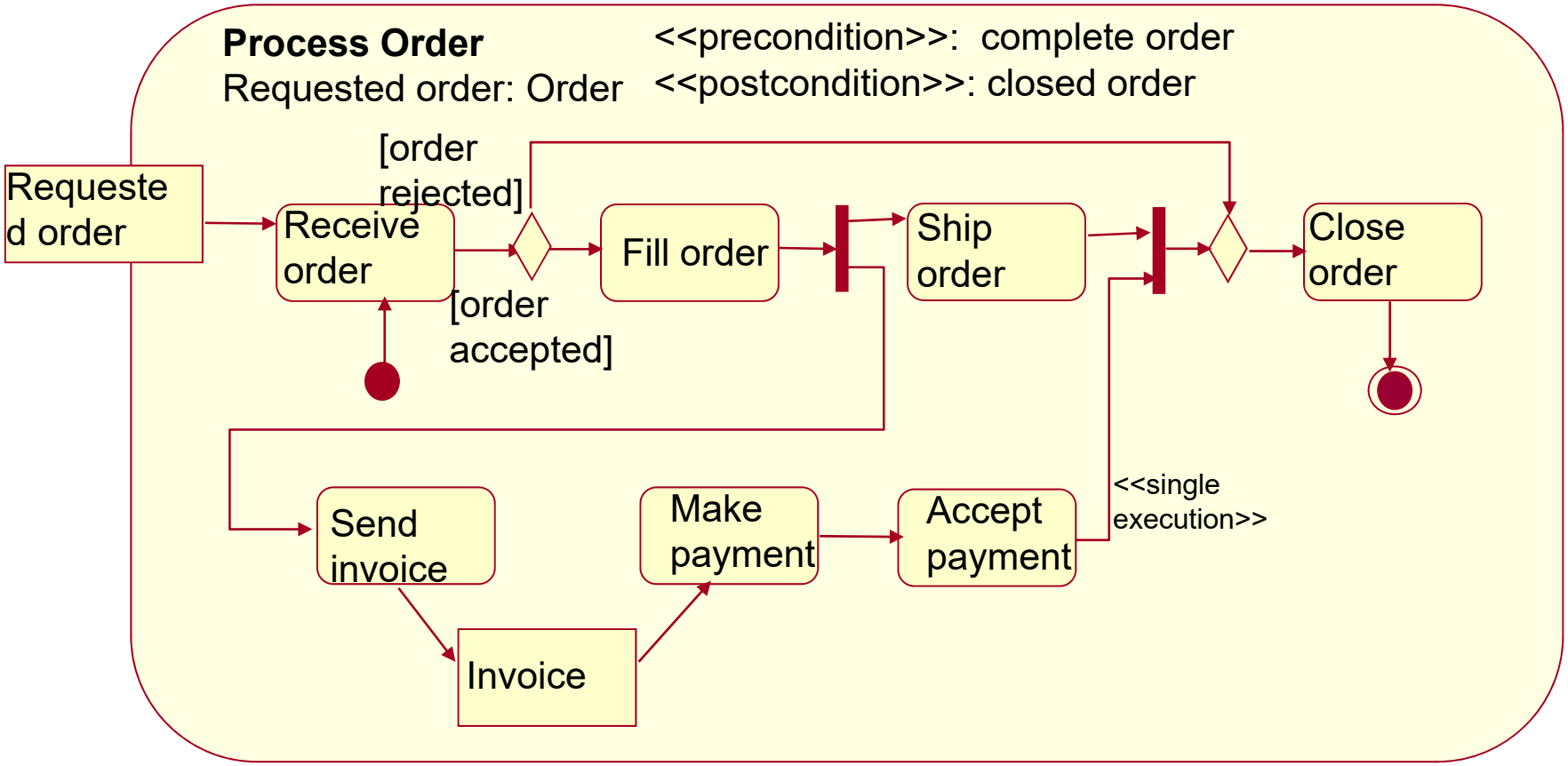
Continue the swimlane assignments until the entire diagram has been rearranged within the swimlanes.

Dimensions

- An activity partition may also represent a *dimension* of the problem. A dimension is a highest-level view of the activity that may contain partitions. It addresses a perspective on the activity as opposed to an assignment of responsibility.



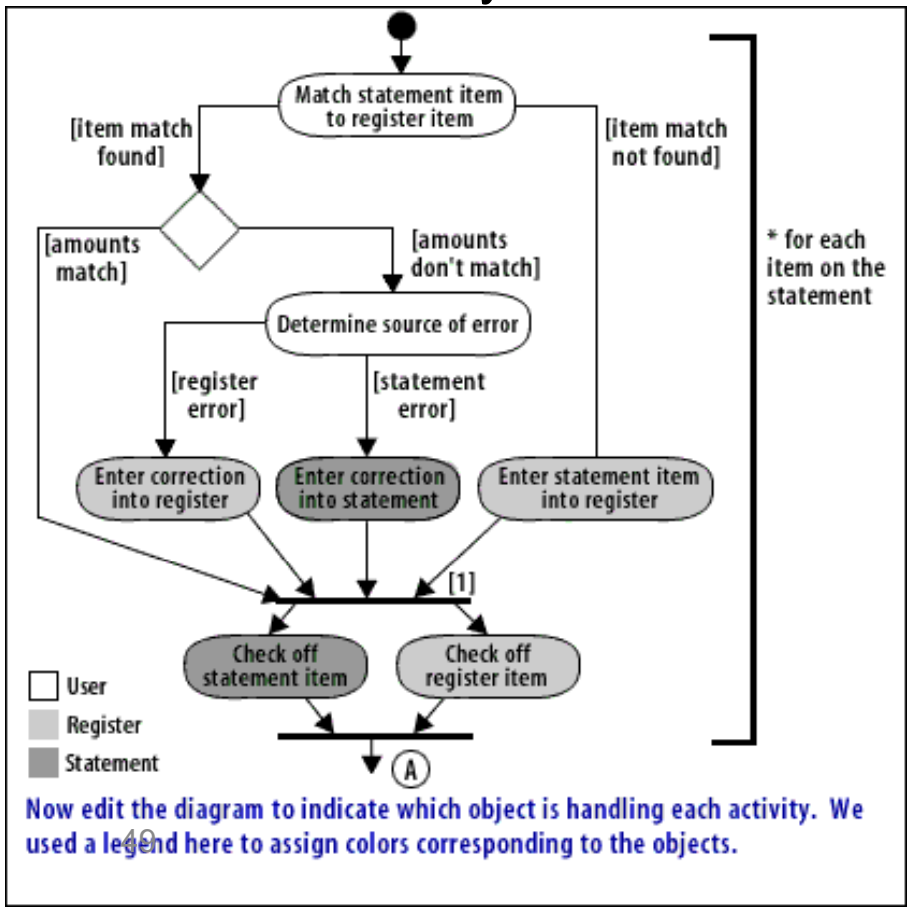
Example of an activity with a parameter



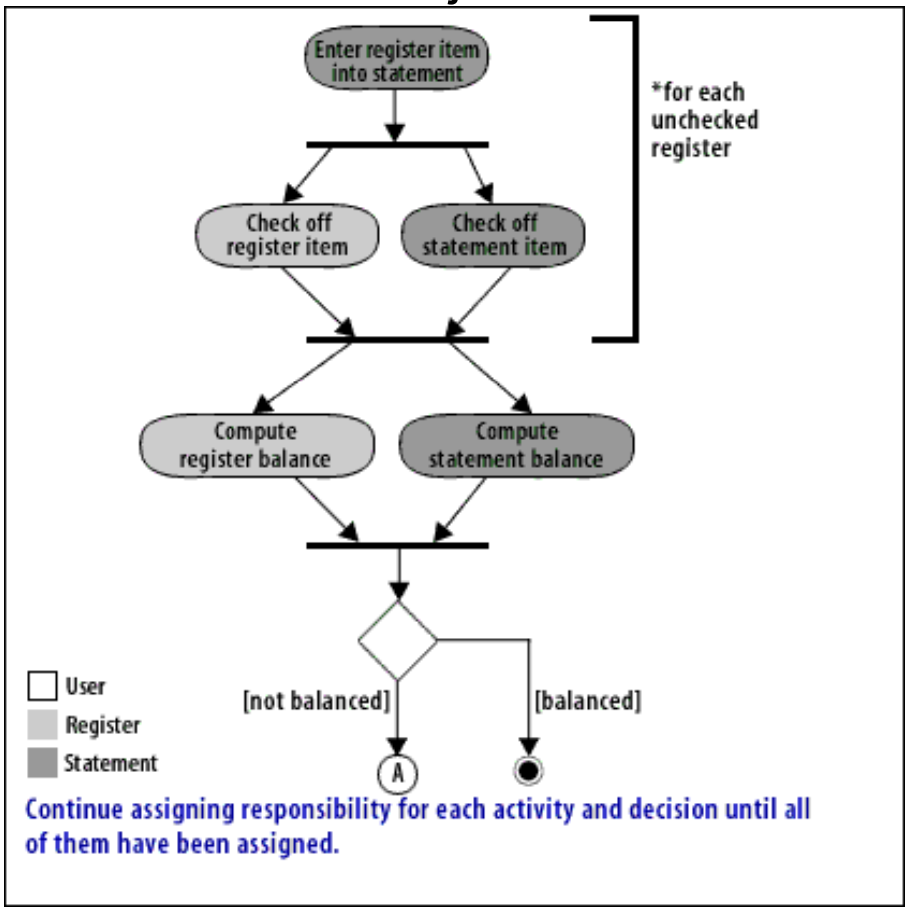
Map activities to objects using swimlanes

Once the activity diagram is complete and verified, you will need to decide which object will be responsible for each activity and decision. The UML accomplishes this mapping by placing columns or swimlanes over the activity diagram.

1. Identify the objects involved by activities and assign swimlanes to them.
2. Then each activity is moved into the swimlane of the object that owns it.

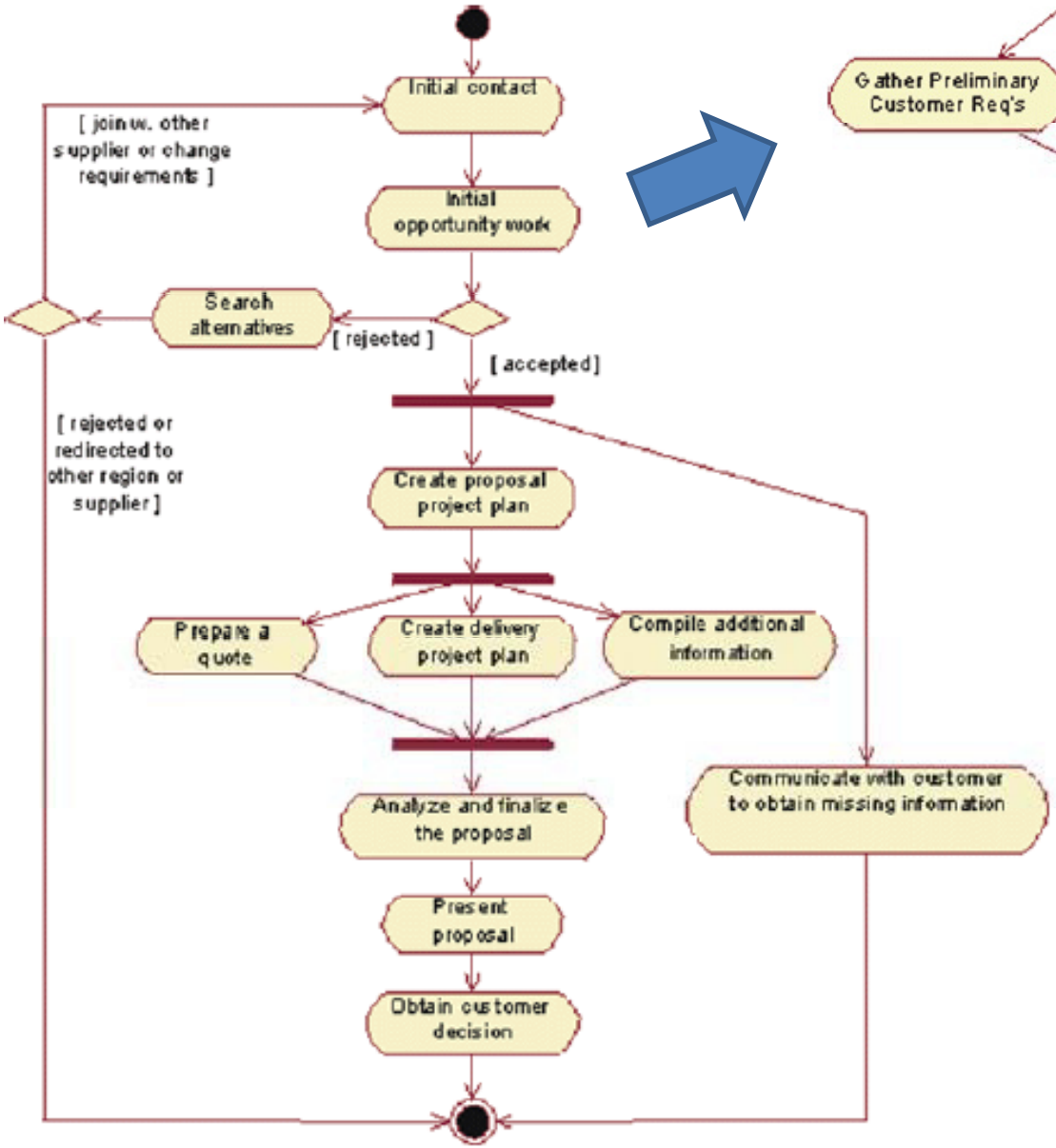


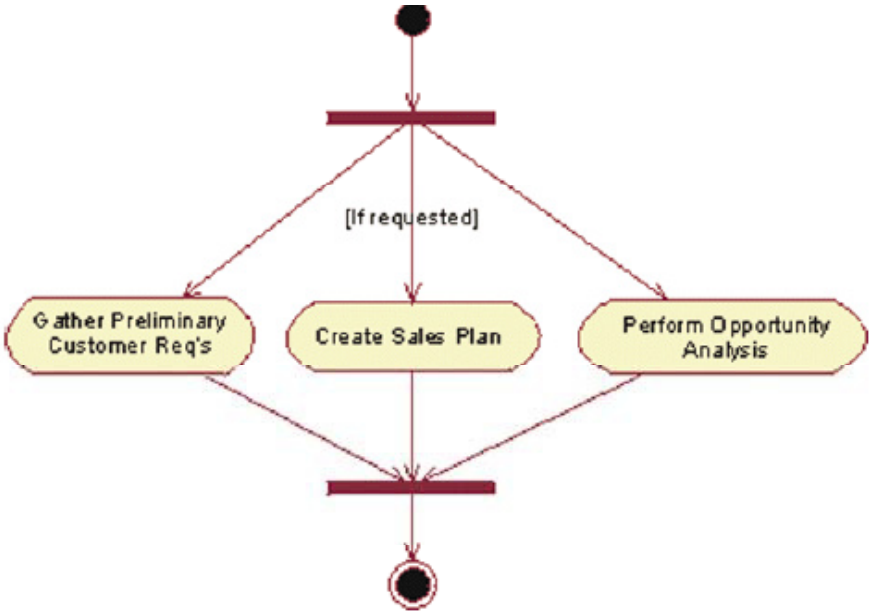
Now edit the diagram to indicate which object is handling each activity. We used a legend here to assign colors corresponding to the objects.



Continue assigning responsibility for each activity and decision until all of them have been assigned.

Activity Diagram with sub-activity Diagram





Etudes de cas

Le système Point of Sale (POS)



POS est une applications informatisée qui servent notamment à l'enregistrement des ventes et au traitement des règlements. Ils sont constitues de composants matériels — ordinateurs et lecteurs de codes a barres — et d'un logiciel. Ils sont connectes a diverses autres applications telles que des systèmes de calcul des taxes et de gestion des stocks. Ils doivent posséder une certaine tolérance aux pannes 1 en cas de défaillance des services distants, comme le système de gestion des stocks, ils doivent pouvoir continuer à permettre de saisir les ventes et de gérer au moins les règlements en espèces.

Ces systèmes doivent de plus en plus couramment prendre en charge les terminaux et les interfaces de clients divers. Il peut s'agir de clients légers comme des navigateurs Web, d'ordinateurs personnels dotes d'interfaces graphiques Java Swing, d'écrans tactiles, de PDA (Personal Digital Assistant, assistant personnel) sans fil, etc.

De plus, nous sommes en phase de création d'un système que nous destinons à différents types d'utilisateurs, dont les besoins en termes de règles de gestion sont variables. Chacun d'eux aura besoin d'une logique distincte qui sera exécutée à certains points prévisibles des scenarios d'utilisation du système, comme le début d'une vente ou l'ajout d'un nouvel article.

Utilisez le développement itératif: l'analyse des besoins, l'analyse orientée objet, la conception et l'implémentation

Déclaration de problème : Système de Vente des Billet en ligne

- Notre entreprise a besoin d'un système pour soutenir les ventes de billets pour les spectacles à notre lieu. Le lieu est une facilité d'auditorium seul. Les sièges sont arrangés dans le lieu dans les sections, dans les rangs, et dans les nombres de siège pour que nous pouvons identifier uniquement chaque siège. Nous présentons une grande variété d'événements comme les jeux, les concerts, et les événements sportifs. Certains de nos événements courent longtemps avec le multiple montre chaque semaine. Quelques événements ont seulement un spectacle, comme un événement sportif. Nous utilisons que nous appelons une stratégie d'évaluation pour chaque spectacle. Une stratégie d'évaluation de spectacle est composé d'un ou les rangées de plus de prix ou les niveaux d'évaluation. Chaque rangée de prix définit un prix pour chaque type de prix que nous offrons, par exemple, l'adulte, l'étudiant, et les prix enfant. Chaque siège dans un spectacle est évalué par associer chaque siège à une des rangées de prix pour ce spectacle. La stratégie d'évaluation peut inclure aussi une remise, mais pas tous spectacles offrent un rabais. Aussi, la même stratégie d'évaluation peut être utilisée pour n'importe quel nombre de spectacles. Par exemple, nous utilisons la même stratégie d'évaluation pour toutes nos matinées de samedi.
- La plupart de nos billets sont vendus par les agents. Les agents doivent avoir un contrat légal avec nous avant que nous pouvons ayons fait des affaires. Une fois les agents ont un contrat, les séries de directeur de facilités sur un accord de ventes avec l'agent. L'accord de ventes donne à la permission d'agent pour vendre de certains sièges pour une période de temps. L'agent peut vendre des billets pour les sièges assignés pour n'importe quels spectacles qui interviennent pendant cette période. Les clients individuels peuvent acheter aussi des billets directement de nous. Ils peuvent accéder à seulement les sièges qui ne sont pas assigné aux agents. Les clients et les agents regardent les sièges disponibles par un graphique plaçant. Quand un client ou l'agent choisit un siège sur le graphique plaçant, il ou elle met efficacement une prise sur le siège pour que personne d'autre peut le choisir jusqu'à ce que le client ou le relâche ou l'achète. Une fois le client ou l'agent achète le billet, le billet est distribué et est envoyé au client.

FIN DE LA TROISIÈME PARTIE

Chap. 4
Analyse des besoins

Les besoins sont les conditions auxquelles un système et, plus généralement, un projet doivent satisfaire.

Il ne s'agit pas, comme dans les méthodes en cascade, de définir et de stabiliser les besoins une fois pour toutes des la première phase du projet avant de commencer à développer .

C'est mieux d'employer « une approche systématique à la conclusion, documenter, organiser, et traquant les besoins changeantes d'un système ».

Les types et les catégories de besoins

- La principale difficulté de l'analyse des besoins consiste à rechercher, communiquer et mémoriser ce qui est réellement nécessaire, sous une forme compréhensible par le client et l'équipe de développement.

les besoins sont classes selon le modèle FURPS+

  - Fonctionnalité (Functionality). Ponctions, capacités et sécurité.
  - Aptitude à l'utilisation (Utilisability). Facteurs humains, aide et documentation.
  - Fiabilité (Reliability). Fréquence des pannes, possibilités de récupération et prévisibilité.
  - Performance (Performance). Temps de réponse, débit, exactitude, disponibilité et utilisation des ressources.
  - Possibilités de prise en charge (Supportability). Adaptabilité, facilite de maintenance, internationalisation et configurabilité.

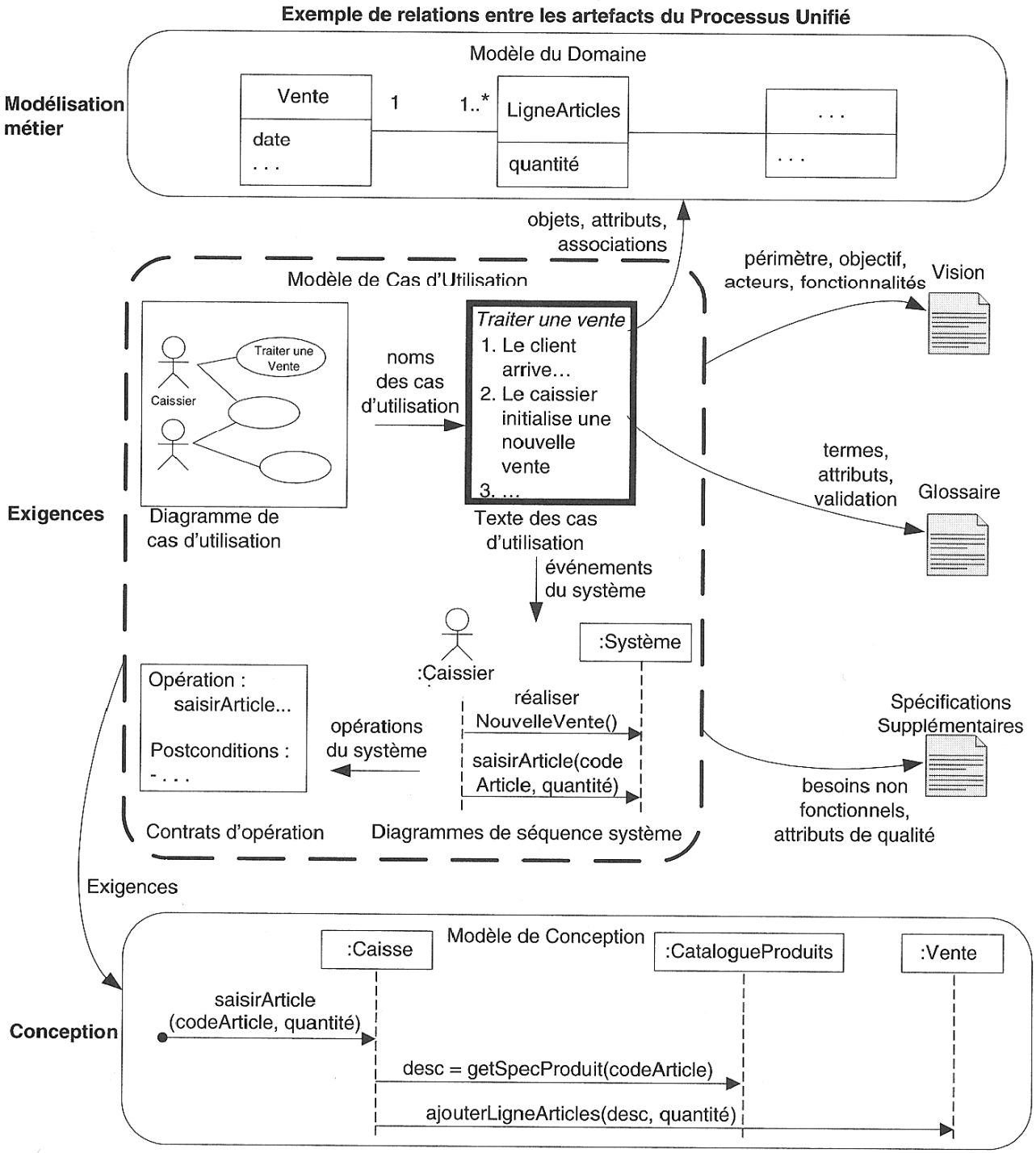
quality attributes, quality requirements or non-functional requirements

- Certaines de ces spécifications sont collectivement dénommées attributs de qualité, spécifications de qualité

Les besoins dans les artefacts du Processus Unifié

- Le Processus Unifie propose plusieurs artefacts pour organiser les besoins.
- Modèle de Cas d'Utilisation. Ensemble de scenarios typiques de l'utilisation d'un système. Il concerne avant tout les besoins fonctionnels (comportementaux).
- Spécifications Supplémentaires. Elles contiennent en substance tout ce qui ne se trouve pas dans les cas d'utilisation et concernent surtout les besoins non fonctionnels (les exigences de performance ou les licences).
- Glossaire. Il définit les termes les plus importants. Il peut détailler virtuellement n'importe quel élément : un attribut d'un objet, un paramètre d'une méthode, l'agencement d'un état, etc.
- Vision. Il résume les spécifications de haut niveau élaborées dans le Modèle de Cas d'Utilisation et dans les Spécifications Supplémentaires, ainsi que l'étude d'opportunité du projet.
- Règles métier ou règles du domaine, décrivent les besoins ou les politiques qui ne s'appliquent pas qu'à un seul projet. Elles intéressent l'intégralité d'un domaine ou d'un métier. Les réglementations fiscales en constituent un excellent exemple.

Cas d’utilisation

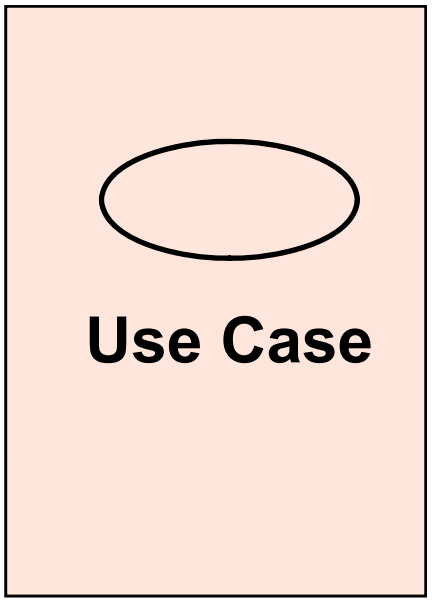


Influence des cas d'utilisation sur les artefacts du Processus Unifié

Modèle de cas d'utilisation

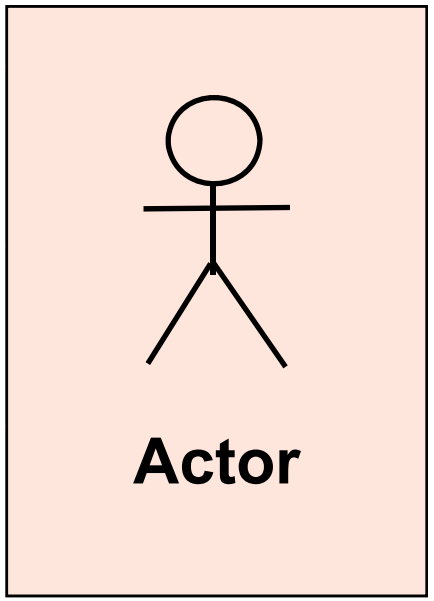
- Un modèle de cas d'utilisation est un modèle des fonctions du système voulu (les cas d'utilisation) et ses entités environnantes (les acteurs).
- Les cas d'utilisation constituent toutes façons possibles d'utilisations du système.
- Les acteurs représentent des utilisateurs et/ou n'importe quel système qui réagit réciproquement avec le système vous construisez . Un **acteur** est une entité qui a un comportement, par exemple une personne identifiée par un rôle.
- Un scénario est une suite spécifique d'actions et d'interactions entre un ou plusieurs acteurs et le système. On le nomme également instance de cas d'utilisation. C'est une « histoire » particulière de la façon dont on utilise un système, ou l'un des cheminements possibles dans un cas d'utilisation.
- Un cas d'utilisation est donc une collection de scenarios de réussite ou d'échec qui décrit la façon dont un ou plusieurs acteurs utilisent un système pour atteindre un but.
- Les cas d'utilisation sont des besoins. Il s'agit avant tout de besoins fonctionnels ou comportementaux, qui indiquent ce que le système devra faire.
- Les sources d'Informations pour les cas d'utilisation:
  spécification de système /« problem statement », littérature sur le domaine, interviews avec des experts de domaine , systèmes « legacy ».

Cas d'utilisation



- Un cas d'usage est une unité seule de travail logiquement unitaire. Il fournit une vue de haute-niveau de comportement observable à quelqu'un situé hors du système
- Un cas d'utilisation est aussi un flux complet et significatif d'événements .
- Un cas d'utilisation est instancié par un acteur pour invoquer un certain fonctionnalité dans le système.
- Les cas d'usage ne sont pas de diagramme, ils sont principalement les documents de texte: une histoire de texte de quelque acteur utilisant un système pour rencontrer des objectifs.
- Les cas d'utilisation servent comme un fil unifiant à travers le développement de système: l'analyse de besoins, la conception, et le test.

Quels sont les trois types d'acteurs ?



Un acteur est une entité quelconque ayant un comportement, comme une personne (identifié par le rôle), le système informatique, ou l'organisation. Les acteurs ne sont pas la partie du système, ils représentent des rôles un utilisateur du système peut jouer

L'acteur principal = Il a des buts d'utilisateur qu'il atteint en utilisant les services du système. Ce peut être, par exemple, le caissier.

Pourquoi l'identifier ? Pour découvrir les buts utilisateurs, qui pilotent les cas d'utilisation.

L'acteur auxiliaire = Il fournit un service (par exemple une information) au système. Le système de paiements automatisé en est un.

Pourquoi l'identifier ? Pour clarifier les protocoles et les interfaces externes.

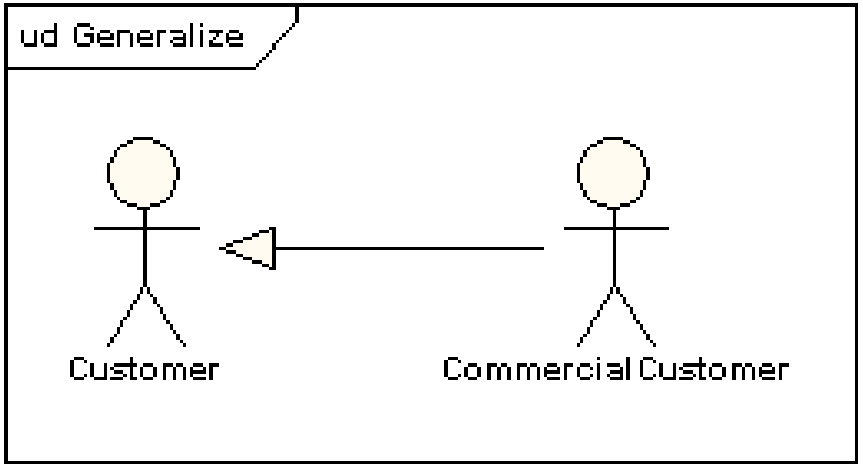
L'acteur hors champ = Il a un intérêt dans le comportement du cas d'utilisation, sans être un acteur principal ou auxiliaire.

Pourquoi l'identifier ? Pour s'assurer que tous les intérêts ont bien été identifies et qu'ils seront satisfaits.

Comment découvrir les acteurs?

Répondez a des questions utiles :

Qui est intéressée dans une certaine besoin?
Où dans l'organisation le système est-il utilisé ?
Qui fournira le système avec ces informations, utiliser ces informations, éliminer ces informations ?
Qui utilisera cette fonction ?
Qui s'occupera de la gestion et de la maintenance du système ?
Le système utilise-t-il une ressource externe ?
Quels acteurs ont-ils besoin de ce cas d'utilisation ?
Un acteur joue-t-il plusieurs rôles différents ?
Plusieurs acteurs jouent-ils le même rôle ?



Un acteur peut généraliser un autre acteur

Les trois formats des cas d'utilisation

Abrégé. Résumé succinct qui présente généralement le scenario de base (succès) dans un paragraphe.

— Quand? Lors clé la première étude de besoins, pour se faire rapidement une idée du sujet et de son périmètre. Quelques minutes peuvent suffire à le créer.

Informel. Les différents scenarios sont décrits dans plusieurs paragraphes.

— Quand ? Comme précédemment

Détaillé. Toutes les étapes et les variantes sont indiquées en détail, de même que les préconditions et les garanties en cas de succès.

— Quand ? Lorsque de nombreux cas d'utilisation ont été identifies et rédigés au format abrégé, puis durant le premier atelier d'expressi0n des besoins, quand environ 10 % des cas architecturalement significatifs et prioritaires pour l'entreprise ont été rédigés en détail.

Example de cas d'utilisation abrégé

Traiter une vente.

Un client arrive a la caisse avec les articles qu'il souhait acheter. Pour enregistrer chaque article, le caissier utilise le système POS, lequel présente le détail des articles et le montant total des achats. Le client fournit les informations nécessaires pour le règlement. Le système valide et enregistre ces informations, puis met a jour les quantités en stock et imprime le ticket de caisse destine au client. La vente est terminée et le client peut quitter le magasin.

Example de cas d'utilisation informel

Traiter un retour

Scénario principal {succès) :

Un client arrive à la caisse avec les articles qu'il veut retourner. Le caissier utilise le système POS pour enregistrer chaque article retourne. ..

Autres scénarios :

Si le client a payé a crédit d'avance et que la demande de remboursement sur sou compte est rejetée, l'en informer et le rembourser en espèces.

Si le code de l'article n'est pas reconnu par le système, informer le caissier et lui suggérer de saisir le code manuellement (l'emballage est peut-être endommagé).

Si le système ne parvient pas à communiquer avec le système comptable externe...

Modèle d’un cas d’utilisateur détaillé

Section du cas d'utilisation	**Commentaire**
Nom du cas d'utilisation	Commencer par un verbe
Périmètre	Le système en cours de conception
Niveau	But utilisateur ou << sous-fonction »
Acteur principal	Fait appel au système pour obtenir un service
Parties prenantes et intérêts	Qui s'intéresse a ce cas d'utilisation et que souhaite-t-il ?
Pré conditions	Quelles sont les conditions qui doivent être vraies initialement et que le lecteur a besoin de connaitre?
Garanties en cas de succès	Quelles sont les conditions qui doivent être vraies lorsque le cas d'utilisation se termine avec succès et que le lecteur a besoin de connaitre?
Cas nominal (succès)	Scenario de succès type, sans conditions
Extensions	Autres scénarios de succès ou d'échec
Spécifications particulières	Besoins non fonctionnels associés
Liste de variantes des données	Différentes méthodes d'E/S et formats de données et des technologies
Fréquence d'occurrence	Influence sur l’investigation, les tests et la planification de l’implémentation
Divers	Par exemple, questions demeurant ouvertes

Exemple d'un cas d'utilisation detaillé (I)

Cas d'utilisation CU1 : Traiter une vente
Périmètre :Application POS NextGen
Niveau : but utilisateur
Acteur principal : le Caissier
Parties prenantes et intérêts :
- **Le Caissier**. Il veut un moyen de saisie exact et rapide et aucune erreur de paiement, car les différences de caisse sont retenues sur son salaire.
- **Le Vendeur**. Il veut que les commissions sur les ventes soient mises à jour
- **Le Client**. Il veut pouvoir réaliser des achats aisément, obtenir un service rapide en fournissant un minimum d'efforts. Il veut également une preuve d'achat pour le cas ou il devrait rendre un article.
- **L'Entreprise**. Elle veut enregistrer correctement les transactions et satisfaire les souhaits des clients. Elle veut être sure que les effets à recevoir du Service d'autorisation des paiements sont enregistres. Elle veut une certaine tolérance a l'erreur afin de pouvoir traiter les ventes même si l'un des composants du serveur est indisponible (par exemple, la validation du crédit a distance). Elle veut une mise à jour automatique et rapide de la comptabilité et de l'inventaire.
- **Les Services fiscaux**. Ils veulent enregistrer les taxes sur chaque vente.
- **Le Service d'autorisation des paiements**. Il veut recevoir des demandes d'autorisation numérique conformes à un format et a un protocole donnés. Il veut une comptabilité exacte des effets à payer au magasin.

Exemple d'un cas d'utilisation detaillé (II)

Préconditions :
Le Caissier est identifie et authentifié.
Garanties en cas de succès (postconditions) :
La vente est enregistrée. La taxe est correctement calculée. La comptabilité et les quantités en stock sont mises à jour. Les commissions sont enregistrées. Le reçu est généré. Les autorisations de paiement sont enregistrées.
Scenario principal (succès) :
1. Le Client arrive à la caisse avec des marchandises et/ou des services à acheter.
2. Le Caissier commence une vente.
3. Le Caissier entre le code de l'article,
4. Le Système enregistre l'article et présente sa description, son prix et le total en cours. Le prix est calcule a partir d'un ensemble de règles.

Le Caissier répète les étapes 3 et 4 jusqu'à ce que tous les articles soient saisis.

5. Le Système présente le total incluant les taxes calculées.
6. Le Caissier communique le montant total au Client et en demande le paiement.
7. Le Client pale et le Système traite son règlement.
8. Le Système enregistre la vente et transmet les informations la concernant et celles concernant le règlement au système de Comptabilité externe (pour la mise à jour de la comptabilité et le calcul des commissions) et au système de Gestion de stock (pour la mise a jour des quantités en stock).
9. Le Système génère un reçu.
10. Le Client quitte le point de vente avec ses achats et son reçu.

Exemple d'un cas d'utilisation detaillé (III)

Extensions (ou scénarios alternatifs) :

*a. A tout moment, si le Manager demande une opération réservée:
    1. Le Système entre en mode Manager.
    2. Le Manager ou le Caissier exécute une opération du mode Manager, par exemple une modification du solde de la caisse, la reprise d'une vente commencée sur une autre caisse, l'annulation d'une vente, etc.
    3. Le Système revient en mode Caissier.

1a. Le Client ou le Manager demande à reprendre une vente suspendue.
    1. Le Caissier exécute l'opération de reprise et tape le code pour récupérer la vente.
    2. Le Système affiche l'état de la vente avec son sous-total.
        2a. Vente non trouvée.
            1. Le Système signale l'erreur au Caissier.
            2. Le Caissier démarre probablement une nouvelle vente et ressaisit tous les articles.
    3. Le Caissier poursuit la vente (probablement en saisissant d'autres articles ou en demandant le total).

2-4 a. Le Client indique au Caissier que son statut l'exempte de certaines taxes (par exemple seniors, détaxe à l'exportation) :
    1. Le Caissier vérifie l'exactitude de la déclaration du Client et saisit le code correspondant.
    2. Le système enregistre le code (qui sera utilise lors du calcul des taxes).

3a. Code article invalide (non trouvé par le Système):
    1. Le Système signale l'erreur et rejette la saisie.
    2. Le Caissier traite l'erreur: . . . . .

Exemple d'un cas d'utilisation detaillé (IV)

Spécifications particulières

- Interface utilisateur a écran tactile sur grand écran plat. Le texte doit être visible à un mètre.
- Obtention de la réponse a la demande d'autorisation en moins de trente secondes. dans 90 % des cas.
- Une forme quelconque de récupération fiable quand l'accès a un service distant (stock, par exemple) échoue.
- internationalisation du texte affiché.
- Possibilité d'introduction de règles de gestion dans les étapes 3 et 7.

Liste de variantes de données et de technologies :

*a. Pour réaliser les opérations réservées, le Manager doit insérer une carte dans un lecteur dédié ou un code particulier via le clavier.

3a. Le code de l'article est saisi par un lecteur laser (si le code a barres est présent) ou au clavier.

3b. Le code de l'article peut obéir à n'importe quel schéma de codage UPC, EAN,JAN ou SKU.

7a. Les informations concernant les règlements à crédit sont saisies par un lecteur de cartes ou sur le clavier.

7b. Actuellement, en cas de paiement a crédit, le Client signe un reçu papier. Nous prévoyons que d'ici deux ans, de nombreux clients pourront saisir une signature numérique.

Fréquence d'occurrence :

Pourrait être pratiquement continue.

Questions ouvertes :

- Qu'en est-il des variations de taux des taxes?
- Etudier le problème de la récupération en cas d'inaccessibilité d'un service distant.

Une variante sur deux colonnes

Cas d'utilisation CU1 : Traiter une vente
Acteur principal :
...inchangé...
Scénario principal (succès) : Responsabilités du système

Action Acteur (ou intention)	
1. Le Client arrive à la caisse avec des marchandises et/ou des services à acheter.	
2. Le Caissier commence une vente.	
3. Le Caissier entre le code de l'article.	
	4. Le Système enregistre l'article et présente sa description, son prix et le total en cours.
Le Caissier répète les étapes 3 et 4 jusqu'à ce que tous les articles soient validés,	
	5. Le Système présente le total avec les taxes calculées.
6. Le Caissier communique le montant total au Client et en demande le paiement.	
7. Le Client paie.	
	8. Le Système traite le paiement.
	9. Le Système enregistre la vente et transmet les informations au système de Comptabilité externe (pour la comptabilité et les commissions) et au système de Gestion de stock (pour la mise à jour des quantités en stock), Il génère un reçu.

Comment découvrir les cas d'utilisation ?

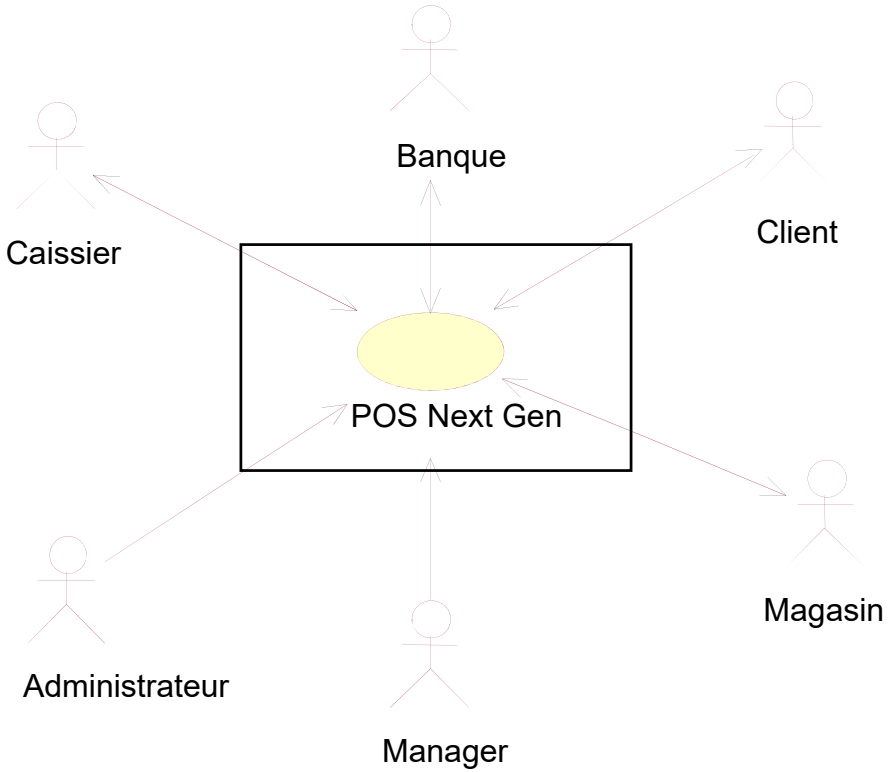
1. Délimiter le système. S'agit-il d'une application logicielle, de l'ensemble matériel et application, de cet ensemble plus la personne qui l'utilise ou de toute une organisation ?
2. Identifier les acteurs principaux, ai savoir ceux qui atteignent leurs buts en utilisant les services du système.
3. Identifier les buts de chaque acteur principal.
4. Définir des cas d'utilisation qui correspondent aux objectifs des utilisateurs et les nommer conformément à ce but. Les cas d'utilisation et les buts sont généralement dans une relation « un à un ».

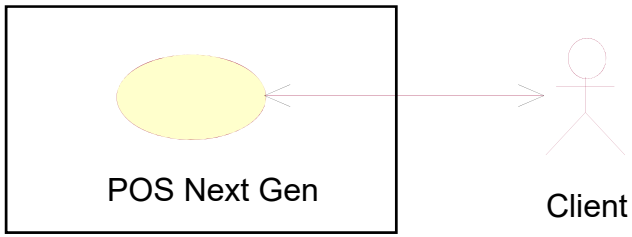
Diagramme de contexte

Normalement on décrit les cas d'utilisation dans le système et les acteurs comme hors du système: les frontières de système.

Les frontières typiques d'un système peuvent être :

- Un appareil ou un système connecté à notre système
- Une personne,
- Une division dans une entreprise ou une organisation entière





Identifier les acteurs principaux et auxiliaires

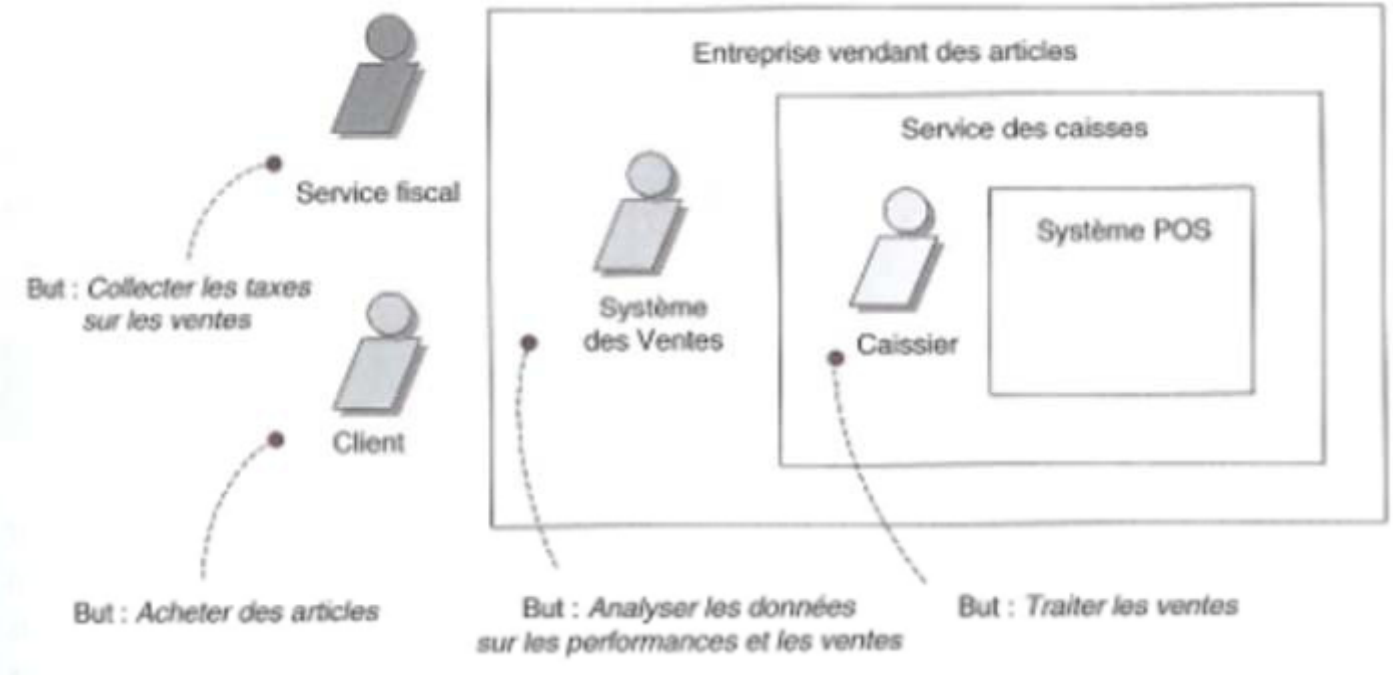
Principe de base : se concentrer sur la recherche des acteurs principaux, ce qui fournira le cadre des investigations ultérieures.

Questions qui permettent d'identifier acteurs et buts

1. Qui démarre et arrête le système ?
2. Qui est responsable de la gestion des utilisateurs et de la sécurité ?
3. Existe-t-il un processus de surveillance qui restaure le système s'il tombe en panne ?
4. Comment les mises à jour logicielles sont-elles gérées ?
5. Qui assure l'administration du système ?
6. Le « temps » est-il un acteur (par exemple, le système exécute-t-il des tâches par rapport a un événement temporel) ?
7. Qui évalue l'activité ou les performances du système ?
8. Qui évalue les journaux ? Sont-ils récupérés à distance ?
9. Outre les acteurs principaux humains, y a—t-il des sesterces externes, logiciels ou matériels, qui font appel aux services du système ?
10. Qui reçoit les notifications en cas d'erreur ou de panne ?

L'acteur principal est-il le Caissier ou le Client ?

- La réponse dépend des limites du système à concevoir, et du destinataire premier du système.
- Si on considère l'entreprise ou le service des caisses comme un agrégat, alors le client est un acteur principal dont le but est d'acquérir des marchandises et de les emporter.
- Mais, du seul point de vue du système POS, il sert le but du caissier (et du magasin) qui est de réaliser des ventes.
- Le client est un acteur, mais, dans le contexte du système POS, ce n'est pas un acteur principal.



Comment organiser les acteurs et les buts ?

Les acteurs ont des buts et utilisent des applications pour les atteindre.

L'objectif consiste à découvrir ces acteurs et ces buts, et à créer des solutions qui produisent des résultats apportant de la valeur.

Acteur	**But**	**Acteur**	**But**
Caissier	traiter les ventes	Administrateur système	ajouter des utilisateurs
	traiter les locations		modifier des utilisateurs
	traiter les retours		supprimer des utilisateurs
	ouvrir la caisse		gérer la sécurité
	fermer la caisse		gérer les tables
Manager	démarrer le système	Système des Ventes	analyser les données sur les ventes et les performances
	arrêter le système		

Le nom d'un cas d'utilisation de ce niveau doit refléter le nom de ce but, pour mettre ce point de vue en évidence.

Objectif : capturer ou traiter une vente -> Cas d'utilisation : Traiter une vente.

Définir les cas d'utilisation

- On définit généralement un cas d'utilisation pour chaque but utilisateur, et on le nomme d'après ce but. Exemple : si le but est de traiter une vente, le cas d'utilisation se nomme Traiter une vente.
- Nommez les cas d'utilisation en commençant par un verbe à l'infinitif.

Les tests que permettent d'identifier des cas d'utilisation valides

Négocier un contrat avec un fournisseur.

- Trop vaste et trop long pour un processus métier élémentaire. Pourrait être utilise comme un cas d'utilisation métier et non comme un cas d'utilisation système.

Gérer les retours.

-Le patron est content. Présente les caractéristiques d'un un processus métier élémentaire. La taille est correcte.

Gérer les retours.

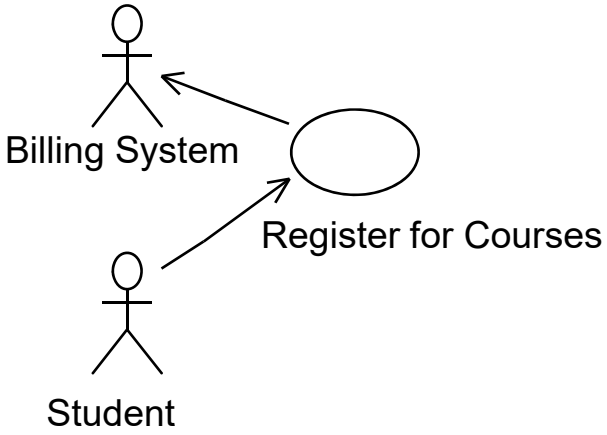
- Le patron n'est pas content si vous y avez passé la journée.

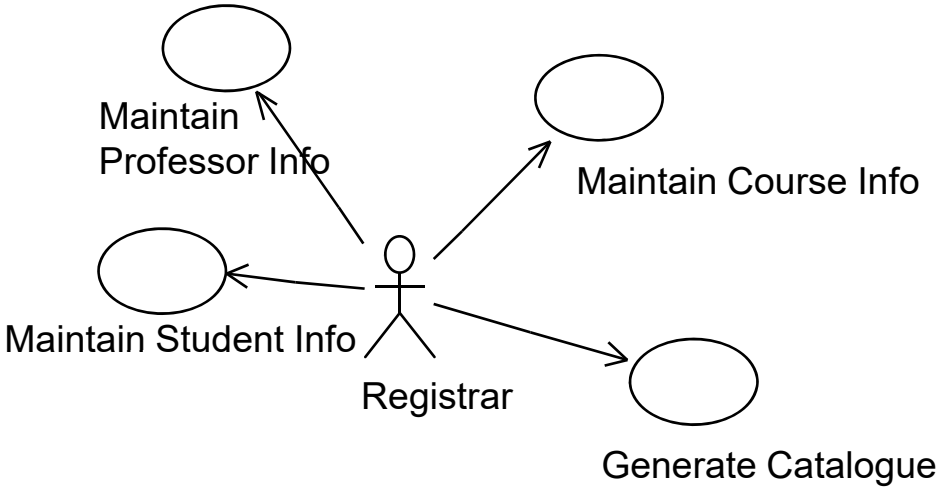
Déplacer un pion sur le plateau de jeu.

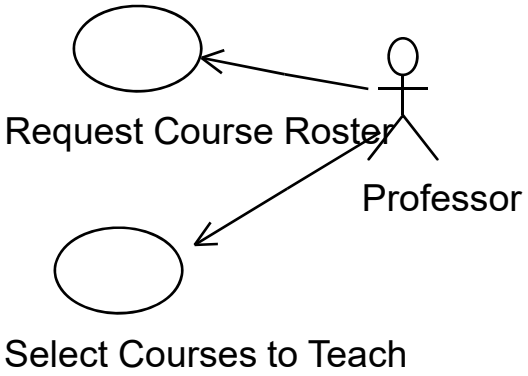
-Ce n'est qu'une étape. La taille est trop réduite.

Use Case Identification

- Student
  - Register for courses
- Registrar
  - Maintain course information,
  - Maintain student information,
  - Maintain professor information,
  - Generate catalogue
- Professor
  - Request course roster,
  - Select courses to teach







Register for Courses Use Case

Brief Description -- Register for Courses Use Case

- This use case is initiated by a student. It provides the capability for the student to create, delete, modify and/or review a course schedule for a given semester.

Flow of Events -- Register for Courses Use Case

- This use case begins when the student enters the student id number. The system verifies that the student id number is valid and prompts the student to select the current semester or a future semester. The student enters the desired semester. The system prompts the student to select the desired activity:
  - Create a schedule
  - Review a schedule
  - Change a schedule
    - Delete a course
    - Add a course
- The student indicates that the activity is complete. The system will print the student schedule and notify the student that registration is complete. The system sends billing information for the student to the billing system for processing.

Flow of Events -- Register for Courses Use Case

- Alternate flow
  If an invalid id number is entered, the system will not allow access to the registration system.
  If an attempt is made to create a schedule for a semester where a schedule already exists the system will prompt for another choice to be made.
- Create a schedule
  The student enters 4 primary course offering numbers and 2 alternate course offering numbers. The student then submits the request for courses. The system then
  1: Checks that prerequisites are satisfied for the requested course
  2: Adds the student to the course offering if the course offering is open.
  Alternate flow
  If a primary course offering is not available, the system will substitute an alternate course offering.
- Review a Schedule
  The student requests information on all course offerings in which the student is registered for a given semester. The system displays all course offerings for which the student is registered including course name, course number, course offering number, days of the week, time, location, and number of credit hours.

Flow of Events -- Register for Courses Use Case

- Change Schedule -- Delete a Course Offering
  The student indicates which course offering to delete. The system checks that the final date for changes has not been exceeded. The system deletes the student from the course offering. The system notifies the student that the request has been processed.
- Change Schedule -- Add a Course
  The student indicates which course offerings to add. The system checks that the final date for changes has not been exceeded. The system then
  1: Verifies that the maximum course load for the student has not been exceeded
  2: Checks that prerequisites are satisfied for the requested course
  3: Adds the student to the course offering if the course offering is open.

Exemples of scenarios

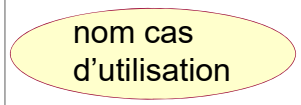
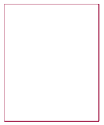
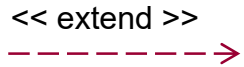
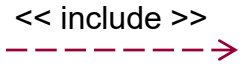
Primary scenario

- John enters the student ID number 369 52 3449 and the system validates the number. The system asks which semester. John indicates the current semester and chooses create a new schedule.
- From a list of available courses, John selects the primary courses English 101, Geology 110, World History 200, and College Algebra 110. He then selects the alternate courses Music Theory 110 and Introduction to Java Programming 180.
- The system determines that John has all the necessary prerequisites and adds him to the course rosters.
- The system indicates that the activity is complete. The system prints the student schedule and sends billing information for four courses to the billing system for processing.

Secondary scenarios :

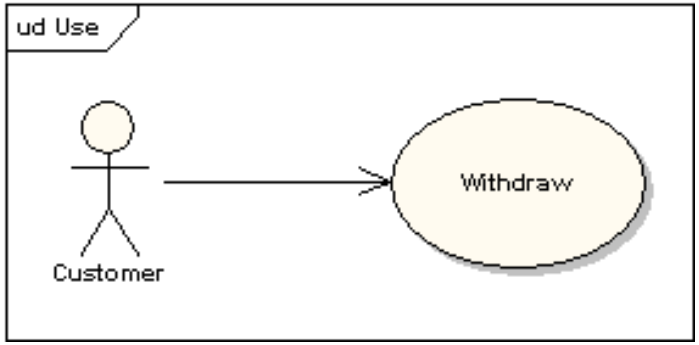
- Student does not select 4 primary courses
- Primary course is not available
- Primary and secondary courses are not available
- Cannot add student to the course roster
- Cannot create the student schedule.

Composants des diagrammes de cas d'utilisation

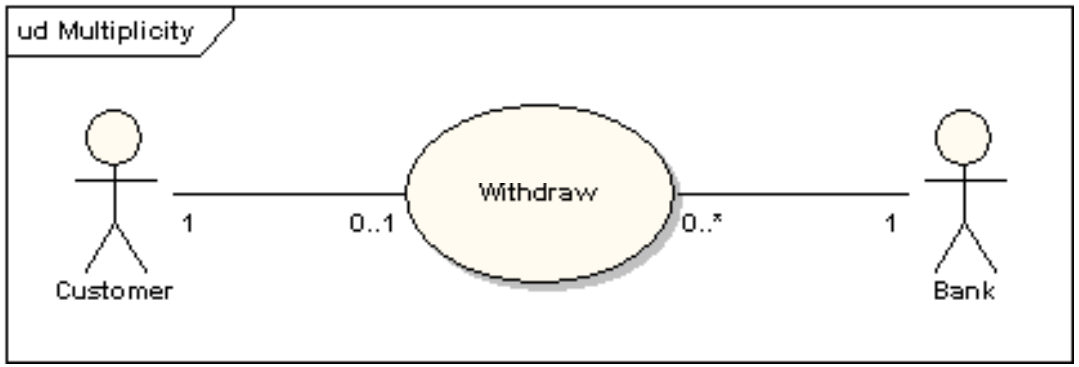
Composant	Description	Syntaxe
cas d'utilisation	Une séquence d'actions, y compris les variantes, qu'un système peut exécuter en réagissant réciproquement avec les acteurs de système.	
acteur	Un ensemble cohérente de rôles que les utilisateurs d'un cas d'utilisation jouent quand ils réagissent réciproquement avec les cas d'utilisation.	 Nom acteur
frontière de système	Elle représente la frontière entre le système et les acteurs qui réagissent réciproquement avec le système.	
association	La participation d'un acteur dans un cas d'utilisation, c'est-à-dire la communication entre une instance d'acteur et les instances d'un cas d'utilisation.	
généralisation	Les relations entre un cas d'utilisation plus général et un cas d'utilisation plus spécifique	
extend	La relation entre une extension d'un cas d'utilisation et le cas d'utilisation de base, indiquant que le comportement d'extension peut être inséré dans le comportement de cas d'utilisation dans un point d'extension	
include	La relation entre un cas d'utilisation de base et un cas d'utilisation « inclus » qui spécifie comment le comportement de cas d'utilisation inclus est inséré dans le comportement de cas d'utilisation de base	

Notation pou les diagrammes de cas d'utilisation

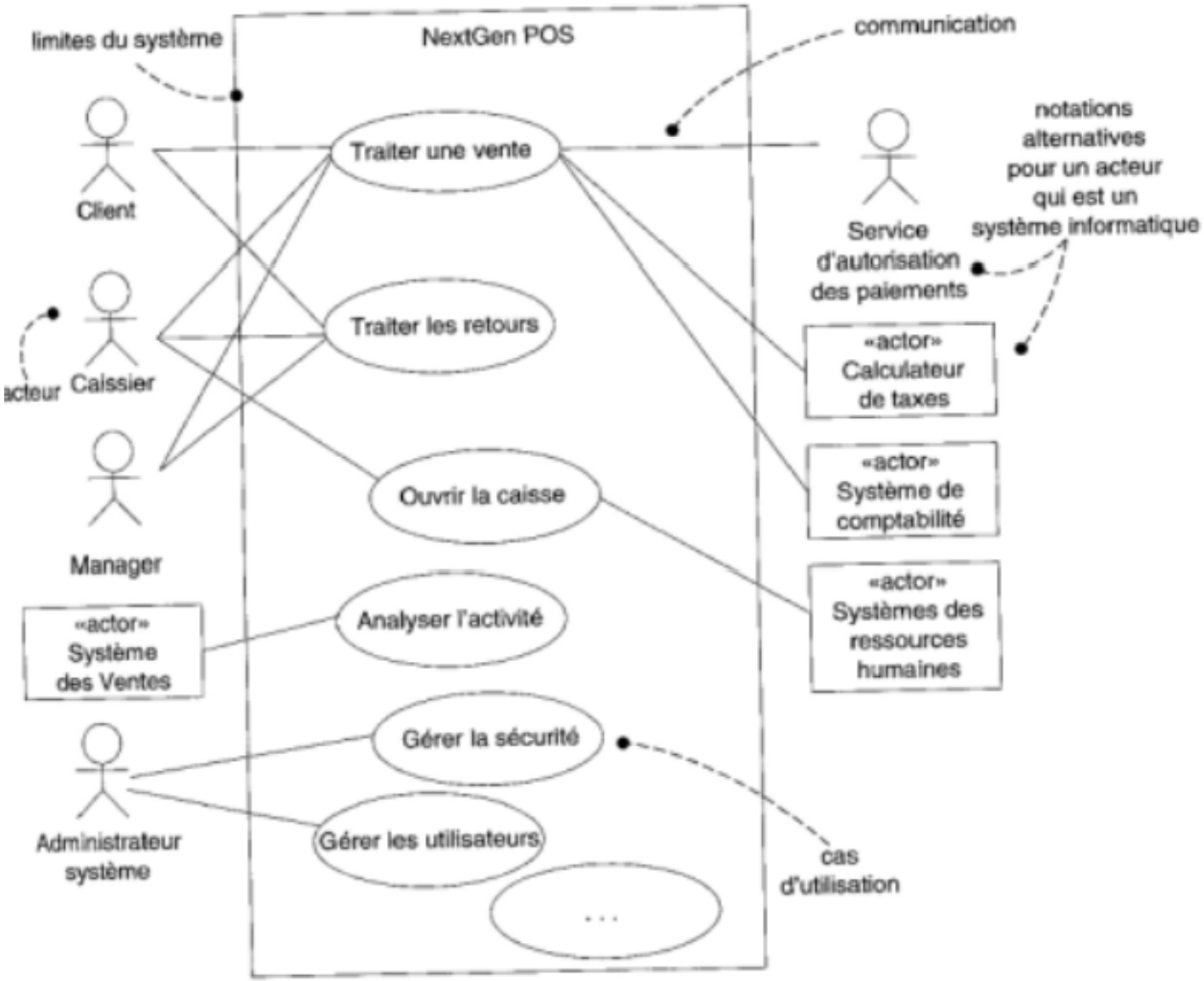
- La notation pour utiliser un cas d'utilisation est une ligne que connecte avec une flèche facultative montrant la direction de contrôle. Le diagramme suivant indique que l'acteur Client utilise le cas d'utilisation Retire.



- Le connecteur « uses » peut avoir facultativement les valeurs de multiplicité à chaque fin, comme dans le diagramme suivant qui montre qu'un client peut avoir seulement une séance de retrait à la fois, mais une banque peut avoir n'importe quel nombre de clients faisant des retraits simultanément.



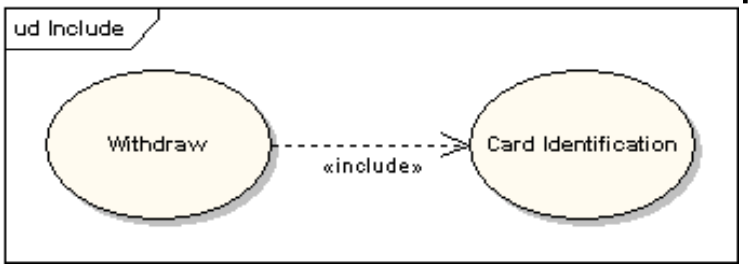
Diagrammes de cas d'utilisation UML



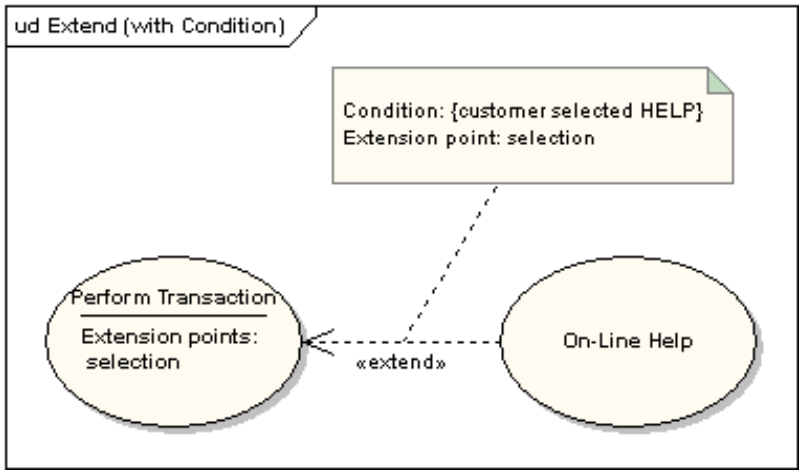
Les diagrammes de cas d'utilisation et les relations entre cas d'utilisation sont secondaires.
Les cas d'utilisation sont des documents textuels. Les modéliser signifie avant tout rédiger.

Relations entre cas d'utilisation

Relation "include". Les cas d'utilisation peuvent contenir la fonctionnalité d'un autre cas d'utilisation comme une partie de leur comportement normal. En général il est supposé que le cas d'utilisation inclus sera appelé chaque fois que le scenario fondamental est exécuté. Dans l'exemple l'exécution du cas d'utilisation <Card Identification> est exécuté comme une partie d'un cas d'utilisation <Withdraw>.



Relation "extend" . Un cas d'Utilisation pourrait être utilisé pour étendre le comportement d'un autre; ceci est typiquement utilisé dans des circonstances exceptionnelles.



Par exemple, si avant modifier un type particulier d'ordre d'un client, un utilisateur doit obtenir l'approbation de quelque plus haute autorité, alors le cas d'utilisation « Get Approval» peut étendre facultativement le cas d'utilisation régulier « Modify Order »

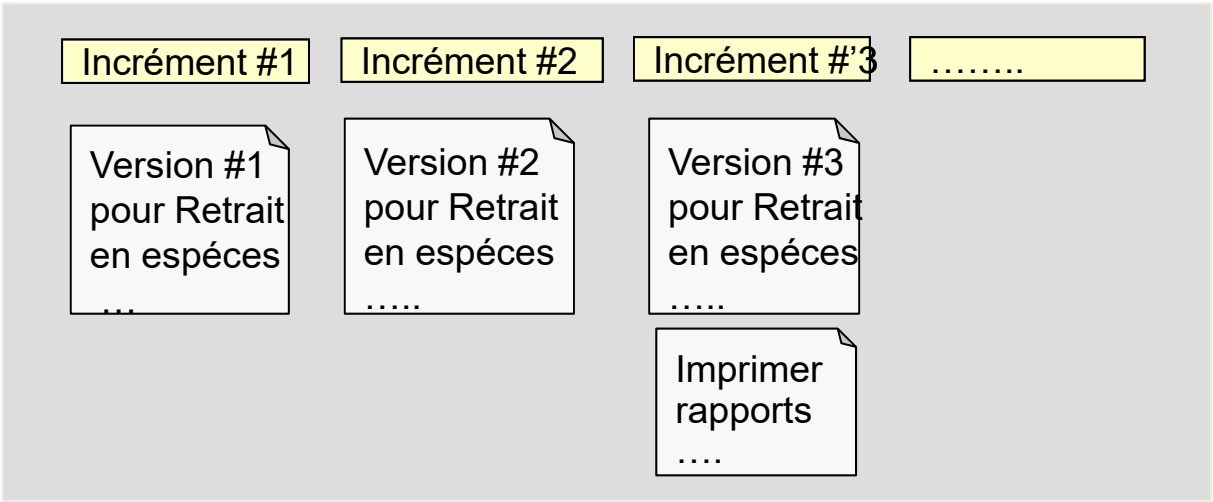
Développement guidé par cas d'utilisation

Critères d'évaluation de la priorité des cas d'utilisation

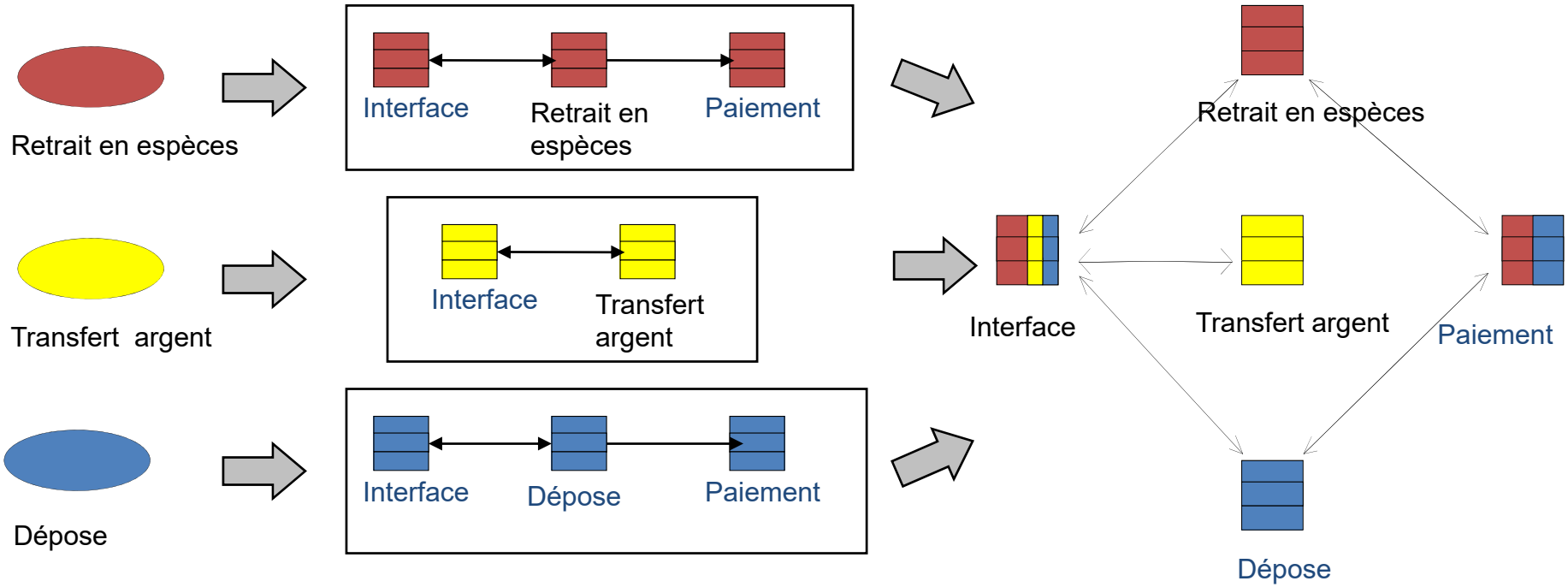
a. Quel est l'impact sur la conception d'architecture ? Des nouveaux classes ou des services persistants pourraient être ajoutés.
b. Informations importantes sont incluses
c. Fonctions de risque sont incluses (critiques comme complexité ou temps).
d. Besoin de recherche importante, comme une nouvelle ou risquante technologie
e. Représente un processus métier principale.
f. Rendre capable l'abaissement du coût ou/and la hausse du revenu.

Exemple:

Cas d'utilisation	a.	b.	c.	d.	e.	f.	Totale
Retrait en espèces	5	5	2	1	5	1	19
Imprimer rapports	2	4	2	0	4	0	12



Cas d'utilisation et Objets



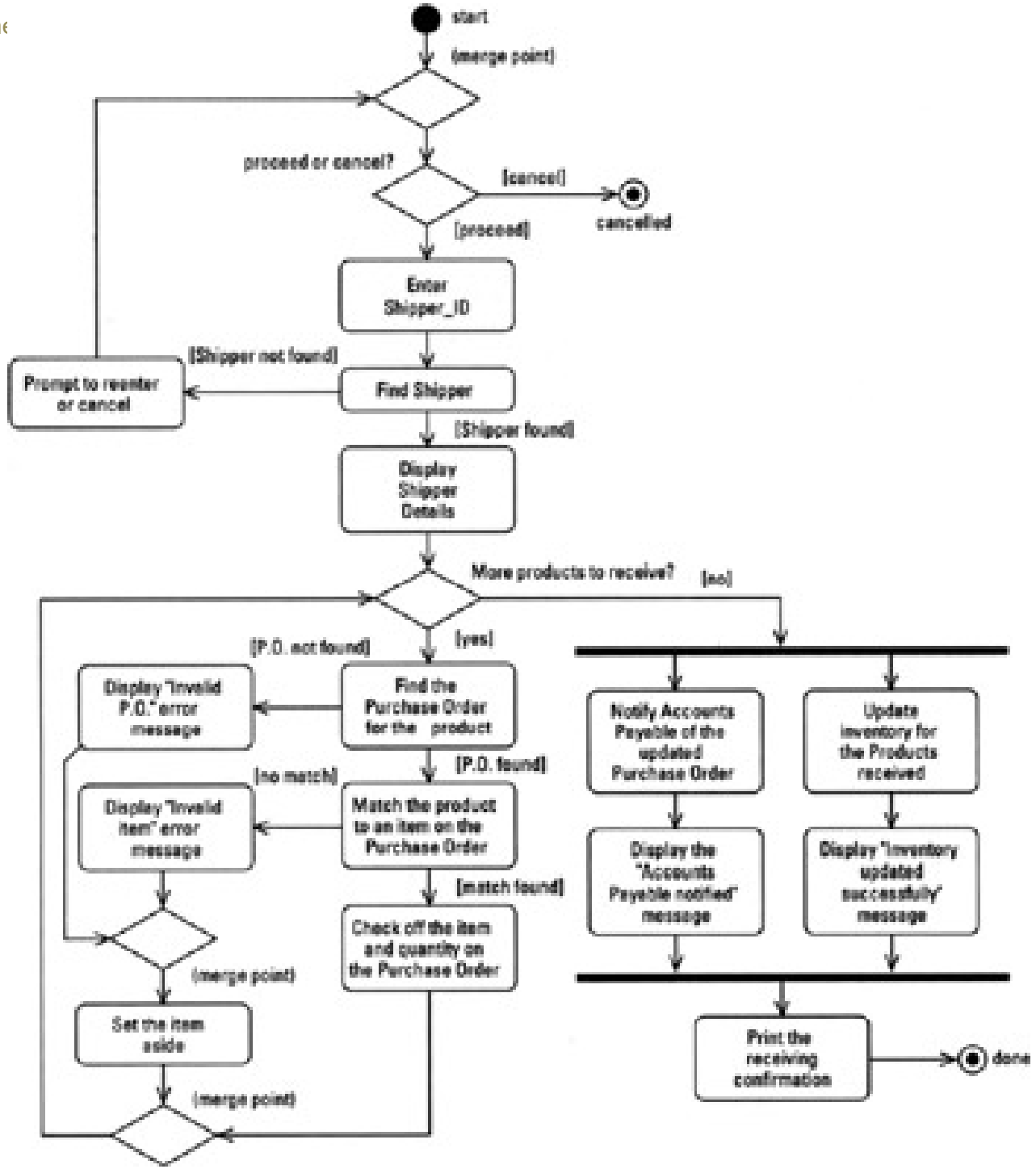
Indications pour l'identification de classes et objets dans les cas d'utilisation de système

- Tout cas d'utilisation est réalisé par une coopération entre certains objets.
- Le même objet peut jouer différents rôles dans des cas d'utilisation différents.
- La responsabilité d'un objet est l'intégration de la somme de ces rôles.

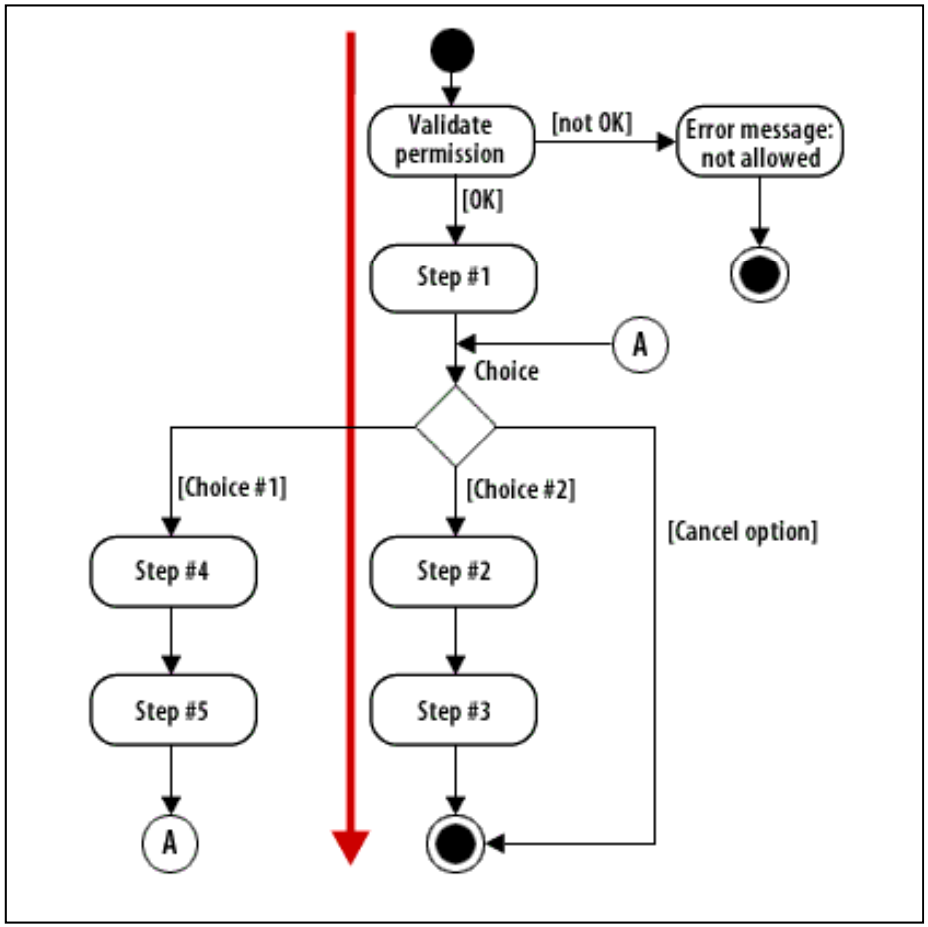
Example de cas d'utilisation

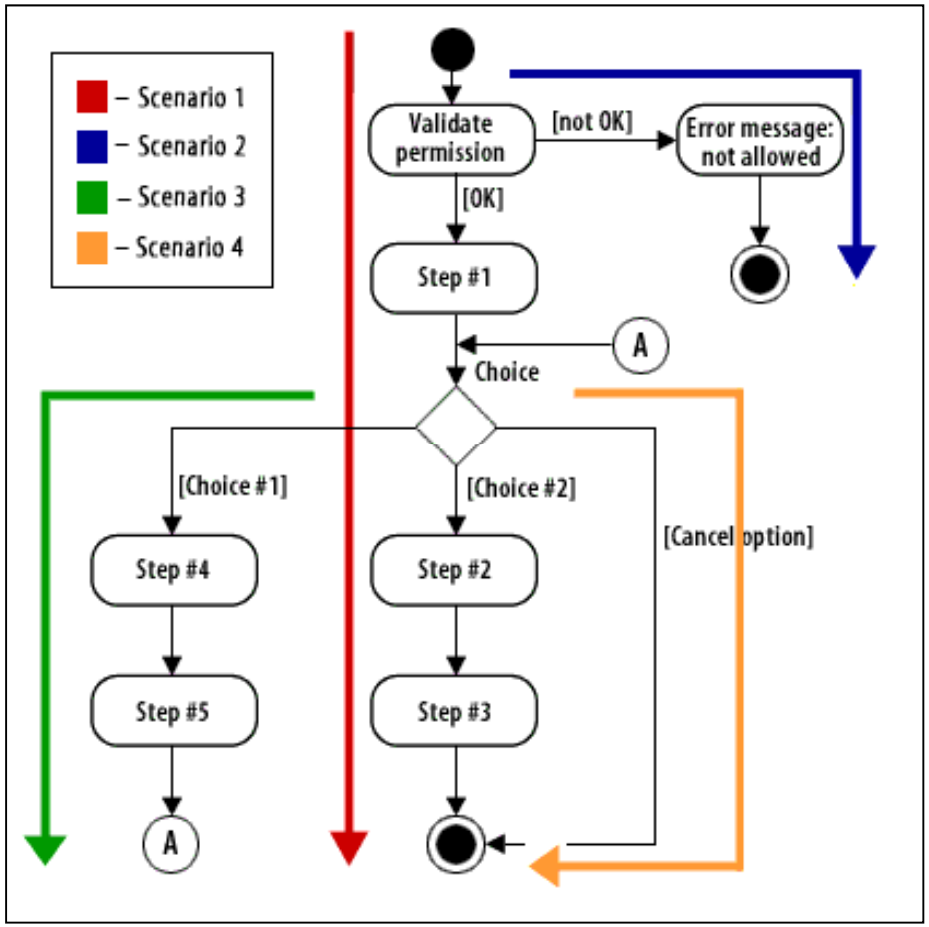
Use case Name	Receive Product
Number	2.0
Author	Joe Analyst
Last update	01/01/03
Assumptions	The user has authority to use this transaction
Preconditions	Shipment is from a valid shipping company
Use case initiation	This use case starts on demand
Use case dialog	The user enters the shipper identification number (ShipperID) If not found, display the "Shipper not found" error message Prompt to re-enter or cancel Else (found) proceed Display the Shipper details Enter the Shipment details (date, sender, number of pieces) For each product in the Shipment Find the Purchase Order using the PO_nbr If not found, display "invalid P.O." error message and set the item aside Else (P.O. found) Match the product to an item on the Purchase Order If not matched, display "invalid item" error message and set the item aside Else check off the item and qty on the Purchase Order Notify Accounts Payable of the updated Purchase Order Display the "Accounts Payable notified" message And Update inventory for the items Products received Display "inventory updated successfully" message Print the receiving confirmation
Use case termination	This use case ends when: The system displays the message that it could not find the Shipping company and the user cancels Accounts Payable has been notified and the inventory has been updated The user cancels the transaction
Post conditions	Upon successful completion of the transaction: Update inventory Notify Accounts Payable Print a confirmation to go with the receiving documents Upon receiving the Cancel option: Returns to its initial state prior to this use case

Description d'un cas d'utilisation



Trouver les scénarios dans un diagramme d'activité





- On peur deriver les cas de test à partir des scenarios.

Diagrammes de séquences de système

SSDs font partie du modèle de cas d'utilisation. Ils sont une visualisation des interactions implicites dans les scénarios de cas d'utilisation.

SSD sont aussi très utile lorsque vous voulez comprendre l'interface et les collaborations entre systèmes existants ou pour documenter l'architecture.

Comportement de systèmе

Use cases describe how external actors interact with the software system we are interested in creating. During this interaction an actor generates system events to a system, usually requesting some system operation to handle the event.

Basically, a software system reacts to three things:

1. external events from actors (humans or computers),
2. timer events, and
3. faults or exceptions (which are often from external sources).

It is useful to know what, precisely, are the external input events the system events. They are an important part of analyzing system behavior.

S system event initiates an operation upon the system.

System events should be expressed at the abstract level of intention rather than in terms of the physical input device.

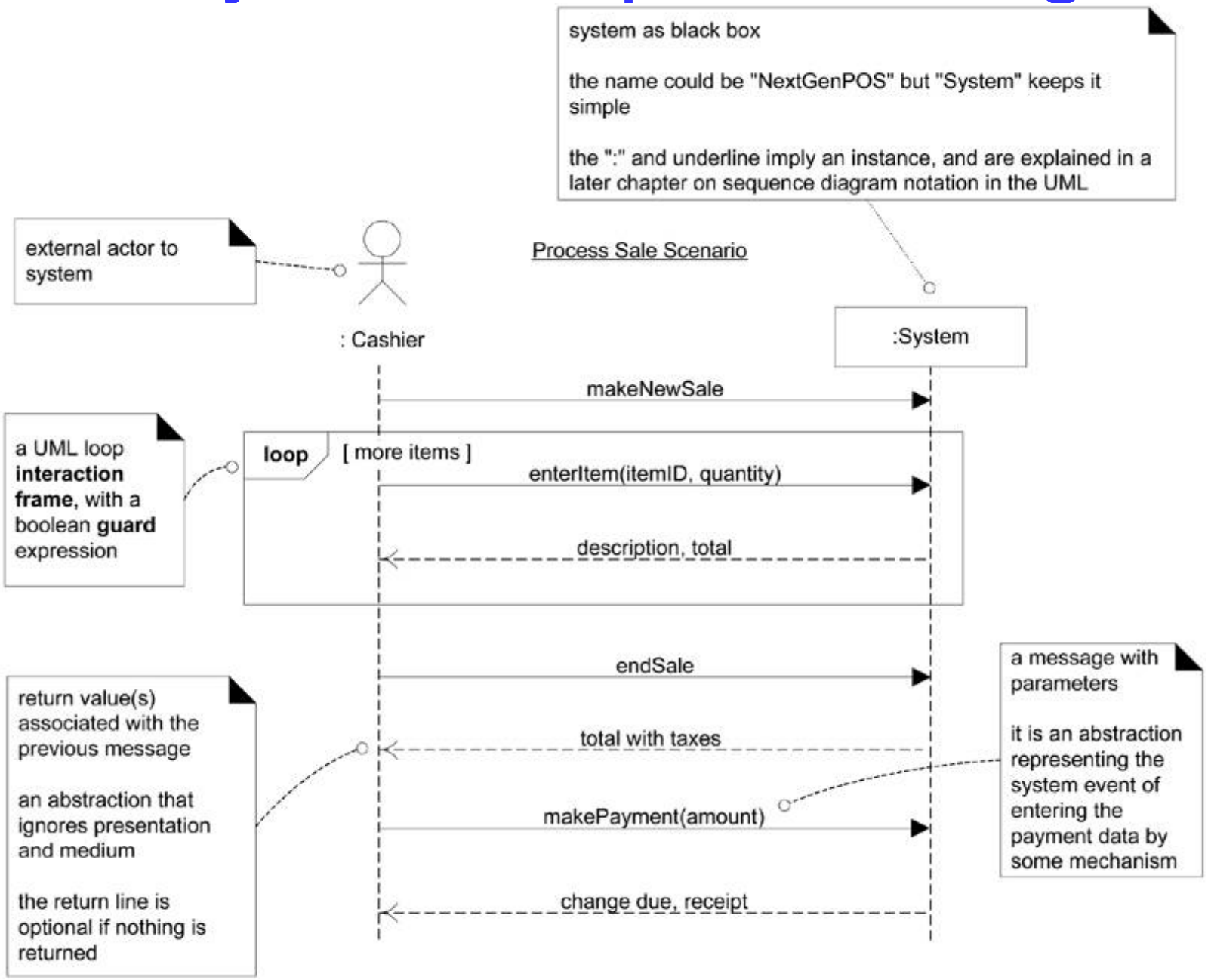
Thus "enterItem" is better than "scan" (that is, laser scan) because it captures the intent of the operation while remaining abstract and noncommittal with respect to design choices about what interface is used to capture the system event.

SSD-System Sequence Diagram

- Use cases describe how external actors interact with the software system we are interested in creating. During this interaction an actor generates system events to a system, usually requesting some system operation to handle the event.
- An SSD is a picture that shows, for one particular scenario of a use case, the events that external actors generate, their order, and inter-system events. All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems.
- An SSD shows, for a particular course of events within a use case, the external actors that interact directly with the system, the system (as a black box), and the system events that the actors generate. Time proceeds downward, and the ordering of events should follow their order in the scenario.
- The example is for the main success scenario of a cash-only Process Sale scenario. It indicates that the cashier generates makeNewSale, enterItem, endSale, and makePayment system events.

Draw an SSD for a main success scenario of each use case, and frequent or complex alternative scenarios.

POS: System Sequence Diagram

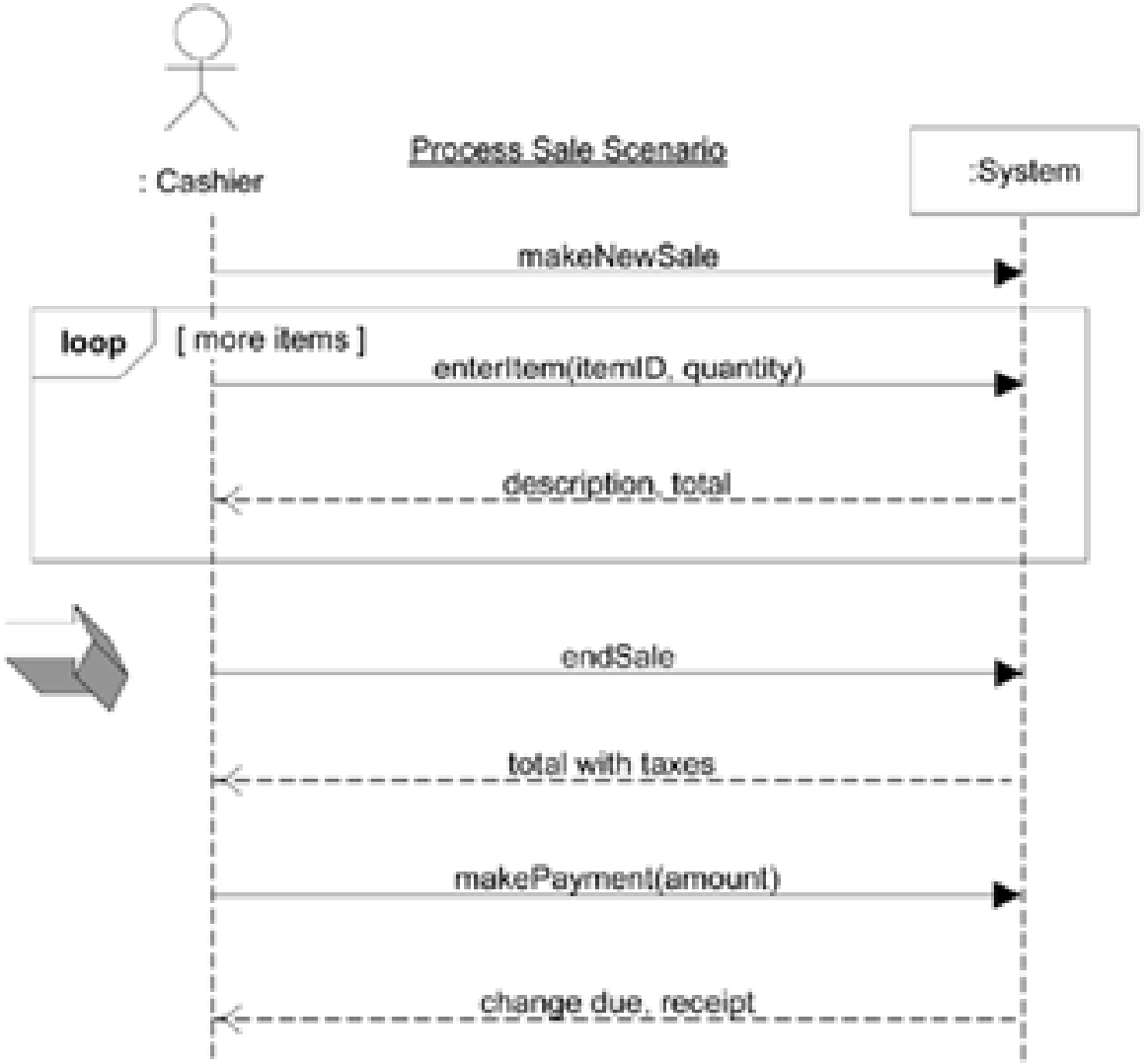


Relationship Between SSDs and Use Cases

- An SSD shows system events for one scenario of a use case, therefore it is generated from inspection of a use case

Simple cash-only Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...



Contrats d’opération

Contrat d'opération

- Use cases or system features are the main ways in the UP to describe system behavior, and are usually sufficient. Sometimes a more detailed or precise description of system behavior has value. Operation contracts use a pre- and post-condition form to describe detailed changes to objects in a domain model, as the result of a system operation.
- The prime inputs to the contracts are the system operations identified in SSDs , the domain model, and domain insight from experts. The contracts can in turn serve as input to the object design, as they describe changes that are likely required in the software objects or database.

Exemple

Opération:	saisirArticle(codeArticle: CodeArticle, quantité: Entier)
Références croisées:	Use Cases: Traiter une vente
Préconditions:	Il éxiste une vente en cours.
Postconditions:	1. Une instance *lar* de LigneArticles a été créée (création d'instance) 2. *lar* a été associée à la vente en cours (formation d'association). 3. *lar.quantité* est devenue *quantité* (modification d'atribut) 4. *lar* a été associé à DéscriptionProduit sur la base de la correspondance avec codeArticle (formation d'association)

System Operations

- Operation contracts may be defined for system operations.
- System operations = operations that the system as a black box component offers in its public interface. System operations can be identified while sketching SSDs.
- The SSDs show system events or I/O messages relative to the system. Input system events imply the system has system operations to handle the events, just as an OO message (a kind of event or signal) is handled by an OO method (a kind of operation).
- The entire set of system operations, across all use cases, defines the public *system interface*, viewing the system as a single component or class.

Postconditions

- Postcondition = description ofchanges in the state of objects in the domain model. Domain model state changes include instances created, associations formed or broken, and attributes changed. they are observations about the domain model objects that are true when the operation has finished .
- postconditions fall into these categories:
  - Instance creation and deletion.

```
A SalesLineItem instance sli was created .
```

  - Attribute change of value.

```
sli.quantity became quantity .
```

  - Associations (to be precise, UML links) formed and broken.

```
sli was associated with the current Sale.
sli was associated with a ProductDescription, based on itemID match .
```

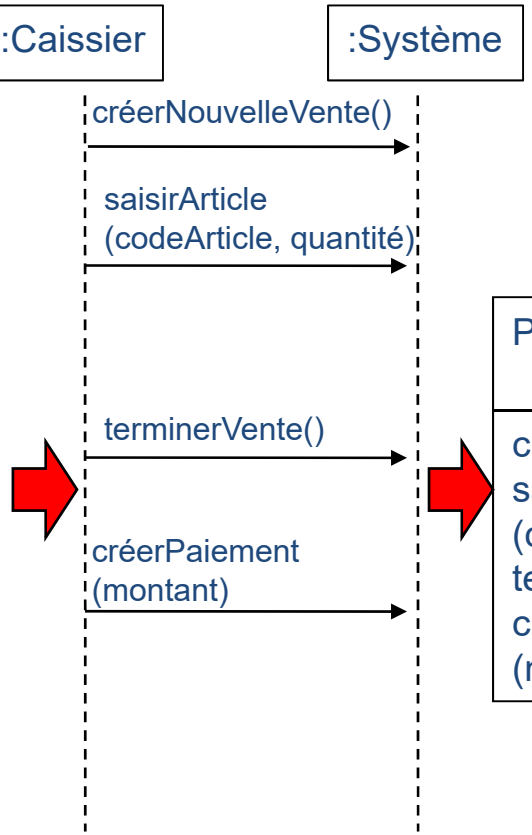
To create contracts:

1. Identify system operations from the SSDs.
2. For system operations that are complex and perhaps subtle in their results, or which are not clear in the use case, construct a contract.
3. To describe the postconditions, use the following categories:
   - instance creation and deletion
   - attribute modification
   - associations formed and broken

Dès cas d'utilisation aux contrats d'opération

Cas d'utilisation:
Traiter une Vente
Flux principale d'évènements
1. Le Client arrive à la caisse …..

2. Le Caissier débute une vente ….



POS

créerNouvelleVente()
saisirArticle
(codeArticle, quantité)
terminerVente()
créerPaiement
(montant)

Opération:
saisirArticle()

Postconditions:
1. Une instance *lar* de LigneArticles a été créée (création d'instance)
2. *lar* a été associée à la vente en cours (formation d'association).
3. *lar.quantité* est devenue *quantité* (modification d'atribut)
4. *lar* a été associé à DéscriptionProduit sur la base de la correspondance avec codeArticle (formation d'association)

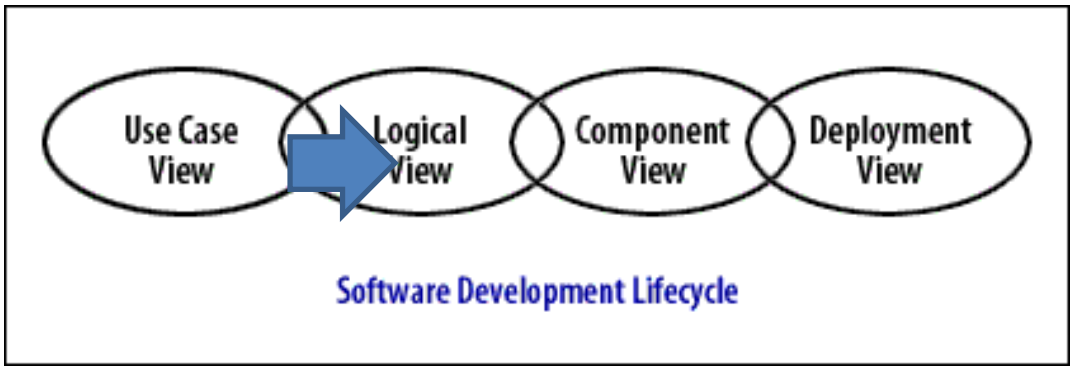
Opération:
terminerVente

Postconditions:
1.
…..

Cas d'utilisation | **Diagramme de Séquence Système** | **Système** | **Contrat**

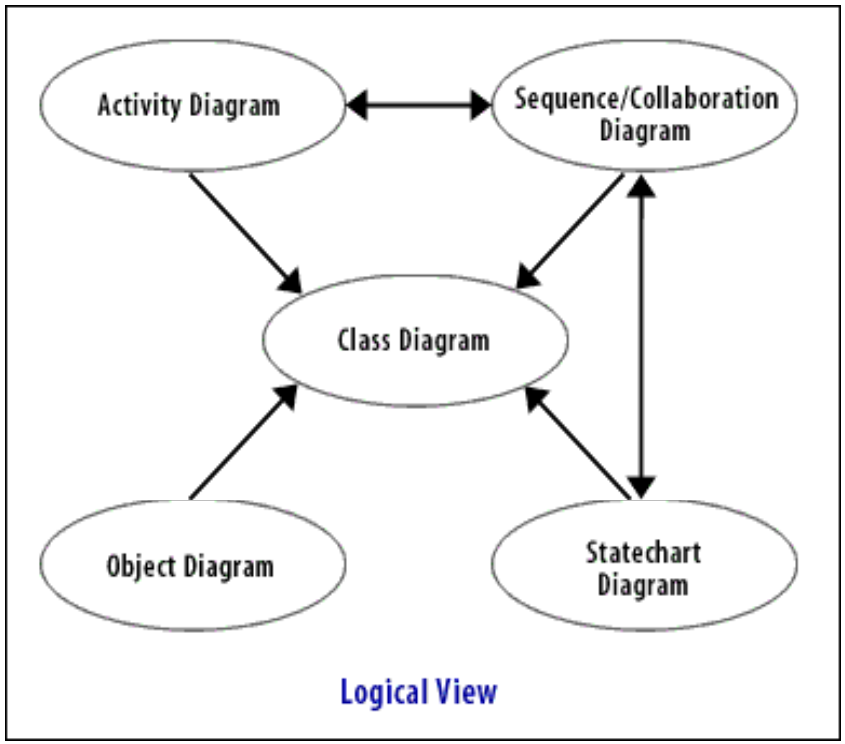
FIN DU QUATRIÈME CHAPITRE

Chap. 5. Analyse de domaine



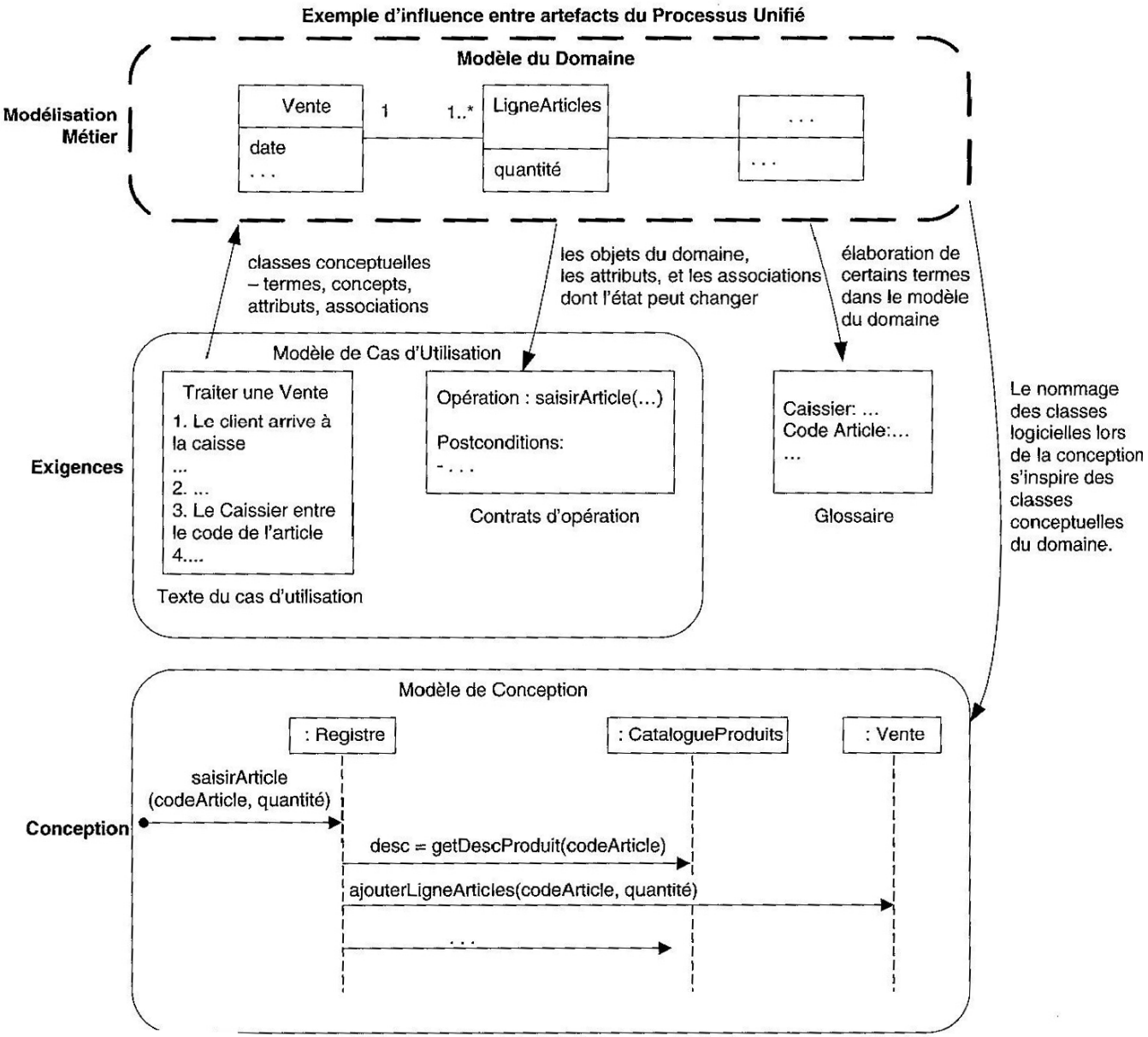
L'objectif d'analyse de domaine est de définir le but et la structure de chaque ressource de domaine du problème.

Vue logique de la modélisation



1. Le diagramme primaire est le diagramme de classes, qui est la fondation pour la modélisation des objets, la source pour la génération de code, et la cible pour reverse engineering.
2. Le diagramme d'objets illustre des objets au lieu de des classes. Le diagramme d'objet peut être utilisé pour tests ou pour comprendre un diagramme de classe.
3. Pour modeler des interactions entre objets, la vue logique inclut les diagrammes de séquences et de communication. Ceux-ci sont deux vues très similaires pour l'identification des interfaces e la communication des objets.
4. Pour modeler le comportement interne d'objet, le diagramme statechart fournit un regarde comment un objet réagit aux stimuli externes et gère ses changements internes.
5. Le diagramme d'activitées décrit des processus y compris la séquence, la décision, et la concurrency. Ce diagramme est similaire à un organigramme, mais il a été amélioré pour l'usage avec la modélisation des objets.

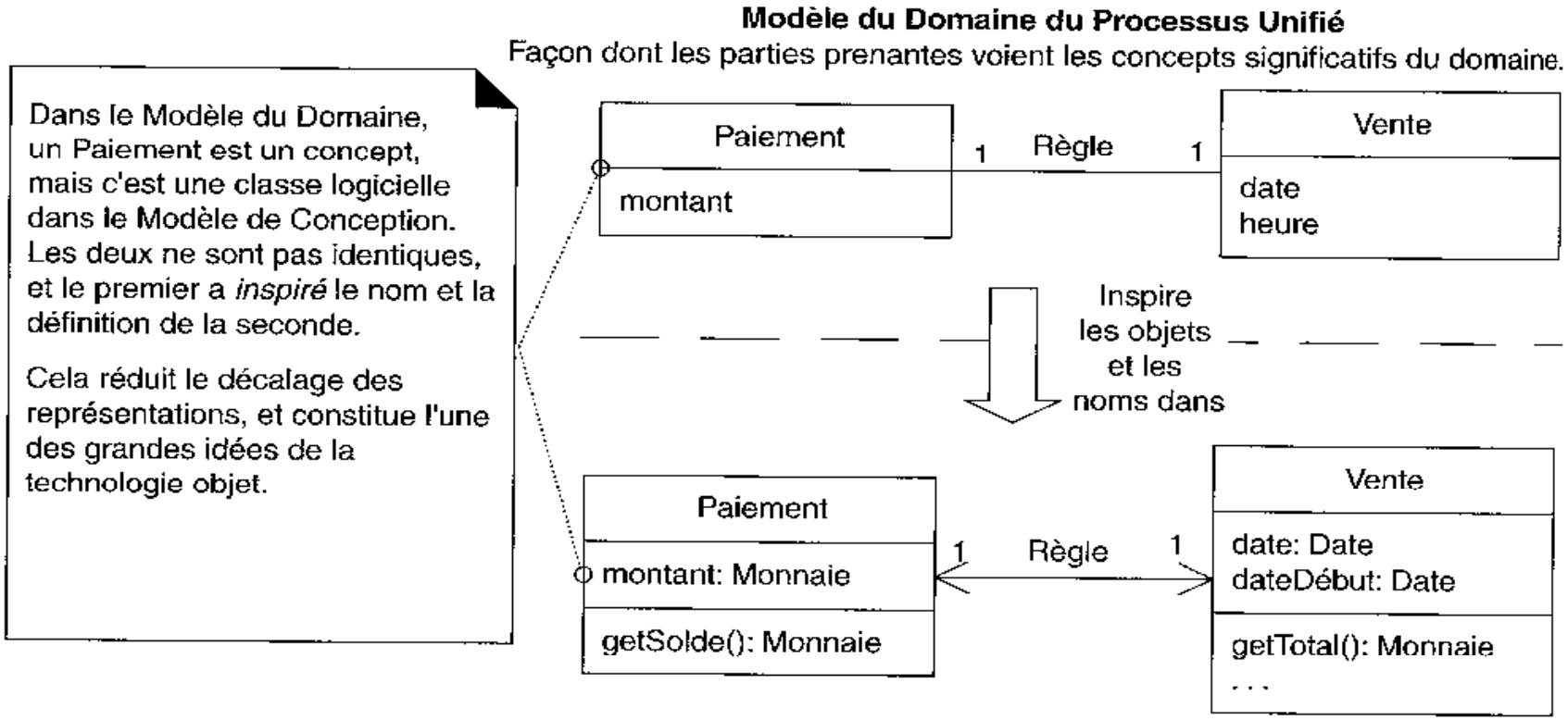
Influence entre artefacts du Processus Unifie.



Qu'est-ce qu'un modèle du domaine?

- La modélisation du domaine joue un rôle central dans la compréhension de l'actuel environnement du système et la planification des activités pour la création du système future.
- La quintessence de l'étape d'analyse orientée objet réside dans la décomposition d'un domaine en concepts fondamentaux ou objets.
- Un modèle du domaine est un modèle d'objet métier « qui se concentre sur l'explication des entités et des produits importants pour un domaine métier.» Le modèle du domaine est une spécialisation du modèle des objets métier. Il est un des artefacts qui peuvent être crées dans la discipline Modélisation métier: le BOM est un modèle étendu, couvrant plusieurs domaines, généralement de grande envergure et difficile a créer puisqu'il englobe la totalité d'un métier et de ses sous-domaines.
- L'identification d'un riche ensemble de classes conceptuelles est au cœur de l'analyse orientée objet.
- Le modèle du domaine *ne signifie pas*:
  - une série de diagrammes qui décrivent des classes de logiciel,
  - la couche de domaine d'une architecture de logiciel, ou
  - un objets de logiciel avec les responsabilités.
- Un modèle du domaine n'est pas un modèle de données (c'est-à-dire la définition des objets persistants stockés quelque part).

Réduction du décalage entre le monde réelle et les représentations avec la modélisation objet



Modèle de Conception du Processus Unifié

Le développeur orienté objet s'est inspiré du domaine du monde réel pour créer les classes logicielles.

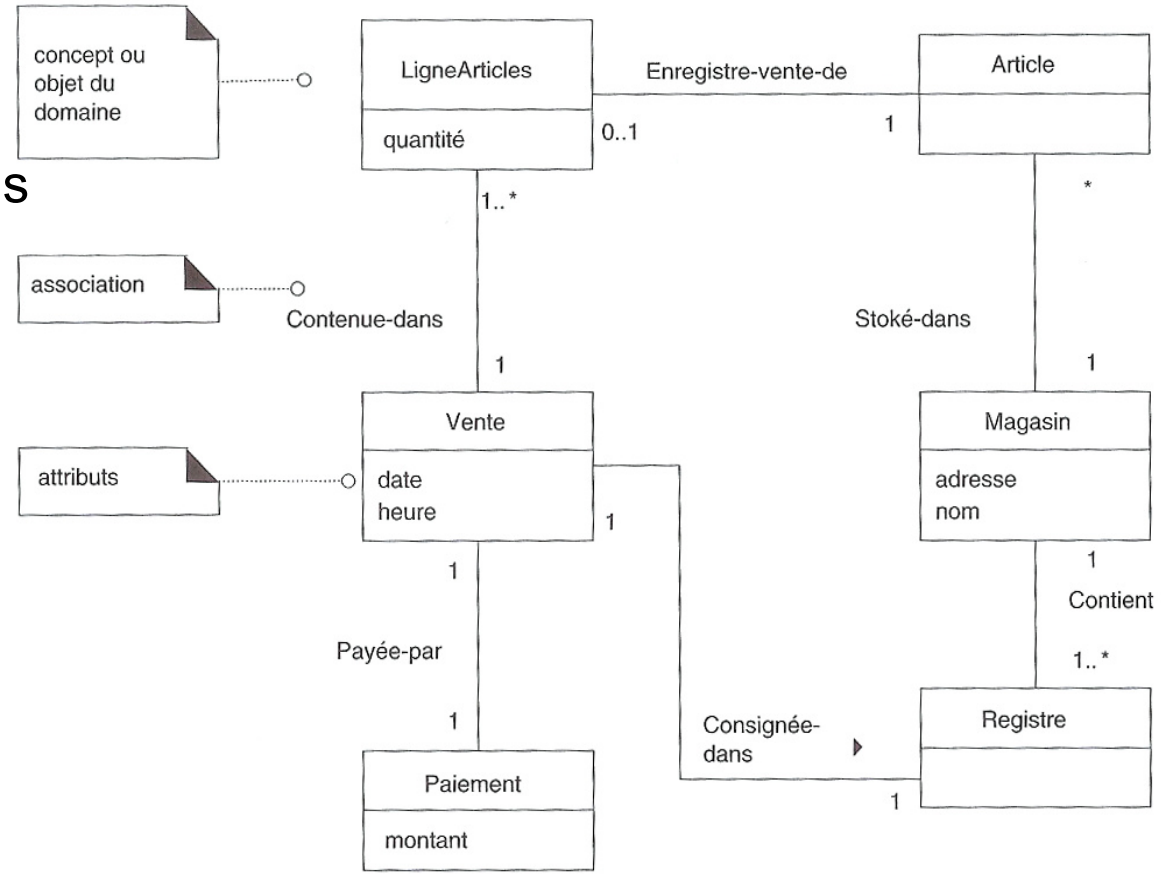
En conséquence, le décalage des représentations entre la façon dont les parties prenantes voient le domaine et sa traduction logicielle a été réduit.

Modèle du domaine (I)

Modèle du domaine (modèle conceptuel, modèle des objets du domaine et modèle des objets d'analyse) = une représentation des classes conceptuelles (une idée, une chose ou un objet appartenant au monde réelle), **sans inclure les objets du logiciel**.

Un modèle de domaine est illustré avec une série de diagrammes de classes dans laquelle les opérations ne sont pas définis.
Il fournit une perspective conceptuelle et représente les éléments suivants :

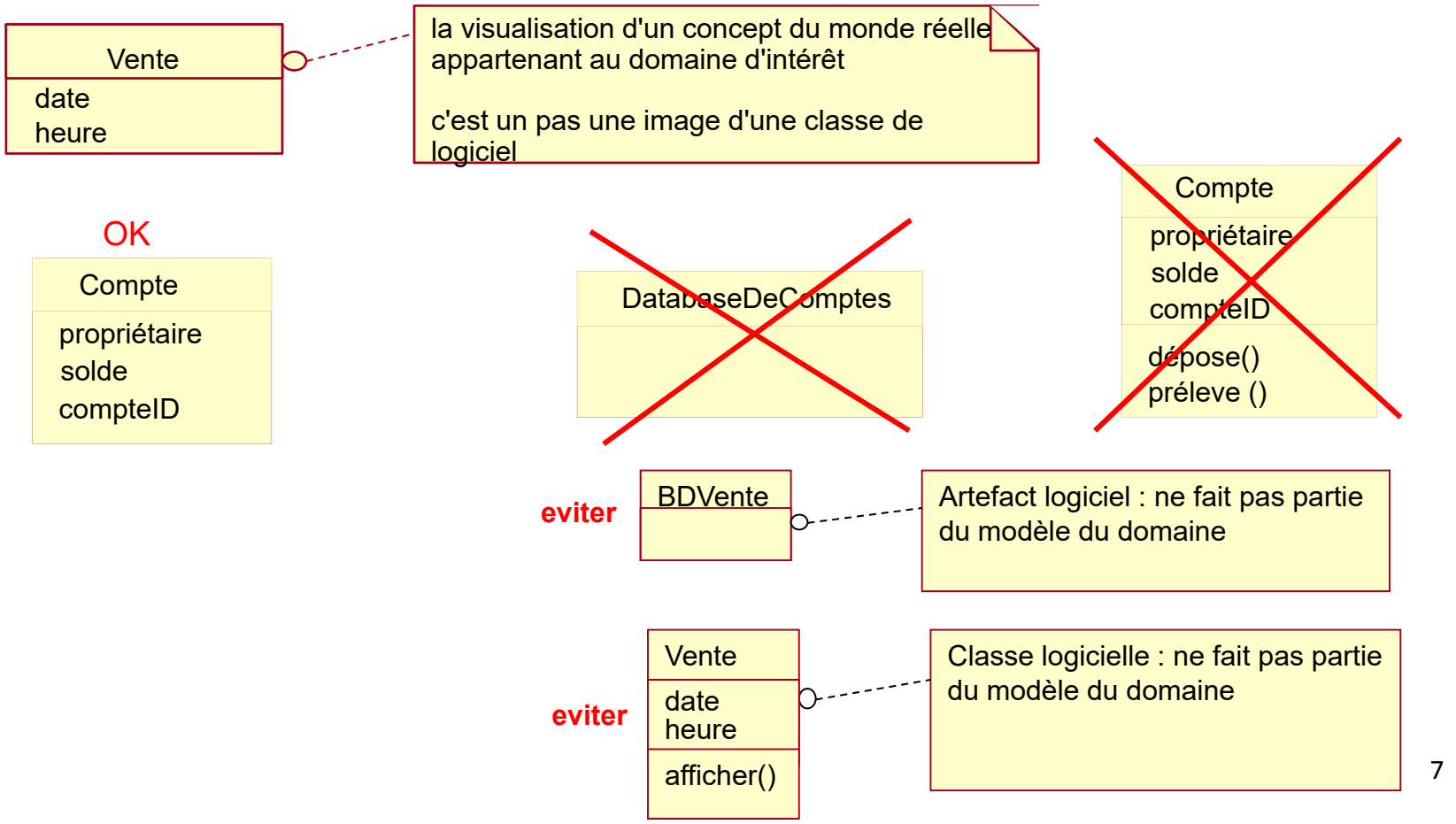
- les classes conceptuelles ou les objets du domaine;
- les associations entre classes conceptuelles;
- les attributs des classes conceptuelles.



Modèle du domaine (II)

Q. Que qualifie un objet pour devenir une classe de domaine de problème ?
R. Le critère fondamental est que les utilisateurs doivent reconnaître l'objet comme une ressource.

- Une ressource est quelque chose utilisée par les acteurs. Une ressource pourrait être acquise, maintenu, utilisé, probablement fait et vendu et liquidé.



Comment créer un modèle du domaine ?

Une procédure avec trois activités:

1. Identifier les classes conceptuelles (voir le principe suivant).
2. Représenter les classes conceptuelles sous forme de classes dans un diagramme UML.
3. Ajouter des attributs et des associations.

1. Comment identifier les classes conceptuelles ?

a. réutiliser ou modifier des modèles existants (excellent méthode!)
b. utiliser une liste de catégories
c. identifier les classes conceptuelles à l'aide de groupes nominaux.

Liste de catégories de classes conceptuelles

Catégorie de classe conceptuelle	Exemples
1.transactions métier Principe : Elles sont critiques : débutez par les transactions.	Vente, Paiement Réservation LigneArticles
2.lignes d'une transaction Principe : Souvent le début d'une transaction, à considérer juste âpres les transactions.	
3.produit ou service lle a une transaction ou a une ligne d'une transaction Principe: Les transactions ont lieu pour quelque chose, à considérer juste âpres les lignes d'une transaction.	Article, Vol Siège, Repas
4.où la transaction est-elle enregistrée ? Principe : Important.	Registre, GrandLivre, ManifesteDeVol
5.rôle des personnes ou des organisations liées à la transaction ; acteurs dans les cas d'utilisation Principe: Nous avons habituellement besoin de connaitre les parties impliquées dans une transaction.	Caissier, Client, Magasin, Passager CompagnieAérienne
6.lieu de la transaction ; lieu du service	Magasin, Aéroport, Avion, Siege
7.événements notables, souvent a un moment ou dans un lieu que vous devez mémoriser	Vente, Paiement Vol
8. objets physiques Principe.·Particulièrement important lorsque nous devons réaliser des logiciels de contrôle d'équipements divers ou effectuer des simulations.	Article, Registre
9.descriptions d'entités	DescriprionProduit, DescriptionVol
10.catalogues Principe : Les descriptions se trouvent souvent dans des catalogues.	CatalogueProduits CatalogueVols
11.conteneurs (d'objets physiques ou d'informations)	Magasin, Rayonnage, Plateau, Avion
12.contenu d'un conteneur	Article, Passager
13.autres systèmes externes	SystémeAutorisationPaiementsACrédit ControleDuTraficAérien
14.documents financiers, contrats, documents légaux	Reçu, GrandLivre,JournalDeMaintenance
15.instruments financiers	Espèces, Cheque, i.igneDeCrédit
16.plannings, manuels, documents régulièrement consultés pour effectuer un travail	MiseAJourTarifs PlanningRéparations

Identifier les classes conceptuelles à l'aide de groupes nominaux

Pas 1 : Trouver les noms

– Lire la description du problème (problem statement), les cas d'utilisation et la documentation associée et soulignent les noms;
– Traduire les noms en classes ou objets, attributs ou valeurs d'attribut. De la même manière, les verbes révèlent souvent des opérations ou des associations.

Pas 2: Evaluer les noms pour trouver des classes

Questions pour demander d'évaluer des noms pour trouver des classes:

1. Le terme est-il eu exigée dans l'espace occupé par (étendue) du projet ?
2. Si oui, qualifie alors le nom comme une ressource de domaine du problème?
3. Autrement, le nom est-il un exemple d'une classe, par exemple, un objet, une instance d'une classe? Quelle classe? La notez.
4. Autrement, est-il le nom d'un attribut d'une classe? Quelle classe? La notez.
5. Autrement, le nom est-il un exemple d'un attribut, une valeur d'attribut ? Notez ce fait et réfère à lui quand vous devez tester les données.

Pas 3: Définir le but

– Décrire pourquoi la classe est une ressource exigée du domaine du problème.

Identification des concepts avec la description du problème

« Notre auditorium est utilisé pour différents types d'événements. Certains des événements sont des événements uniques, non répétables, comme les jeux de basketball. Les autres événements consistent en un nombre de spectacles, comme les jeux ou comme les concerts. »

1. Souligner les noms :

   « Notre auditorium est utilisé pour différents *types* d'événements. Certains des événements sont des *événements* uniques, non répétables, comme les jeux de basketball. Les autres *événements* consistent en un nombre de spectacles, comme les jeux ou comme les concerts. «

2. Construire une table comme ceci :

Candidate Term	Not a class because	Class Name	Class Purpose
Auditorium		Auditorium	The facility in which the events are held and for which tickets are sold
Event		Event	A performance negotiated by the event planner and presented in one or more shows
Basketball game	it is a type of event		
Show		Show	A single presentation of an event
Play	it is a type of event		
Concert	it is a type of event		

Identification des concepts avec les cas d'utilisation

Scenario principal (succès) :

1. Le **Client** arrive a la **caisse** avec des **marchandises** et/ou des **services** a acheter.
2. Le **Caissier** commence une **vente**.
3. Le **Caissier** entre le **code de l'article**.
4. Le Système enregistre **l'article** et présente sa **description**, son **prix** et le **total** en cours. Le prix est calcule a partir d'un ensemble de règles.

Le Caissier répète les étapes 3 et 4jusqu'd· ce que tous les articles soient valides.

5. Le Système présente le total incluant les **taxes** calculées.
6. Le Caissier communique le montant total au Client et en demande le **paiement**.
7. Le Client paie et le Système traite son règlement.
8. Le Système enregistre la **vente** et transmet les informations la concernant et celles concernant le règlement au système de **Comptabilité** externe (pour la mise a jour de la comptabilité et des **commissions**) et au système de **Gestion de stock** (pour la mise a jour des quantités en stock).
9. Le Système génère un **reçu**.
10. Le Client quitte le point de vente avec ses achats et son **reçu**.

Extensions (ou scenarios alternatifs) :

7a. Paiement en espèces :

1. Le Caissier saisit le **montant reçu** en espèces.
2. Le Système affiche le **solde dû** et ouvre le **tiroir-caisse**.

3- Le Caissier encaisse le montant et rend la monnaie en espèces au Client.

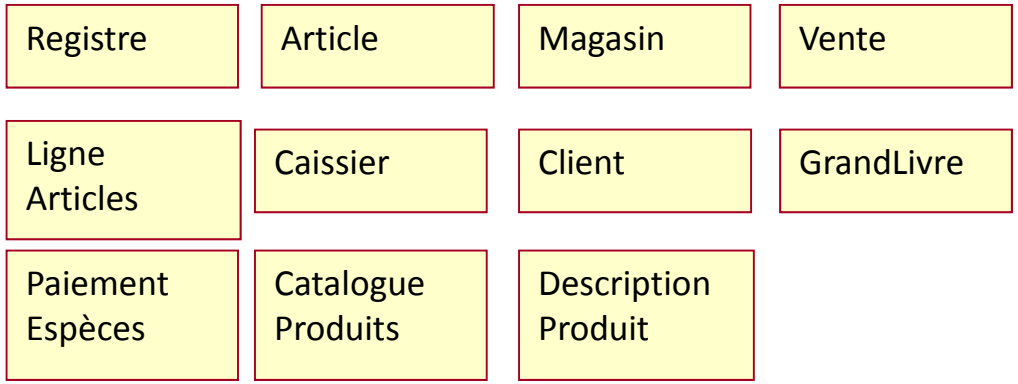
4. Le Système enregistre le paiement en espèces.

Identifier et représenter les classes conceptuelles

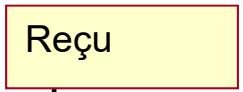
La liste clé catégories et l'analyse des groupes nominaux permettent de générer une liste des classes conceptuelles possibles pour ce domaine:

Vente
PaiementEnEspèces
LigneArticles
Article
Registre
GrandLivre
Caissier
Client
Magasin
DescriptionProduit
CatalogueProduits

Modèle du domaine préliminaire :



Faut-il faire inclure Reçu dans le modèle ?



- Non, parce que est redondante: dérive d`autres sources (Vente et Paiement)
- Oui, parce que il confère à son porteur le droit de retourner les articles.

Additionner associations au modèle

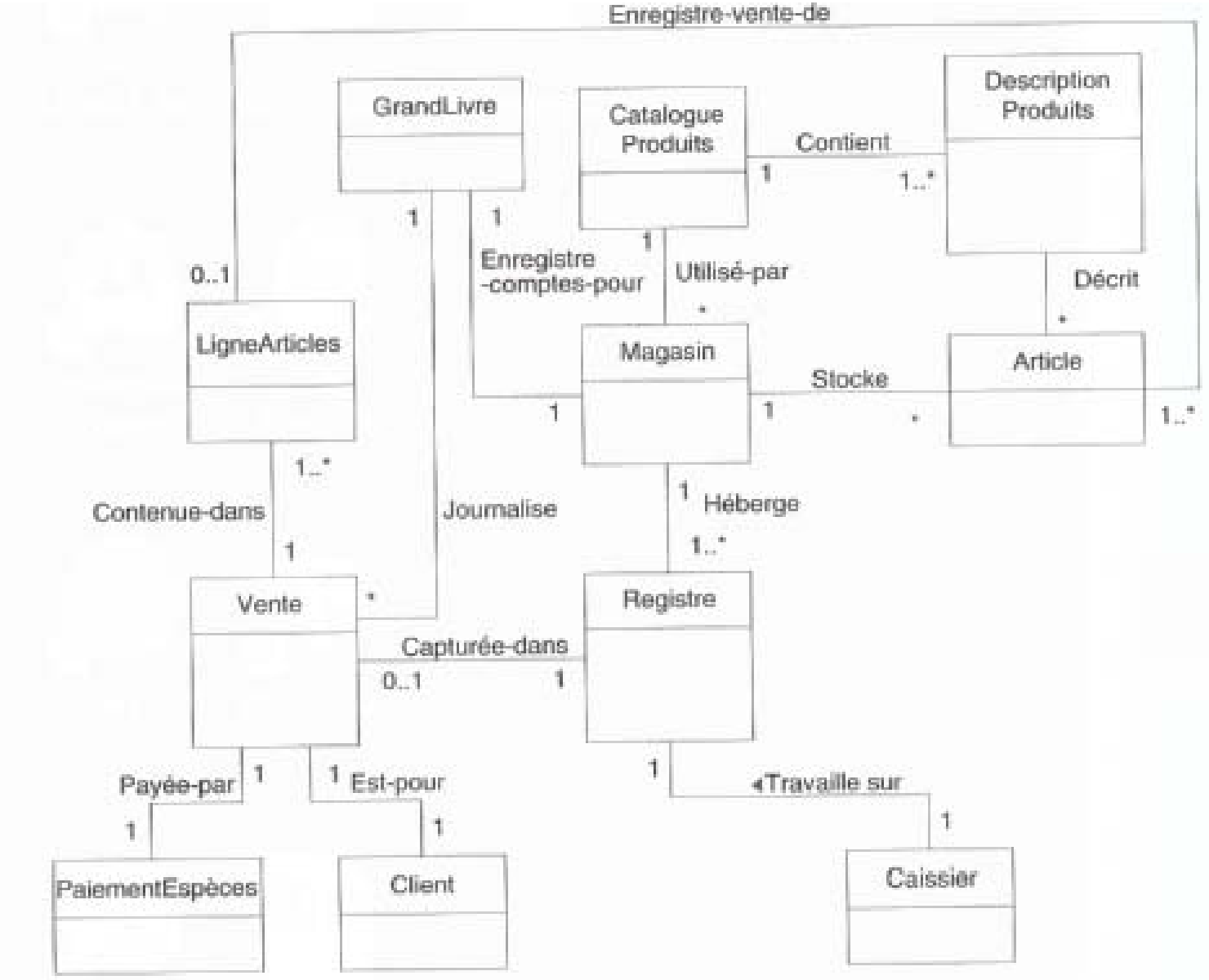
- Considérer dans un modèle de domaine les suivantes associations :
  - Les associations pour qui la connaissance de la relation a besoin d'être conservé pour quelque durée (associations «besoin-de-se-souvenir-de»)
  - Les associations dérivées de la List des catégories d'associations .
- Pendant la modélisation du domaine une association n'est pas une spécification des fluxes de données, des relations avec les «foreign keys» dans la base de données ou connexions d'un objet dans un logiciel ; c'est une déclaration qu'une relation entre deux concepts est significative dans une perspective purement conceptuelle sur le domaine réel .
- Exemples:
  - Une transactions relative à une autre transaction(Vente *Payée-par* PaiementEnEspeces)
  - Ligne-Articles d'une transaction (Vente *Contient* LigneArticles).
  - Produit pour une transaction (ou ligne-article) (LigneArticles *Enregistre-la-vente-de* Article).
- Une communication entre deux objets indique une association entre les deux classes.



Liste d'associations

Catégorie	Exemple
1. A est une transaction liée a une transaction B	PaiementEnEspeces - Vente Reservation - Annulation
2. A est un élément d'une transaction B	LigneArticles - Vente
3. A est un produit pour une transaction (ou un élément de transaction) B	Article - LigneArticles (ou Vente) Vol-Réservation
4. A est un rôle lié a une transaction B	Client - Paiement, Passager - Billet
5. A est une partie logique ou physique de B	Tiroir- Registre, Siège -Avion
6. A est physiquement ou logiquement contenu dans B	Registre - Magasin Passager- Avion
7. A est une description clé B	DescriptionProduit - Article, Description Vol - Vol
8. A est connu/consigné/enregistré/saisi dans B	Vente - Registre, Réservation - ManifesteDeVol
9. A est un membre de B	Caissier – Magasin, Pilote - CompagnieAérienne
10.A est une sous-unité organisationnelle de B	Rayon -Magasin Maintenance -CampagnieAerienne
11.A utilise, gére ou posséde B	Caissier -Registre, Pilote -Avion
12.A est voisin de B	Article – Article, Ville -Ville

Modèle du domaine avec relations



Ajouter Attributs

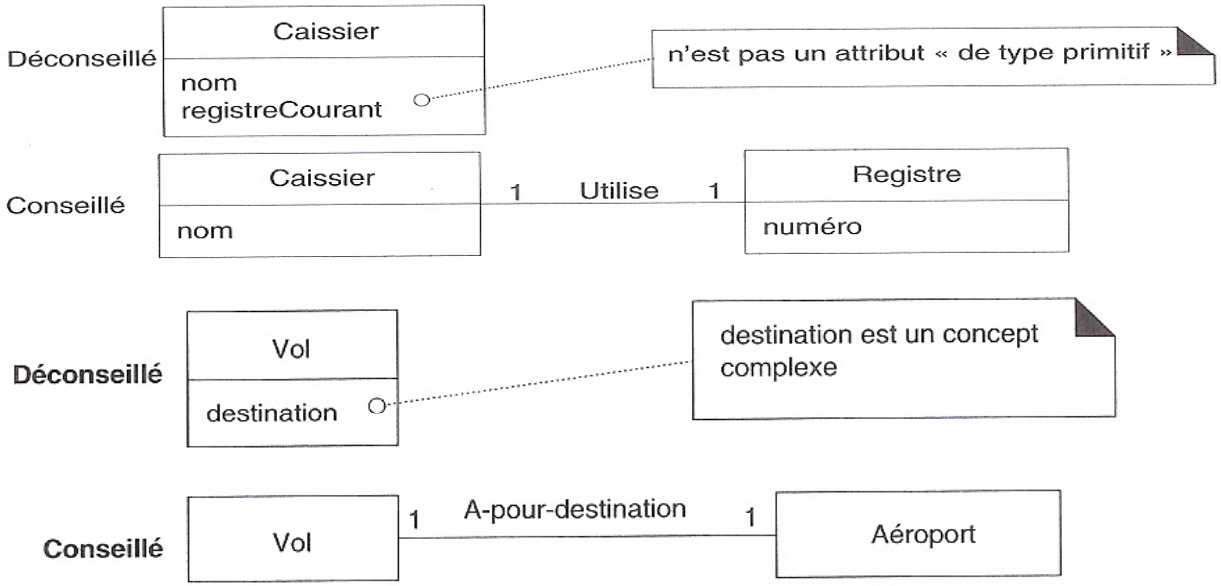
Un **attribut** est la valeur d'une donnée logique d'un objet.

Nous incluons des attributs dans le modèle de domaine quand les exigences (comme les cas d'utilisation) suggèrent la nécessité de mémoriser des informations.

- *Vente* a besoin d'un attribut *dateHeure*.
- *Magasin* a besoin d'un *nom* et d'une *adress*.
- *Caissier* a besoin an *ID*.

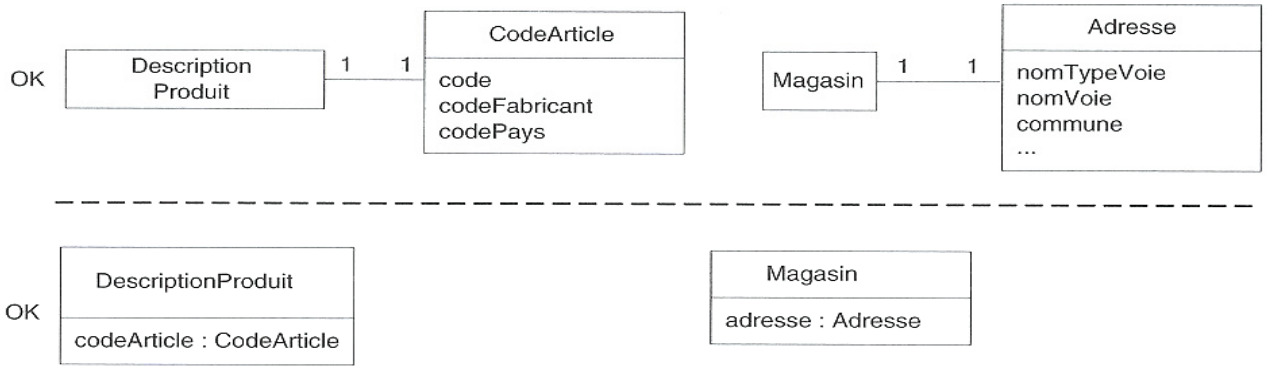
Types d'attributs valides

Normalement, le type d'un attribut ne doit pas être un concept complexe du domaine, comme Vente ou Aéroport. Ils sont types de données « primitifs », par exemple les nombres ou les booléens (non nécessairement ceux des langages de programmation).

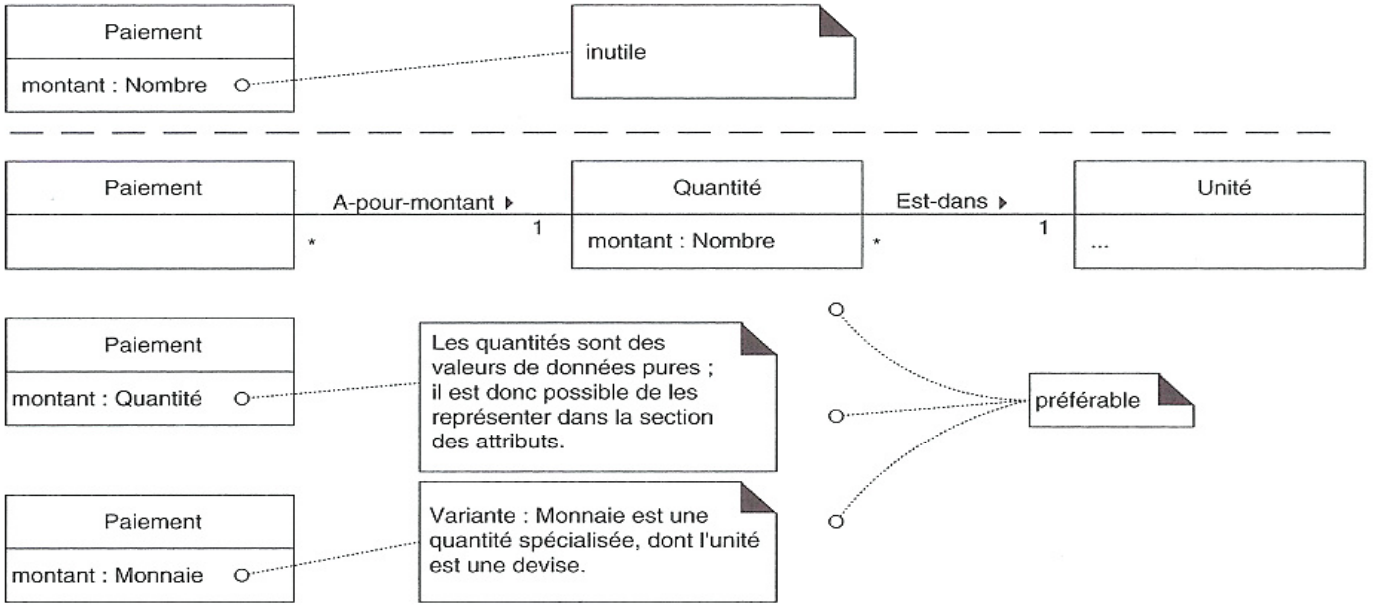


Représenter les attributs (I)

Doit-on représenter la classe CodeArticle comme une classe distincte dans le modèle du domaine ? Tout dépend de ce qu'on veut mettre en valeur dans le diagramme.

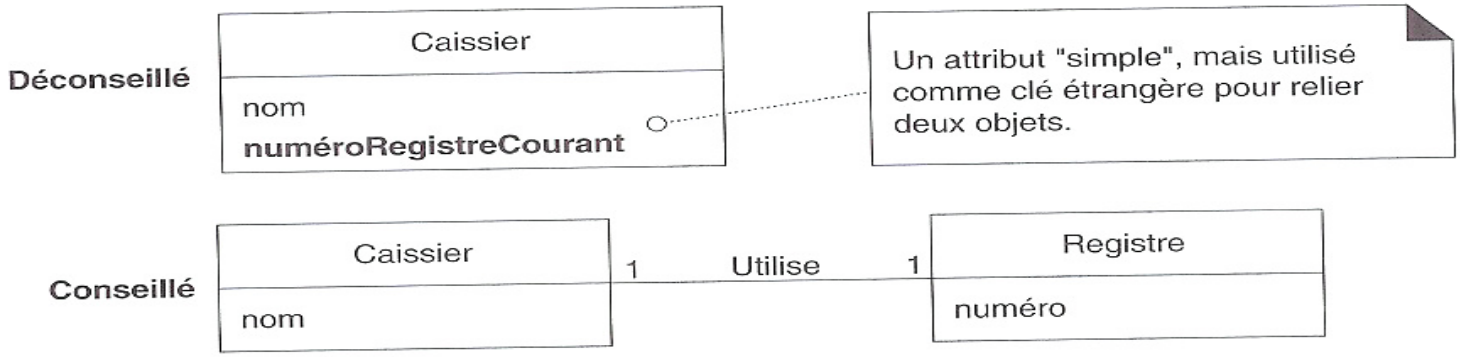


Modélisation des quantités et des unités:



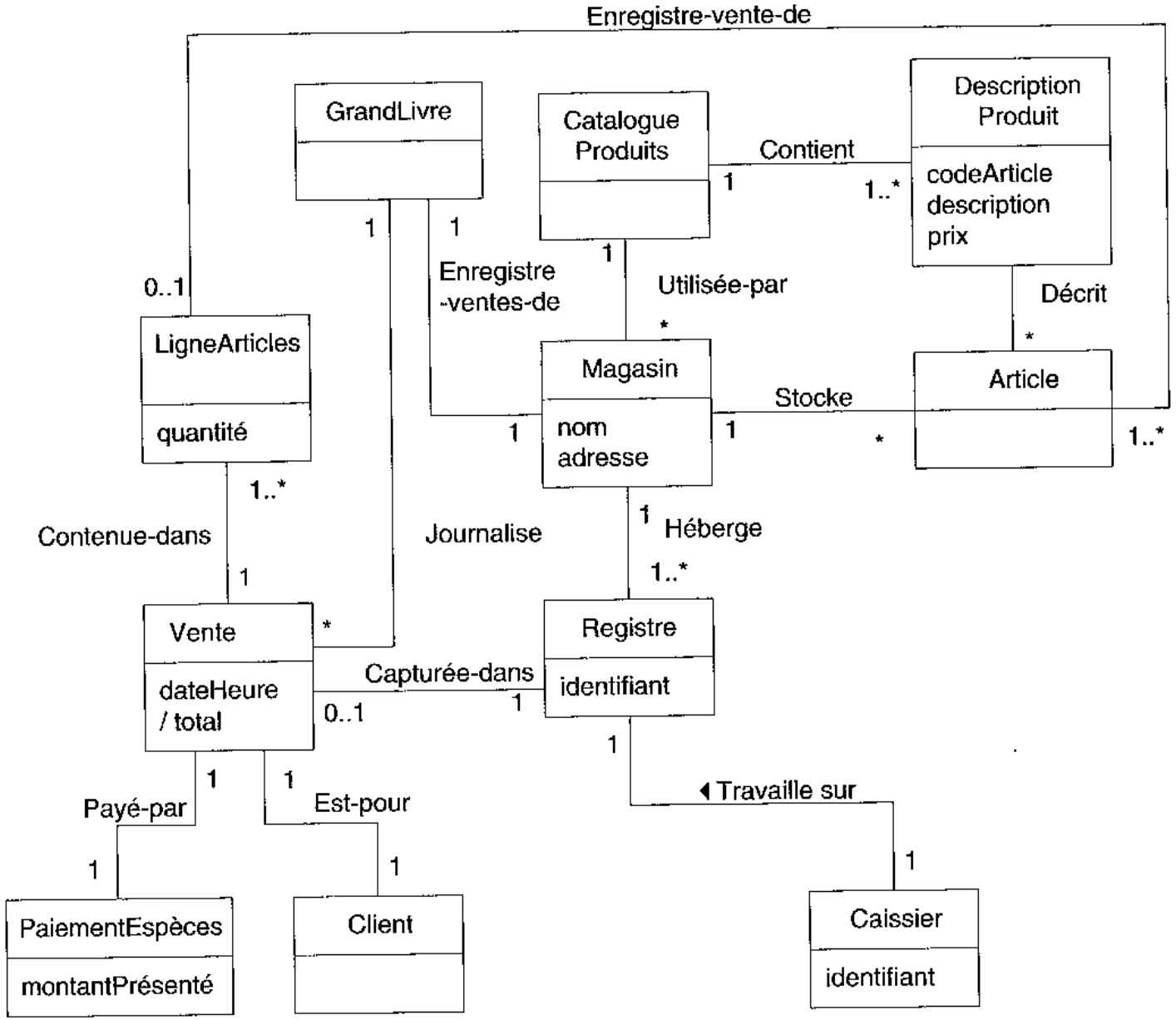
Représenter les attributs (II)

- Pas d'attributs pour représenter les clés étrangères



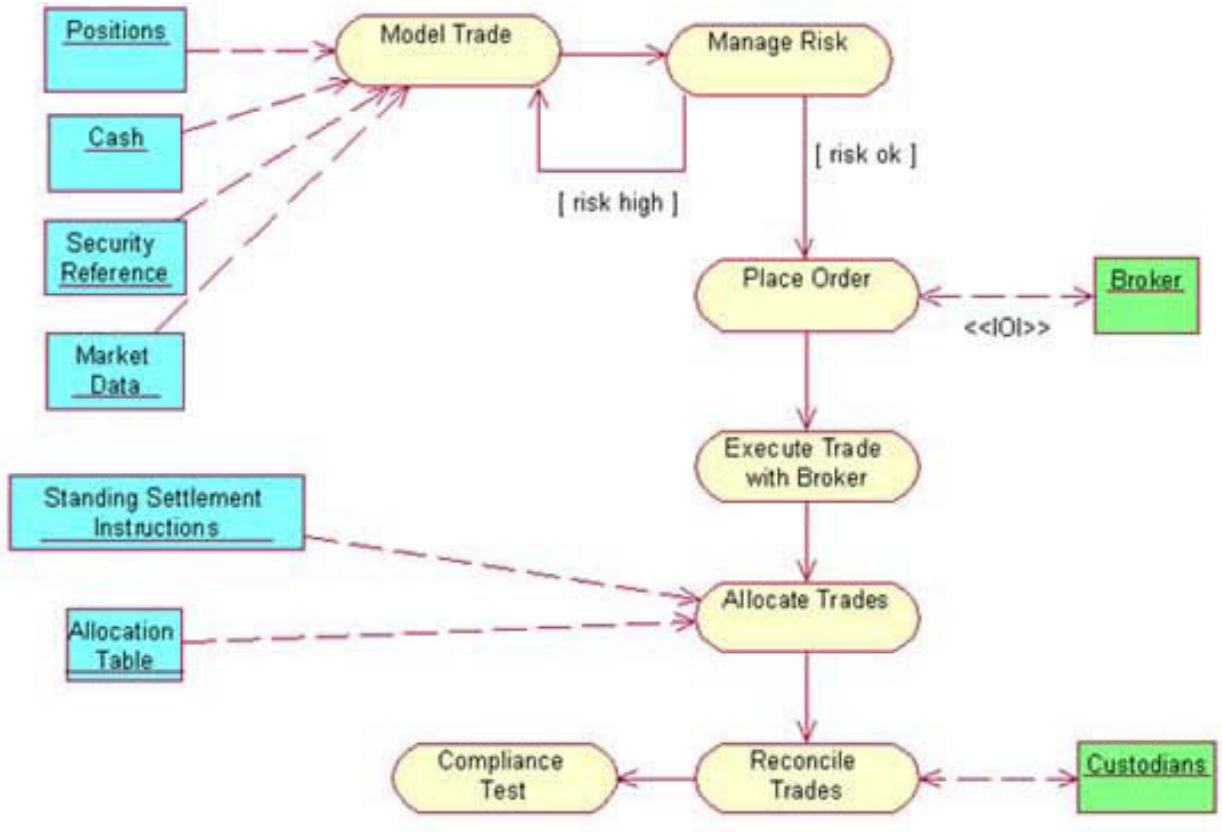
PaiementEnEspéces	*montant* -Pour déterminer si la somme fournie est suffisante et calculer la monnaie à rendre, il est nécessaire de capturer un montant.
DescriptionProduit	*descriptif* - Pour afficher le descriptif en l'écran ou l'imprimer sur le reçu.
	code - Pour pouvoir effectuer une recherche dans les DescriptionProduit.
	prix - Pour afficher le prix de chaque ligne et calculer le total.
Vente	*date, heure* - Un reçu est l'état d'une vente sur papier. Il indique généralement une date et une heure.
LigneArticles	*quantité* - Pour enregistrer la quantité entrée lorsqu'il y a plusieurs exemplaires d'un même article (par exemple cinq paquets de pruneaux).
Magasin	*nom, adresse* - Le nom et l'adresse du magasin doivent figurer sur le reçu.

Modèle du domaine avec attributs

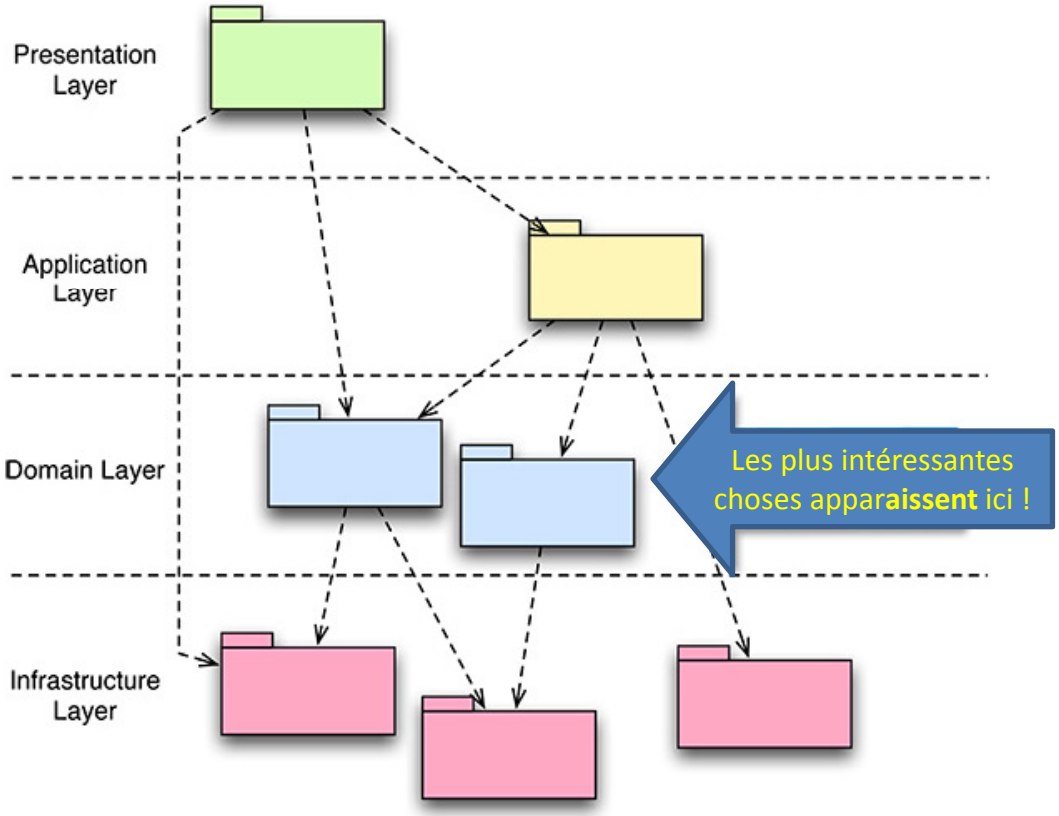


Comment utiliser le Modèle de Domaine

- L'analyste peut souhaiter perdre informations supplémentaires sur les processus métier clés dans le domaine en complétant la vue statique des collaborations du système avec les diagrammes d'activité ou séquence pour montrer le comportement.
- Le diagramme d'activité suivant fournit un contexte d'haut niveau pour décrire le traitement d'un transaction d'achat-vente dans une organisation qui gère investissements
- Le diagramme inclue classes du domaine (à gauche) que sont ressources pour les activités.



Architecture en couches de référence

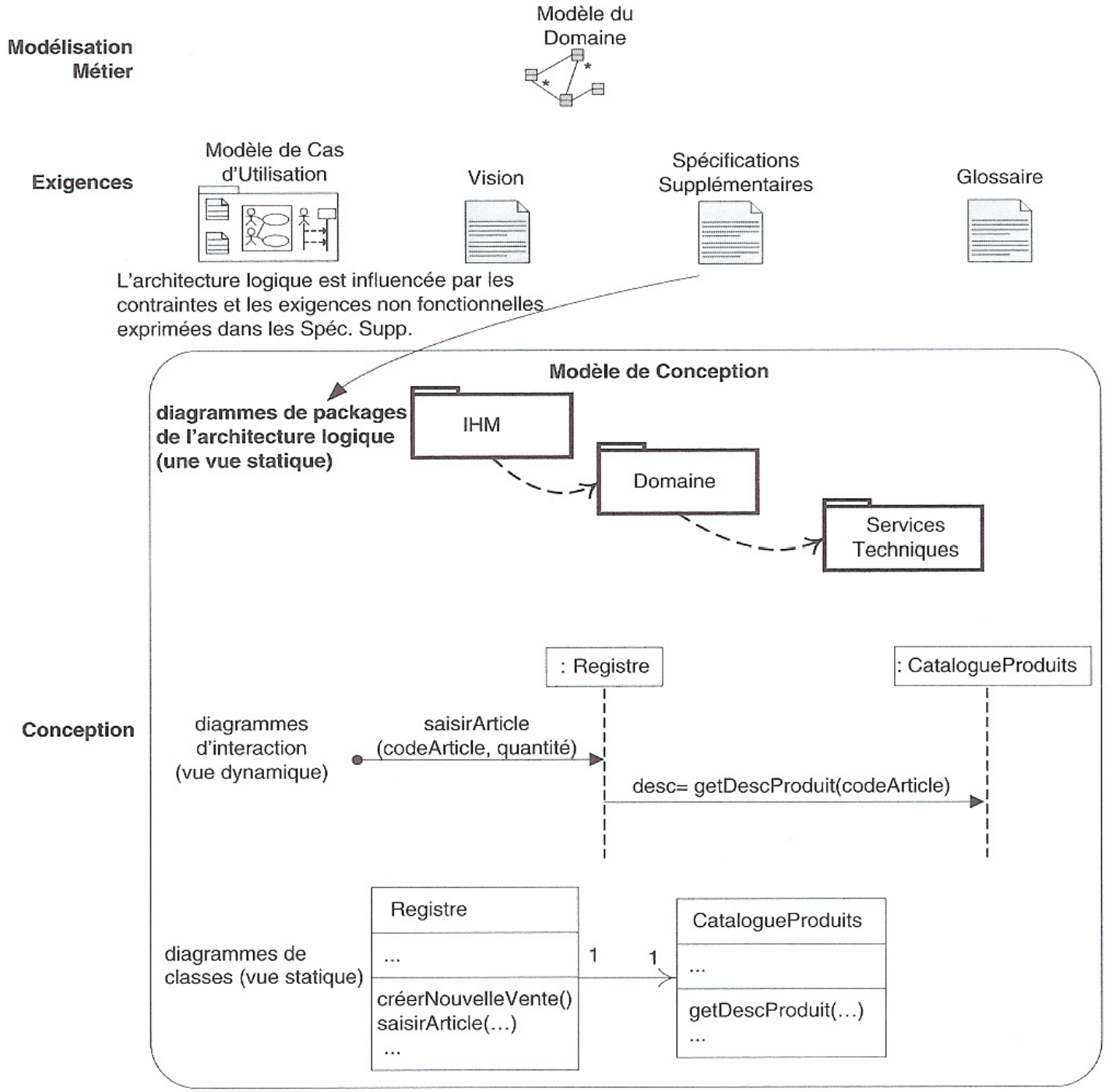


- Un modèle de domaine est un modèle métier « incomplet », dans qu'on omet les responsabilités des agents individuelles.
- Le but de la modélisation de domaine est de fournir « la grande image » des relations entre les entités métier dans l'organisation.
- Le modèle de domaine montre typiquement les entités métier majeures, leurs responsabilités fonctionnelles et les relations entre les entités.
- Il fournit aussi une description d'haute niveau des données que chaque entité fournit.

FIN DU CINQUIÈME CHAPITRE

Chap. 6. Architecture Logicielle

M. Shaw, D. Garlan, *Software Architecture. Perspective on an Emerging Discipline*, Prentice Hall, 1996



Influence des artefacts du Processus Unifié

Architecture logique (AL) peut être spécifiée par:
- diagrammes de packages UML, et
- le Document d'Architecture du Logiciel.

AL s'inspire principalement des contraintes architecturales documentées dans les Spécifications Supplémentaires.

Architecture logicielle

Architecture logicielle = l'ensemble des décisions importantes relatives à l'organisation d'un système logiciel:

- la sélection des éléments structuraux et leurs interfaces par lesquelles le système est composé, ainsi que
- leurs comportements comme spécifié dans la collaboration entre ces éléments,
- la composition de ces éléments structurels et comportementaux en sous-systèmes progressivement plus importantes, et
- le style architectural qui guide cette organisation de ces éléments et de leurs interfaces, leurs collaborations, et leur composition.

Un **style d'architecture** définit une famille de systèmes en termes de patterns de l'organisation du système. Il doit contenir:

- un vocabulaire des *composants* et des types de *connecteurs*,
- un ensemble de *contraintes* sur la façon dont elles peuvent être combinées,
- un ou plusieurs *modèles sémantiques* qui précisent comment les propriétés globales d'un système peuvent être déterminées à partir des propriétés de ses parties.

Styles d'architecture

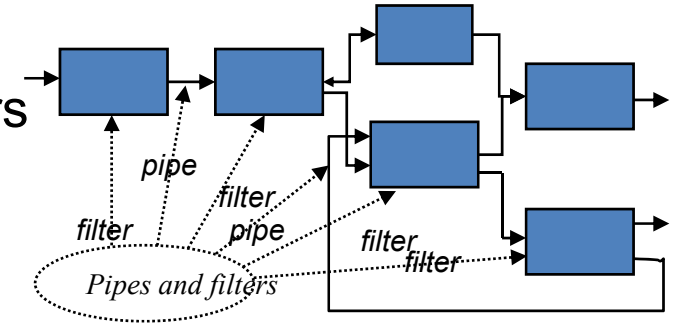
- Dataflow systems (systèmes de flux de données)
  - Batch sequential
  - Pipes and filters
- Call-and-return systems
  - Main program and subroutine
  - Object-oriented systems (systèmes appel-réponse)
  - Hierarchical layers (architecture en couches)
- Independent components
  - Communicating processes
  - Event systems
- Virtual machines
  - Interpretors
  - Rule-based systems
- Repositories
  - Databases
  - Hypertext systems
  - Blackboards

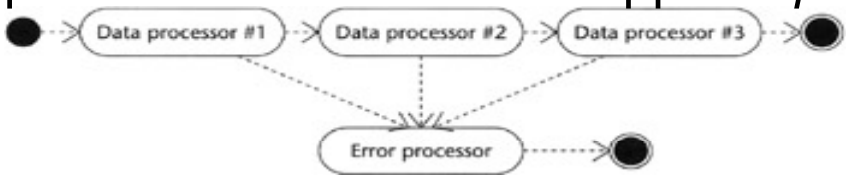
Systèmes de flux de données

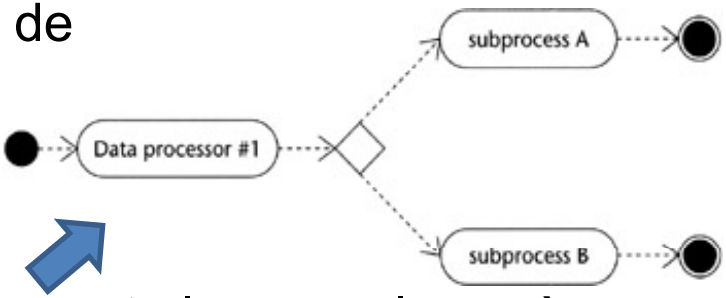
- Systèmes de flux de données (dataflow systems) sont caractérisés par la façon dont les données se déplacent à travers le système. Les architectures de flux de données ont deux ou plusieurs processeurs de données (*filtres*) que chaque transforme les données d'entrée en données de sortie.



- Le style architectural *Pipes-and-filters*.
  Il est commun dans l'architecture pipes-and-filters que un composant de traitement a deux sorties, une sortie standard et une sortie d'erreur et une entrée unique appelé entrée standard. De manière générique, l'entrée et les mécanismes de sortie pour un *filter* donné sont appelés *ports*.

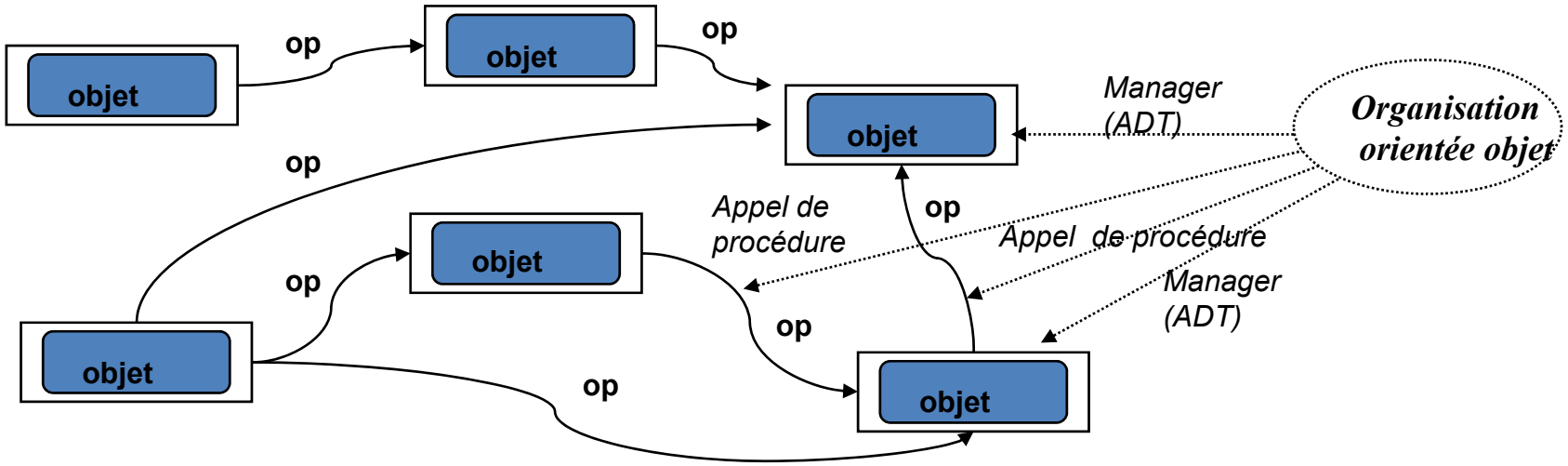






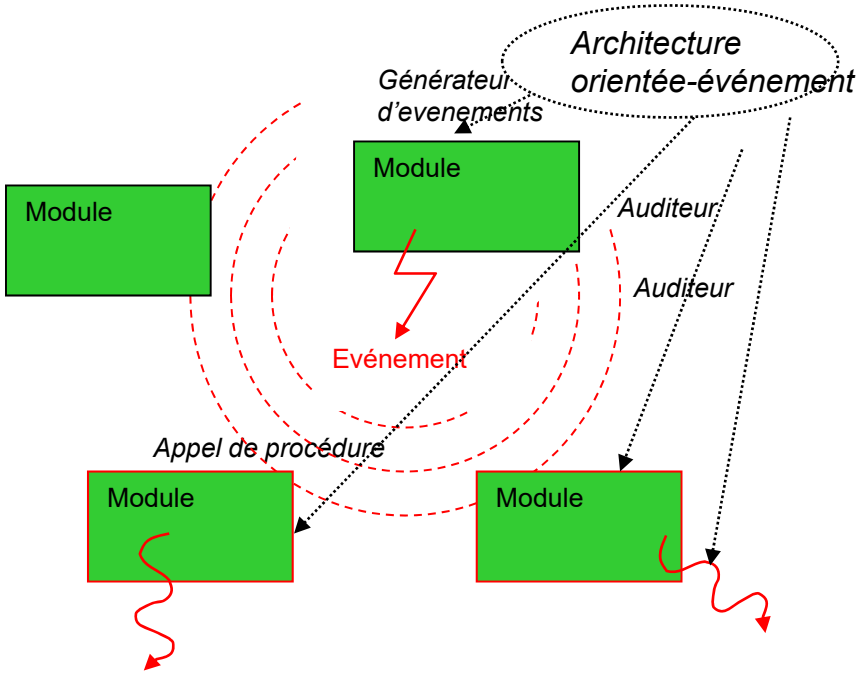
- Il est possible de définir des éléments de traitement plus complexes à produire des arrangements encore plus complexes et les flux de données.
- Flux de données avec des composants de traitement complexes.
- Le style architectural *Pipeline* est contraint à une séquence linéaire de filtres. Versions: *Bounded pipes, Typed pipes* .

Appel-réponse (Call-reply)



- Les données sont encapsulées avec les opérations primitives et associées aux types abstraits de données (ADT) ou aux objets. Leur représentation interne est inaccessible à d'autres objets. Les objets interagissent par appels de fonctions ou de procédures.
- Caractéristique pour l'orientation objet.

Architecture orientée-événement

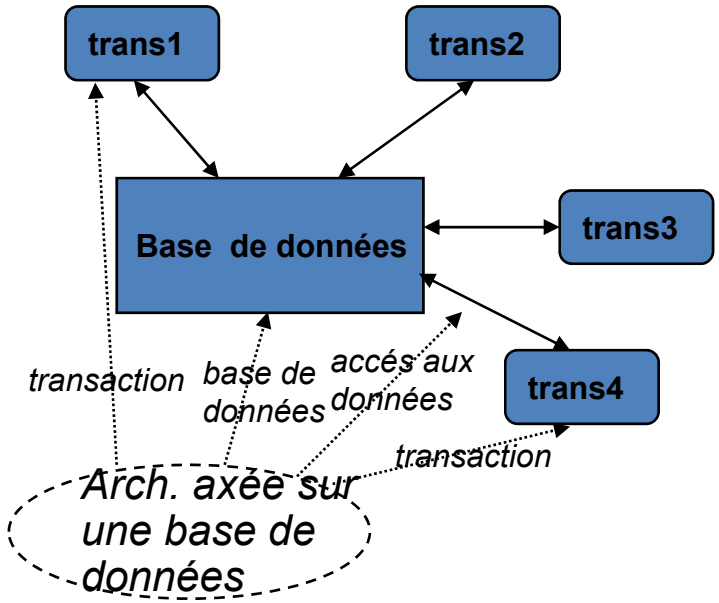


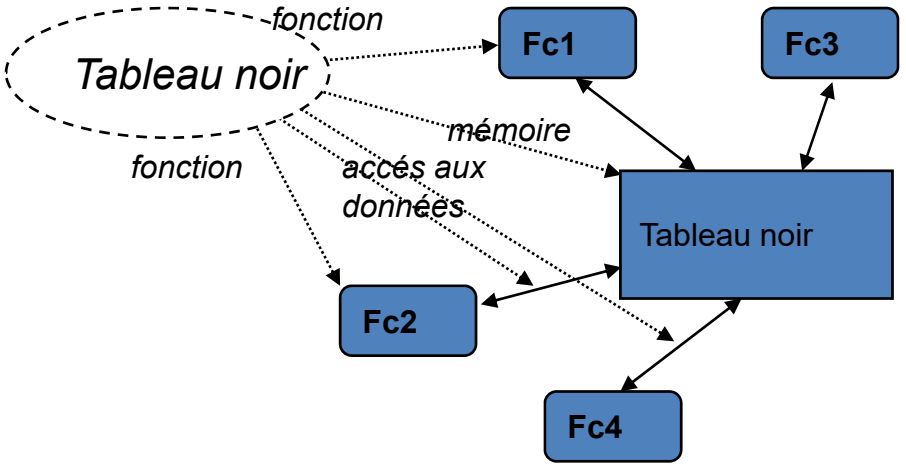
- Une composante peut lancer de nombreux types d'événements.
- Les composants intéressées à un événement doivent s'enregistrer comme «auditeurs» en associant un gestionnaire d'événements. Quand un événement se produit, le système appelle toutes les procédures qui ont été enregistrées pour cet événement.

Architecture axée sur les données

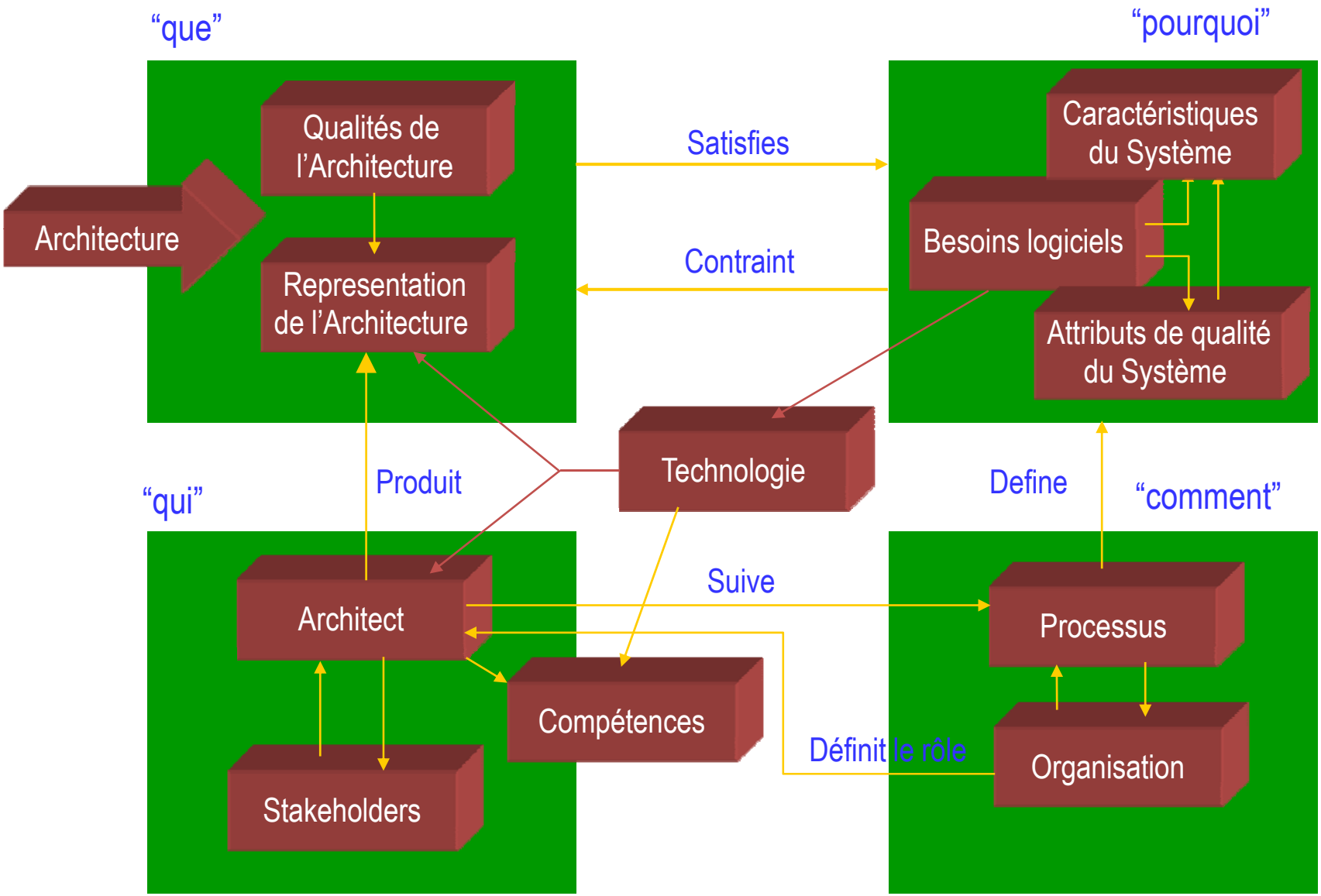
Deux types de composants:

- Une base de données centrale qui représente l'état du système
- Une collection de composants indépendants qui ont accès au dépôt central.



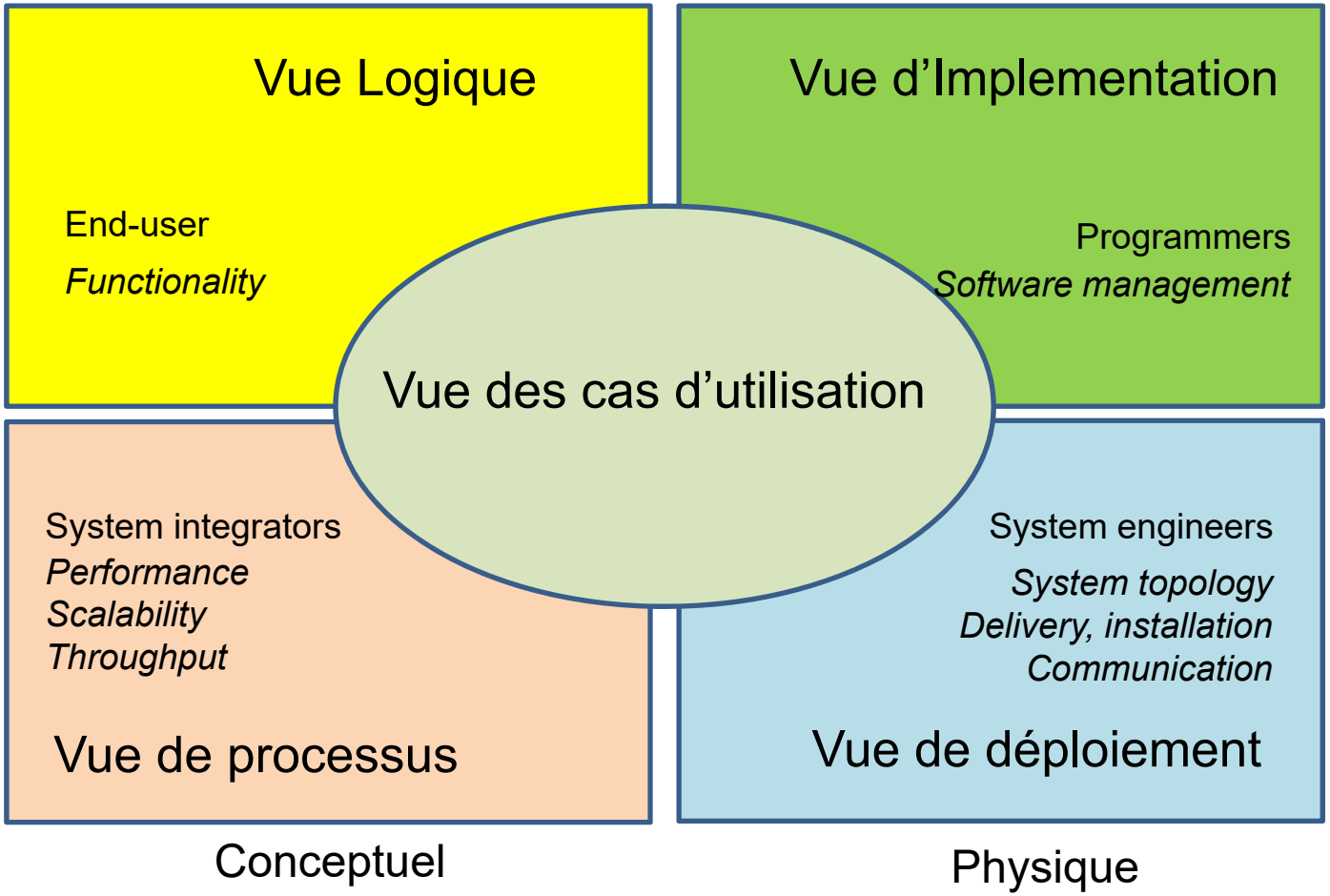


Le domaine de architecturer



Vues d'architecture

- Une vue architecturale est une description simplifiée (une abstraction) d'un système à partir d'une perspective ou un point de vue, couvrant des préoccupations particulières, et en omettant les entités qui ne sont pas pertinents pour cette perspective.



Architecture logicielle

Architecture logicielle = l'organisation à grande échelle des classes logicielles en *packages* (ou *espaces de noms*), *sous-systèmes* et *couches*.

L' architecture est « logique » parce qu'elle n'implique aucune décision quant as la façon dont ces éléments seront déployés sur différents systèmes d'exploitation ou différentes machines physiques sur un réseau. Ces dernières décisions font partie de l'architecture de déploiement.

Couche = un groupement a très forte granularité de classes, packages et sous-systèmes qui a une responsabilité de cohésion pour un aspect majeur du système. En outre, les couches sont organisées de telle sorte que les plus « hautes« fassent appel aux services des plus « basses », mais d'ordinaire pas l'inverse..

1. architecture en couches stricte (fermé)
2. architecture en couches lâche (ouverte)

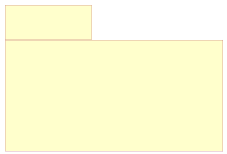
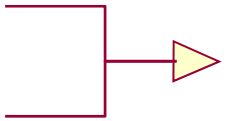
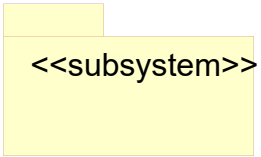
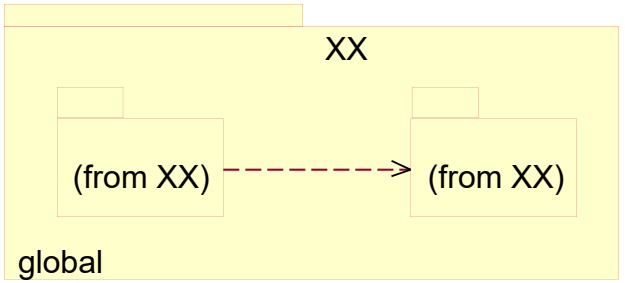
En général, les couches d'un système OO sont les suivantes :

- *Présentation*.
- *Logique applicative et objets du domaine*. Objets logiciels représentant des concepts du domaine qui satisfont aux besoins d'une application.
- *Services techniques*. Objets a usage général et sous-systèmes qui fournissent des services techniques tels que l'accès a une base de données ou un journal des erreurs. Ces services sont généralement indépendants de l'application et réutilisables dans d'autres systèmes.

Une architecture logique ne doit pas être nécessairement structurée en couches, mais c'est une pratique très courante..

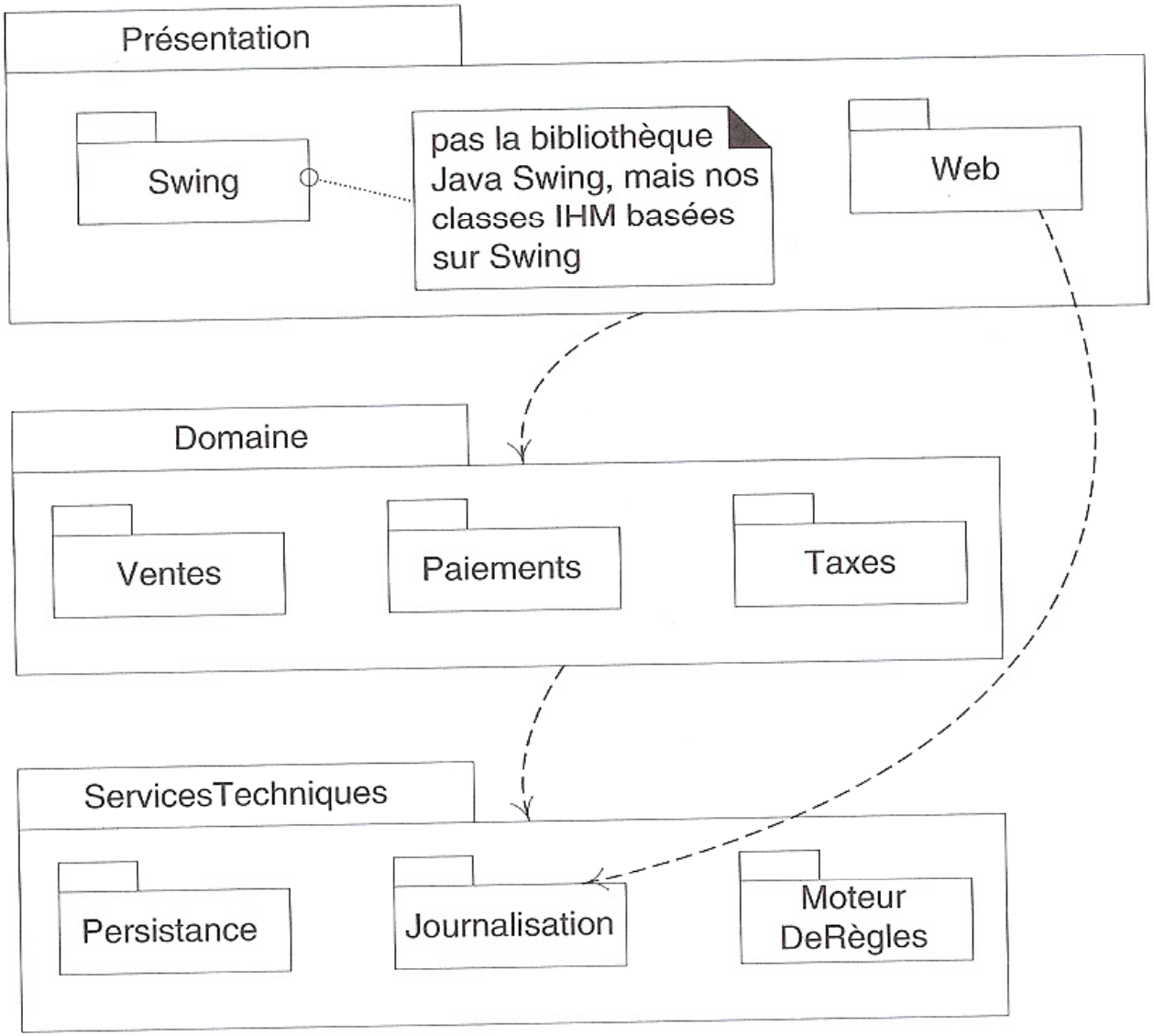
Diagramme de packages

- Les diagrammes de package UML sont souvent utilisés pour illustrer l'architecture logique d'un système, les couches, les sous-systèmes, packages (au sens Java), etc.
- Une couche peut être modélisée sous forme d'un package dans un diagramme UML.
- Un diagramme de package UML permet de grouper éléments: classes, autres packages, cas d'utilisation, etc. Un package UML représente un *espace de noms:* pour que le même nom d'élément puisse être défini dans deux packages. L'imbrication des paquets est très fréquent.

Package	Dépendance	Généralisation
		
Stéréotype	**Relations «contient» et «utilise»**	
		

- Stéréotypes UML standard : façade, framework , stub , subsystem, system .

Diagrammes de packages UML représentant les couches d'une application



Il est courant de représenter les *dépendances* entre packages pour que les développeurs puissent voir les couplages a grande échelle au sein du système: une flèche pointillée partant du package dépendant et se dirigeant vers le package dont il dépend.

Différentes méthodes de représentation des packages imbriqués en UML

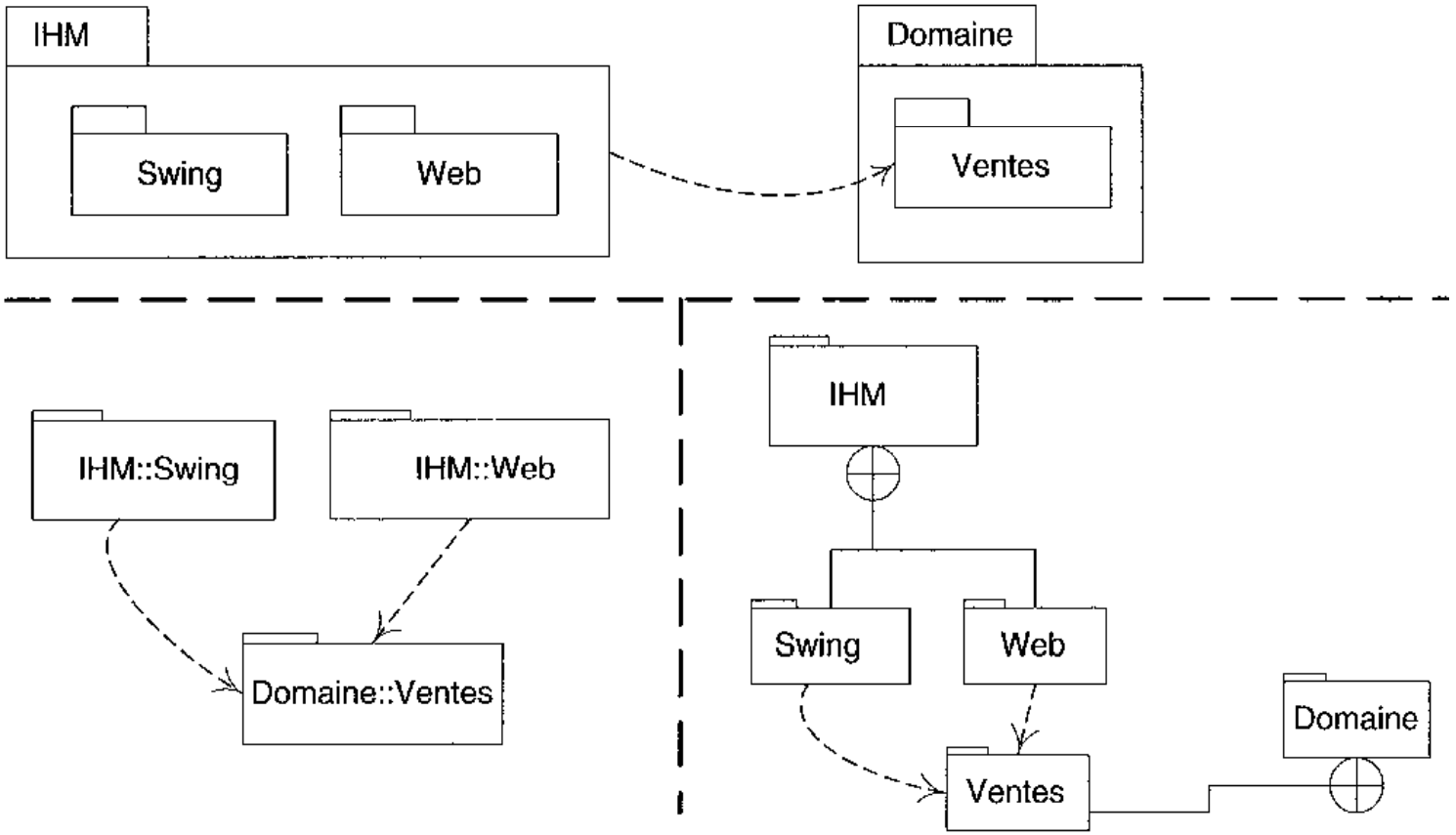


Diagramme de packages - Exemple 1

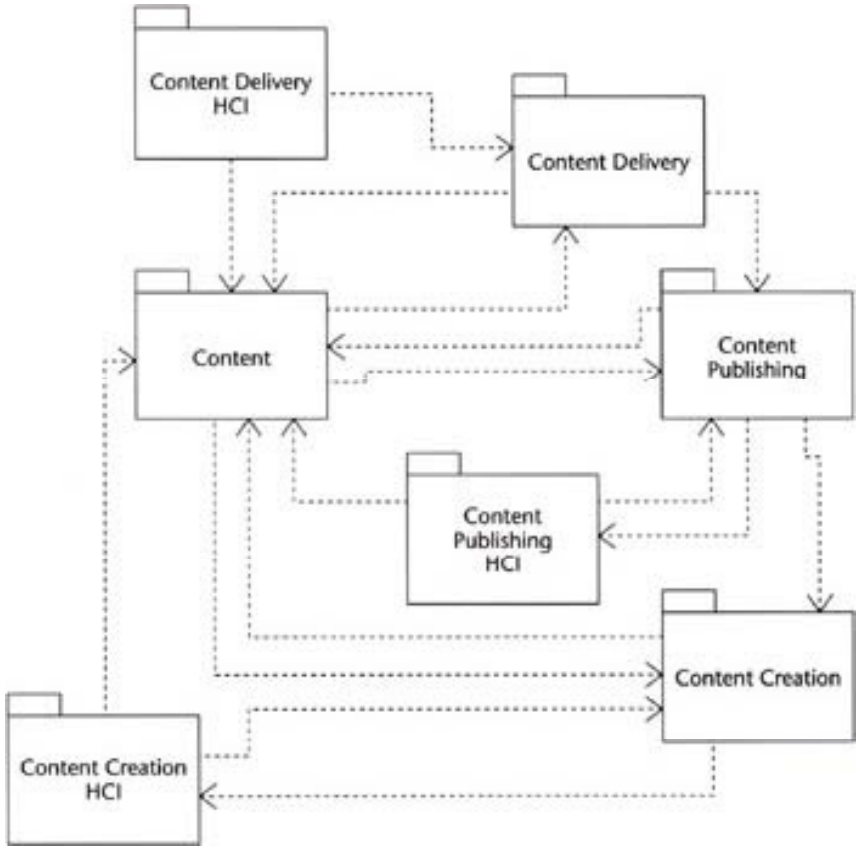


Diagramme de packages - Exemple 2

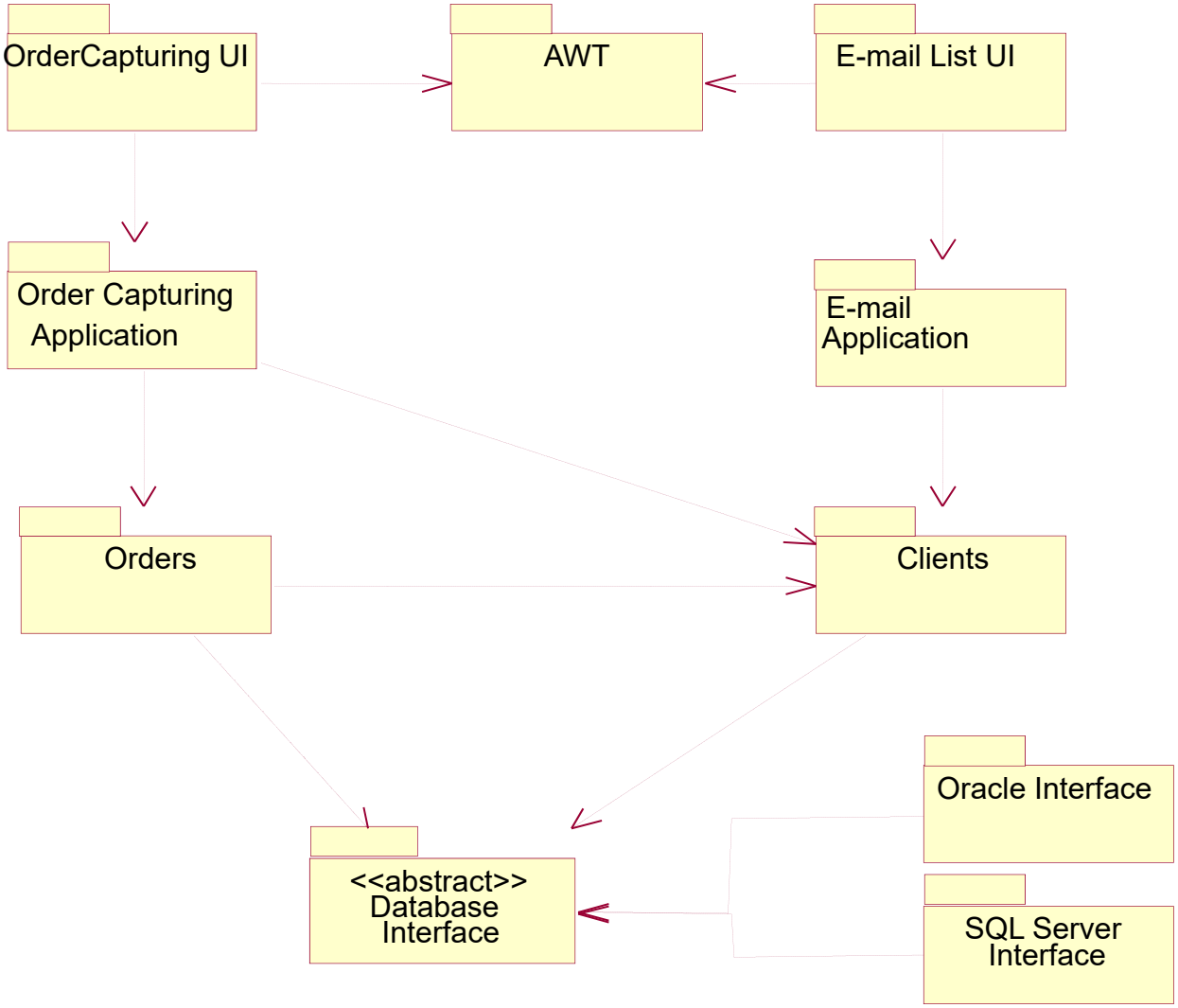
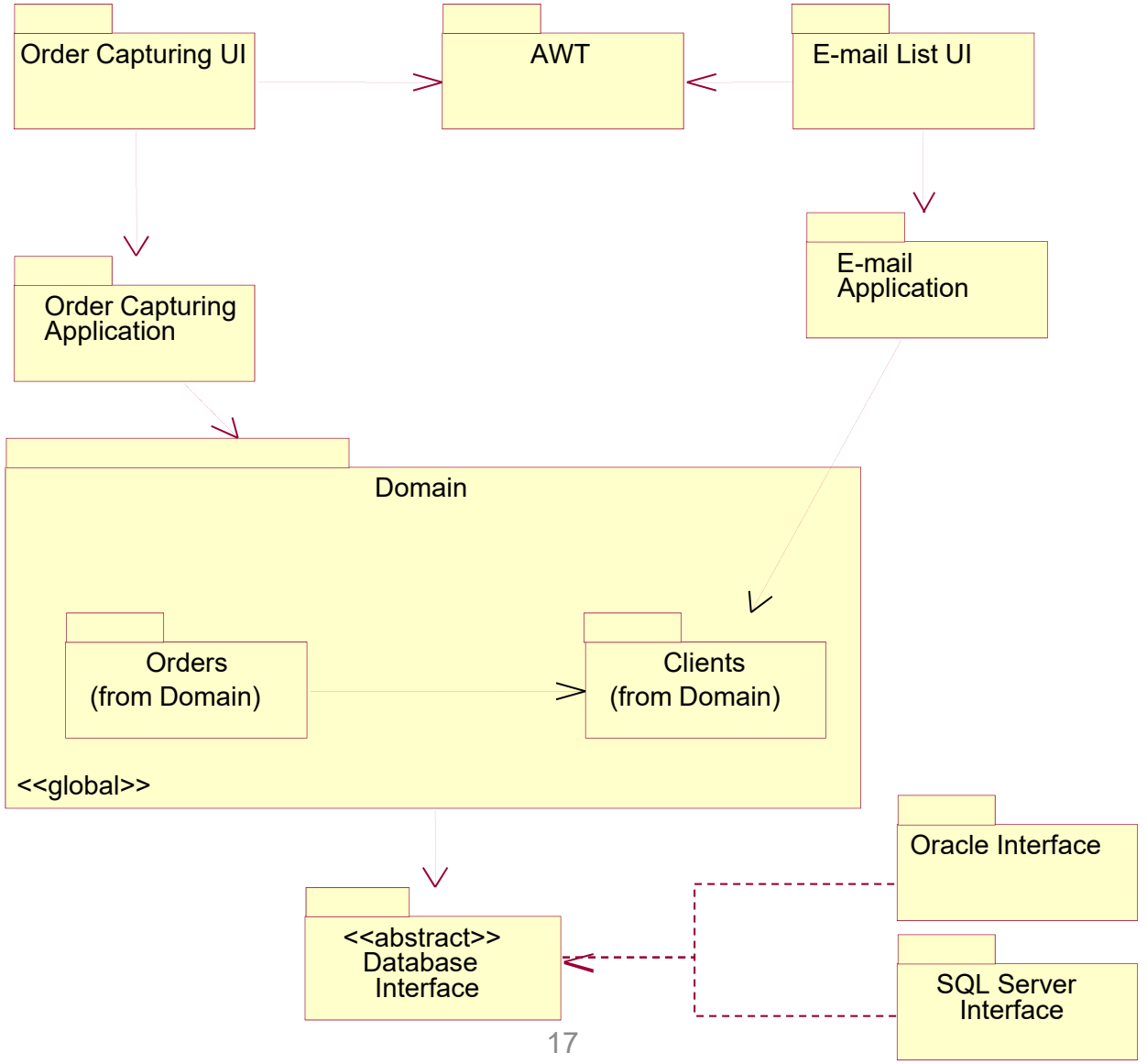
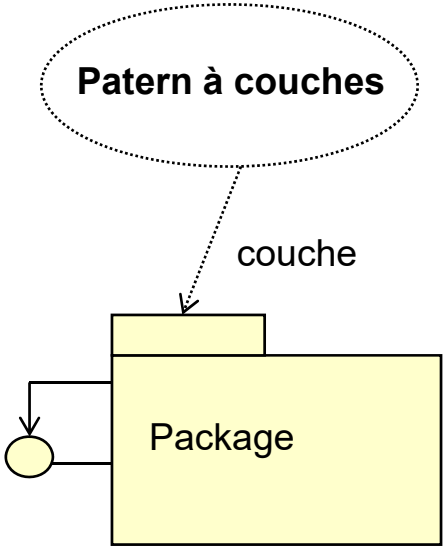


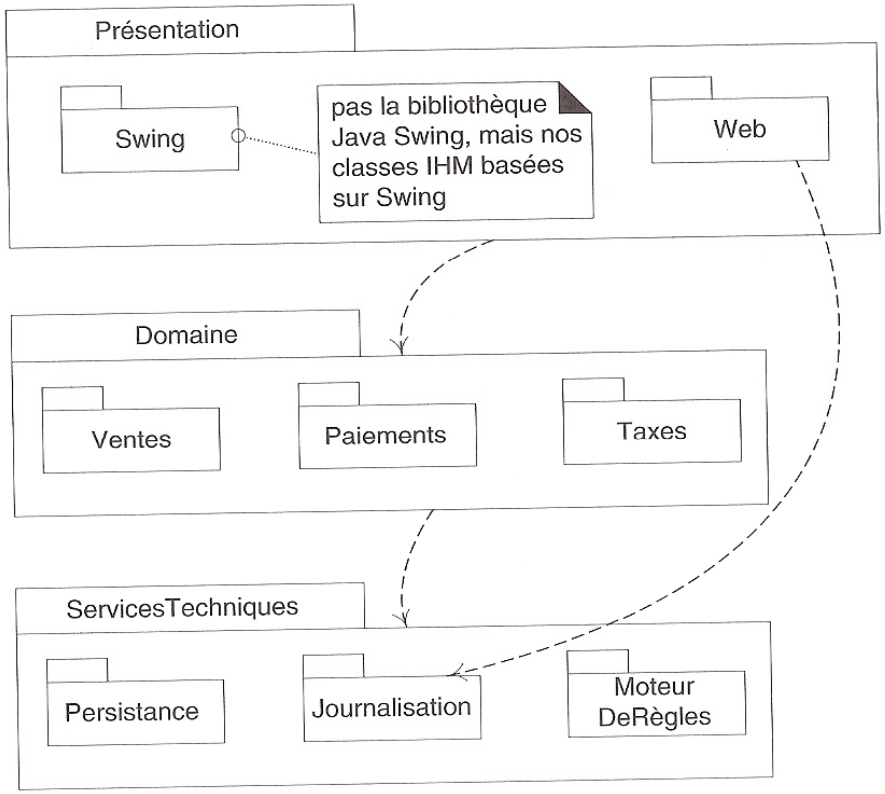
Diagramme de packages imbriqué



Architecture en couches

Architecture en couches





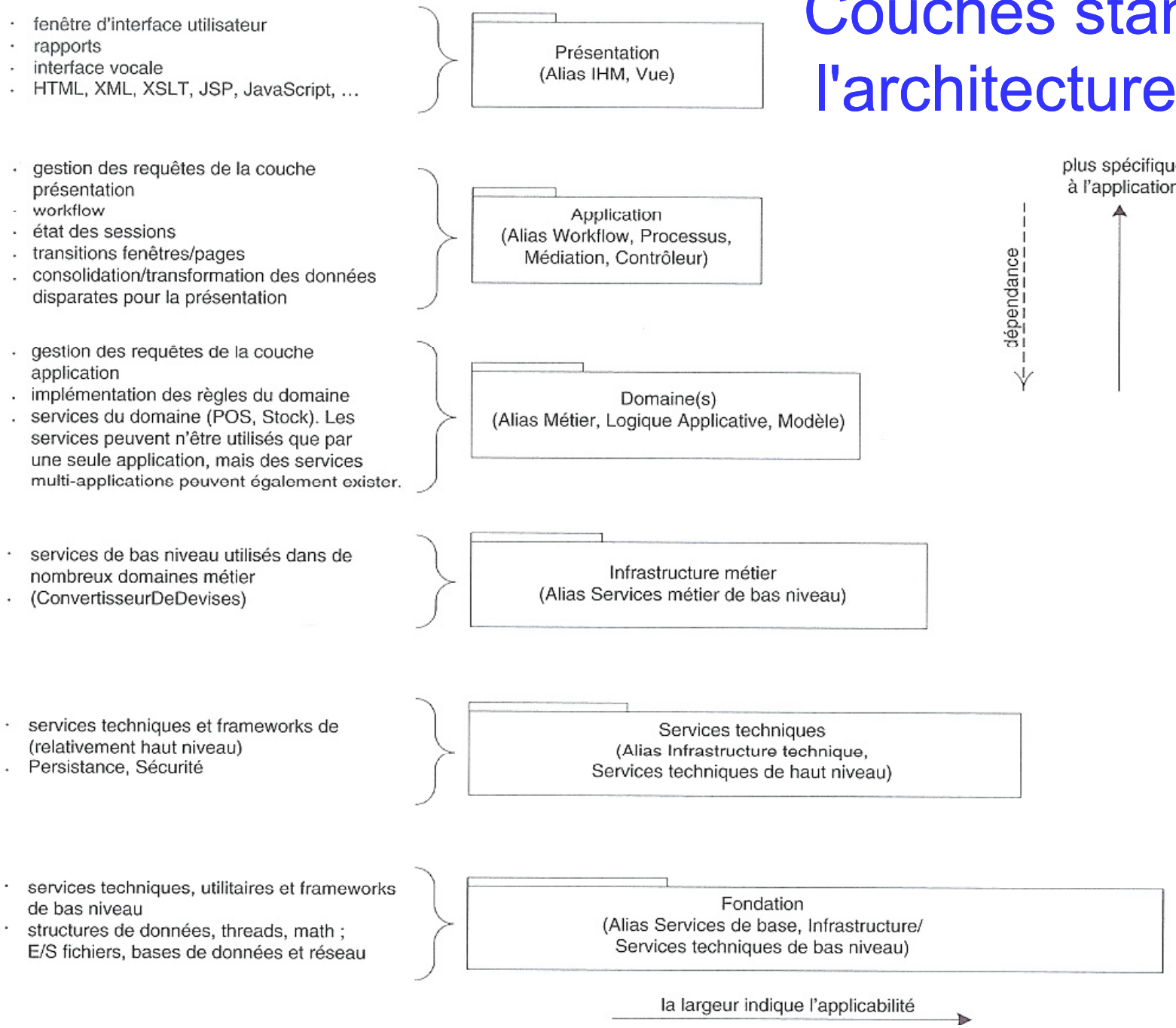
En essence:

- Organisation de la structure logique a grande échelle d'un système en couches discrètes de responsabilités distinctes et réelles , avec une séparation nette et cohésive des attributions : les couches << basses » concernent les services généraux et de bas niveau, les couches supérieures étant plus spécifiques à l'application.
- Collaboration et couplage vont des couches supérieures aux couches inferieures ; on évite de coupler en sens inverse.

Pourquoi les couches?

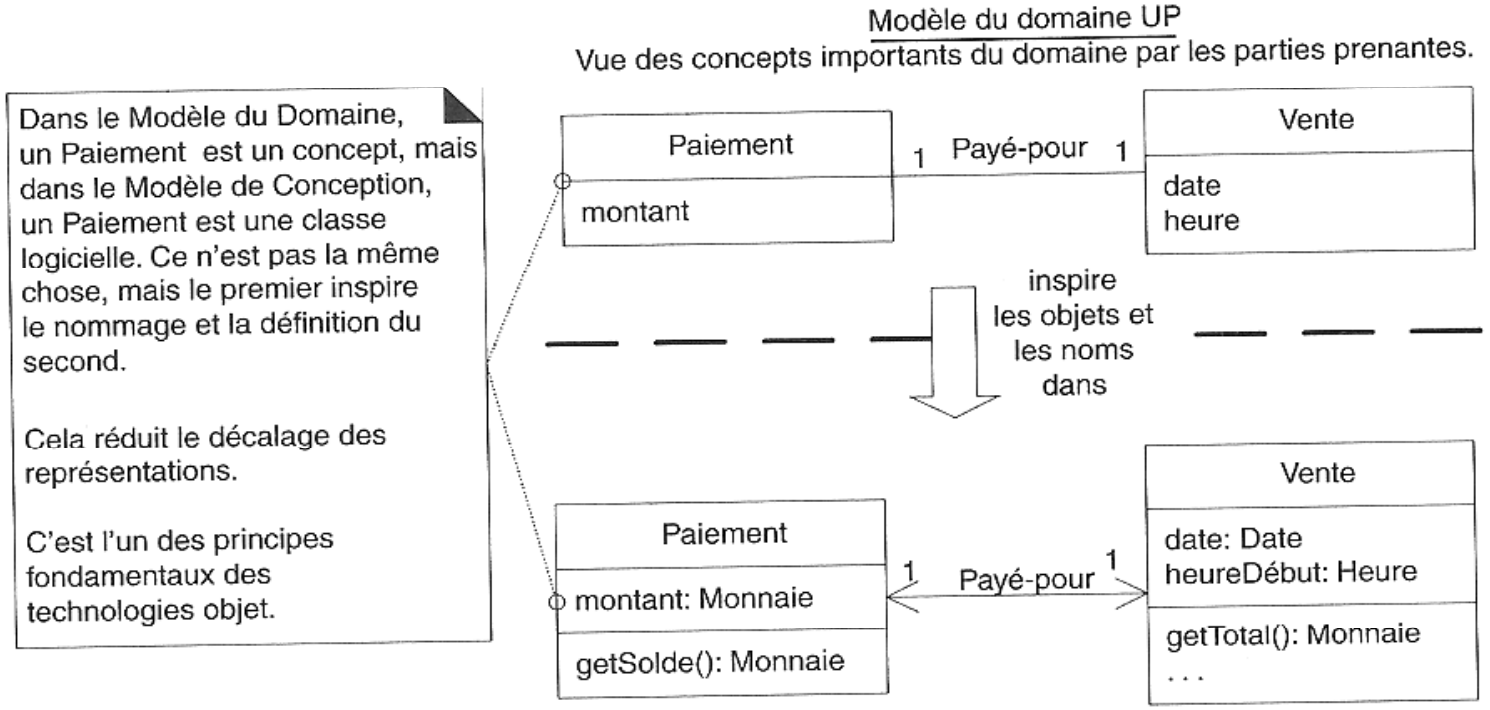
Problèmes de logiciel	Avantages de la modélisation en couches
Les modifications du code source se répercutent dans tout le système, car de nombreuses parties du système sont fortement couplées	Il existe en général une séparation des fonctions, une division entre les services de haut et de bas niveau et entre les services généraux et ceux qui sont spécifiques à l'application. Cela réduit le couplage et les dépendances, améliore la cohésion et augmente la clarté et le potentiel de réutilisation.
La logique applicative est fortement liée à l'interface utilisateur; elle ne peut donc être réutilisée avec une interface différente ou distribuée à un autre nœud de traitement.	On peut remplacer certaines couches par de nouvelles implémentations. C'est généralement impossible pour les couches Services techniques et Fondation, mais c' est le cas des couches Présentation, Application et Domaine.
Les services techniques potentiellement généraux ou la logique métier sont fortement lies à une logique plus spécifique à l'application ; ils ne peuvent donc être réutilises, distribues à un autre nœud ou facilement remplaces par une implémentation différente.	- Les couches les plus basses contiennent des fonctions réutilisables. - Certaines couches (en particulier Domaine et Services techniques) peuvent être distribuées.
Différents domaines d'attribution sont fortement couplés. Il est donc difficile de repartir le travail des développeurs en traçant des frontières nettes.	La segmentation logique facilite le développement en équipes.

Couches standard de l'architecture logique



Relations entre la couche Domaine et le Modèle du Domaine

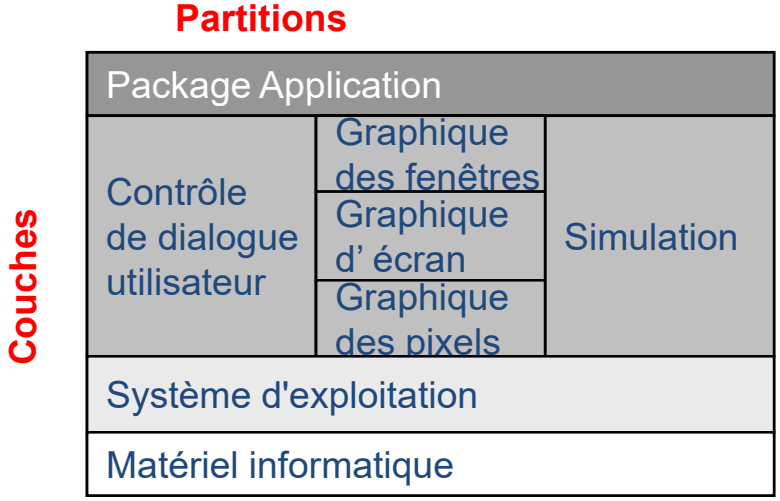
- La couche de domaine fait partie du logiciel et le modèle de domaine fait partie de l'analyse en perspective conceptuelle; donc elles ne sont pas la même chose. Mais en créant une couche de domaine ayant come inspiration le modèle de domaine, nous réalisons un plus petit écart entre le domaine du monde réel et notre conception de logiciel.



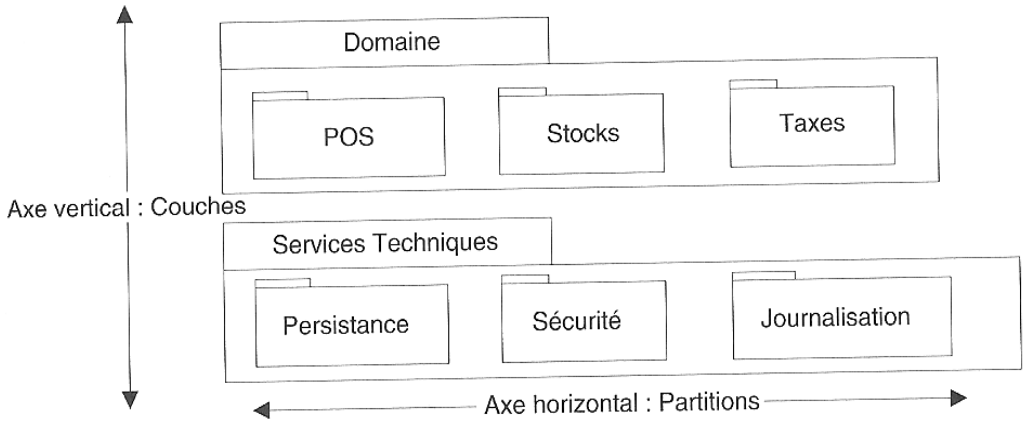
Couches et partitions

- A l'origine, la notion de niveau (*tier*) représentait en architecture une couche logique et non un nœud physique, mais ce terme a été largement employé pour designer un nœud de traitement physique (ou un groups de nœuds), comme dans l'expression << niveau client » (la machine cliente).
- On dit que les couches d'une architecture représentent un découpage vertical, tandis que les partitions représentent une division horizontale des sous-systèmes relativement parallèles dans une couche.

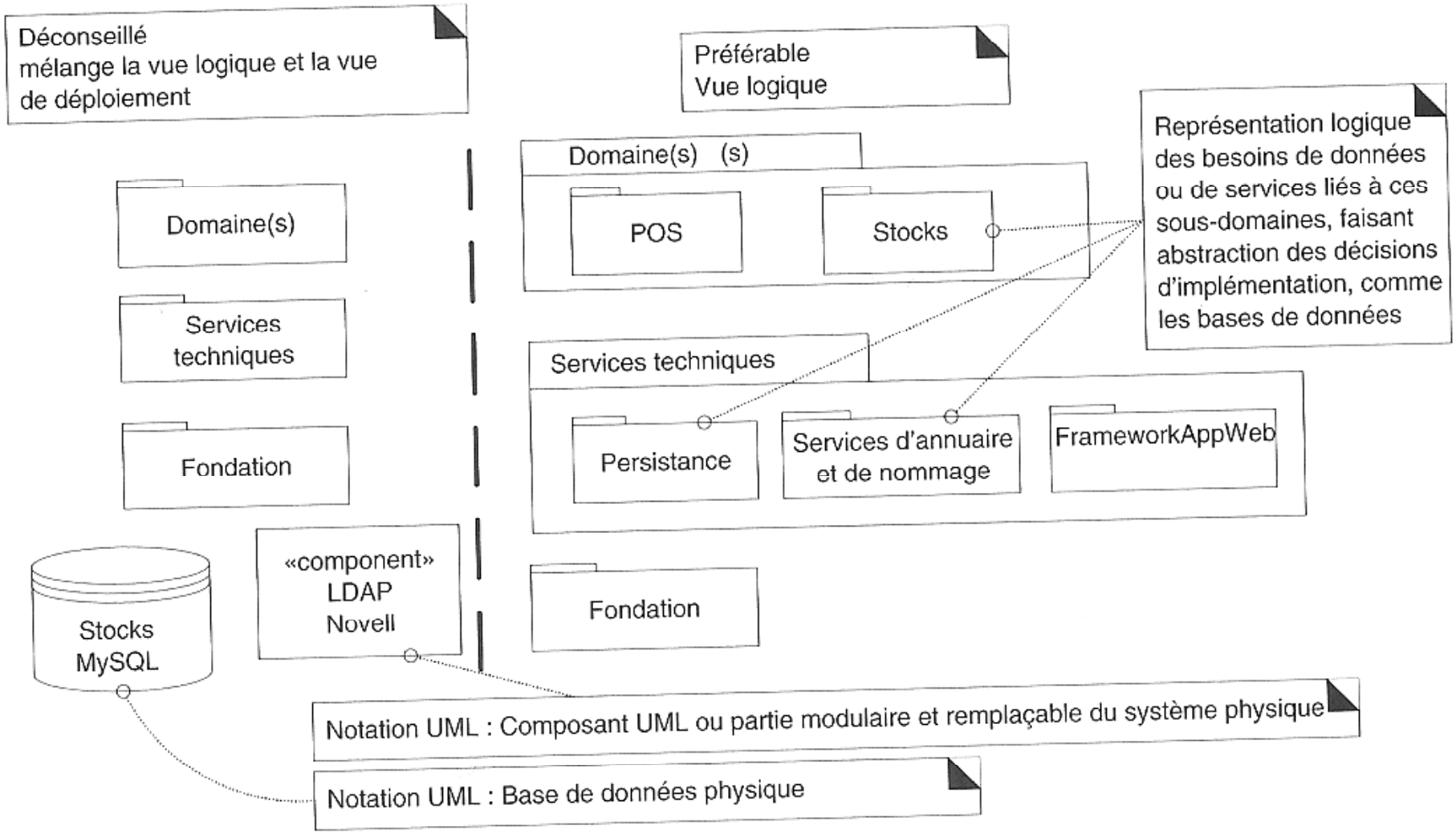
Exemple 1



Exemple 2



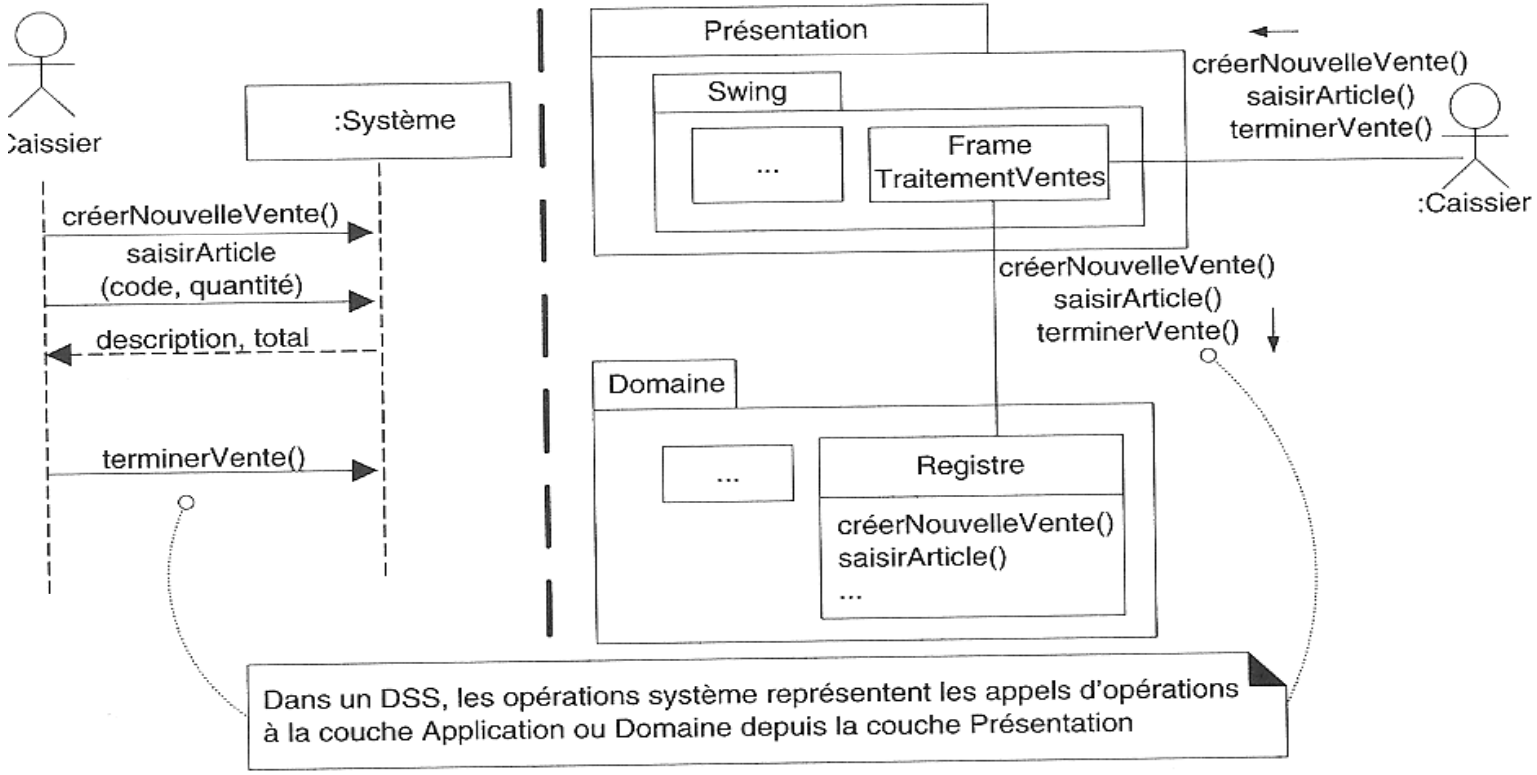
Confusion des vues architecturales logiques et physiques



La relation entre les DSS, les opérations système et les couches

Dans une architecture en couches bien conçue, qui favorise une forts cohésion et une séparation des attributions, les objets de la couche IHM vont donc relayer ou déléguer les requêtes de la couche IHM a la couche domaine qui les traitera.

Les messages émis par la couche IHM vers la couche domaine seront les messages représentés dans les DSS, comme par exemple saisirArticle.



Design Patterns pour la conception architecturale

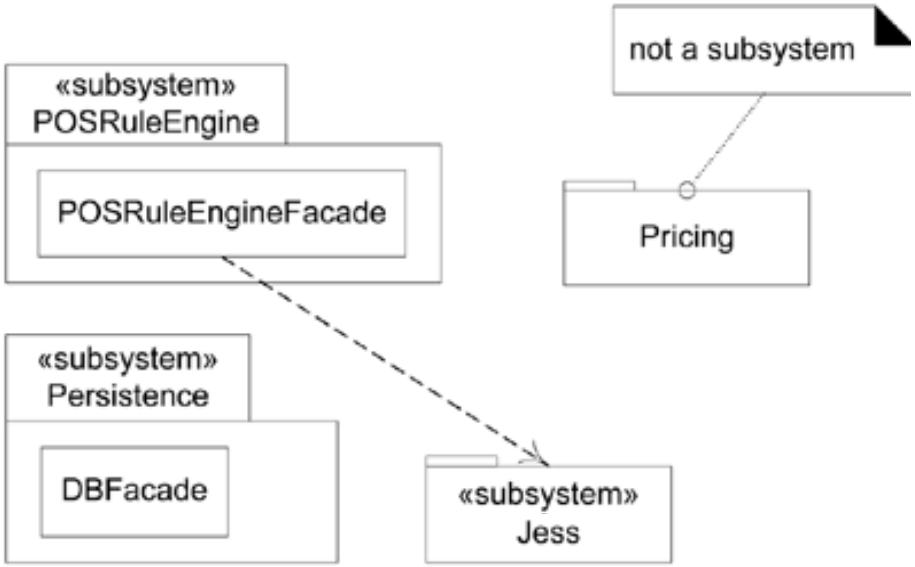
Utilisation de design patterns dans des architectures logicielles

Deux décisions de conception au niveau architecturale:

1. Quelles sont les grandes parties?
2. Comment sont-ils connectés?

Considérant que l'architecture en couches guide nous en définissant les grandes pièces, les modèles de conception micro-architecturales telles que Facade, Controller et Observer sont couramment utilisés pour la conception des connexions entre les couches et les packages.

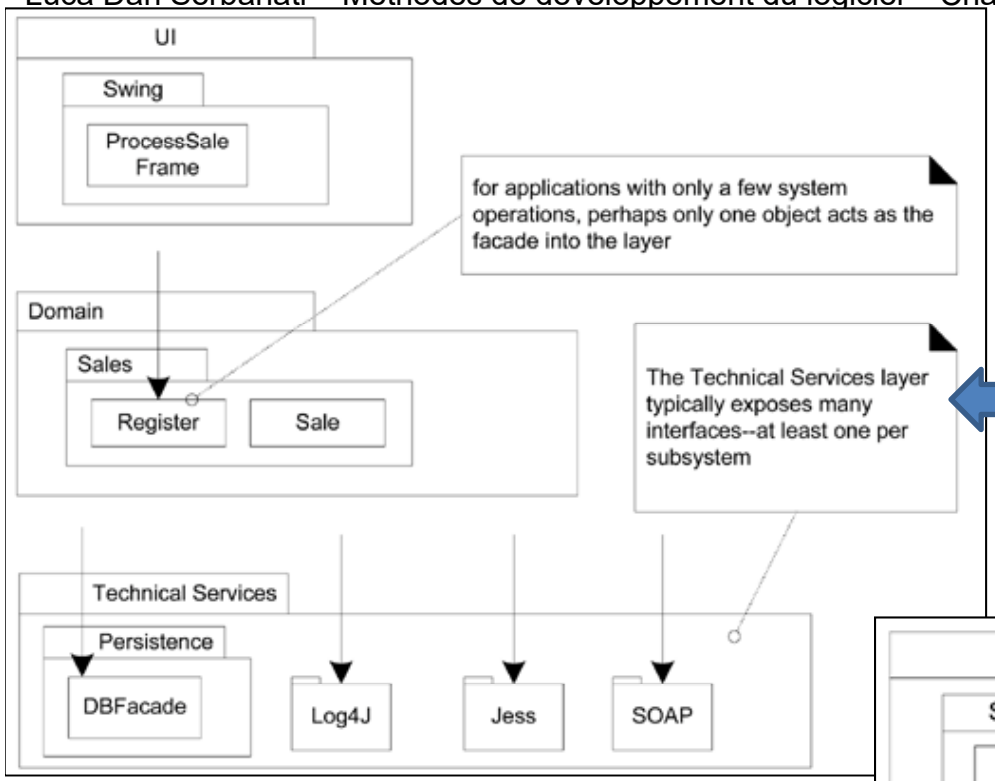
Couches vs. sous-systèms



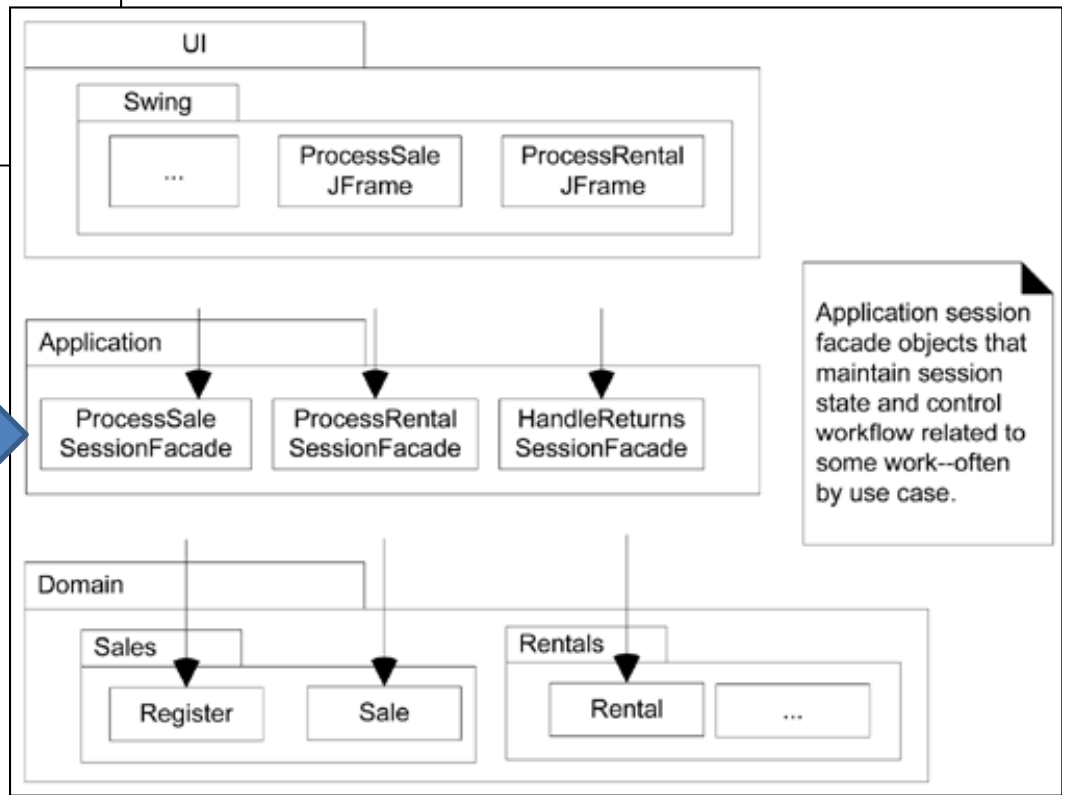
- Certains paquets ou des couches ne sont pas seulement des groupes conceptuelles de choses, mais sont sous-systèmes vrai avec des comportements et des interfaces. Pour le contraste:
- Le package Pricing n'est pas un sous-système, il simplement groupe la classe Factory et les stratégies utilisées dans le calcul du prix. De même avec les packages de la Fondation comme java.util.
- D'autre part les packages Persistance, POSRuleEngine et Jess sont sous-systèmes. Ils sont les moteurs discrets avec des responsabilités de cohésion qui font fonctionner le système.

Façade et Session Façade

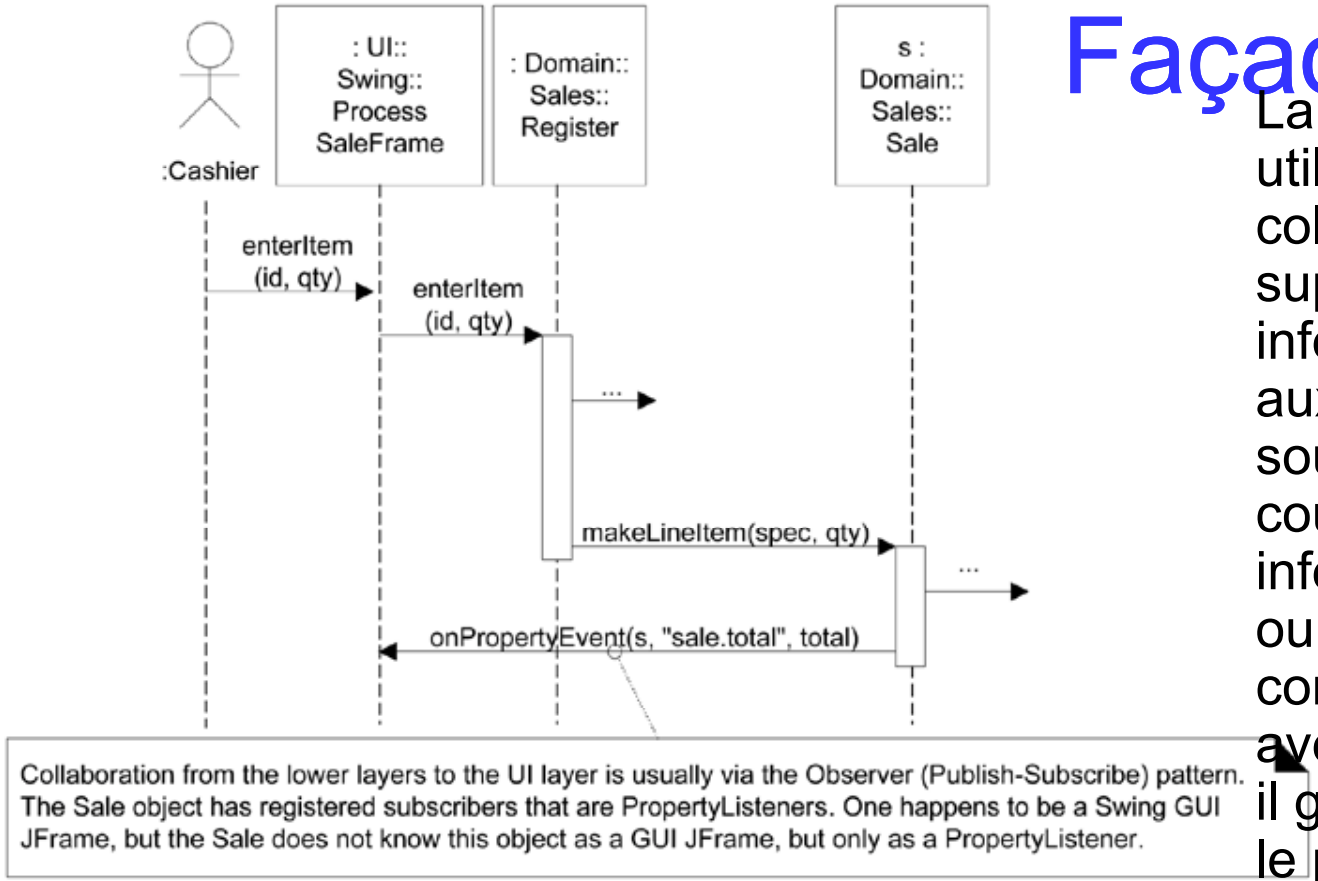
Un objet façade publique définit les services du sous-système. Les clients collaborent avec l'objet façade et pas avec les composants internes du sous-système.



Quand une application a nombreuses opérations de système et soutient nombreux cas d'utilisation, il est courant d'avoir plus d'un objet de médiation entre les couches d'interface et celle du domaine.



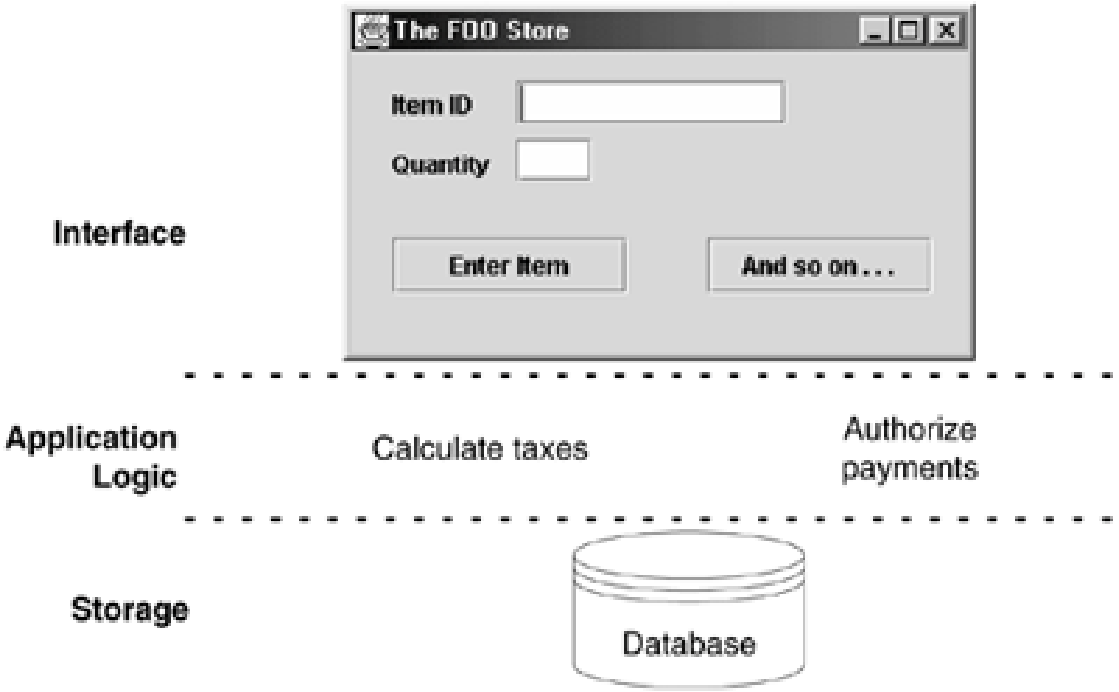
Façade/Observer



La façade est couramment utilisé pour le abaisser la collaboration d'une couche supérieure à une couche inférieure, ou pour l'accès aux services dans un autre sous-système de la même couche. Lorsque la couche inférieure de l'application ou de domaine a besoin de communiquer vers le haut avec la couche d'interface, il généralement est fait via le pattern Observateur

- Objets de l'interface utilisateur dans la couche supérieure de l'interface utilisateur implémente une interface comme PropertyListener ou AlarmListener, et sont abonnés à ou auditeurs des événements (tels que les événements de propriété ou d'alarme) en provenance des objets dans les couches inférieures. Les objets de la couche inférieure envoient directement des messages aux objets de la couche supérieure de l'interface utilisateur, mais le couplage n'est que à des objets considérés comme des entitées qui implémentent une interface, comme PropertyListener, et qui ne sont pas considérés comme des fenêtres GUI spécifiques

Architecture à Trois Tiers

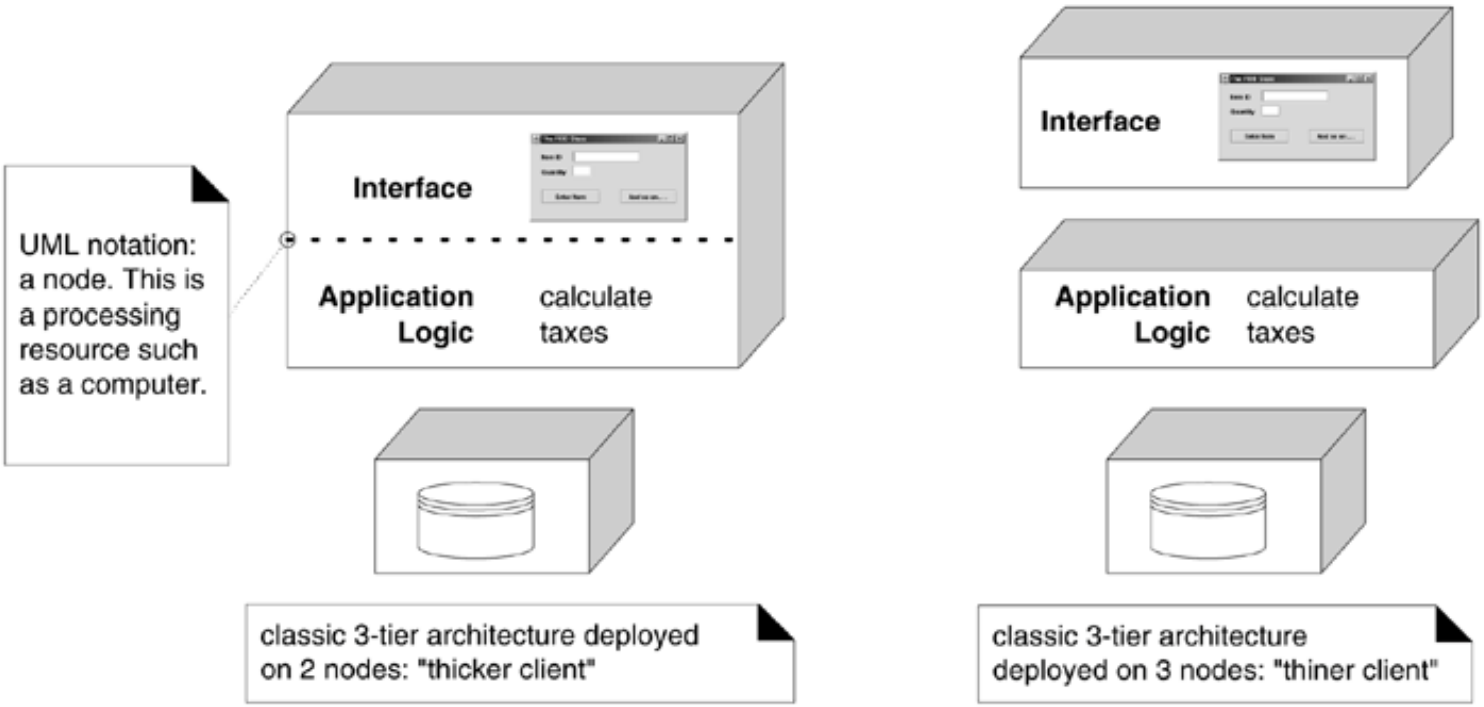


Une description classique des rangées verticales dans une architecture à trois niveaux (tiers) est :

1. Les fenêtres des interfaces graphiques, les rapports etc.;
2. Les tâches de la logique d'application et les règles qui gouvernent le processus;
3. Le mécanisme de stockage persistant des données.

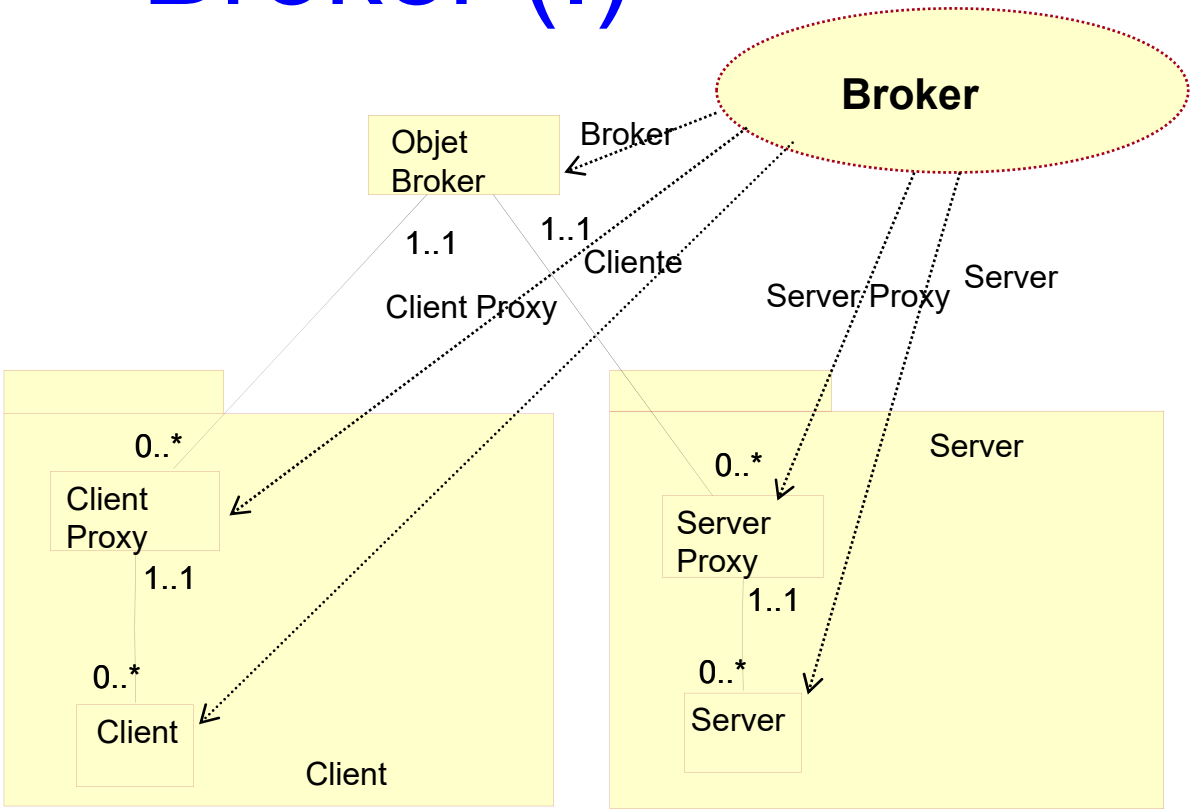
La qualité singulière d'une architecture à trois tiers est la séparation de la logique d'application dans une rangée logique du milieu, distincte, de logiciel. La rangée d'interface est relativement indépendante du traitement des données de l'application: les fenêtres ou les pages Web font des demandes de tâche à la rangée du milieu. La rangée du milieu communique avec la couche de stockage.

Déployer dans une architecture à trois tiers



Il y avait du malentendu que la description a impliqué ou a exigé un déploiement physique sur trois ordinateurs, mais la description voulue était purement logique ; l'allocation des tiers pour les nœuds à calculer pourraient varier de l'un à trois.

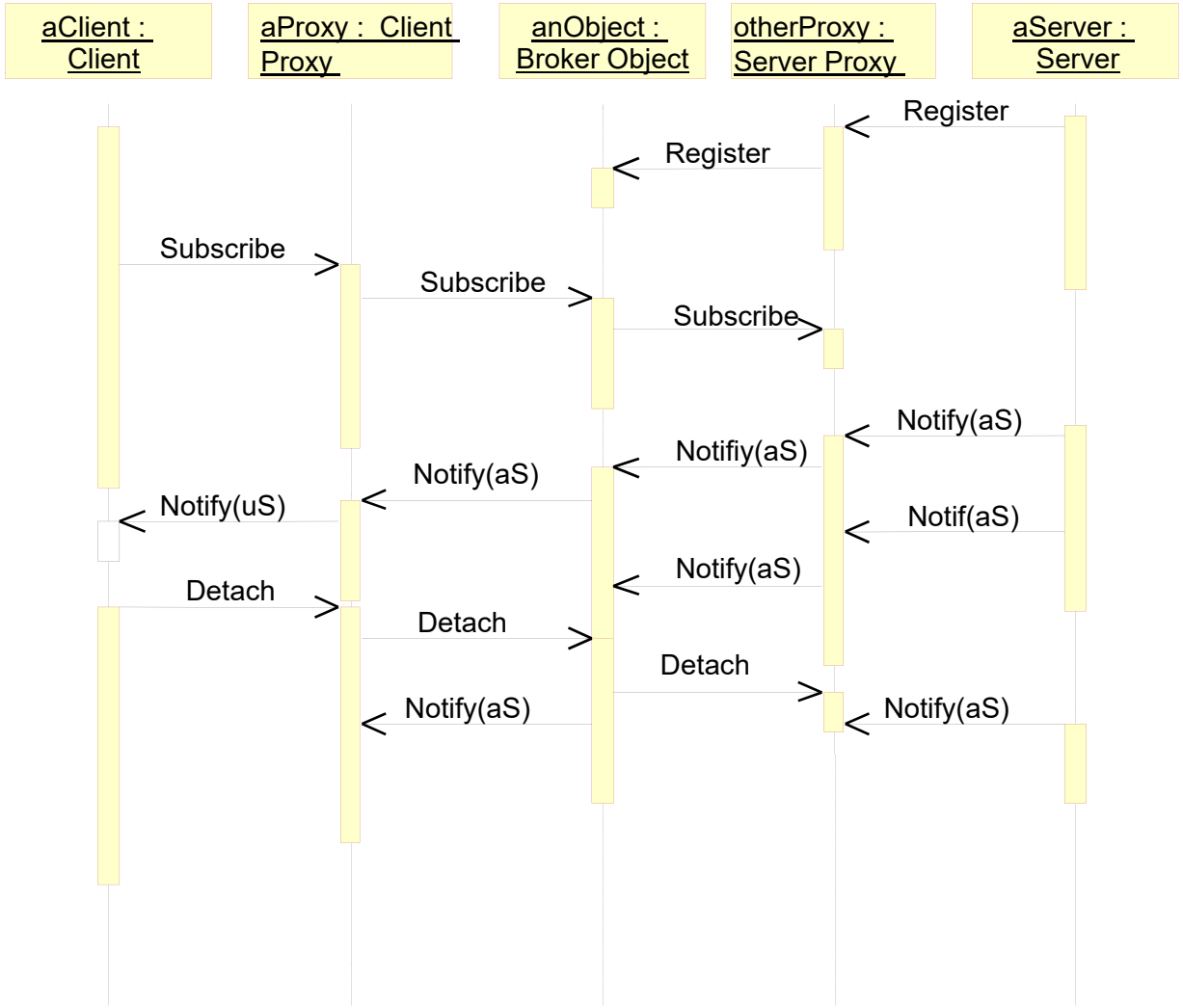
Broker (I)



Caractéristiques

Le modèle Broker accentue le détachement entre le server et ses clients. Un objet Broker est un objet qui sait où sont localisés les objets de système. Cette connaissance peut être apprise au compilation, à l'exécution fondée sur une découverte dynamique ou utilisant une politique mélangée.

Broker (II)



Deux niveaux d'indirection

1. Le premier niveau est composé par proxies qui isolent les clients des servers.
2. Le deuxième niveau est représenté par l'objet Broker qui isole les proxies des servers.

Tout d'abord un server doit enregistrer lui-même au broker. Ainsi le broker pourra répondre à toutes les demandes ayant un server comme la cible.

Quand le client enregistre lui-même au proxy locale, le proxy lui demande d'enregistrer au proxy de server.

La demande est géré exclusivement par le broker qui sait où le server est localisé.

Principe de Séparation Modèle-Vue (I)

Le principe de Séparation Modèle-Vue comprend au moins deux parties :

1. Ne pas connecter ou coupler directement des objets de l'IHM avec d'autres objets. Par exemple, l'objet logiciel Vente (un objet du domaine) ne doit pas posséder de référence La un objet java Frame (une fenêtre). Pourquoi? Parce que les fenêtres sont associées à une application particulière, alors que les autres objets sont susceptibles (dans l'idéal) d'être réutilises dans de nouvelles applications ou rattachés a une nouvelle interface.
2. Ne pas placer la logique applicative (par exemple un calcul de taxe) dans les méthodes d'un objet de l'IHM. Ces derniers doivent uniquement initialiser les éléments de l'interface utilisateur, recevoir des événements (par exemple un clic sur un bouton) et relayer des requêtes à la logique applicative ou a des objets du domaine.

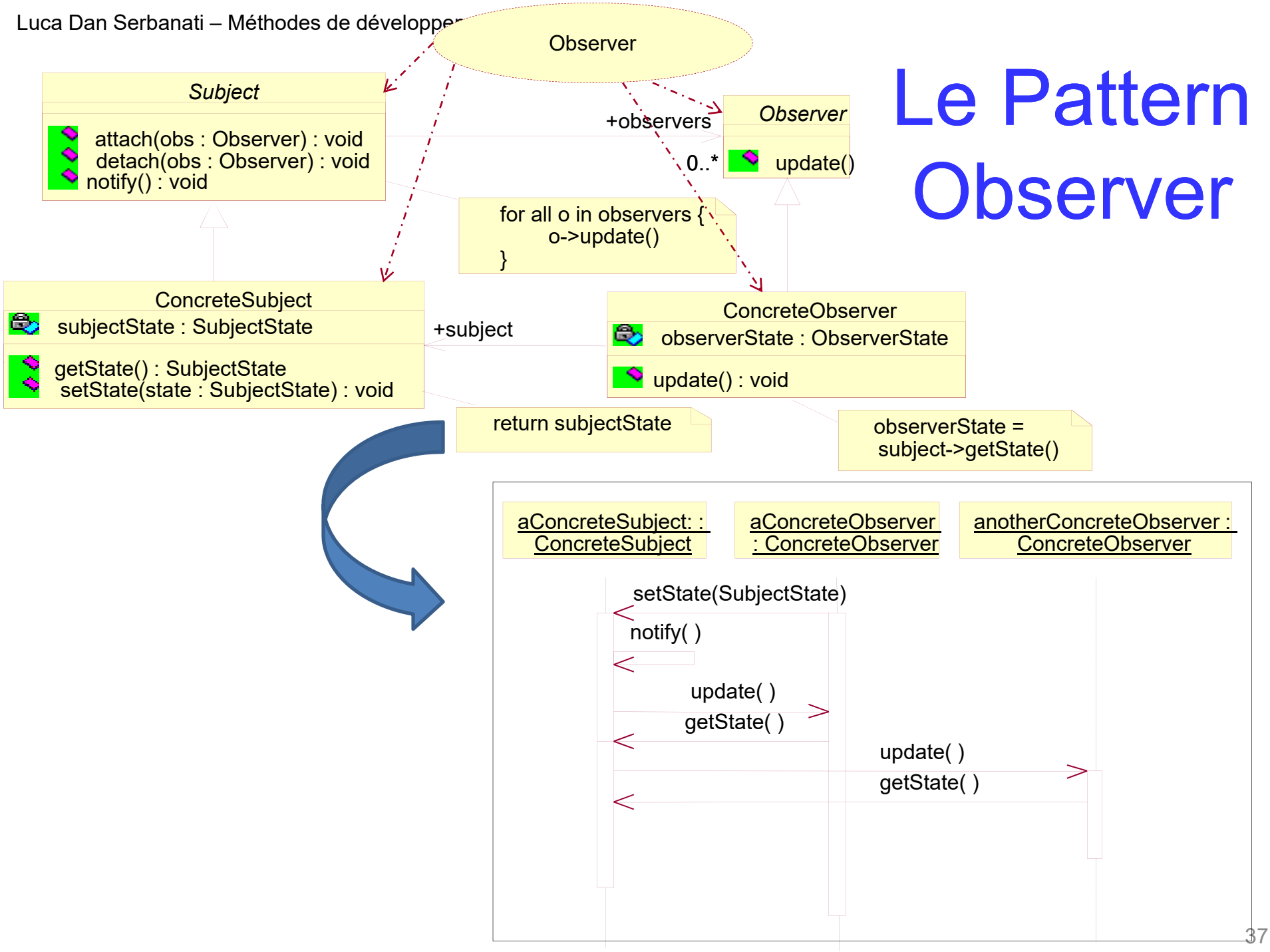
- *Selon le principe de* Séparation Modèle-Vue*, les objets du modèle (les objets du domaine) ne doivent pas connaitre directement les objets vue (les objets de présentation), au moins en tant que tels.*
- Une dérogation légitime à ce principe est celle du pattern Observateur, dans lequel les objets du domaine envoient des messages aux objets IHM vus uniquement en termes d'interfaces

Principe de Séparation Modèle-Vue (II)

Objectifs du principe Séparation Modèle-Vue:

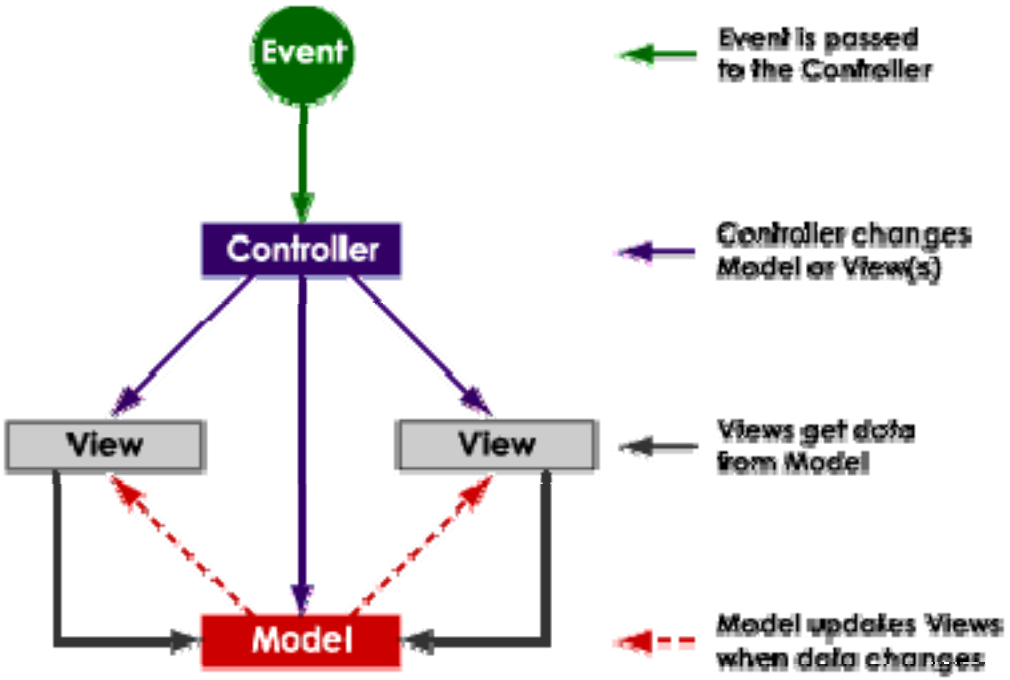
- favoriser des définitions de modèles cohésifs qui se centrent sur les processus du domaine et non sur les interfaces utilisateurs ;
- permettre de développer séparément les couches domaine et présentation ;
- réduire l'impact de l'évolution des besoins de la couche présentation sur la couche domaine ;
- permettre de connecter facilement de nouvelles vues a une couche domaine existante sans affecter celle-ci ;
- permettre plusieurs vues simultanés sur le même objet du modèle, par exemple un tableau et un graphique représentant l'évolution des ventes ;
- permettre d'exécuter la couche modèle indépendamment de l'interface utilisateur, par exemple dans un système de messagerie ou de traitement par lots ;
- favoriser la portabilité de la couche modèle.

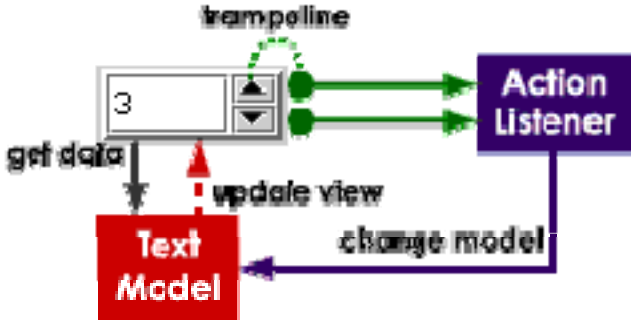
Le Pattern Observer



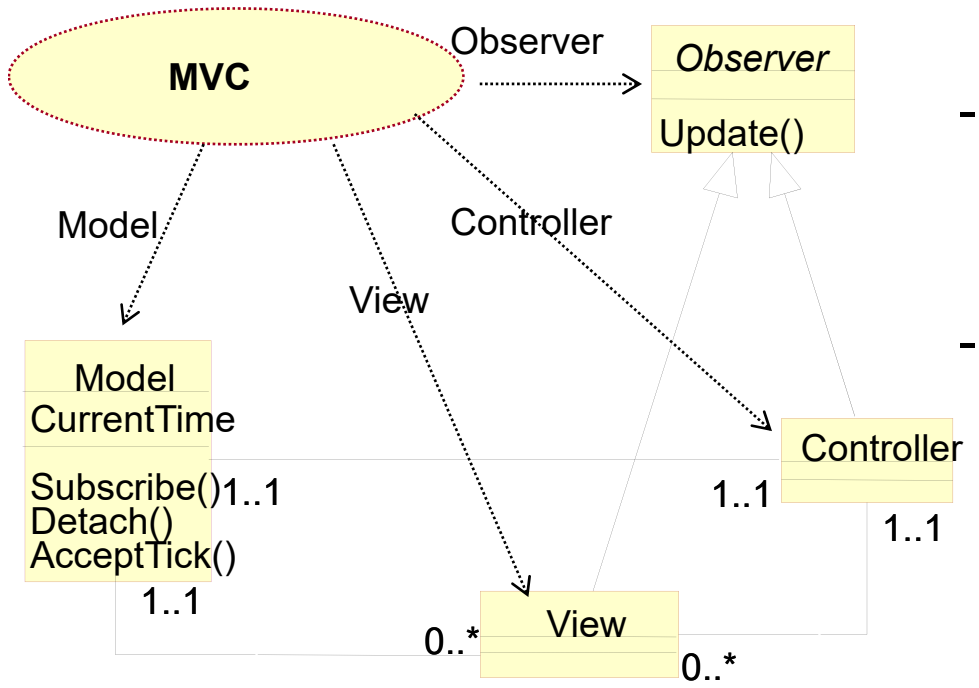
Le Pattern Modèle-Vue-Contrôleur (I)

- Les événements causent typiquement à un contrôleur de changer un modèle ou une vue, ou tous les deux.
- Quand un contrôleur change une donnée du modèle ou les propriétés, toutes les vues que dépendent automatiquement sont mises à jour.
- De même, quand un contrôleur change une vue, par exemple, en révélant des secteurs qui ont été précédemment cachés, la vue obtient des données du modèle fondamental pour se rafraîchir elle-même.





Le Pattern Modèle-Vue-Contrôleur (II)

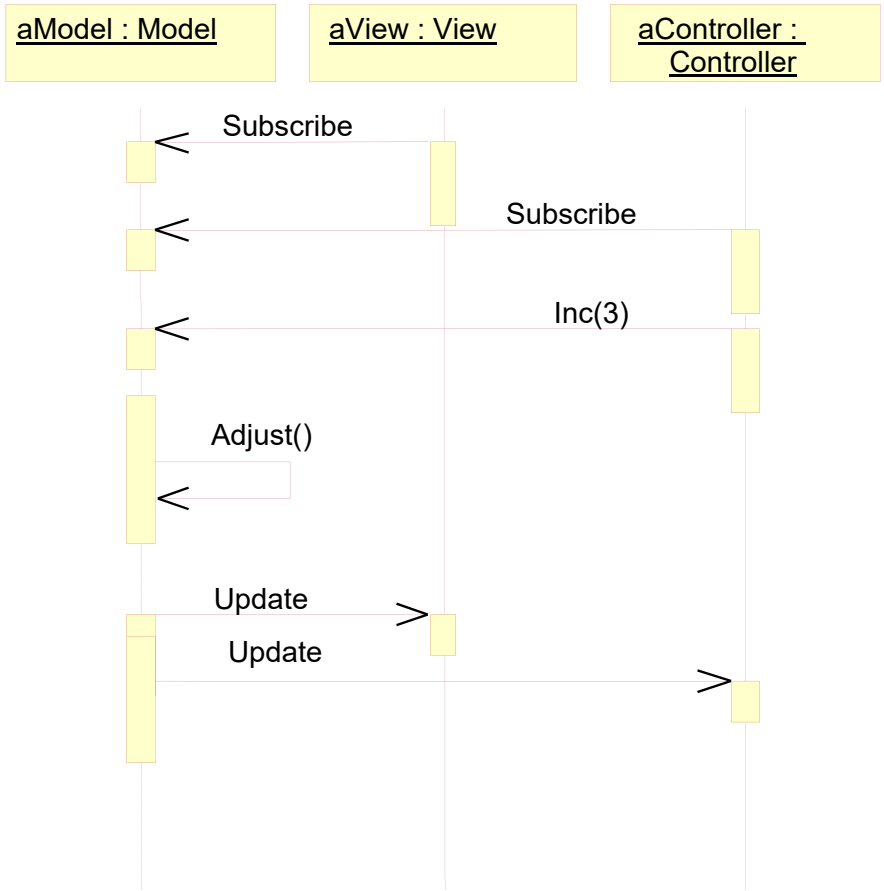


Structure

Le modèle sépare des responsabilités du système dans trois catégories d'objets.

- Le Contrôleur est l'objet qui reçoit, traite et répond à tous les événements produits par les vues et les utilisateurs.
- La Vue est un objet qui présente des données pout être montré au utilisateur. Elle est automatiquement mise à jour régulièrement ou quand les données changent. Un objet de Vue crée le contrôleur parce que ce dernier on devrait être personnalisé selon la vue.
- Le Modèle est l'objet qui rassemble les données de domaine. Il met à jour les vues e contrôleurs, si nécessaire, chaque fois quand un changement de données arrive.

Le Pattern Modèle-Vue-Contrôleur (III)



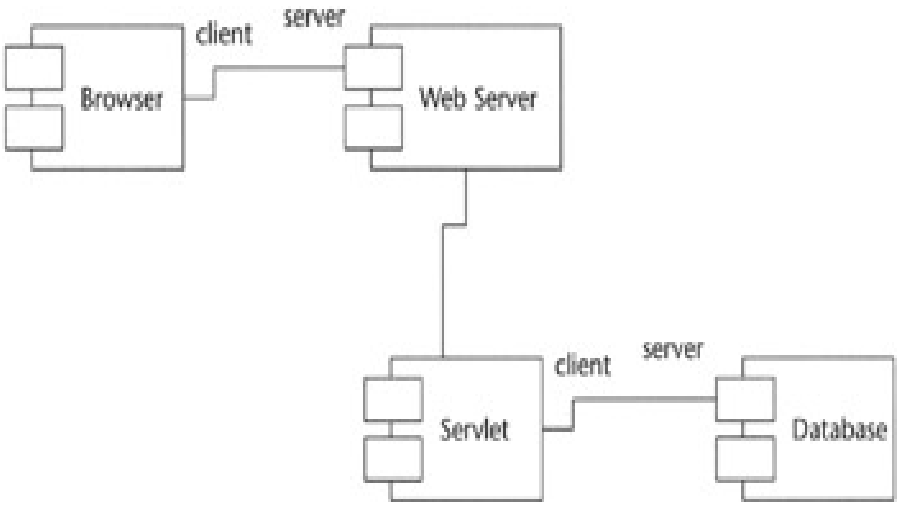
Dans ce scénario c'est le contrôleur que envoie des événements au modèle. Ces événements changent les données dans le modèle. En conséquence, un changements dans le modèle mettent à jour les vues.

Appliquer les Styles Architecturaux au Web (I)

- Considérer une application typique d'entreprise avec une interface utilisateur web (en ligne). Les applications en ligne sont ordinairement organisées comme une architecture disposée en couches ou en tiers.
- La vue en tiers d'une application web:

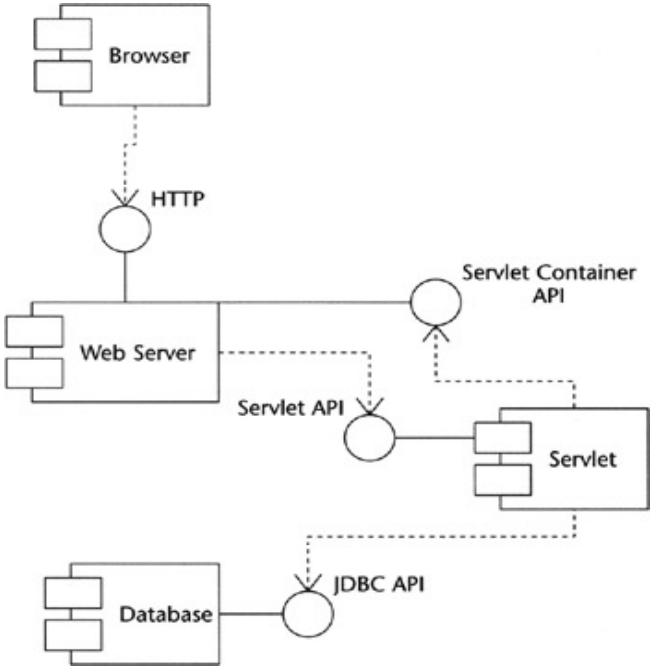


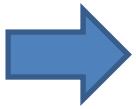
- Les servers Web peuvent invoquer, à la demande du browser, quelque web-component avec une CGI, par exemple servlet, ou quelque autre plug-in.

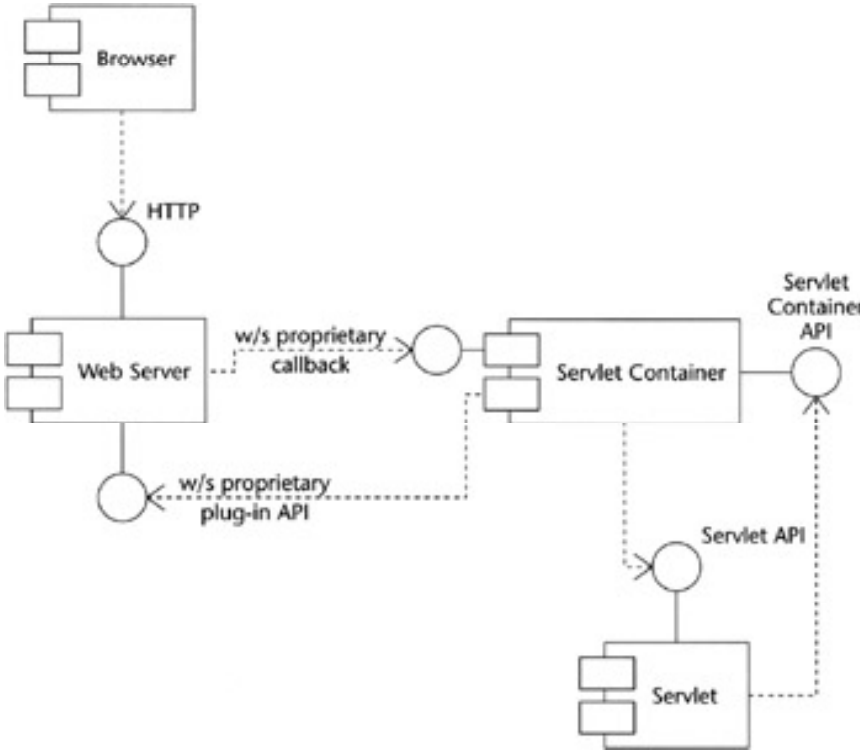


Appliquer les Styles Architecturaux au Web (II)

- Vue physique détaillée d'une application web en tiers :







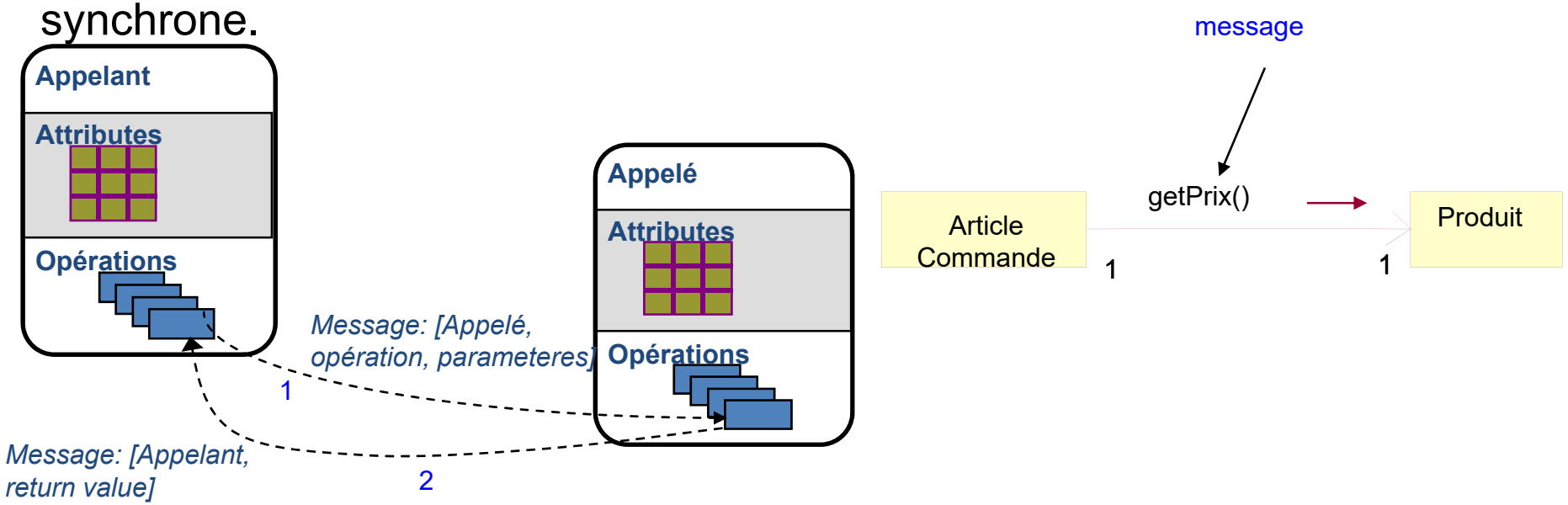
- La vue explicite du « servlet container » :

FIN DU SIXIÈME CHAPITRE

Chap. 7.
Modèle comportemental

Les communications entre objets

- Avoir une opération publique dans une classe signifie que les instances de la classe sont conçues pour recevoir un stimulus ou un message envoyé par autres objets pour demander l'exécution d'opération
- Message = l'instrument avec lequel les objets agissent dans leur environnement. C'est une communication entre deux objets qui transportent des informations d'un objet pour lancer une activité dans l'autre. Dans le orientement objet le modèle de communication est la demande-réponse ou synchrone.

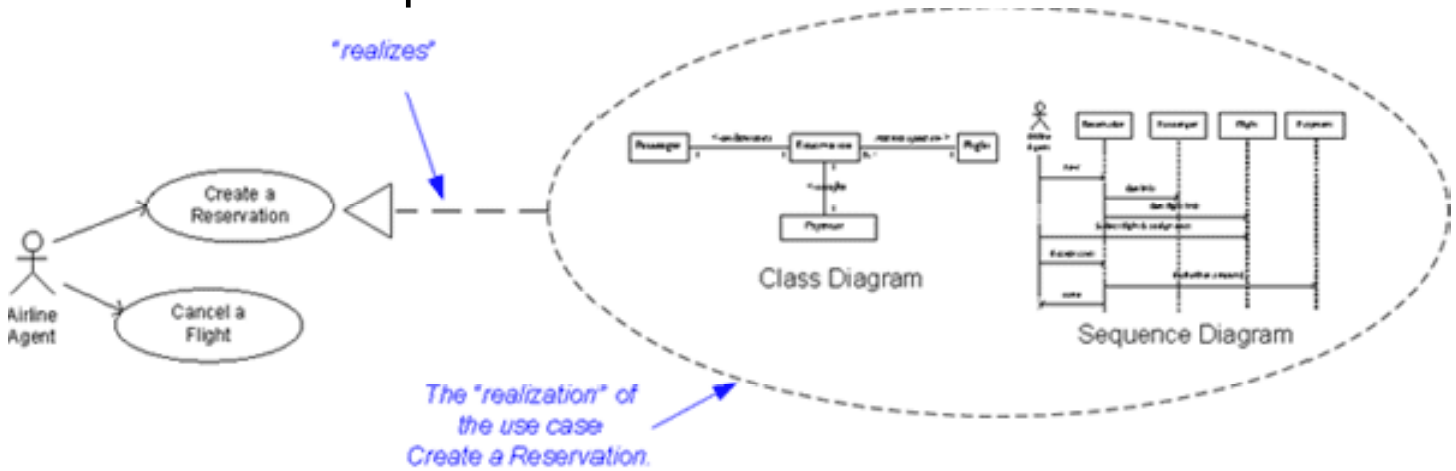


- Un message stimule une réaction dans l'objet appelé: l'exécution de l'opération qui exécute le message. Après l'exécution les objets appelés retournent le contrôle à l'appelant.

Les messages sont la colle dans un système d'OO.

Collaboration

- Les objets dans un système coopèrent typiquement pour produire le *comportement* d'un système. Le comportement est la fonctionnalité que le système est exigé exécuter. Un comportement d'une collaboration sera exposé par une série des instances coopérants qui communiquent mutuellement par l'envoi des signaux ou invocation des opérations.
- Réalisation (dans UML) = Une relation d'abstraction spécialisée entre deux séries d'éléments du modèle, un d'eux représentant une spécification (le fournisseur) et l'autre représente une implémentation du premier (le client). C'est un mélange de généralisation et de dépendance. Il est utilisé dans trois situations :
  - réalisation des Interfaces
  - réalisation des cas d'utilisation
  - réalisation des composants .



Collaboration and Interactions

- La collaboration définit une série de rôles joués par les instances des classes et les liens des associations. Elle décrit comment une opération ou un cas d'utilisation sont exécutés par une structure des éléments (les rôles) qui collaborent, chacun d'eux exécutant une fonction spécialisée, qui collectivement accomplit quelque fonctionnalité désirée. Son but primaire est d'expliquer comment un système travaille et donc typiquement elle incorpore seulement ces aspects de réalité qui est estimé pertinent à la situation.
- Les collaborations nous permettent de décrire seulement les aspects pertinents de la coopération d'une série des instances en identifiant les rôles spécifiques que les exemples joueront.
- Les interactions sont un mécanisme commun pour décrire des systèmes qui collaborent mutuellement. Les plus visibles aspects d'une interaction sont les messages entre les objets. La séquence des messages est considérée important pour la compréhension de la situation.
- Interaction dans OO = la collection de communications (l'échange de messages) entre objets qui incluent aussi la création et la destruction des objets. Les interactions sont chronologiquement ordonnées. Quand deux objets sont connectés une interaction entre eux probablement existe.
- Une interaction peut être décrite dans plusieurs types différents de diagrammes : Diagrammes des séquences (Sequence Diagrams), Diagrammes de vision générale des interactions (Interaction Overview Diagrams) et Diagrammes de Communication (Communication Diagrams).

  Un diagramme d'interaction est un modèle d'interactions entre les objets.

Diagrammes d'interaction

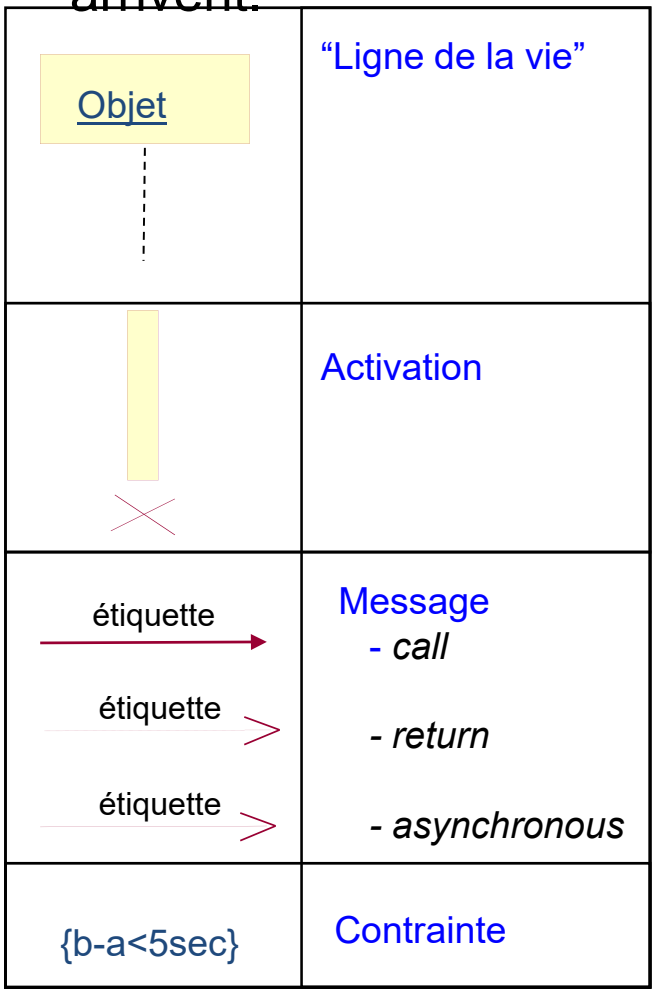
Les interactions nous aident à définir les interfaces de classe.

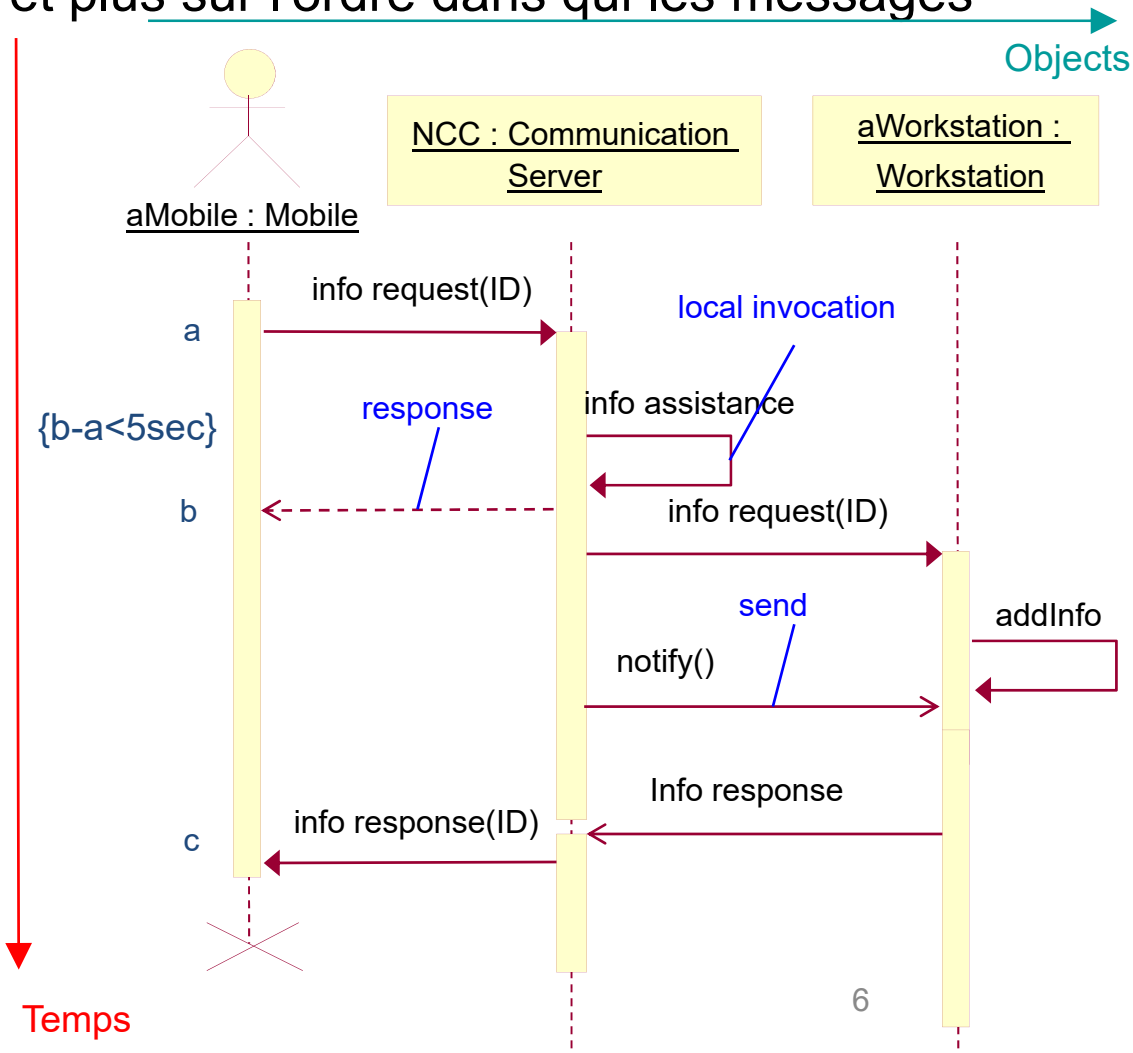
Deux diagrammes :

- Diagramme des Séquences
- Diagramme de Communication

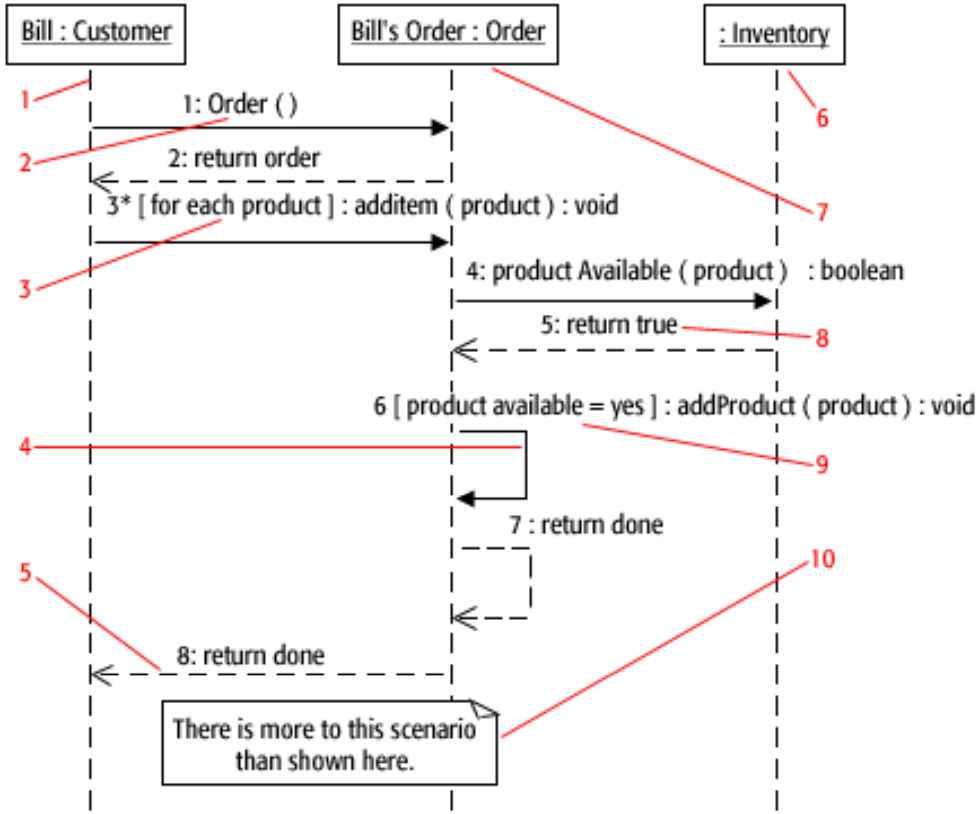
Sequence Diagram

- Le but principal d'un diagramme des séquences est de définir les séquences d'événements qui ont pour résultat quelque issue désirée. L'accent est moins sur les messages se et plus sur l'ordre dans qui les messages arrivent.





Exemple d'un Diagramme de Séquence



1. La ligne de vie de l'objet
2. Message/stimulus
3. Itération
4. Soi-référence
5. Réponse
6. Objet anonyme
7. Nom du objet
8. Numéro de séquence
9. La condition
10. Commentaire

- Le diagramme de séquence illustre des objets qui parlent mutuellement en temps. Le diagramme utilise objets, ligne de vie des objets et flèches pour les message. Le diagramme consiste en une série des objets participants alignés au sommet du diagramme. La ligne de vie est dessinée en bas de chaque objet.

Messages, répétitions, conditions

Syntaxe des message :

prédécesseur expression-séquence valeur-retour := nom-message liste-arguments

A3, B4 / C3.1: res := getLocation (fig)

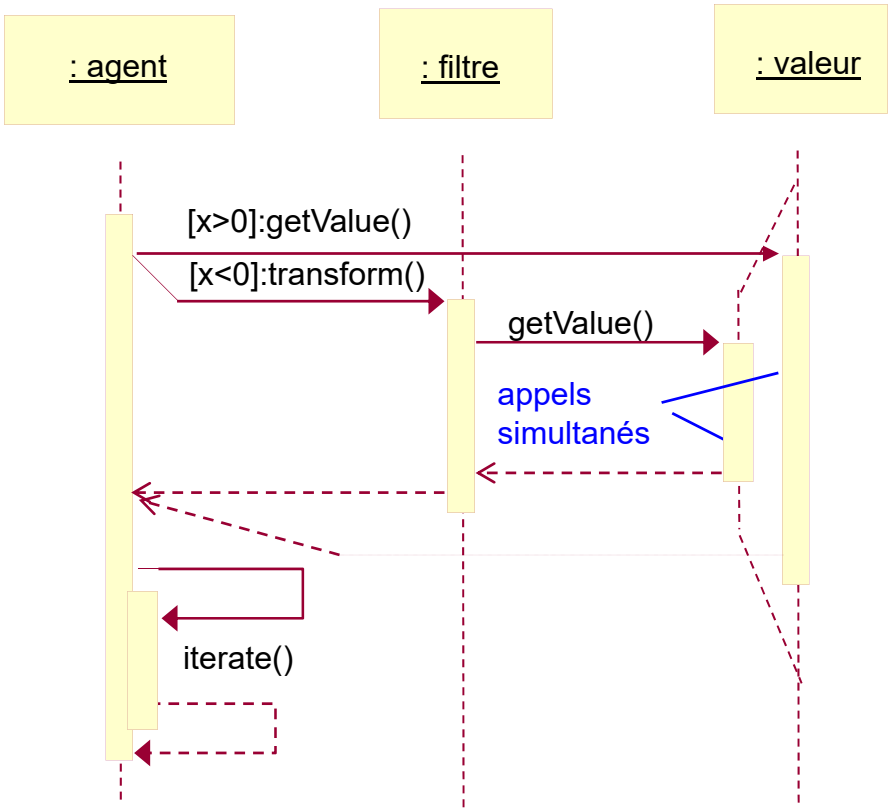
Exemples :

move (5, 7)

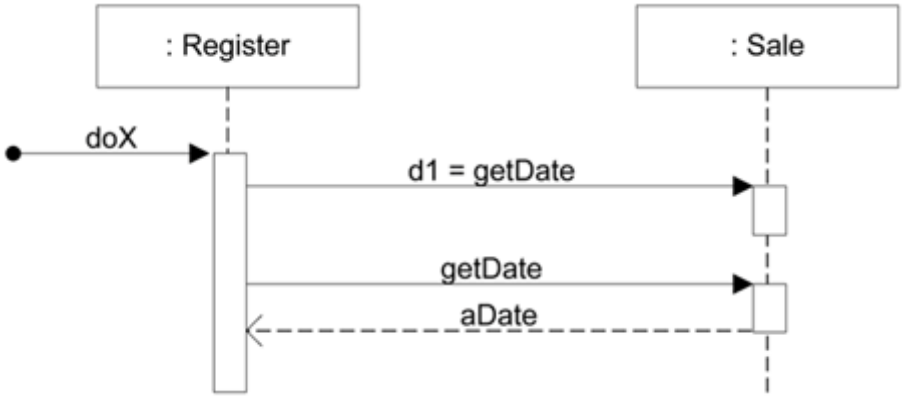
3.7.4 : move (5, 7)

3.7*[1..5] : move (5, 7) itération

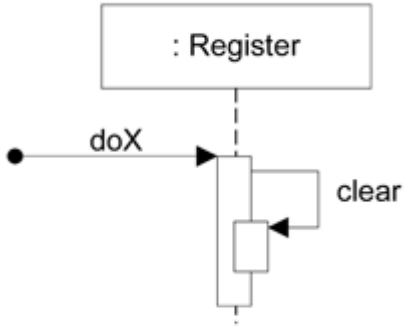
3.7*[z>0] : move (5, 7) condition



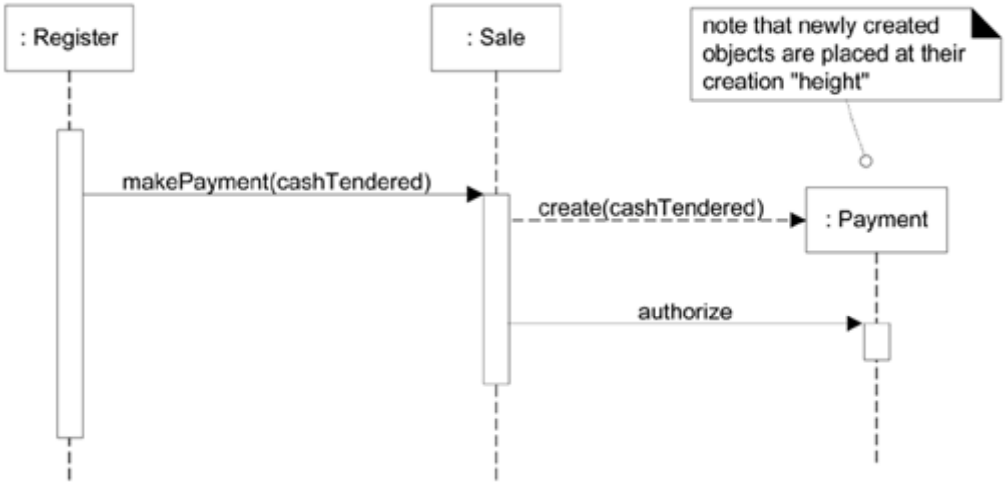
Exemples des Diagrammes de Séquence



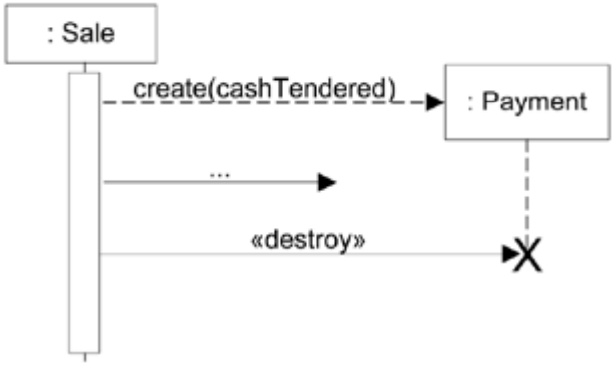
Deux façons pour montrer un résultat



Soi-messages (le message à « this »)

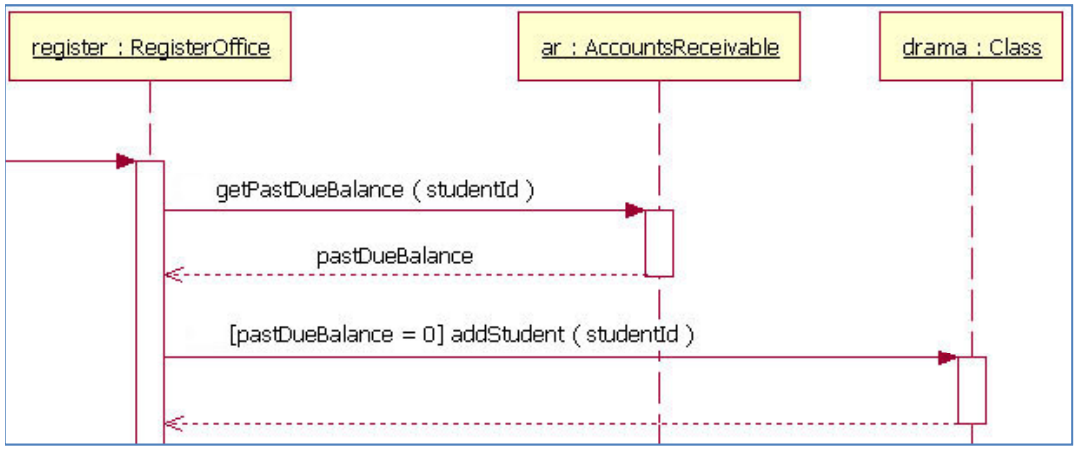


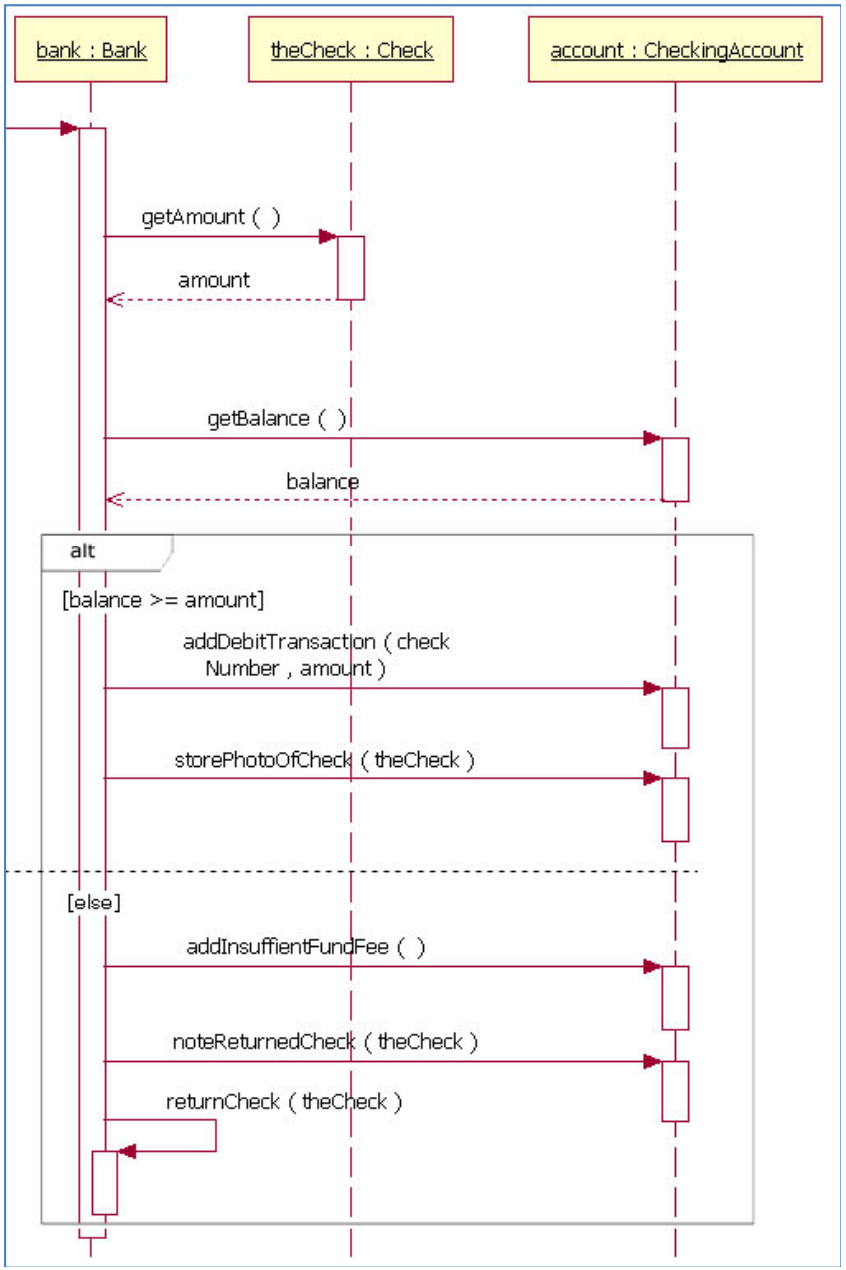
Création d‘une instance



Suppression d‘une instance

Gardes et Alternatives



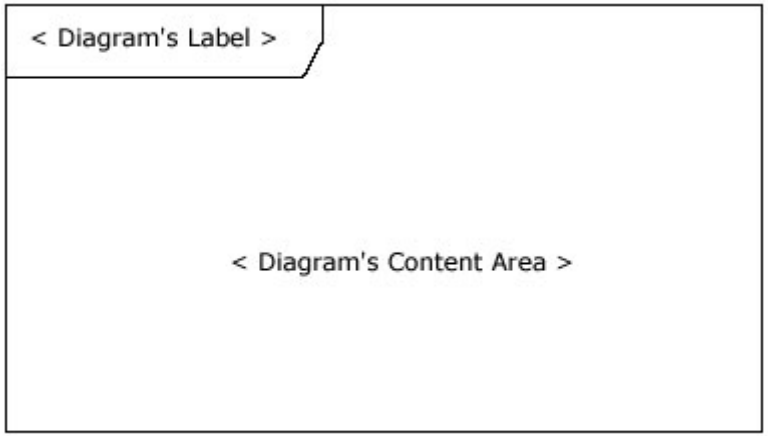


Le but des Diagrammes de Séquences

Le diagramme de séquence est principalement utilisé pour montrer les interactions entre les objets dans l'ordre séquentiel que ces interactions arrivent.

1. Pendant la modélisation métier le personnel peut trouver les diagrammes de séquence utiles à communiquer comment le business actuellement va à travers la présentation du comment les divers objets-métier réagissent réciproquement pendant les divers scénarios représentés dans les cas d'utilisation: au niveau métier un diagramme de séquence peut être utilisé comme la documentation des besoins pour une implémentation future du système. Les cas d'utilisation sont souvent raffinés dans un ou plusieurs diagrammes de séquence.
2. Un personnel technique de l'organisation peut trouver les diagrammes de séquence utiles dans documenter comment un système futur devrait se comporter. Pendant la phase de conception, les architectes et les entrepreneurs peuvent utiliser le diagramme pour illustrer les interactions du objet Système, accentuant ainsi la conception générale du système.
3. Un des usages primaires des diagrammes de séquence sont dans la transition des besoins exprimées comme cas d'utilisation au prochain et plus formel niveau de raffinement.
4. En plus de leur usage dans concevoir nouveaux systèmes, les diagrammes de séquence peuvent être utilisés pour documenter comment les objets dans un déjà existant (appelé « legacy ») système actuellement réagit réciproquement. Cette documentation est très utile quand on transfère le système à une autre personne ou à une autre organisation.

L'Elément Frame (Cadre)



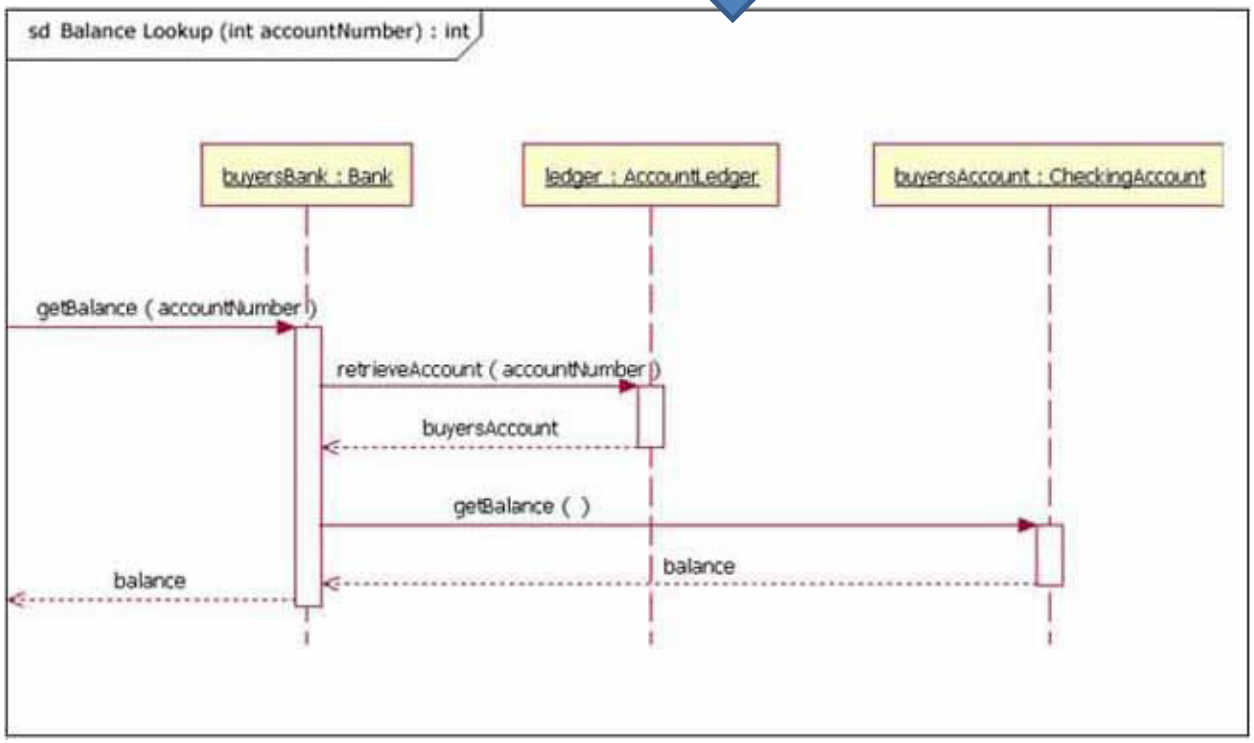
- Un cadre fournit un lieu cohérent pour une étiquette du diagramme, en fournissant une frontière graphique pour le diagramme.

Un cadre vide

Un diagramme de séquence qui a des messages entrants et sortants

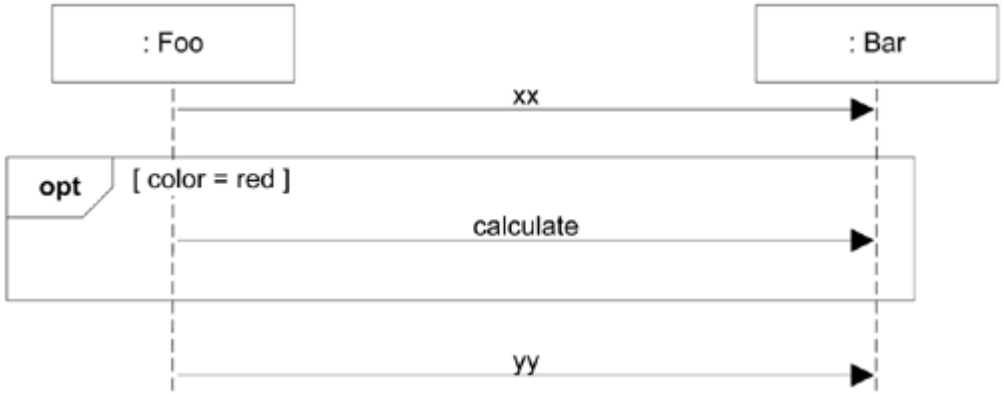
étiquette du diagramme :
<Diagram Type> <Diagram Name>

Type du diagramme :
sd = Diagramme de Séquence
activity =Diagramme d'activité
use case = Diagramme des cas d'utilisation

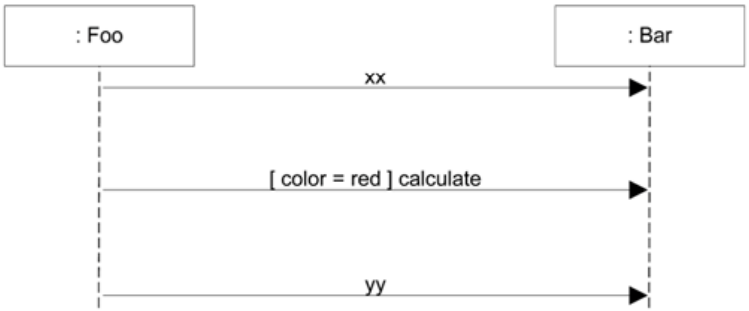


Overview Diagrams (I)



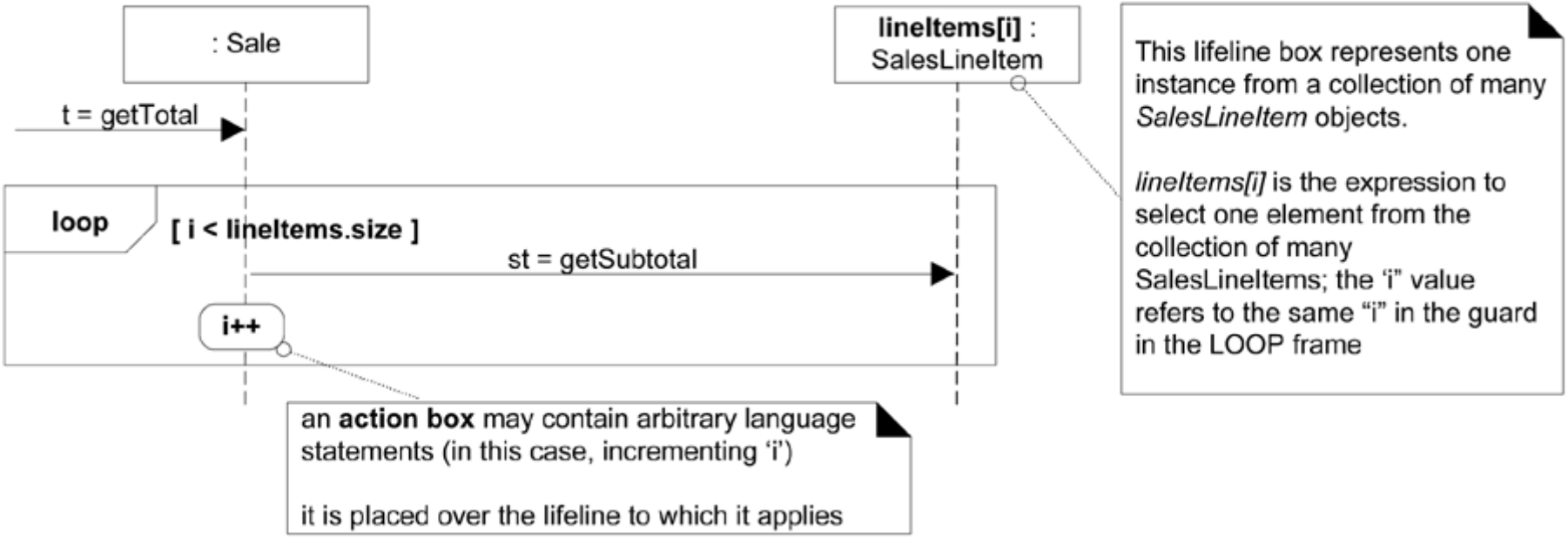


Frame Operator	Meaning
alt	Alternative fragment for mutual exclusion conditional logic expressed in the guards.
loop	Loop fragment while guard is true. Can also write *loop(n)* to indicate looping n times. There is discussion that the specification will be enhanced to define a *FOR* loop, such as *loop(i, 1, 10)*
opt	Optional fragment that executes if guard is true.
par	Parallel fragments that execute in parallel.
region	Critical region within which only one thread can run.

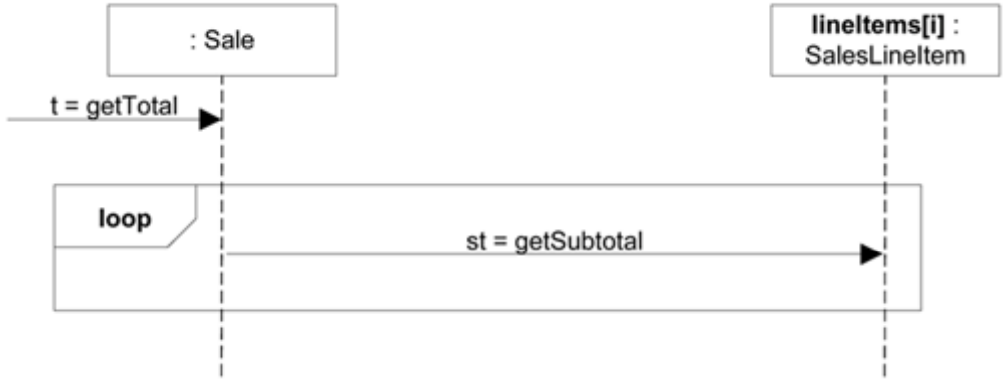


UML 1.x

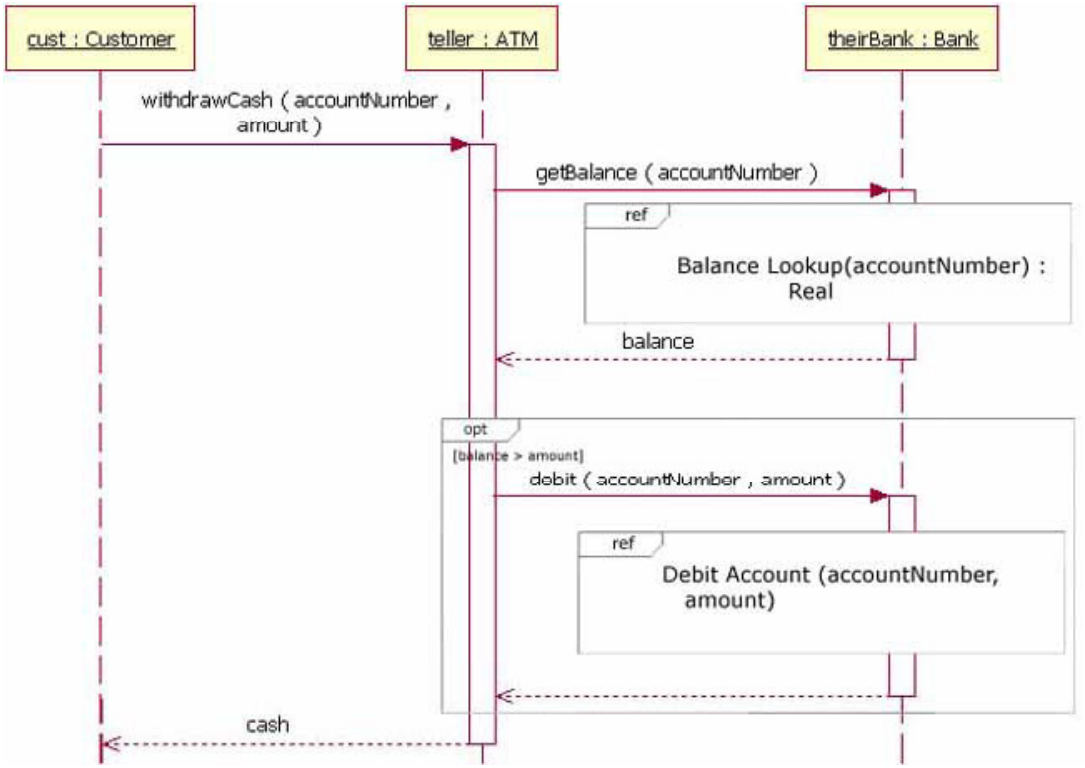
Overview Diagrams (II)



Plus compact :



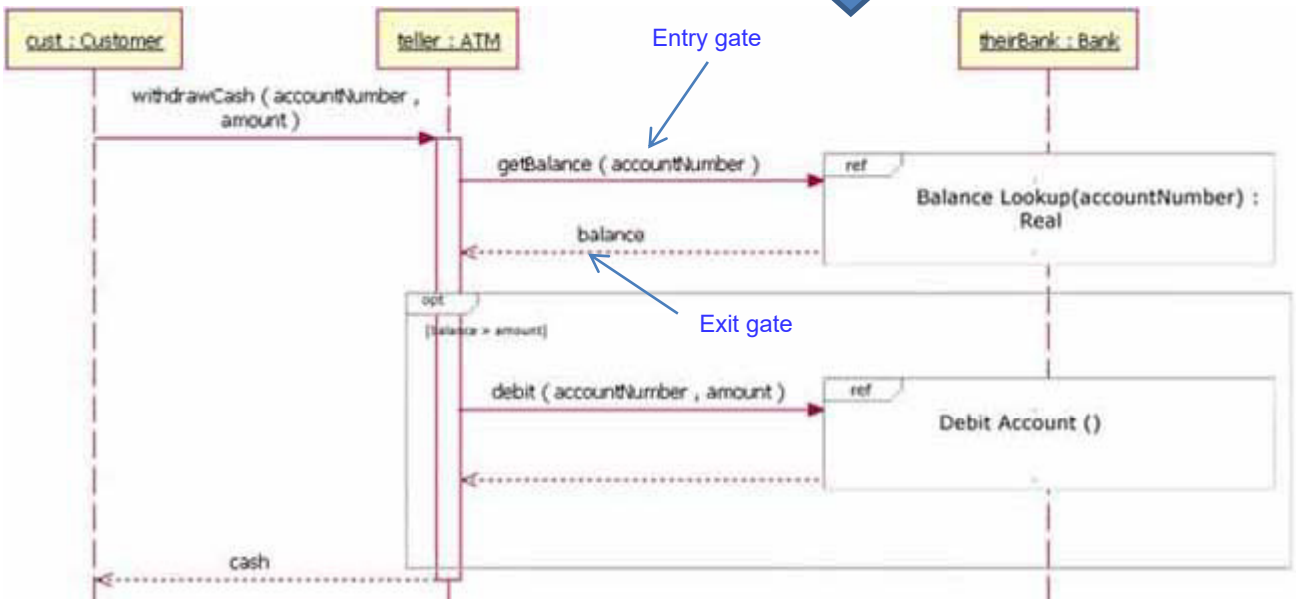
Gates



Un diagramme de séquence qui adresse deux diagrammes de séquence différents

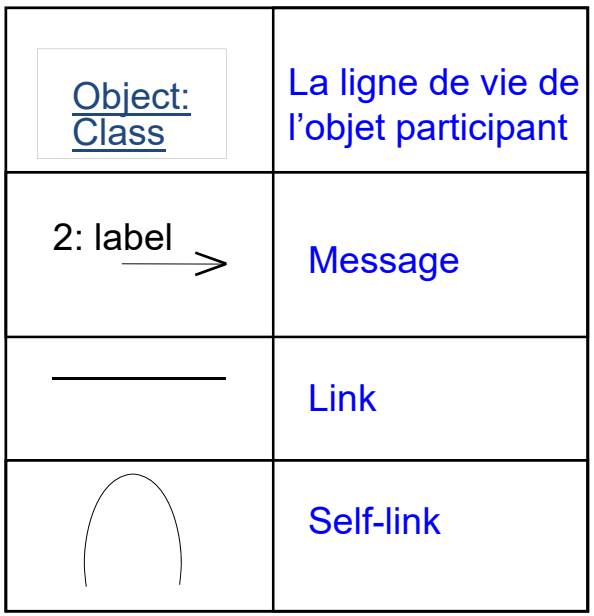
Le même diagramme utilisant gate cette fois

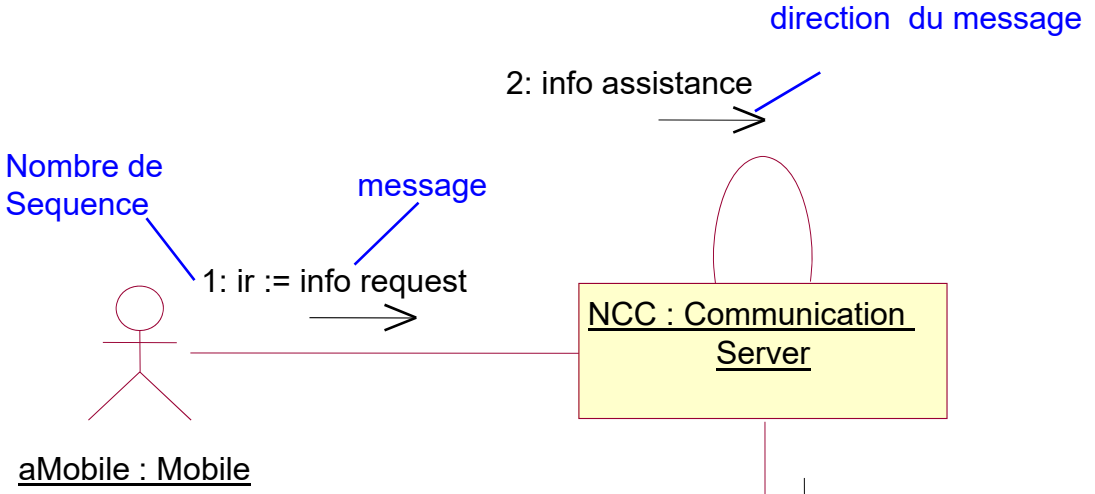
Un gate est un message qui est illustré avec une fin connectée au bord du cadre du diagramme de séquence et l'autre fin connecté à une ligne de vie.

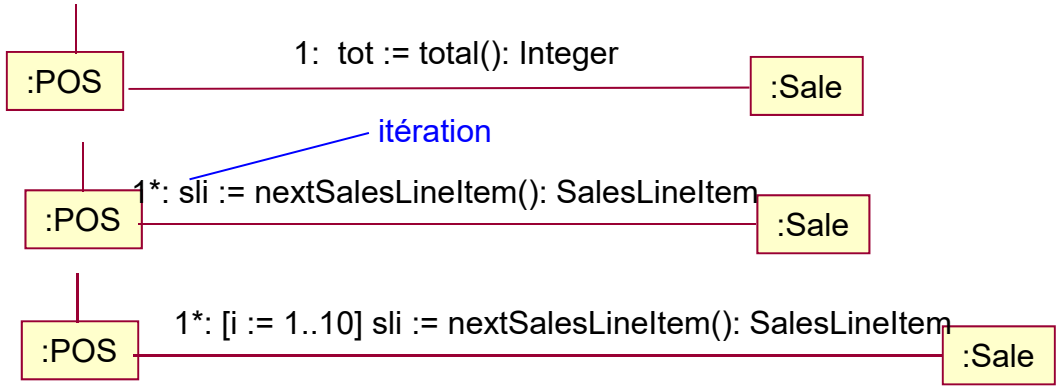


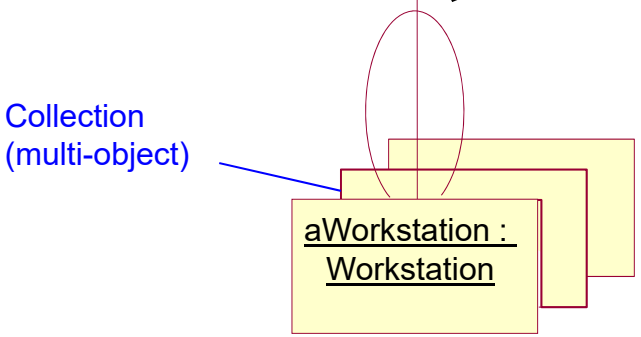
Diagrammes de communication

Les diagrammes de communication se concentrent sur l'architecture de la structure interne et comment ceci correspond avec le passage de message. L'ordre chronologique est indiqué par nombres structurés de séquence.



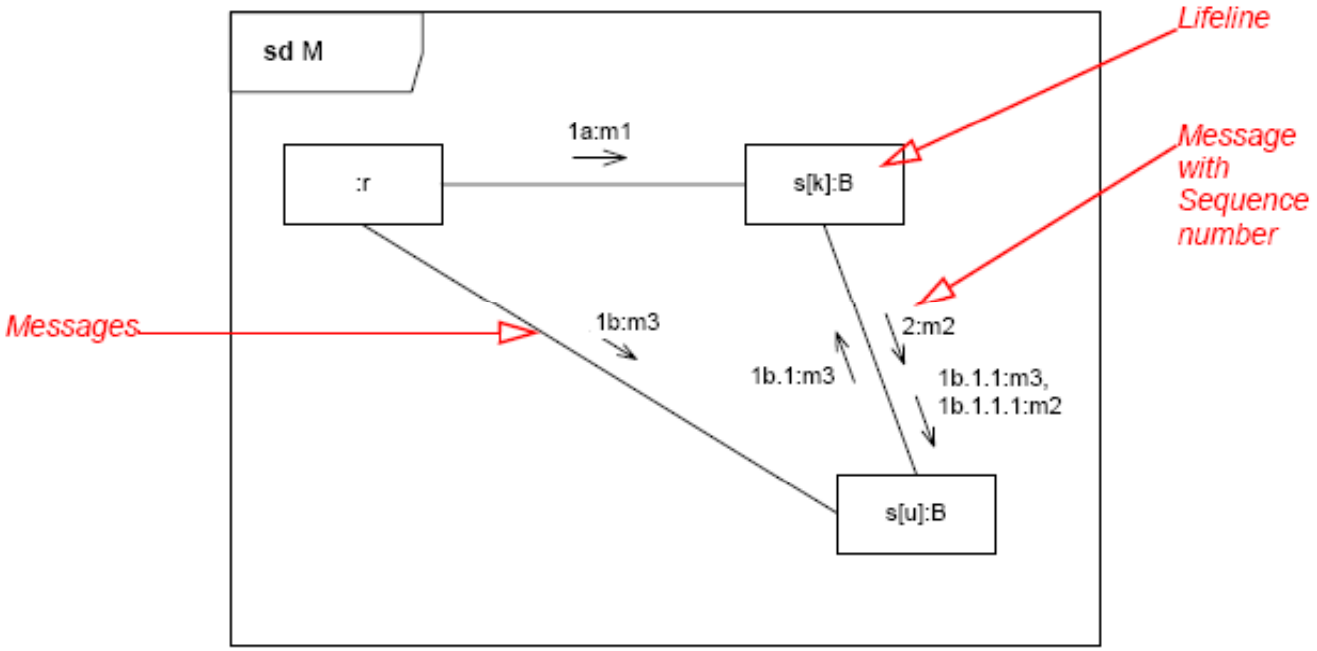






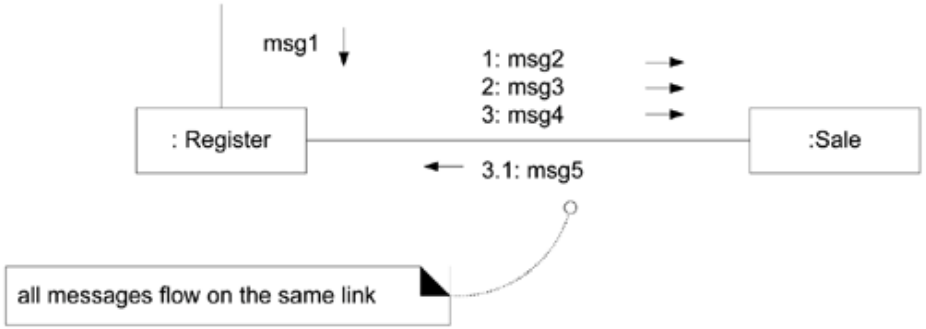
La sequence des messages

- Le diagramme montre les messages m1 et m3 comme concurrents, simultanément envoyés par : r vers deux instances du s. Le nombre de séquence indique comment les autres messages sont séquencés. 1b.1 suit après 1b et 1b.1.1 par la suite etc. 2 suive après 1a et 1b.

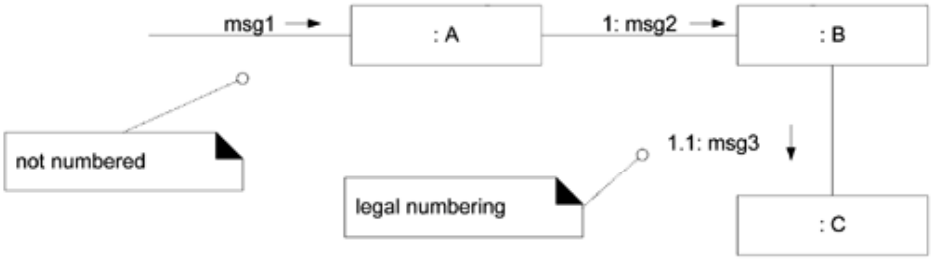


Exemples (I)

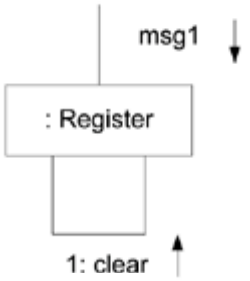
Messages



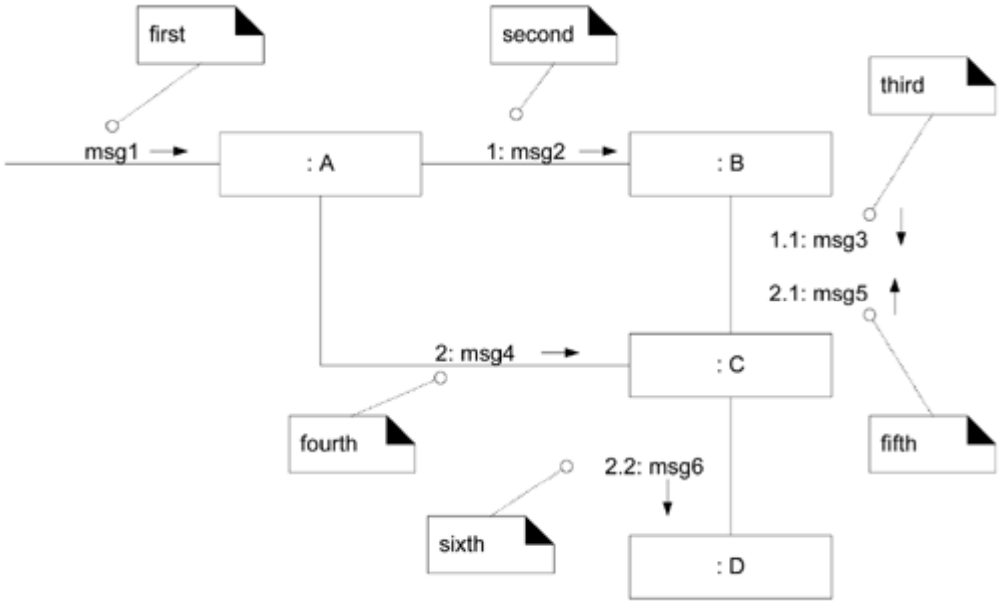
Numérotation des Séquences



Message vers “soi-meme”

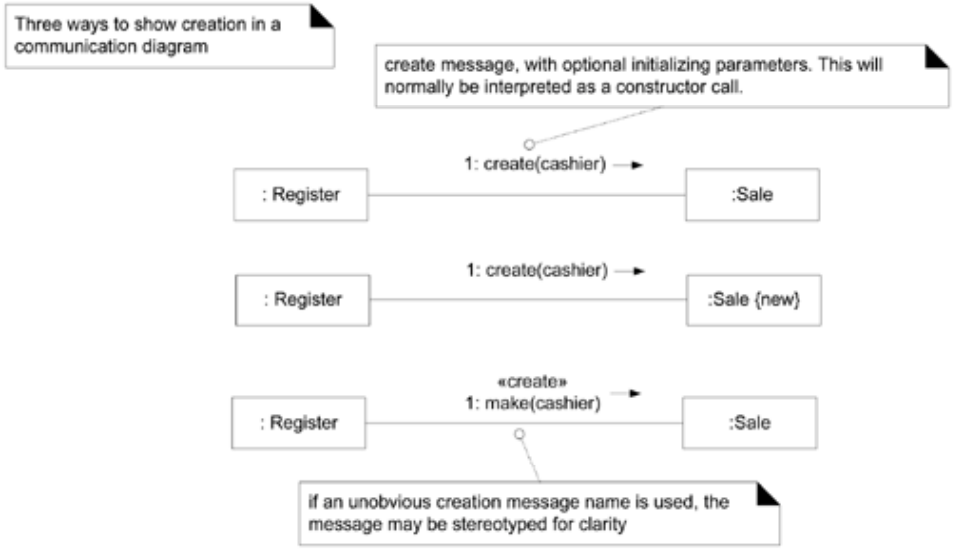


Numérotation complexe des séquences

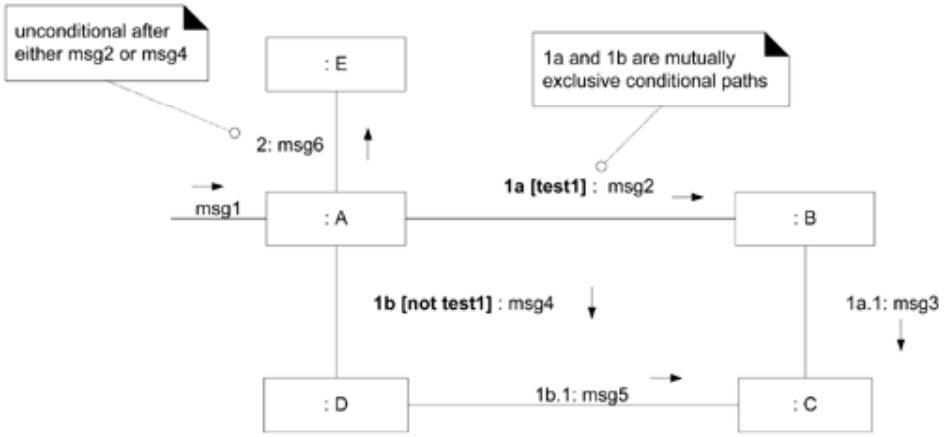


Exemples (II)

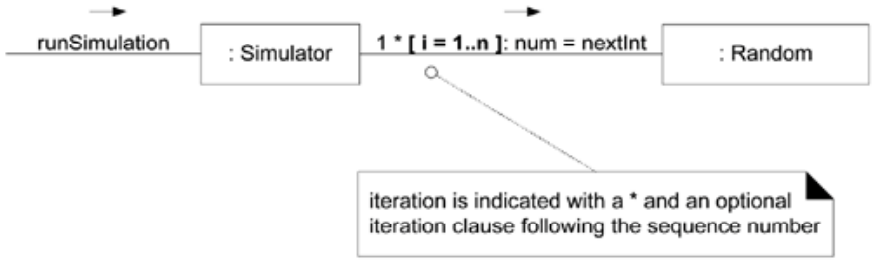
Creation d'une instance



Executions conditionnels mutuelment exclusifs



Répétitions & Cycles



Répétition à travers une Collection

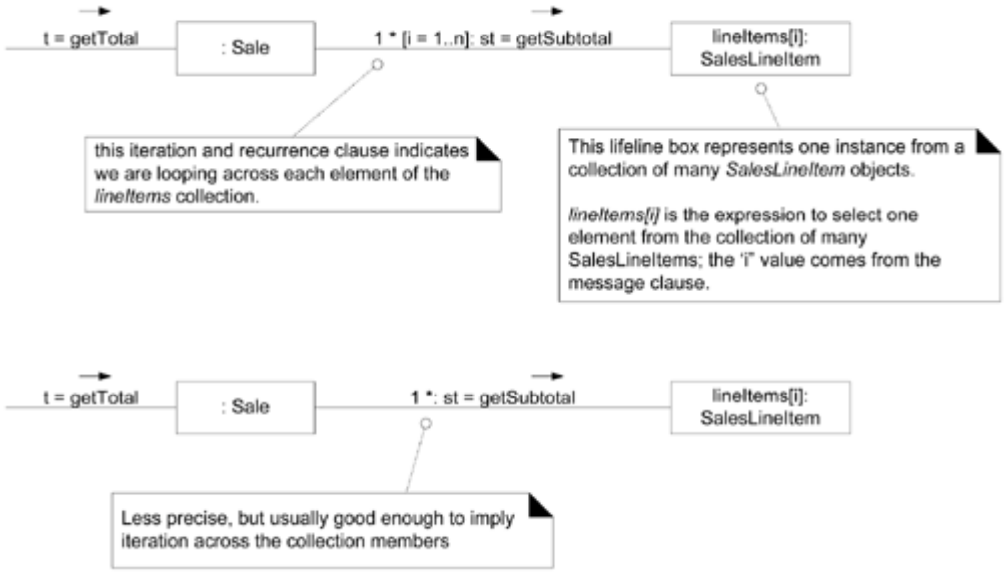


Diagramme de communication

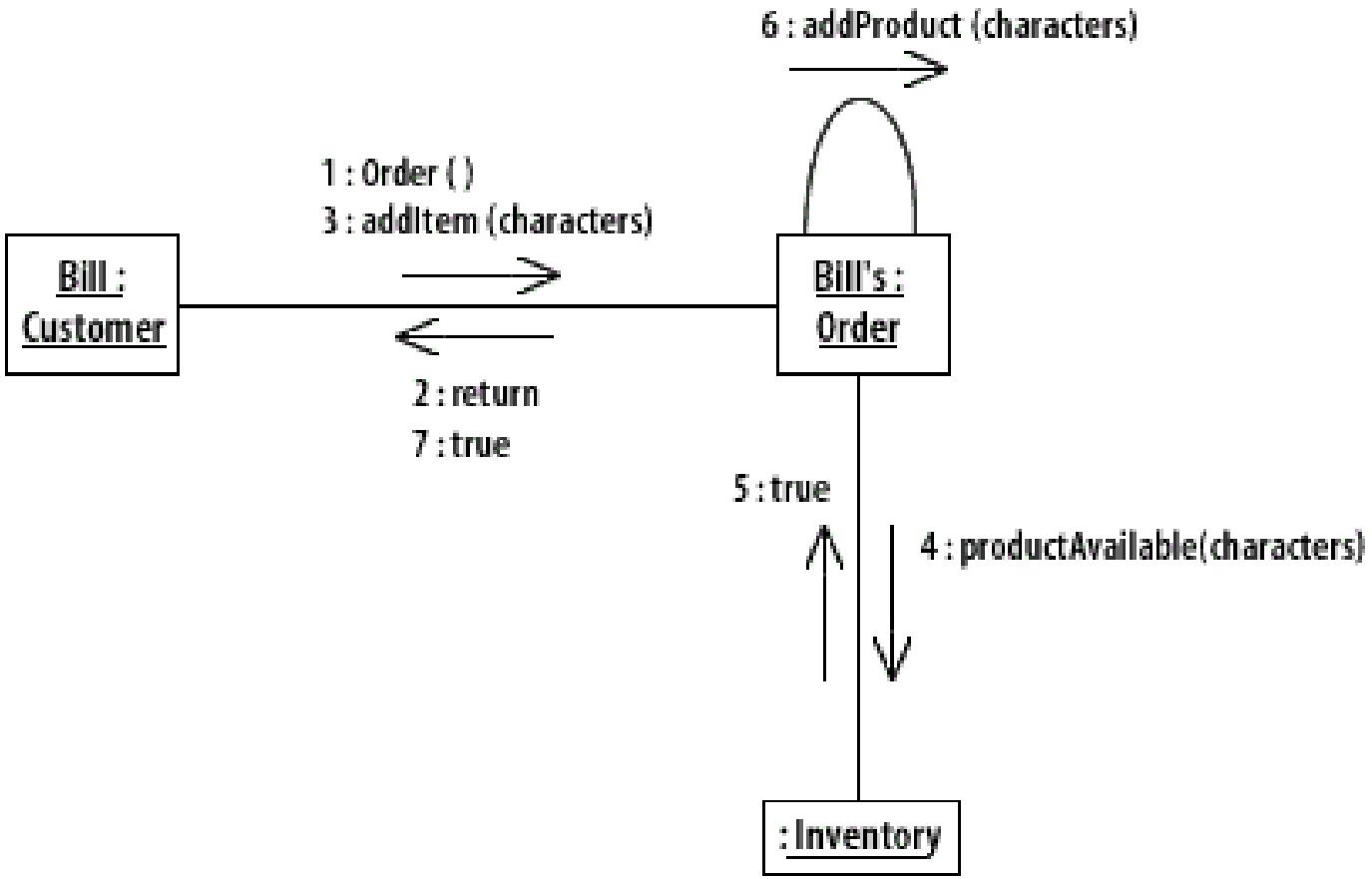
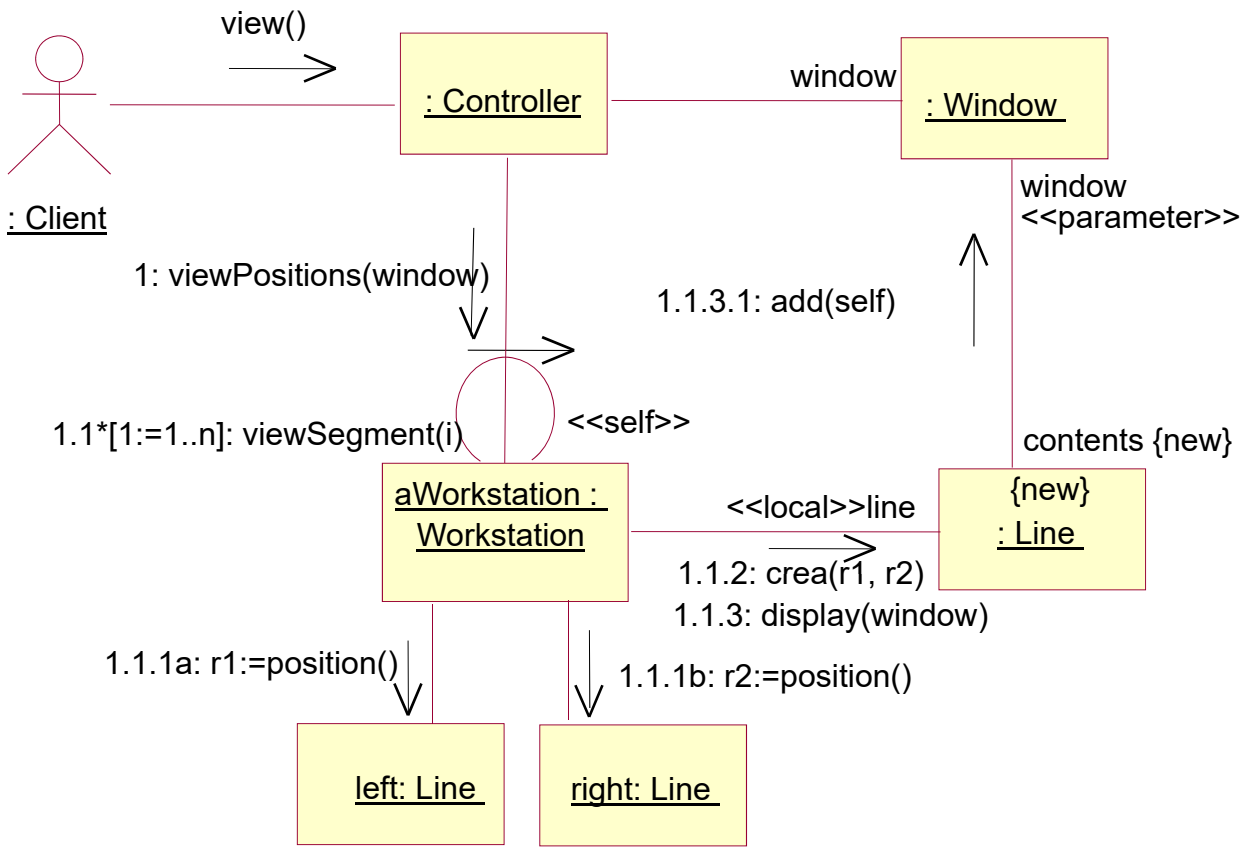
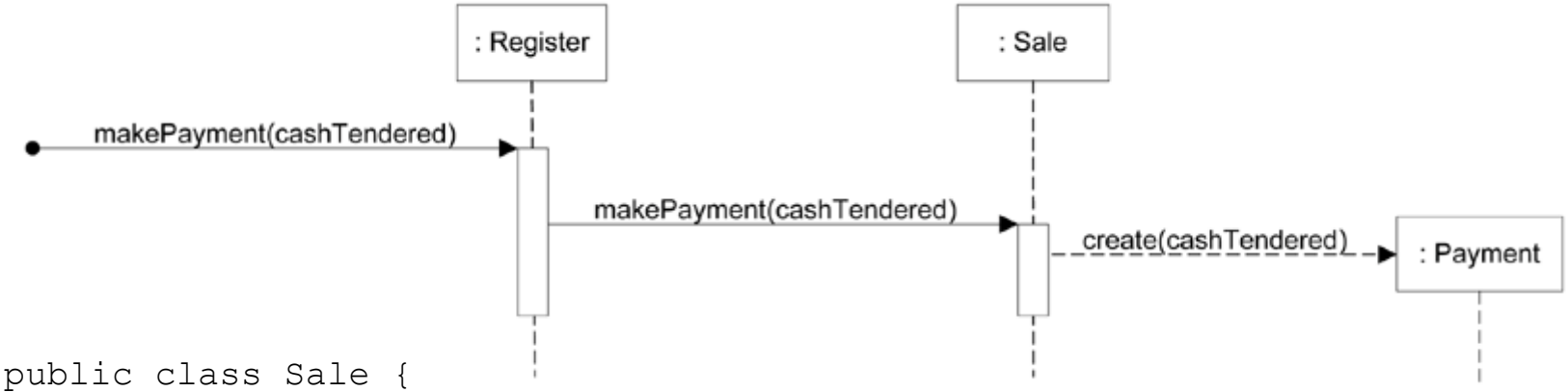


Diagramme de communication . Exemple



Equivalence des Diagrammes d'Interaction




```
public class Sale {
private Payment payment;
 public void makePayment( Money cashTendered ) {
   payment = new Payment( cashTendered );
 //…
 }
 // …
}
```

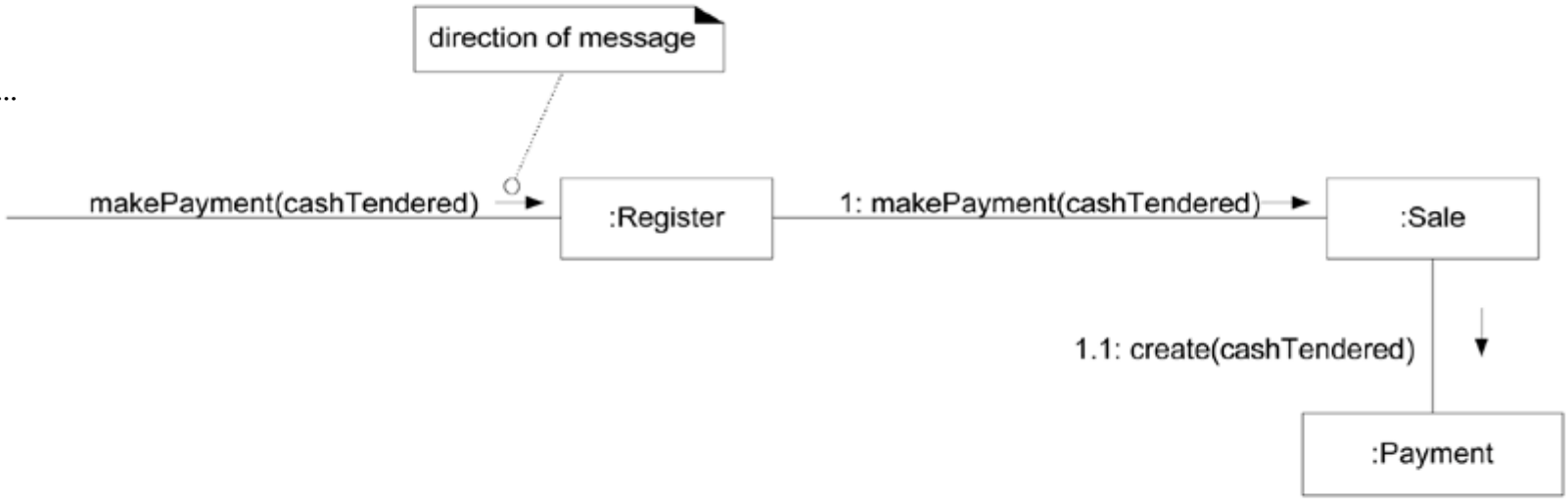


Diagramme de communication de système

1. Créer un diagramme de communication pour chaque opération de système et cas d'utilisation associé.
2. Utilisant les responsabilités de système, les post-conditions de contrat et l'usage-cas associé établit pour concevoir les objets qui appartiennent à la collaboration qui se rend compte le cas d'usage.
3. Si le diagramme est trop complexe, le diviser dans plus diagrammes.
4. Appliquer les design patterns.

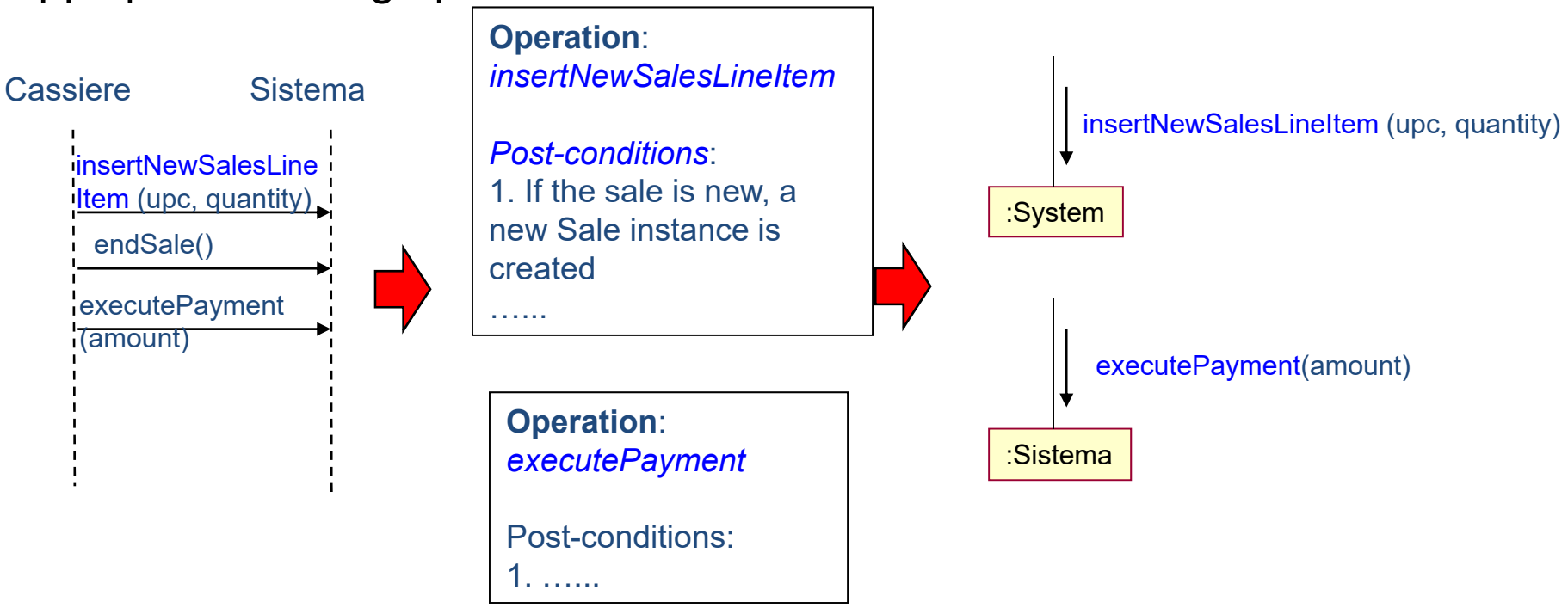
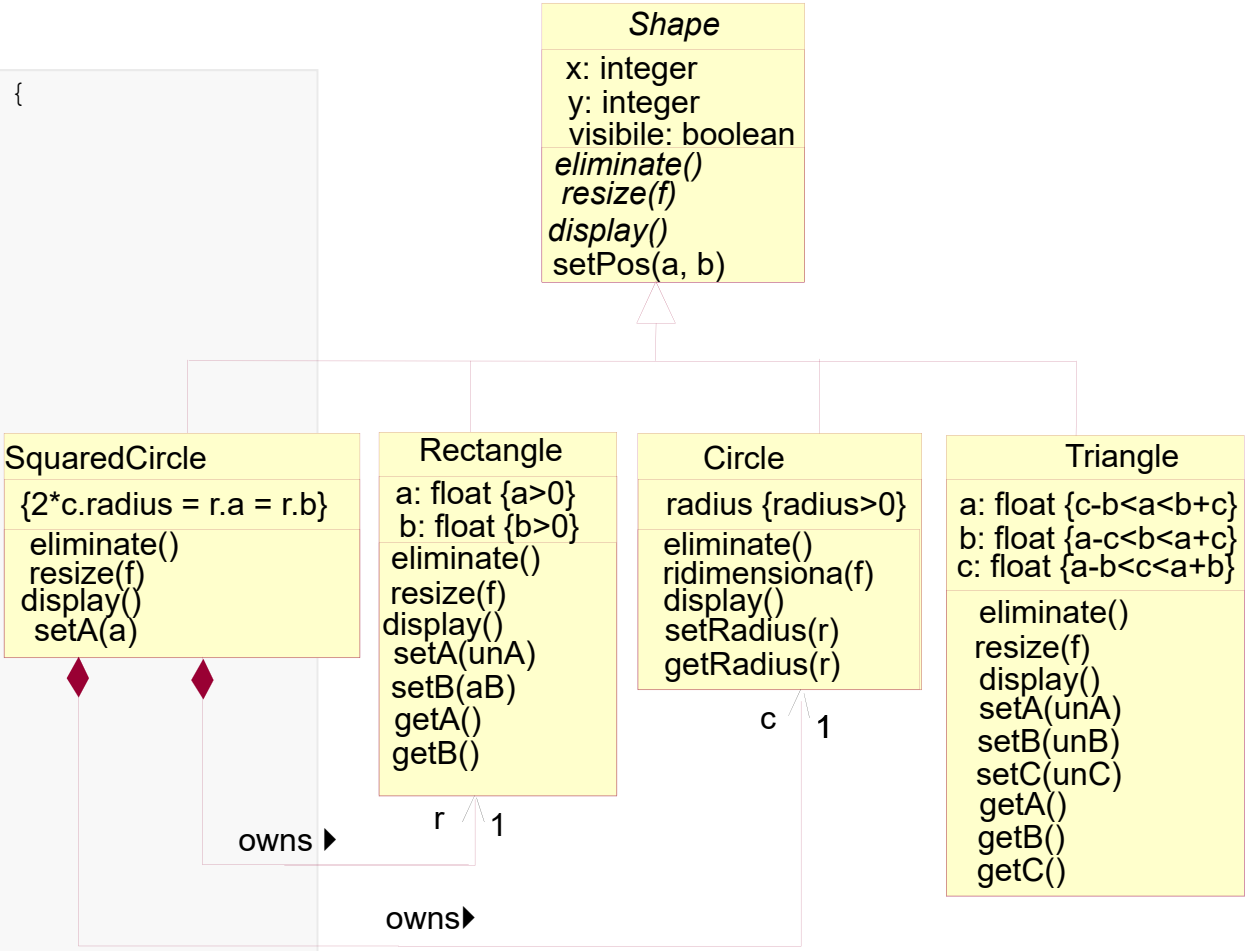


Diagramme de Séquence de système | Contractes | Diagrammes de Communication de système

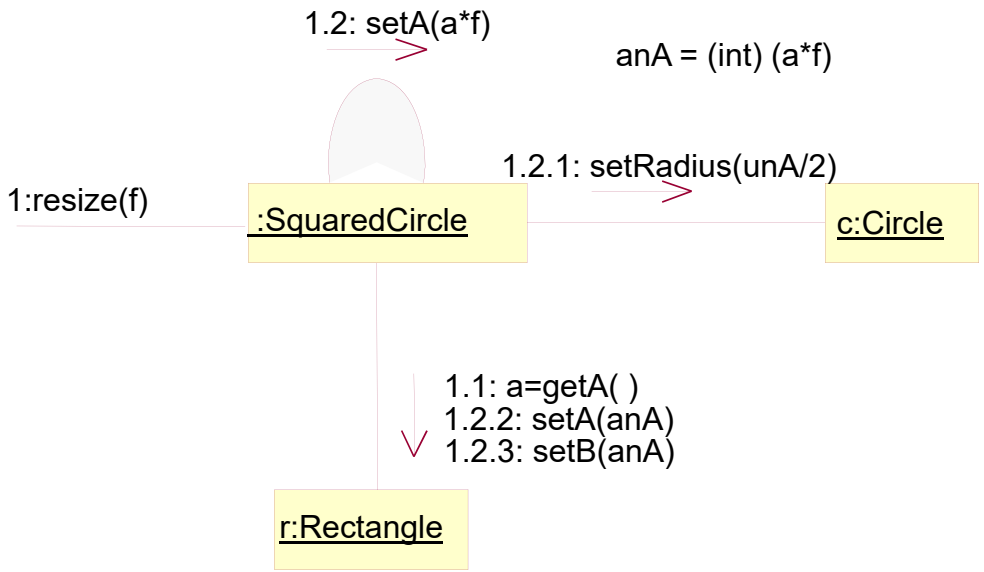
Aggrégation avec collaboration (I)

- Nous voulons ajouter une nouvelle forme géométrique : SquaredCircle qui est composée d'un cercle et un carré.
- Nous voulons aussi réutiliser les classes existantes.
- Un objet de SquaredCircle pourrait agréger un Circle et un Rectangle.

```
class SquaredCircle extends Shape {
  Cercle c;
  Rectangle r;
  public void display() {
    c.display(),
    r.display();
  }
  public void eliminate() {
    c.eliminate();
    r.eliminate();
  }
  public void setA(int a) {
    c.setRadius(a/2);
    r.setA(a);
    r.setB(a);
  }
  public void resize(float f){
    int a;
    a=r.getA();
    setA((int)(a*f));
  }
}
```



Aggrégation avec collaboration (II)



Assigner Responsabilité aux Classes

Responsabilité et collaboration

Responsabilité = Un contrat ou un devoir d'une classe dans un diagramme UML.

Les responsabilités représente les devoirs ou le comportement d'un objet dans les termes de son rôle dans l'application. Fondamentalement, ces responsabilités sont de deux types : faire et savoir.

Les responsabilités « faire » d'un objet incluent :

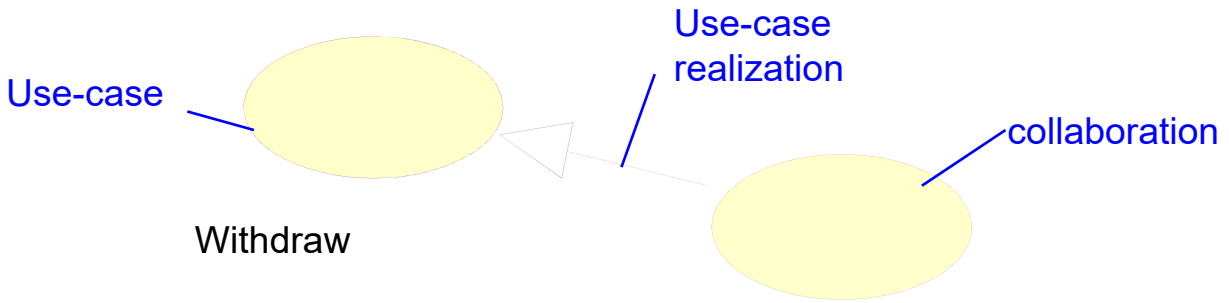
- faire quelque chose lui-même, comme par exemple créer un objet ou faire un calcul
- lancer l'action dans autres objets
- contrôle et coordonnation des activités dans les autres objets

Les responsabilités « savoir » d'un objet incluent :

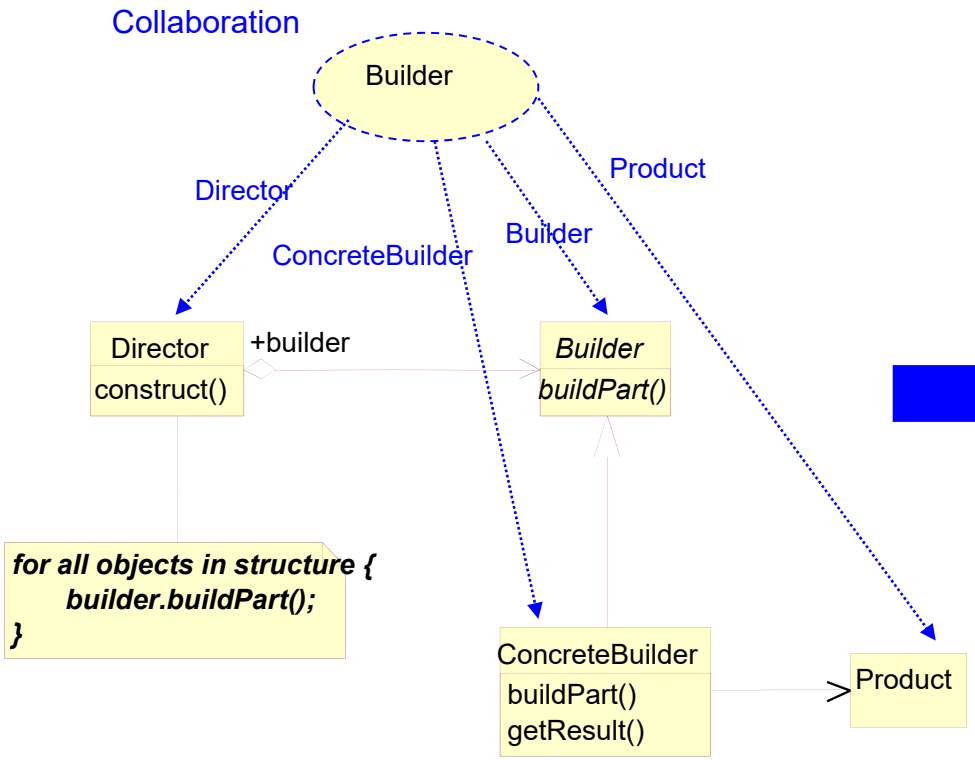
- savoir ses données encapsulées, privées
- savoir des objets liés
- savoir des choses d'où il peut dériver ou calculer résultats.

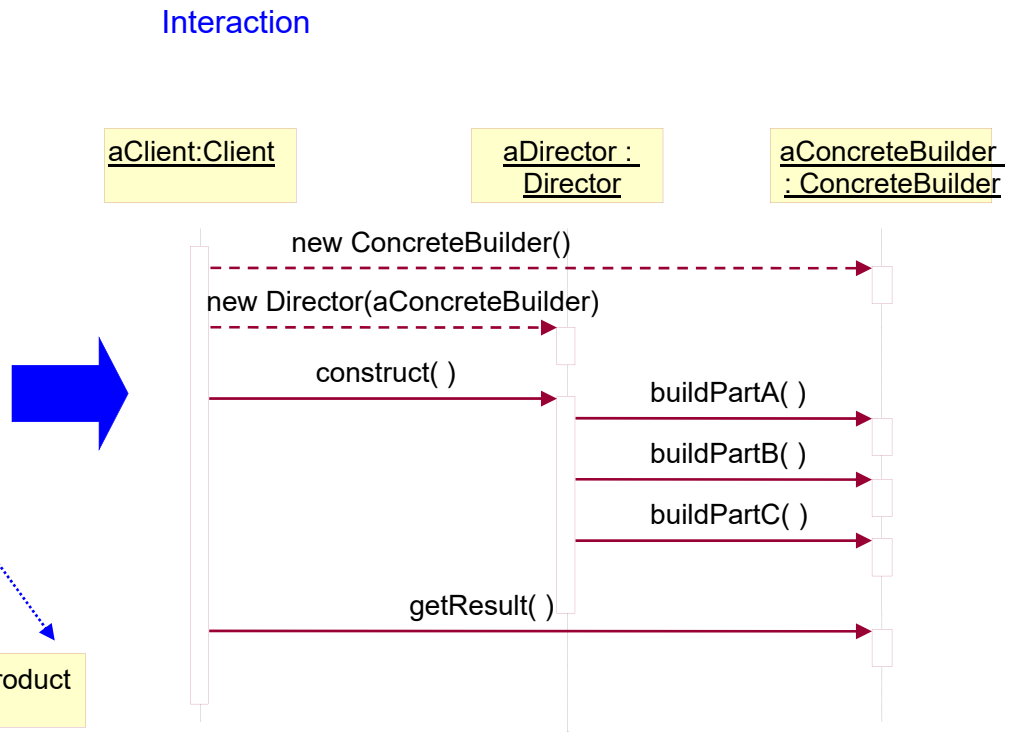
- Les responsabilités sont assignées aux classes d'objets pendant la conception d'objet. Par exemple, nous pouvons déclarer qui « une Vente est responsable de créer de ArticlesVente » (« faire »), ou « une Vente est responsable de savoir son total » (« savoir »).
- Les collaborateurs d'une classe = Toutes les classes connues par la classe et utilisés pour remplir les responsabilités de la classe.

Collaboration vs. Interaction



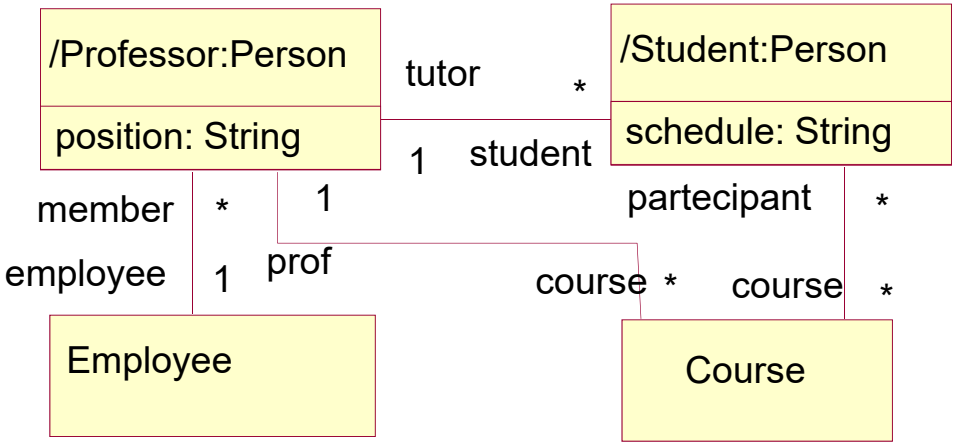
Design Pattern



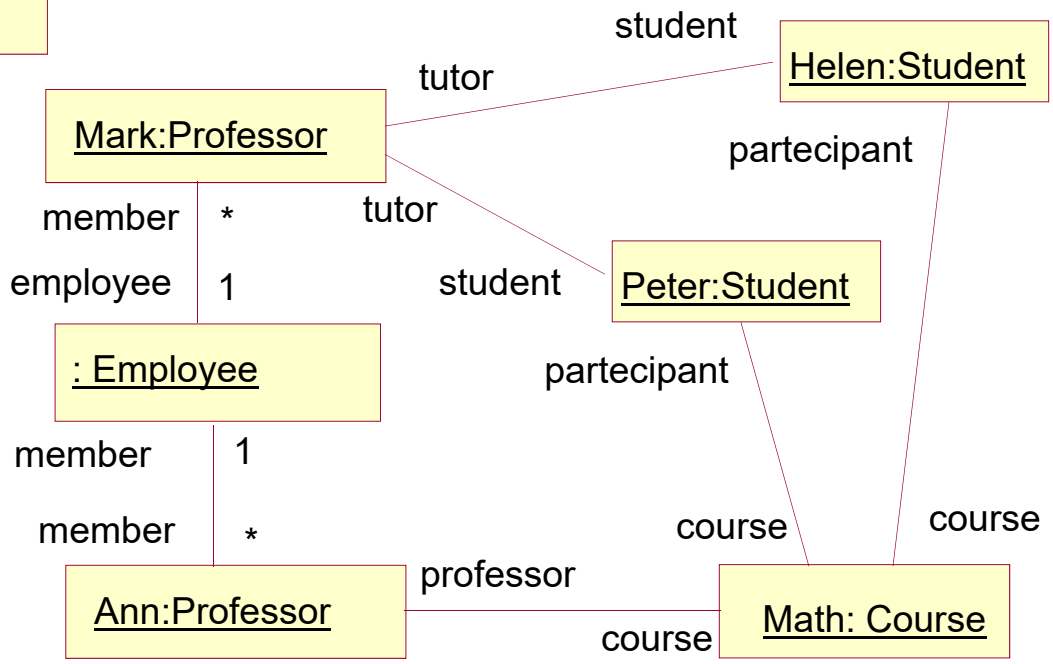


Collaborations

Collaboration - Specification



Collaboration - Instances



Comment utiliser les collaborations

- L'usage de collaborations :
  - pour trouver classe de le et les relations.
  - pour tracer des opérations d'un cas d'utilisation avec les classes identifiées précédemment
  - pour faire correspondre les spécifications à leurs réalisations.

Ne pas oublier :

- La collaboration est à la fois structure et comportement pertinent pour une tâche.
- Un rôle dans une collaboration est une abstraction d'un instance, pas d'une classe !
- Identifier dans une collaboration les rôles suivants :
  - Initiateurs (entités externes)
  - Manipulateurs (entités actives)
  - Manipulé (entités passives)

CRC Cards

- Une technique de modélisation texte-orienté populaire est l'utilisation des cartes de Responsabilité-Collaboration des Classes (CRC) , créées par Kent Beck et Ward Cunningham (aussi les fondateurs des idées de XP et des design patterns).
- CRC cards are paper index cards on which one writes the responsibilities and collaborators of classes. Each card represents one class. A CRC modeling session involves a group sitting around a table, discussing and writing on the cards as they play "what if" scenarios with the objects, considering what they must do and what other objects they must collaborate with.
- Les cartes de CRC sont des fiches en papier sur lequel on écrit les responsabilités et les collaborateurs d'une classes. Chaque carte représente une classe.
- Une séance de modelage de CRC implique une séance d'une équipe autour une table, discutant et écrire sur les cartes pendent on joue des scénarios avec les objets: que-est-ce qu'ils doivent faire et que-est-ce autres objets doivent faire pour collaborer avec eux.

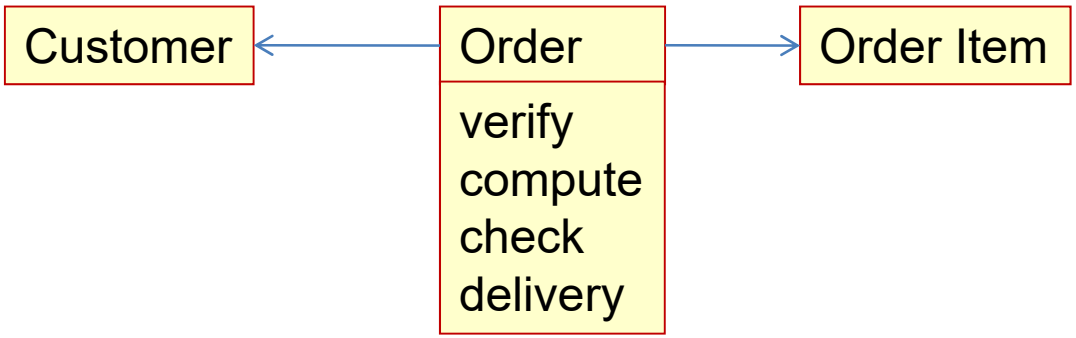
Les Cartes CRC

Class Name		
Class Category (device, property, role, event ….)		
Characteristics (tangibile, atomic, concurrent …)		
Responsibilities		**Collaborators**

Les cartes de CRC sont utilisées pour identifier et structurer les plus importantes classes dans une application, comme elles résultent des besoins.

Responsabilités et collaborations avec CRC

Order		
Thing		
Concrete, aggregate, sequential, persistent, protected		
Responsibility		**Collaborators**
Verify items in the store		Order Item
Compute price		Order Item
Check validity of payment		Customer
Delivery		



Construire les diagrammes de séquence

Le **contexte** d'un diagramme de séquence est la communication entre les objets. L'*étendue (scope)* d'un diagramme de séquence est déterminée par la phase actuelle du cycle de vie de développement. Pendant l'analyse de problème, l'étendue est la communication parmi les acteurs, le système, et les ressources de système.
Quand vous construisez le diagramme de séquence, vous le trouvez utile à séparer le processus dans deux étapes :

Étape 1. Décrit les interactions entre l'acteur et le système. Cette approche maintient le diagramme aussi simple que possible pendant que l'entrepreneur travaille à comprendre comment le cas d'utilisation devrait travailler.

Étape 2. Augmenter le système pour inclure les ressources utilisées par le système. Quand vous êtes certain de comment le cas d'usage doit travailler, vous devez remap les comportements de système pour montrer aux objets de ressource utilisés par le système pour accomplir le comportement.

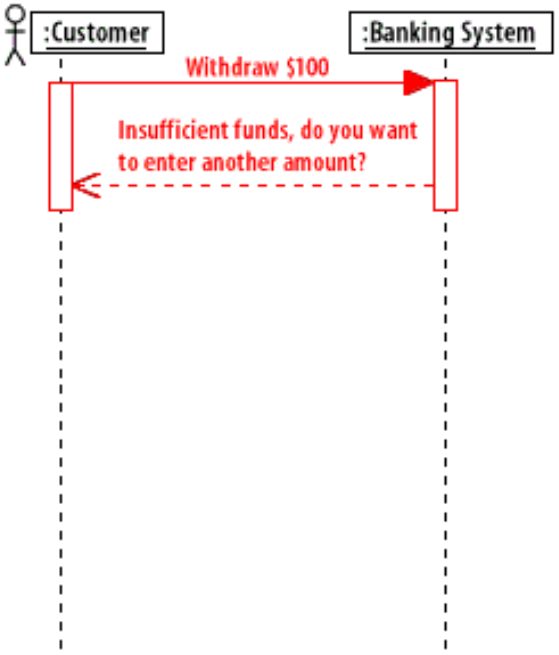
Trouver les objets avec des diagrammes de séquence (I)

Step 1: Describe the interaction between the actor and the system



1) Place the two objects, the actor and the system, on the sequence diagram.

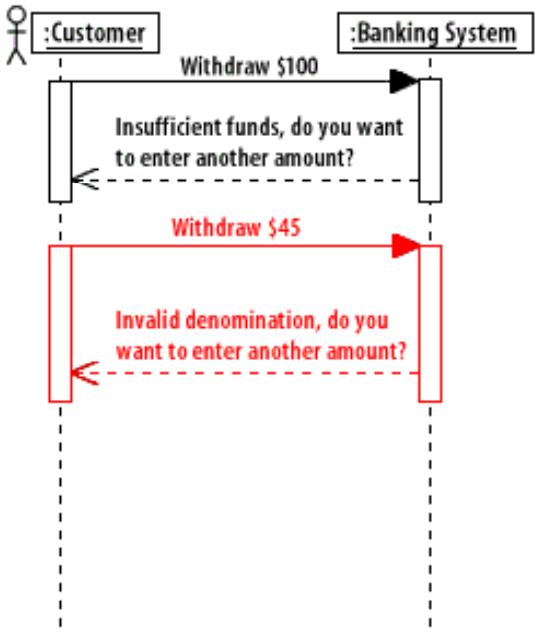
Step 1: Describe the interaction between the actor and the system



3) Draw the events in the conversation between the actor and the system
a) The customer asks for $100
b) The system responds saying that there are insufficient funds and asks for another amount

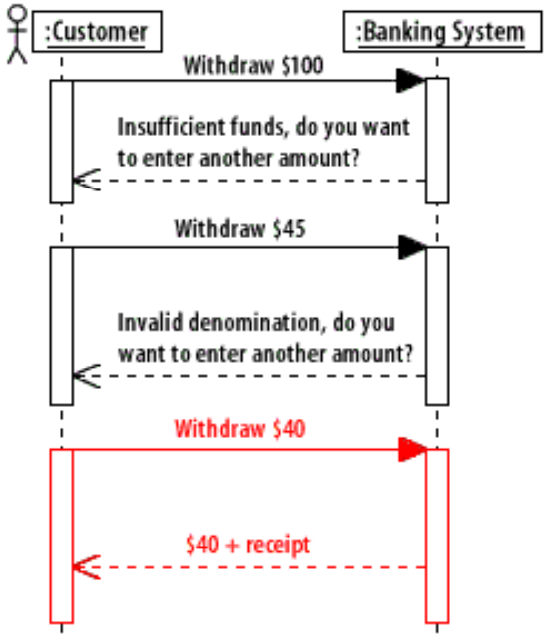
Trouver les objets avec des diagrammes de séquence (II)

Step 1: Describe the interaction between the actor and the system



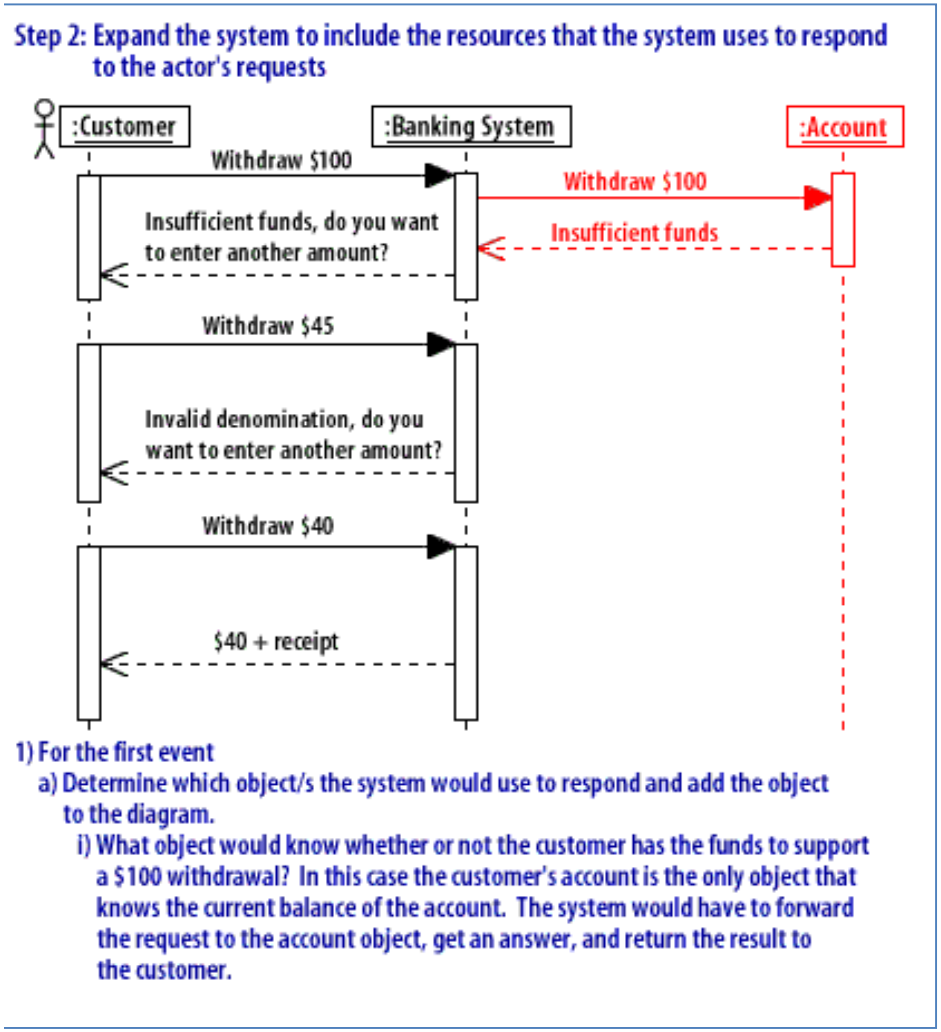
3) Draw the events in the conversation between the actor and the system
c) the customer asks for $45
d) The system responds saying that the denomination requested is invalid and asks for another amount

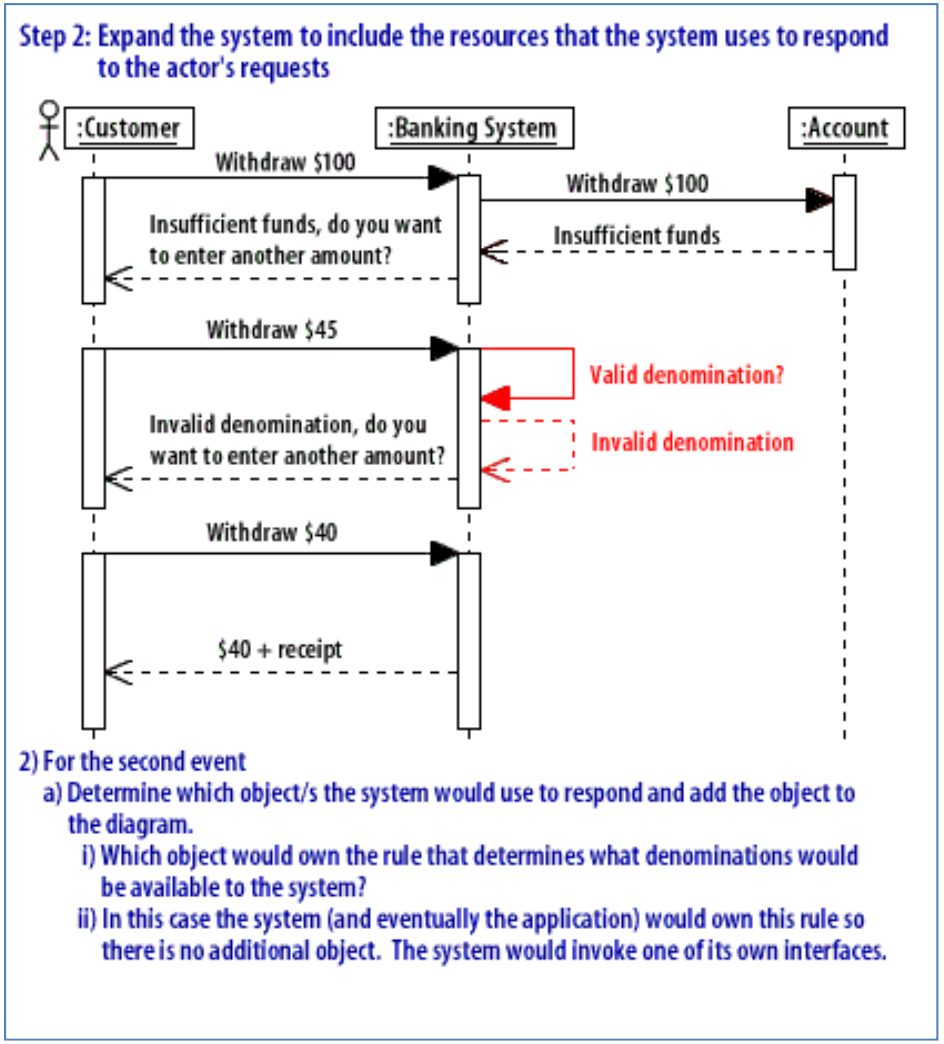
Step 1: Describe the interaction between the actor and the system



3) Draw the events in the conversation between the actor and the system
e) The customer asks for $40
f) The system gives the customer $40 and a receipt

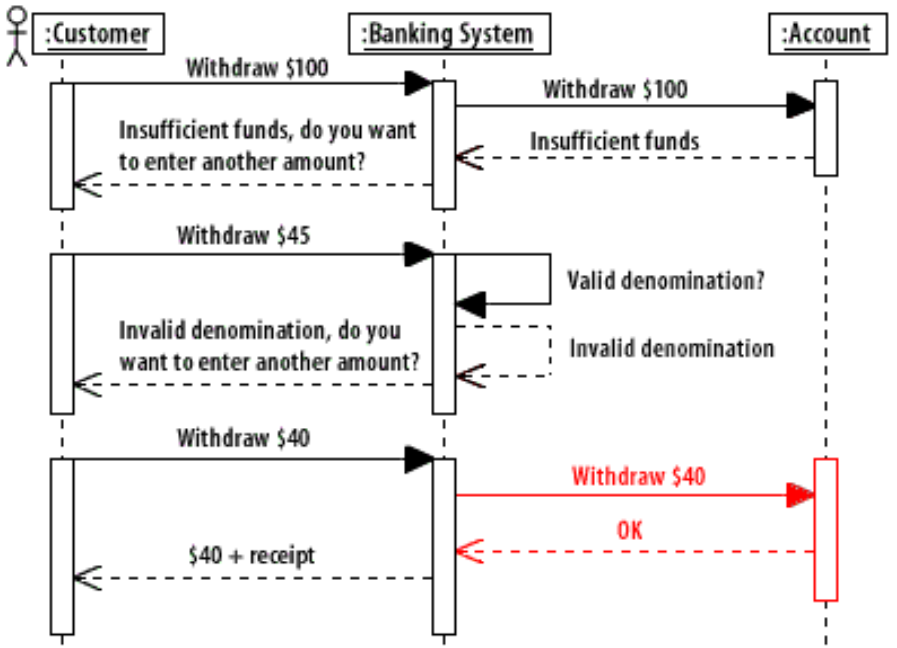
Trouver les objets avec des diagrammes de séquence (III)





Trouver les objets avec des diagrammes de séquence (IV)

Step 2: Expand the system to include the resources that the system uses to respond to the actor's requests



3) For the third event
  a) Determine which object/s the system would use to respond and add the object to the diagram.
    i) What object would know whether or not the customer has the funds to support a $40 withdrawal? The customer's account is the only object that knows the current balance of the account. The system would have to forward the request to the account object, to get an answer.
    ii) The system then decides how to respond to the authorization received from account. In this case it requires the system to provide the money and a receipt.

Scenario vs. Sequence Diagrams

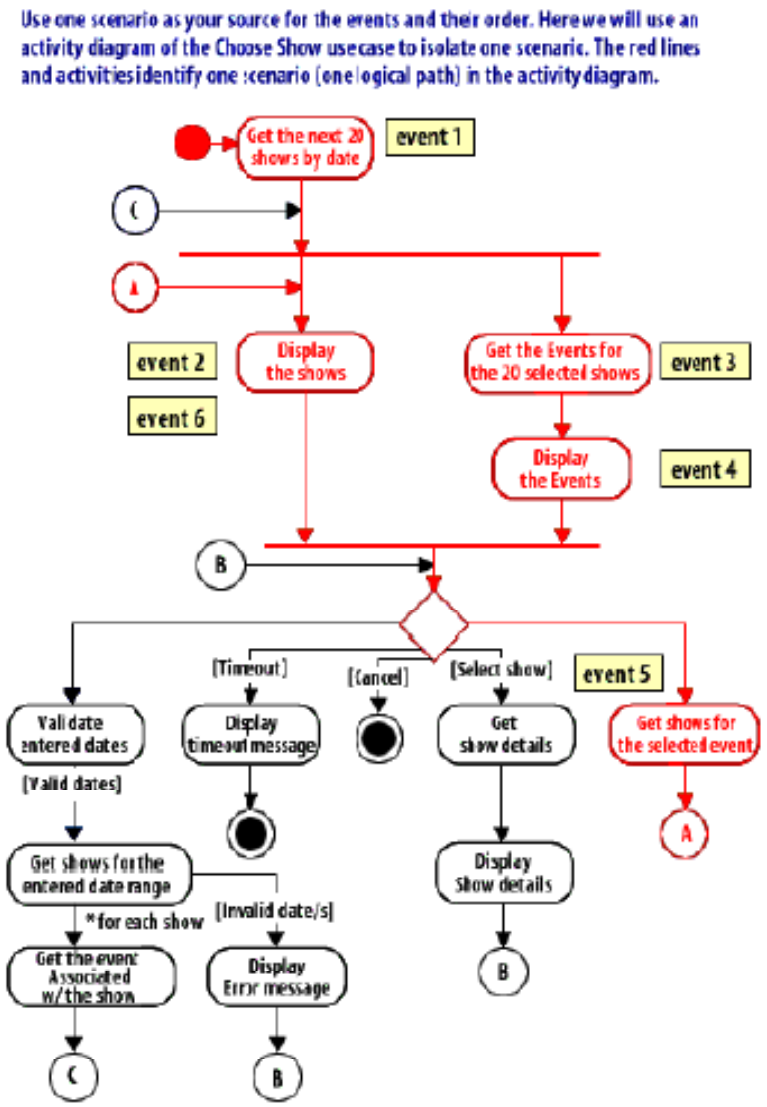
A **scenario** describes an ordered series of events within a use case.

The goal of the sequence diagram is to assign responsibility for events to objects, thus defining the object interfaces.

To build a sequence diagram, **you must match each scenario event to the objects that participate in the event as the sender and the receiver**.

You may map an activity diagram describing a use case to more sequence diagrams that become scenario descriptions.

Modeler les scénarios avec des diagrammes de séquence



To draw a sequence diagram from an activity diagram do the followings:

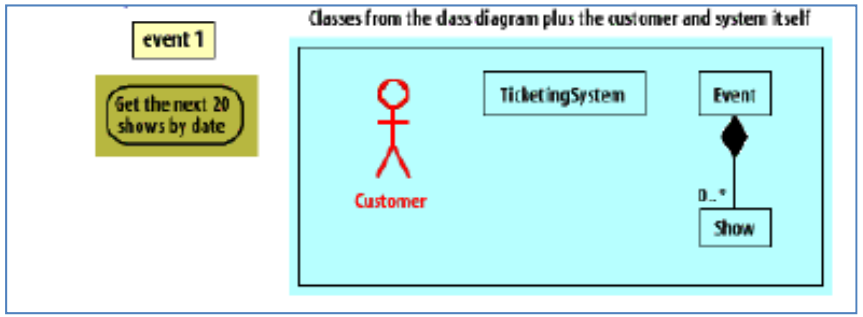
For each scenario in the activity diagram:

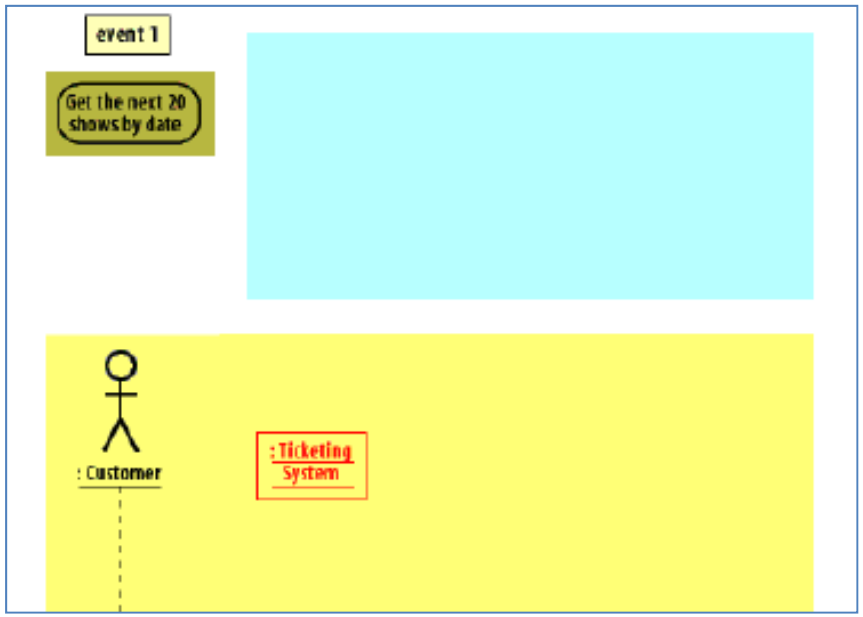
1. Evaluate each event in the scenario and identify the object that initiates the event.
2. Identify the object that is best suited to respond to the event.
3. Draw an event arrow from the initiating object to the responding object.
4. Label the event arrow with the event description.

- As you place events on the sequence diagram, take care to position them on the object lifeline from top to bottom in the order that they occur.

Faire correspondre un diagramme d'activité à un diagramme de séquence(I)

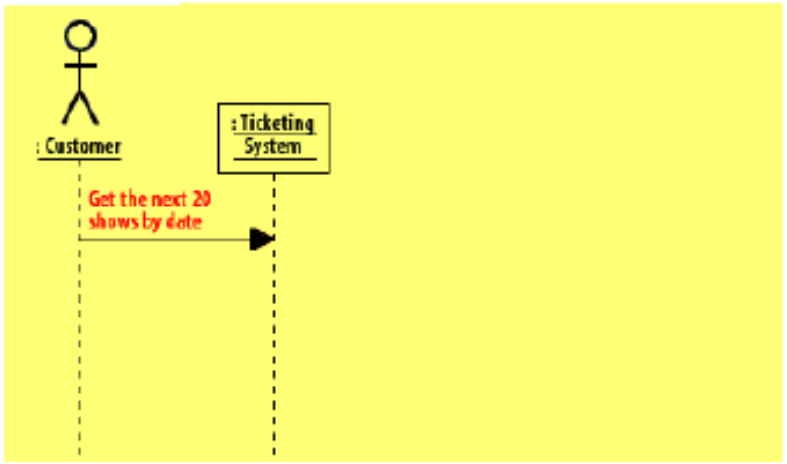
1. Use one scenario in the activity diagram as your source for the events and their order.
2. For the first event, choose the class from the class diagram that describes the object initiating the event. The initiating object may be in a class that represents an actor, the system, or one of the problem domain resources.
3. Draw the object at the top of the diagram.
4. Draw the object lifeline vertically from each object downward.
5. For each event, choose a class from the class diagram that describes the object best suited to receive and respond to the event.
6. Draw the object at the top of the diagram.
7. Draw the object lifeline vertically from the object downward.

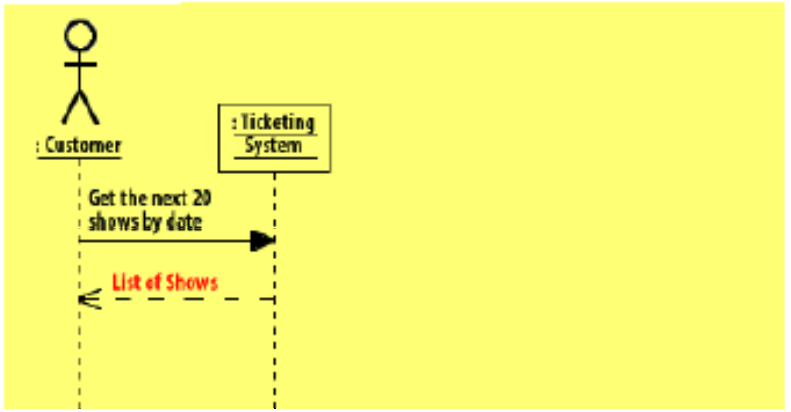




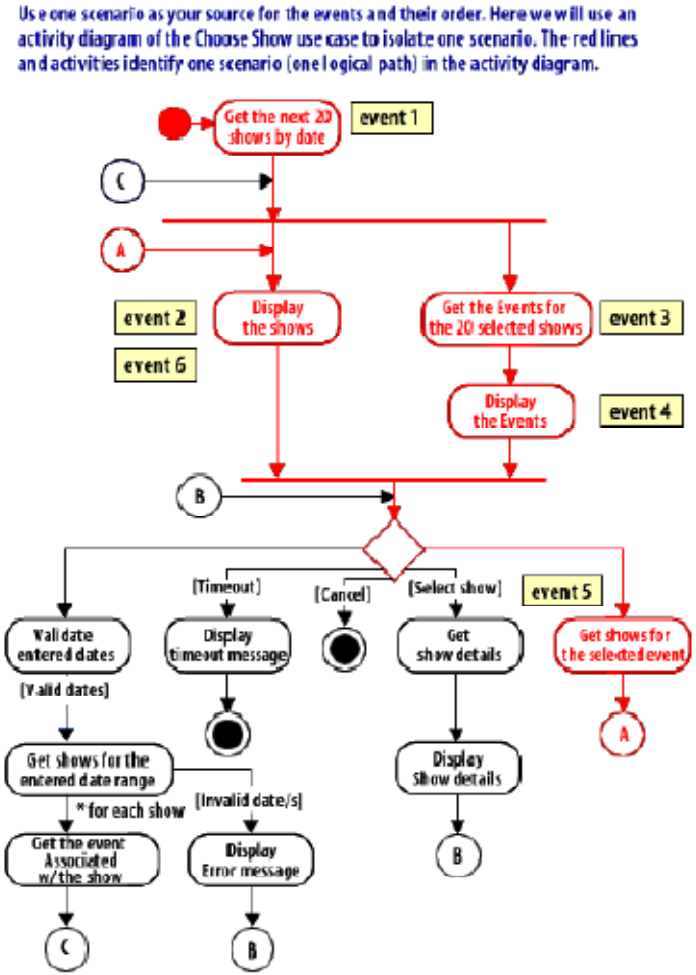
Faire correspondre un diagramme d'activité à un diagramme de séquence (II)

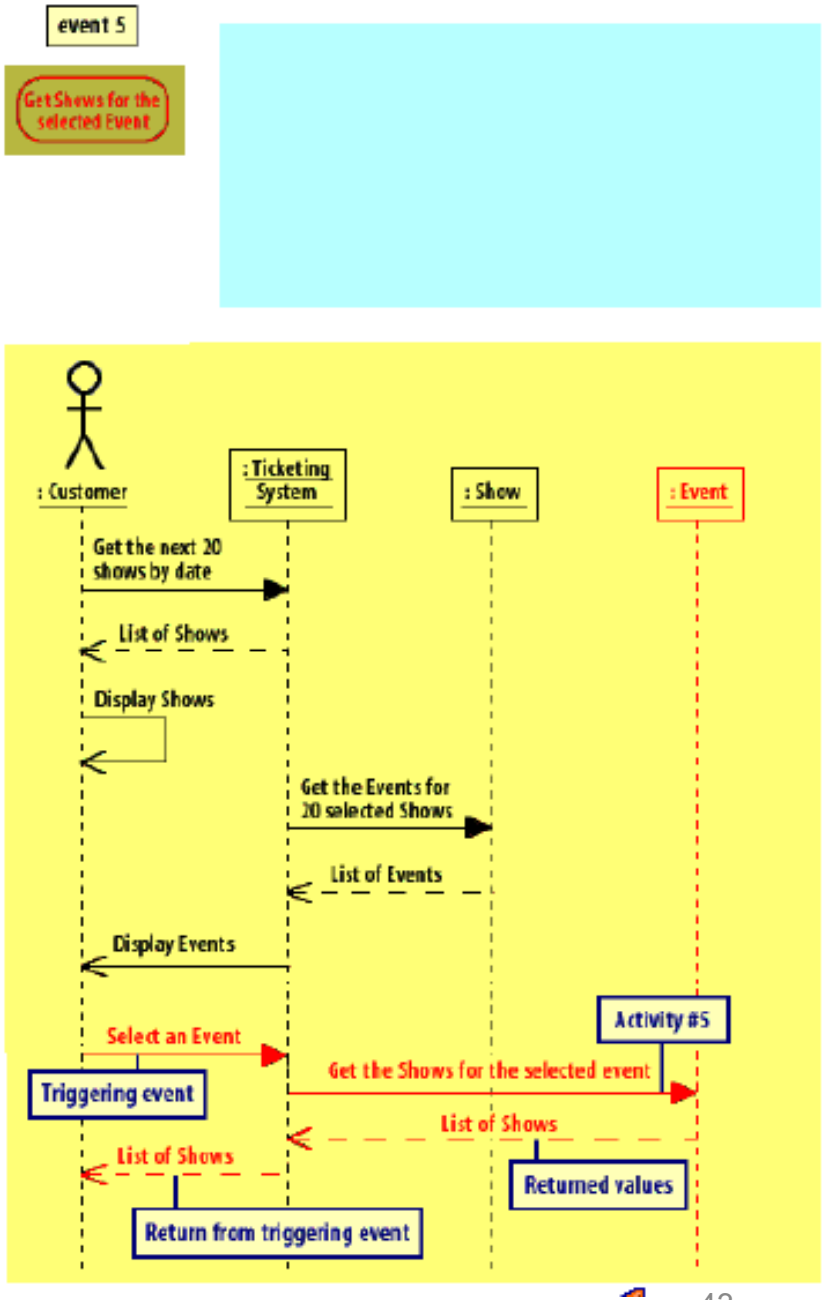
8. Draw a horizontal arrow from the sending object's lifeline to the receiving object's lifeline.
9. Label the event line with the description of the event from the scenario.
10. If the event requires a response, then add an event arrow from the receiving object's lifeline back to the sending object's lifeline.
11. Label the return with the information that is returned. If the event simply returns control, it is common to label the event "return."
12. Repeat this procedure for each event until all events in the scenario have been applied to the sequence diagram.





Faire correspondre un diagramme d'activité à un diagramme de séquence (III)





Mapping events to objects

Assign events to objects

Once you have arranged all the events from one scenario onto a sequence diagram, go back over the diagram and verify that the purpose of the object corresponds to the events that it participates in.

Apply the measure of cohesion to the object. Ask:

1. Does every event initiated by the object support the object's (single) purpose?
2. Does every event that the object responds to fit with object's (single) purpose, or is the object being asked to do something that does not directly relate to its primary purpose?

Apply the measure of coupling to the objects. Interactions implicitly identify dependencies. The questions to ask are:

1. Does the dependency reinforce an appropriate division a labor among the objects, or does it simply complicate communication?
2. Does the dependency reflect the way the real world functions?

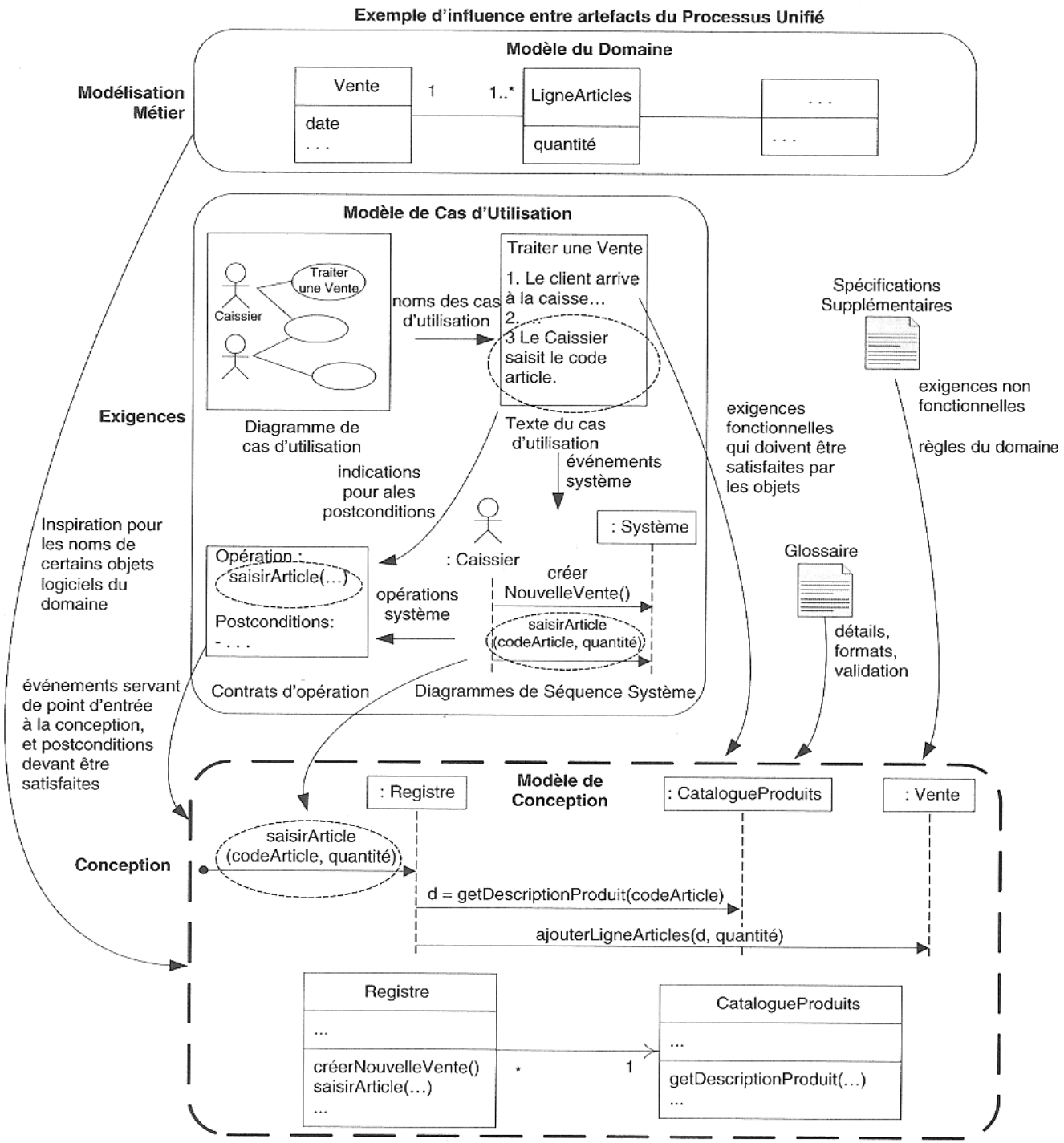
Evaluate object purpose

When you assign a new task to an object, you are in effect adding a new responsibility to the object's "job description." New responsibilities affect objects the same way that adding responsibilities to your job will affect you.

FIN DU SEPTIÈME CHAPITRE

Chap. 8. Conception objet et responsabilités

- Apprendre à appliquer cinq des principes ou patterns GRASP pour la conception orientée objet:
  - Createur ;
  - Expert en Information;
  - Faible Couplage;
  - Controleur ;
  - Forte Cohesion.



Example d'influence entre artefacts du Processus Unifié

Les responsabilités d'un objet

Les responsabilités de faire d'un objet peuvent être :

- faire quelque chose lui-même, par exemple créer un autre objet ou effectuer un calcul ;
- déclencher une action d'un autre objet;
- contrôler et coordonner les activités d'autres objets.

Penser en termes de responsabilités peut s'appliquer a n'reporte quel niveau du logiciel, du plus petit objet au système de systèmes.

Les responsabilités de savoir d'un objet peuvent être :

- connaitre les données privées encapsulées ;
- connaitre les objets connexes ;
- connaitre des éléments qu'il peut dériver ou calculer.

Nous pouvons nommer et expliquer les principes détailles et le raisonnement nécessaires pour saisir une idée fondamentale de la conception objet : l'affectation de responsabilités aux objets.

En UML, la création des diagrammes d'interaction devient l'occasion de réfléchir à ces responsabilités (réalisées sous forme de méthodes).

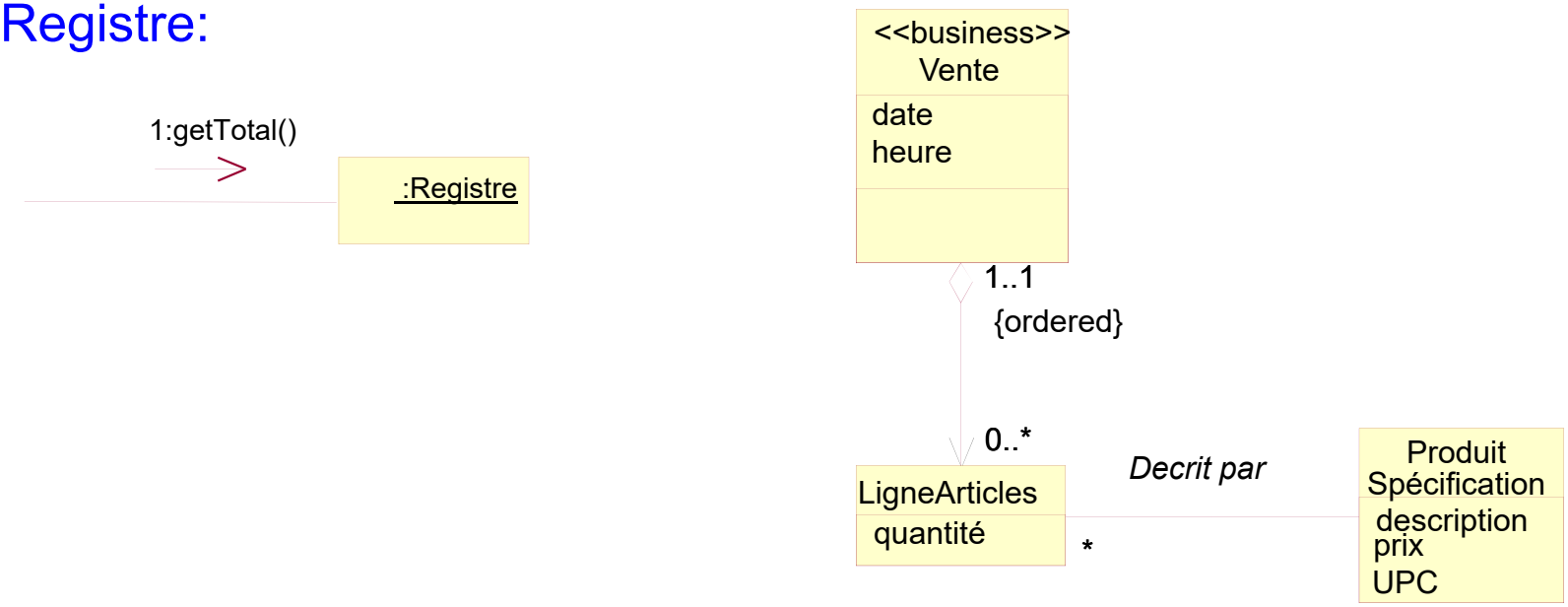
Qu'est-ce qu'un pattern ?

- Les développeurs orientes objet expérimentés (ainsi que d'autres développeurs) construisent des répertoires de principes généraux et de solutions idiomatiques qui les guident lorsqu'ils créent des logiciels.
- Ces principes et ces idiomes, s'ils sont nommés et codifies sous une forme structurée qui décrit le problème et sa solution, peuvent être qualifiés de patterns.
- Exemple
  Nom du pattern : Expert en Information
  Problème: Selon quel principe de base affecter des responsabilités aux objets ?
  Solution : Affecter une responsabilité à la classe qui possède l'information nécessaire pour s'en acquitter.
- En conception orientée objet, un pattern est une description nominée d'un problème et d'une solution susceptible d'être applique dans de nouveaux contextes.
- un pattern est un couple problème/solution nommé et bien connu, qu'on peut appliquer à de nouveaux contextes. Il est accompagne de conseils sur la façon de l'appliquer et d'une présentation de ses inconvénients, implémentations, variantes, etc.

Expert en Information (I)

Quel est le principe général d'affectation des responsabilités aux objets ?
Affecter une responsabilité à l'expert en information: la classe qui possède les informations nécessaires pour s'acquitter de la responsabilité.

Registre:

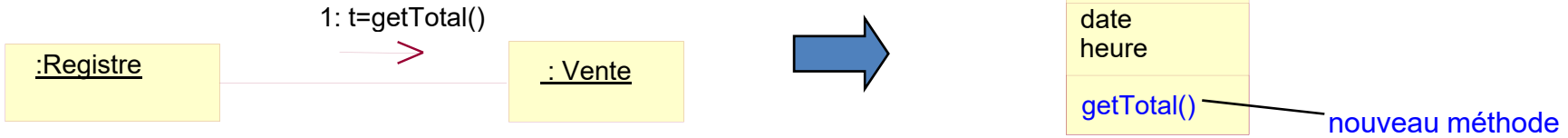


Qui doit avoir la responsabilité de connaitre le total général d'une vente ?

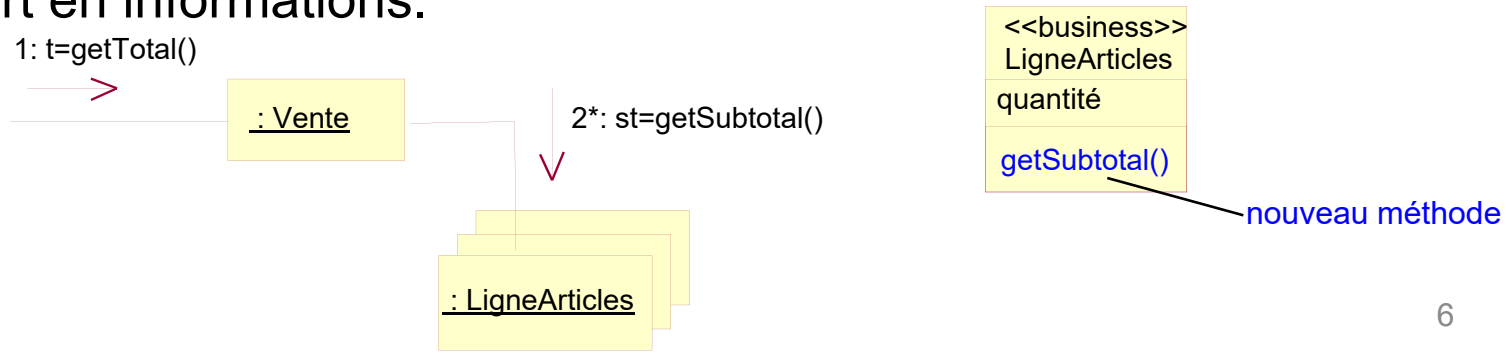
Selon Expert en Information, nous devons rechercher la classe qui possède l'information nécessaire pour déterminer le total.

Expert en Information (II)

1. Problème: Quelles informations sont nécessaires pour déterminer le total ?
   Solution: C'est nécessaire de savoir de tous les instances de LigneArticles d'une vente et la somme de leurs totaux partiels. Un exemple de Vente contient ceux-ci ; donc, par l'indication d'Expert en Information, la Vente est une classe convenable d'objets pour cette responsabilité ; c'est un expert d'informations pour le travail.



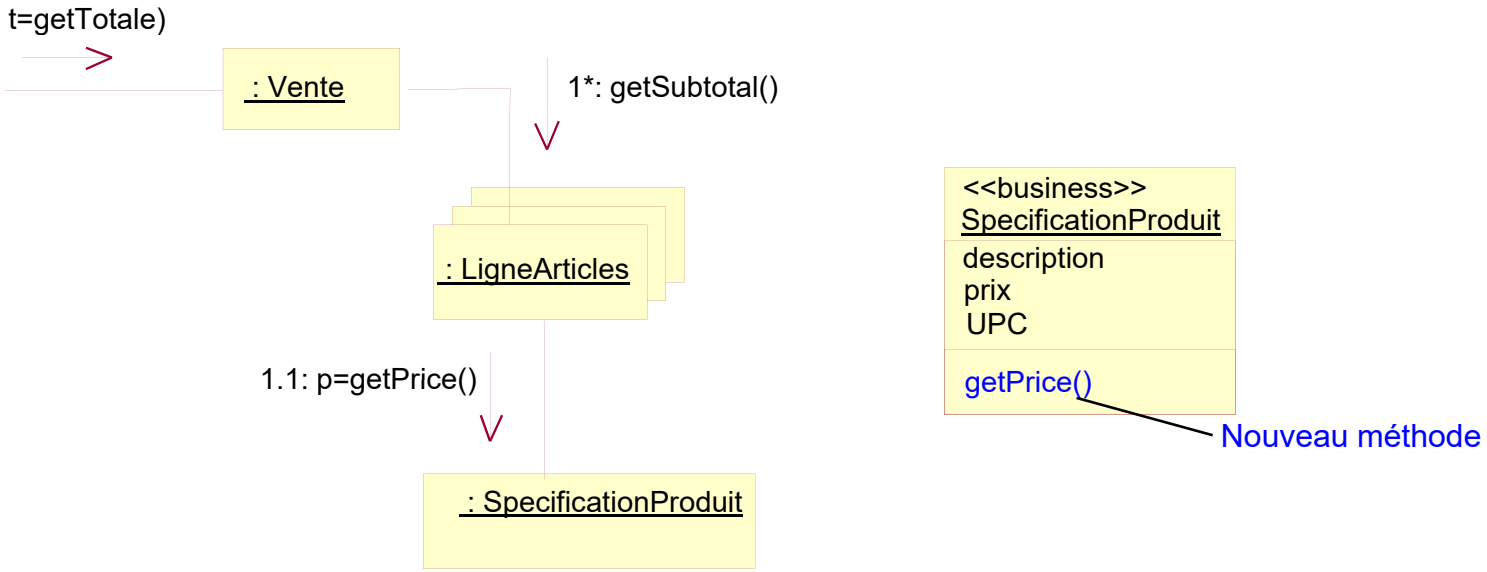
2. Problème: Quelles informations sont nécessaires pour déterminer le total partiel d'article de ligne ?
   Solution : LigneArticles.quantité et SpecificationProduit.prix sont nécessaires.
3. Problème: Qui sait ces informations ?
   Solution : La LigneArticles sait sa quantité et sait sa SpecificationProduit associée; donc, par l'Expert, LigneArticles devrait déterminer le total partiel ; c'est l'expert en informations.



Expert en Information (III)

Pour accomplir la responsabilité de savoir et répondre avec son total partiel, une LigneArticles a besoin de savoir le prix de produit.

4. Problème: Qui sait le prix de produit ?
   Solution: La SpecificationProduit est un expert en information pour répondre à son prix ; donc, un message doit être envoyé à elle demandant son prix.



Classe de conception	Responsabilité
Vente	sait le total de vente
LigneArticles	sait le total partiel des articles d'une ligne
SpecificationProduit	sait le prix de produit

Expert en Information (IV)

Avantages

1. l'encapsulation de l'information est préservée, puisque les objets utilisent leurs = propres informations pour accomplir leurs taches. Le faible couplage l'est généralement aussi, ce qui permet d'obtenir des systèmes plus robustes et plus faciles à maintenir.

2. Le comportement est distribue entre les classes qui possèdent l'information nécessaire, ce qui encourage des définitions de classes « allégées », plus cohésives et plus faciles à comprendre et à maintenir.

Principes ou patterns connexes:

- Faible Couplage
- Forte Cohésion.

Créateur (I)

Problème:

Qui doit avoir la responsabilité de créer une nouvelle instance d'une classe donnée ?

Solution:

Affecter à la classe B la responsabilité de créer une instance de la classe A si une - ou plusieurs - des conditions suivantes est vraie :

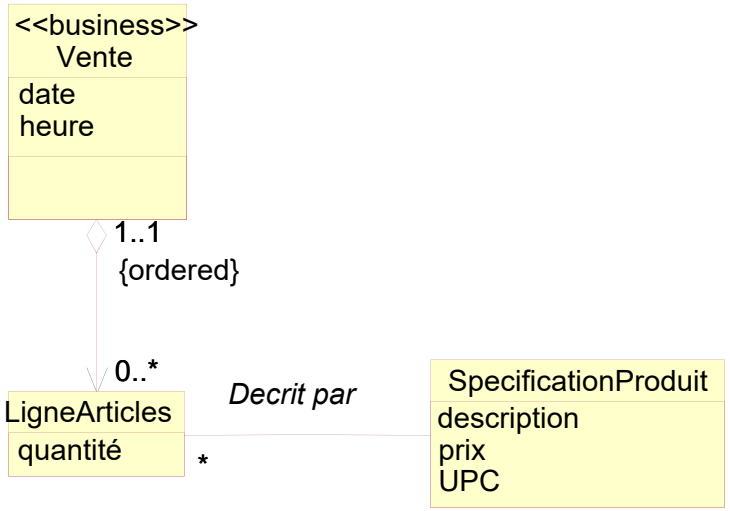
- B *contient ou agrège des objets A.*
- B *enregistre des objets A.*
- *B utilise étroitement des objets A*.
- *B a les données d'initialisation qui seront transmises aux objets A lors de leur création* (B est donc un Expert en ce qui concerne la création de A).

B is a *creator of A objects.*

Si plusieurs conditions s'appliquent, préférez une classe B qui agrège ou contient la classe A.

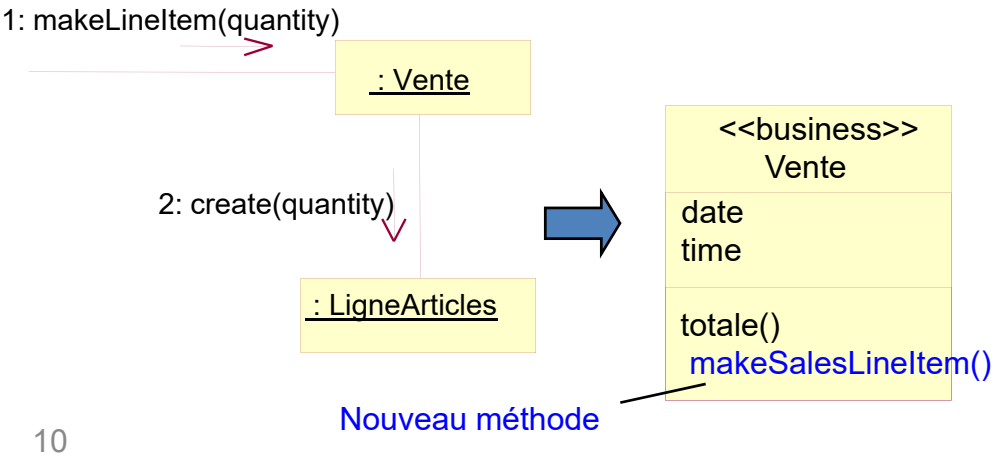
Créateur (II)

Registre:



Problème: Qui devrait être responsable de créer une instance de LigneArticles?

Solution: Par le Créateur, nous devrions chercher une classe qui agrège, contient etc. les exemples de LigneArticles.



Faible Couplage (I)

Couplage en OO: Le couplage est une mesure du degré auquel un élément est lie a un autre, en connait un autre ou a besoin de ses services. Un élément faiblement couple ne dépend pas d'un trop grand nombre d'autres éléments, « trop » étant fonction du contexte. Ces éléments peuvent être des classes, des sous-systèmes ou des systèmes.

Faible Couplage est un principe pour tenir présent pendant toutes les décisions de conception ; c'est un objectif fondamental de considérer continuellement. C'est un principe évaluatif que vous vous appliquez en évaluant toutes les décisions de conception.

Les formes communes d'accouplement de TypeX à TypeY incluent :

- TypaX a un attribut (donnée membre ou variable d'instance) qui reference une instance de TypeY ou TypeY lui-meme.
- Un objet TypeX fait appel au service d'un objet TypeY.
- TypeX a une méthode qui référence d'una façon ou d'una autre une instance de TypeY ou TypeY lui-même. Il s'agit en général d'un paramètre ou d'une variable locale de TypeY ou encore d'une instance de TypeY retournée par un message.
- TypeX est une sous-classe directe ou indirecte de TypeY.
- TypeY est una interface, à TypeX implémenta cette interface.

Seulement les classes avec faible couplage peuvent être facilement réutilisées

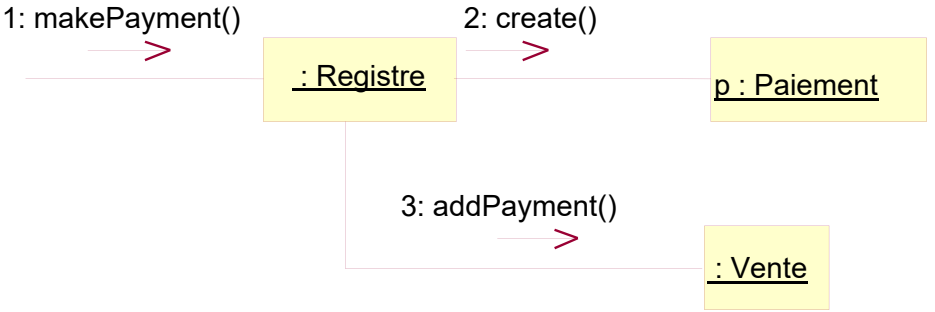
Faible Couplage (II)

Problem: Comment minimiser les dépendances, réduire l'impact des changements et augmenter la réutilisation ?
Solution: Affecter une responsabilité de sorte que le couplage demeure faible. Appliquer ce principe pour évaluer les solutions possibles.

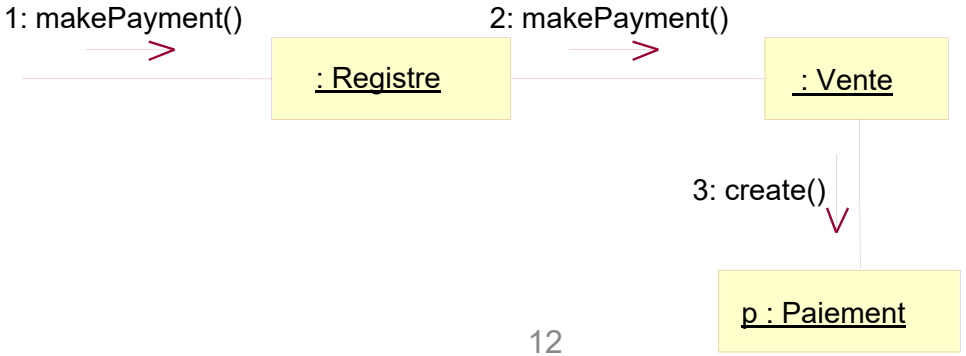


Problème: Supposer que nous avons besoin de créer un exemple de Paiement et l'associe avec la Vente. Quelle classe devrait être responsable de ceci ?
Solution: Deux solutions.
Problème: Quelle conception, fondée sur la tâche de responsabilités, soutient l'accouplement de Faible Couplage?
Solution: Du point de vue d'accouplement, préférer Solution 2 parce qu'il maintient en général bas l'accouplement.

Solution 1:



Solution 2:



Faible Couplage (III)

Avantages:

- non altéré par les modifications des autres composants
- simple à comprendre isolement
- pratique à réutiliser.

Observation: Il ne est pas forte couplage per se qui est le problème, mais il couplage forte des éléments qui sont instables dans quelque dimension, comme leur interface, implémentation, ou la leur simple présence.

Forte Cohésion (I)

Cohésion (en particulier, la cohésion fonctionnelle) est une mesure de comment fortement lié et convergé sont les responsabilités d'un élément. Un élément avec les responsabilités extrêmement liées qui ne font pas une quantité fantastique de travail a une haute cohésion. Ces éléments incluent des classes, les sous-systèmes etc.

Problème: How to keep objects focused, understandable, and manageable, and as a side effect, support Low Coupling?

Solution: Assign a responsibility so that cohesion remains high. Use this to evaluate alternatives.

A class with low cohesion does many unrelated things or does too much work. Such classes are undesirable; they suffer from the following problems:

- hard to comprehend
- hard to reuse
- hard to maintain
- delicate; constantly affected by change

Low cohesion classes often represent a very "large grain" of abstraction or have taken on responsibilities that should have been delegated to other objects.

Forte Cohésion (II)

Problème: Comment s'assurer que les objets restent compréhensibles et faciles a gérer, et, bénéfice second, qu'ils contribuent au Faible Couplage ?

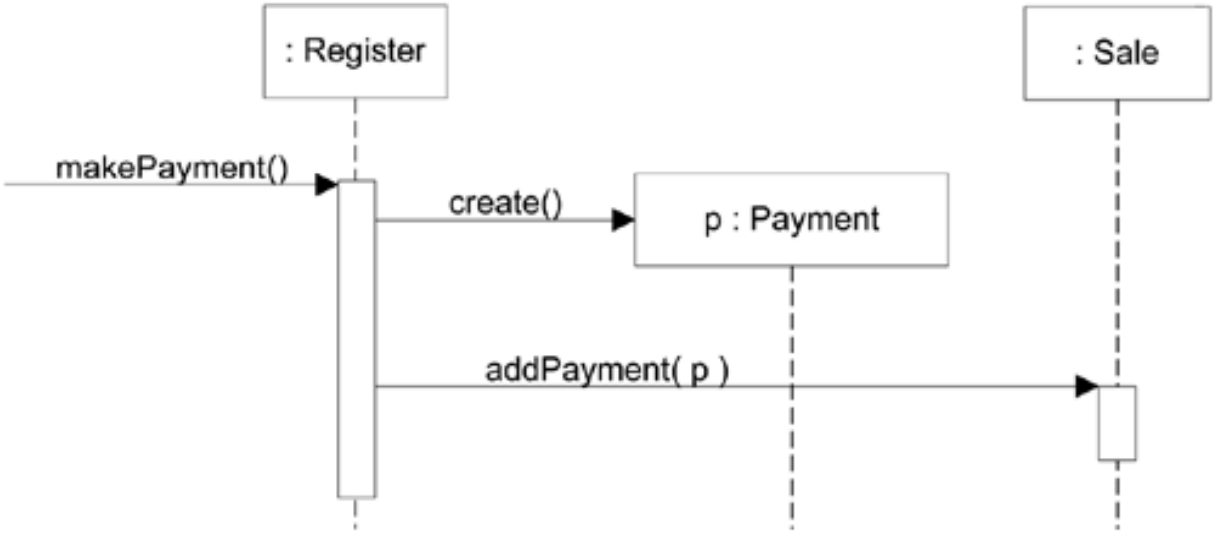
Solution: Affecter les responsabilites pour que la cohesion demeure elevee.

POS:

Assume we have a need to create a (cash) Payment instance and associate it with the Sale. What class should be responsible for this?

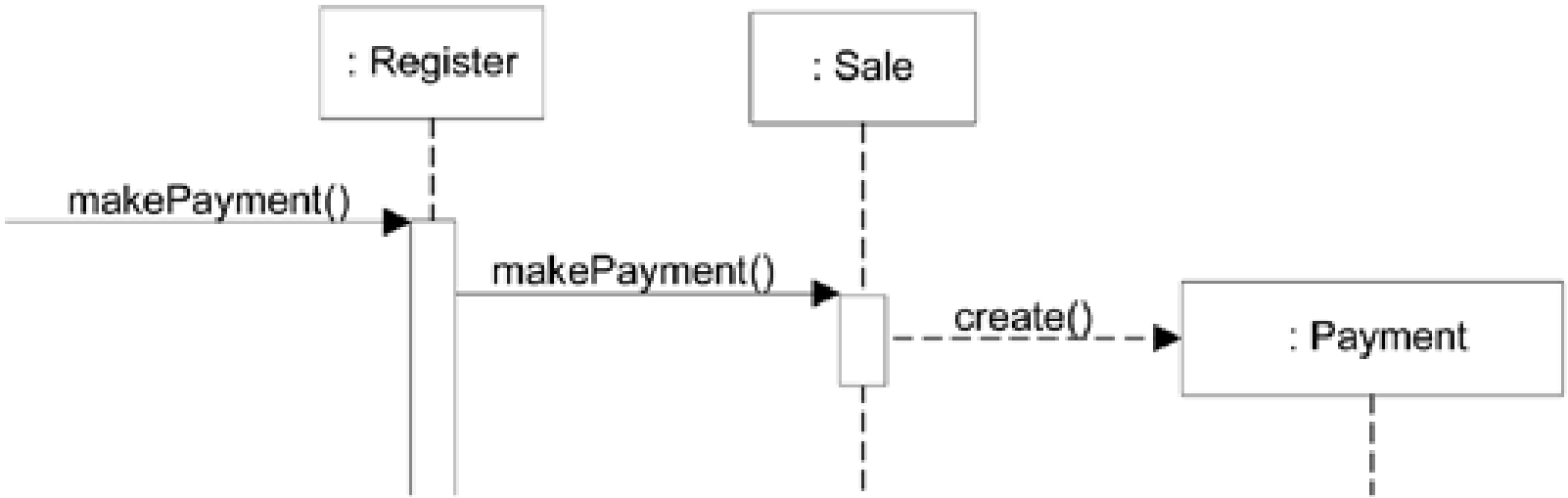
Since Register records a Payment in the real-world domain, the Creator pattern suggests Register as a candidate for creating the Payment. The Register is taking on part of the responsibility for fulfilling the makePayment system operation.

To make the Register class responsible for doing some or most of the work related to more and more system operations, it will become increasingly burdened with tasks and become incohesive.



Forte Cohésion (III)

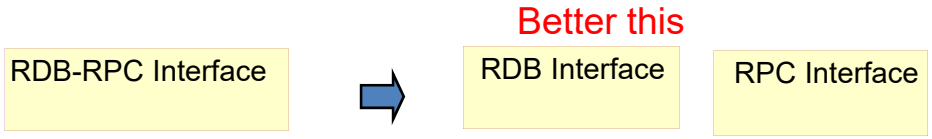
- A second design delegates the payment creation responsibility to the Sale supports higher cohesion in the Register.



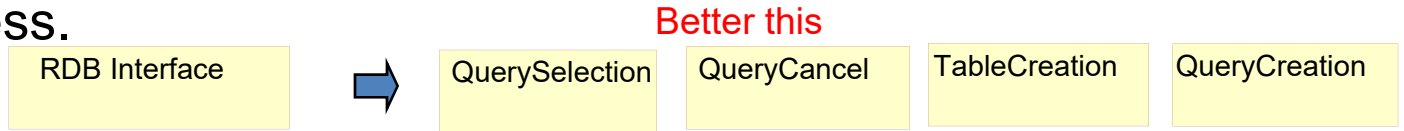
Grades de Cohésion Fonctionnelle (I)

Here are some scenarios that illustrate varying degrees of functional cohesion (Booch):

Very low cohesion. A class is solely responsible for many things in very different functional areas. A class called RDB-RPC-Interface which is completely responsible for interacting with relational databases and for handling remote procedure calls. These are two vastly different functional areas, and each requires lots of supporting code. The responsibilities should be split into a family of classes related to RDB access and a family related to RPC support.

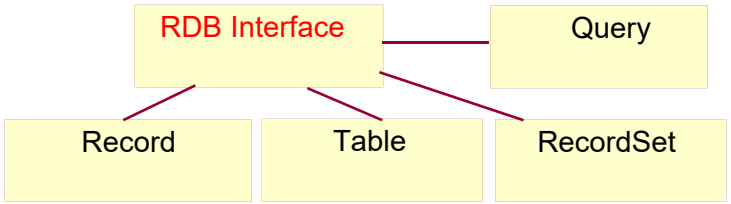


Low cohesion. A class has sole responsibility for a complex task in one functional area. A class called RDBInterface which is completely responsible for interacting with relational databases. The methods of the class are all related, but there are lots of them, and a tremendous amount of supporting code; there may be hundreds or thousands of methods. The class should split into a family of lightweight classes sharing the work to provide RDB access.

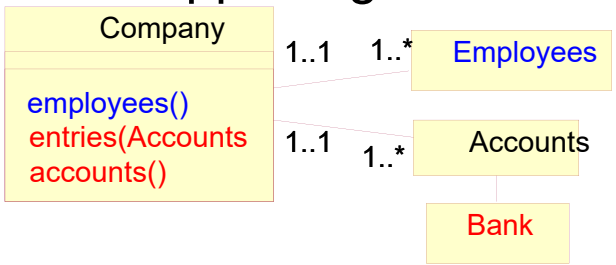


Grades de Cohésion Fonctionnelle (II)

High cohesion. A class has moderate responsibilities in one functional area and collaborates with other classes to fulfill tasks. A class called RDBInterface that is only partially responsible for interacting with relational databases. It interacts with a dozen other classes related to RDB access in order to retrieve and save objects.



Moderate cohesion. A class has lightweight and sole responsibilities in a few different areas that are logically related to the class concept but not to each other. A class called Company that is completely responsible for (a) knowing its employees and (b) knowing its financial information. These two areas are not strongly related to each other, although both are logically related to the concept of a company. In addition, the total number of public methods is small, as is the amount of supporting code.



Contrôleur (I)

Problem: What first object beyond the UI layer receives and coordinates ("controls") a system operation?

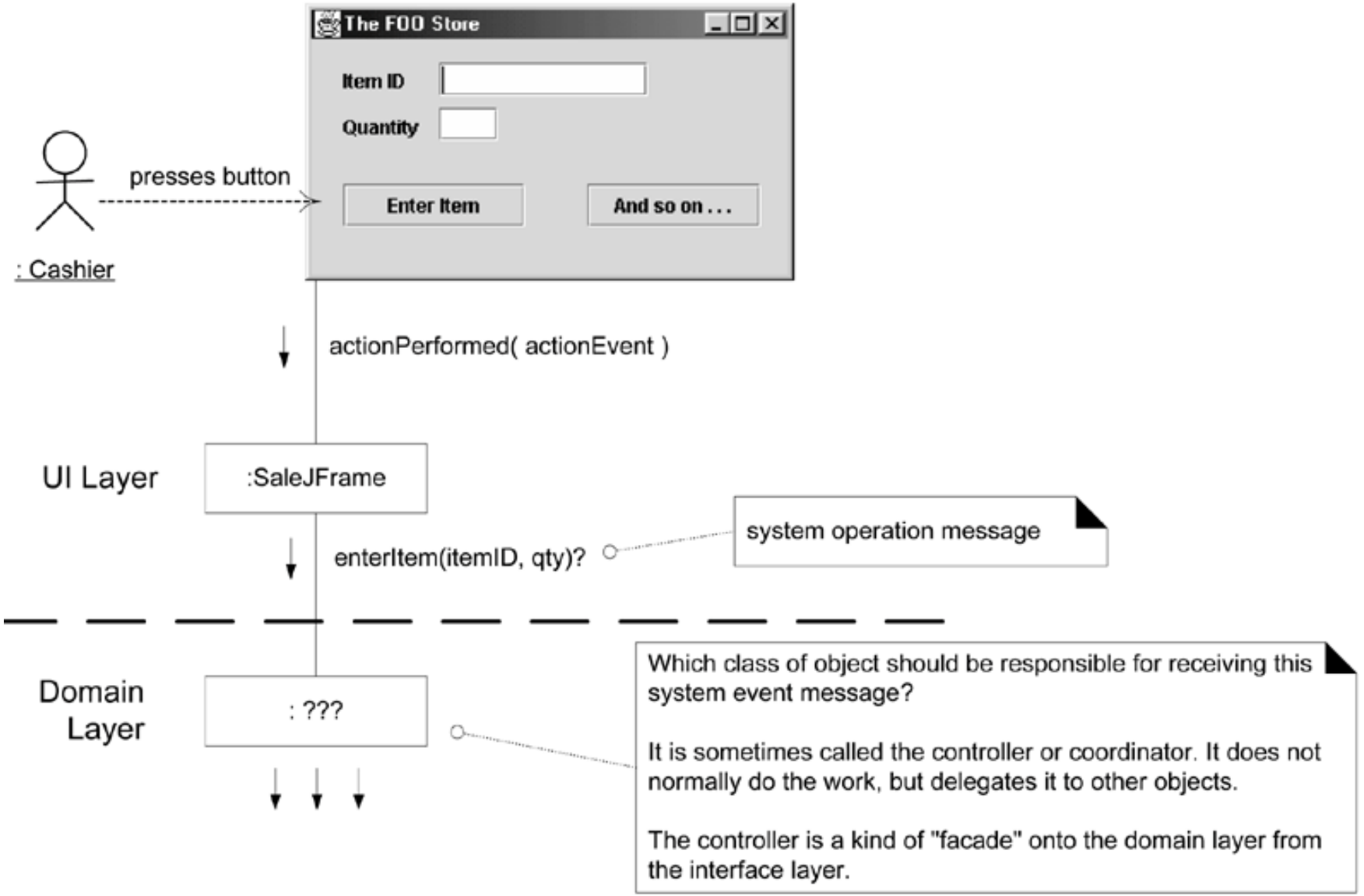
Solution: Assign the responsibility to a class representing one of the following choices:

- Represents the overall "system," a "root object," a device that the software is running within, or a major subsystem (these are all variations of a facade controller).
- Represents a use case scenario within which the system event occurs, often named <UseCaseName>Handler, <UseCaseName>Coordinator, or <UseCaseName>Session (use case or session controller).
  - Use the same controller class for all system events in the same use case scenario.
  - Informally, a session is an instance of a conversation with an actor. Sessions can be of any length but are often organized in terms of use cases (use case sessions).

Corollary: Note that "window," "view," and "document“ should not fulfill the tasks associated with system events; they typically receive these events and delegate them to a controller.

Contrôleur vs. Couche de Présentation (I)

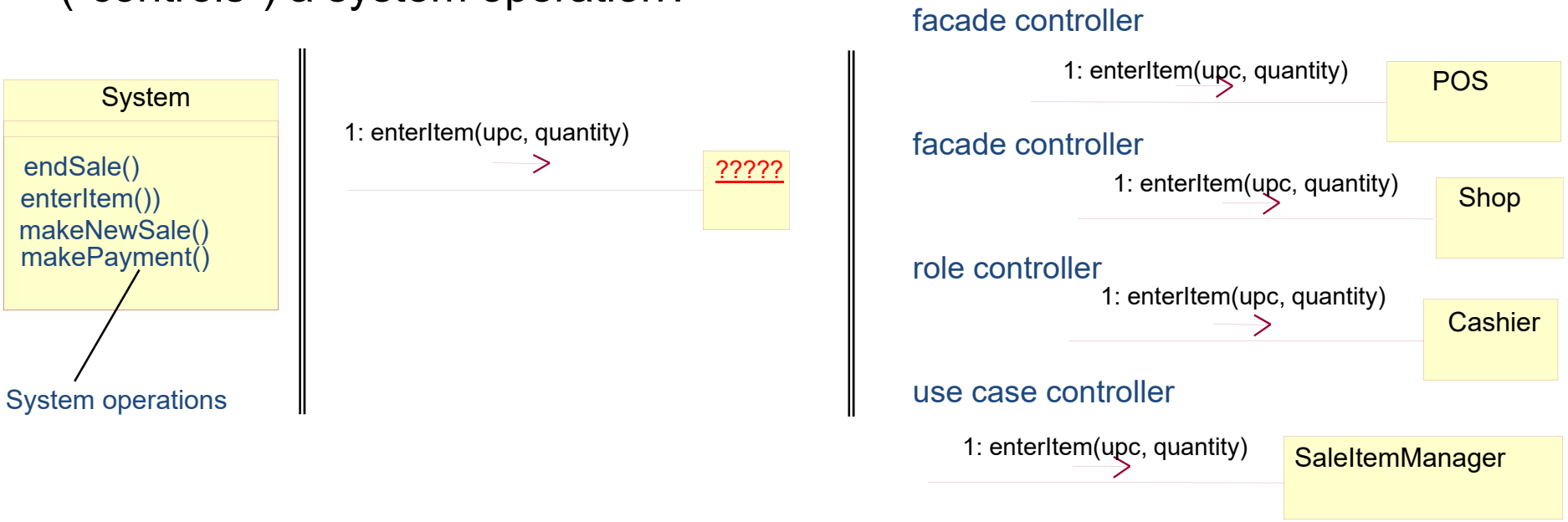
What object should be the Controller for enterItem?



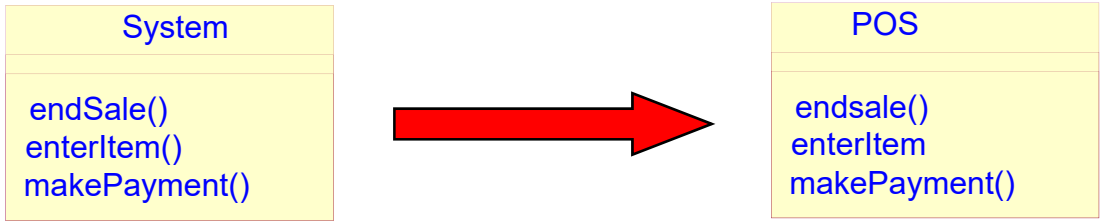
Contrôleur (II)

POS:

Problem: What first object beyond the UI layer receives and coordinates ("controls") a system operation?

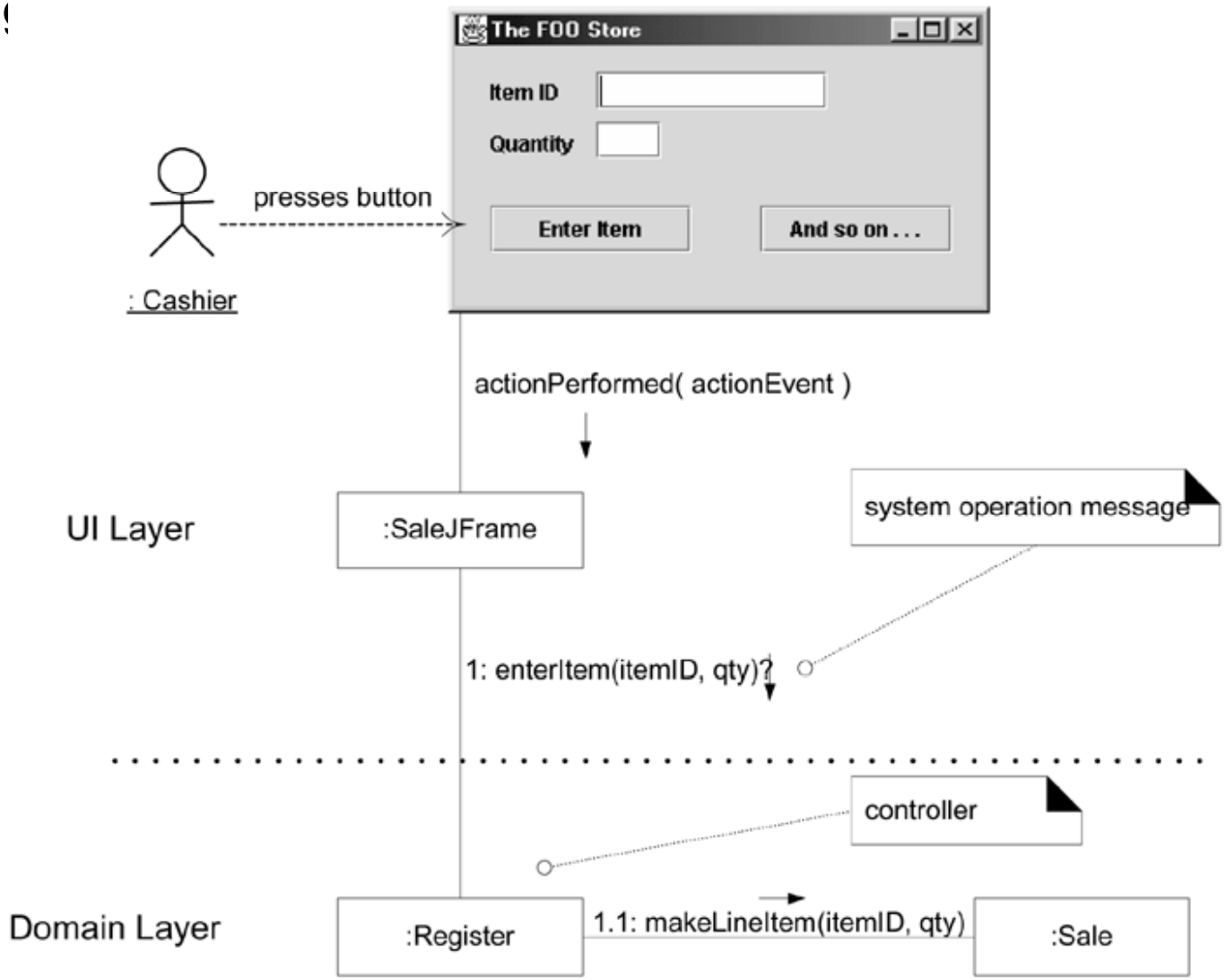


Solution:



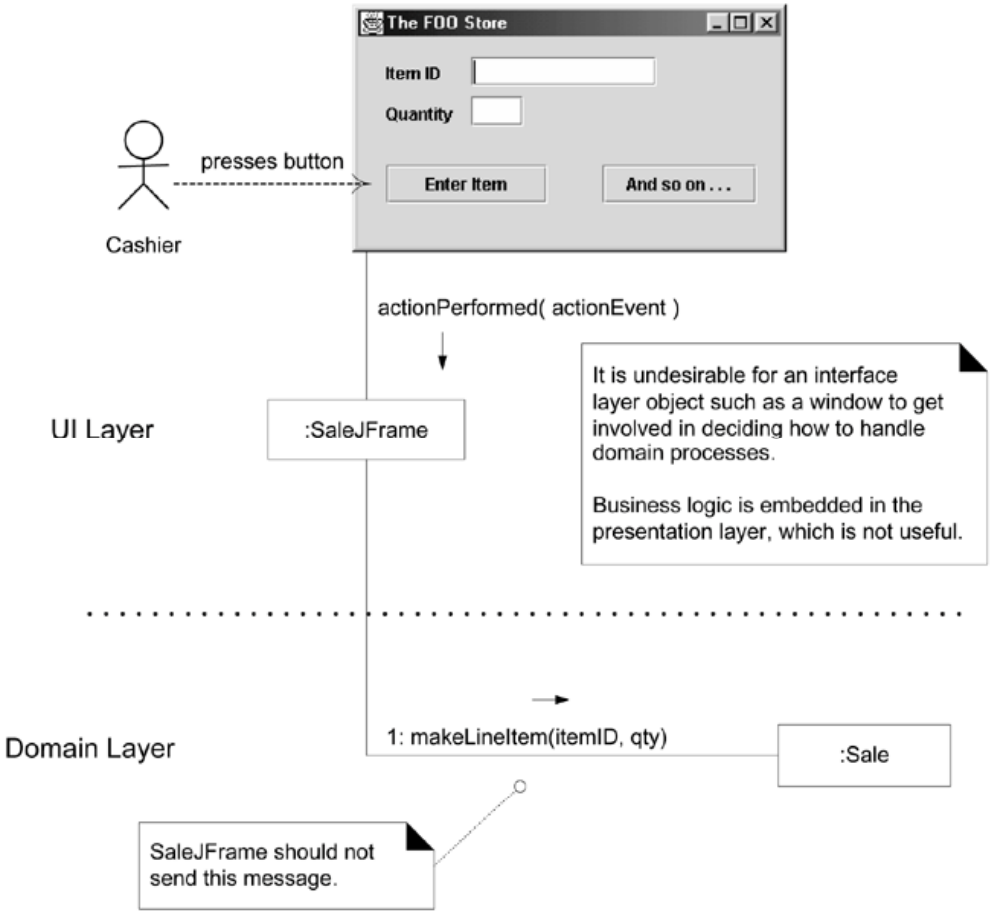
Contrôleur vs. Couche de Présentation (II)

- An important corollary of the Controller pattern is that UI objects (for example, window objects) and the UI layer should not have responsibility for handlin

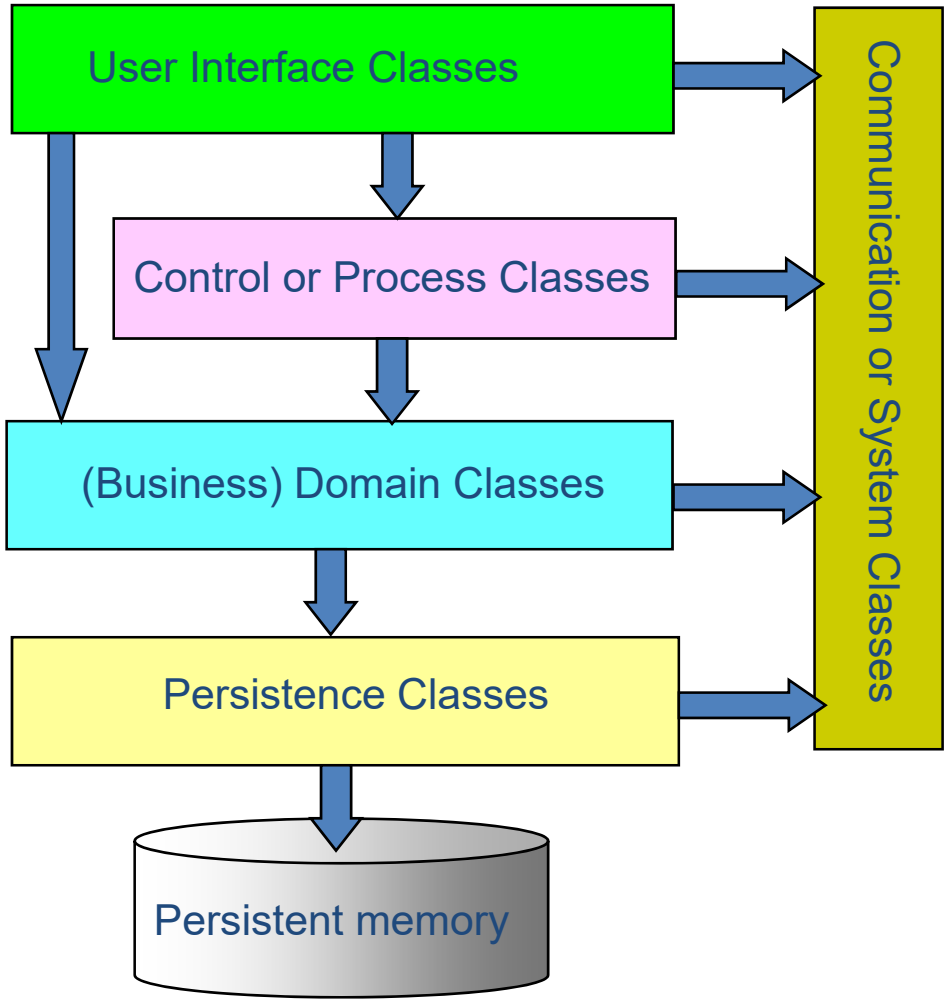


Contrôleur vs. Couche de Présentation (III)

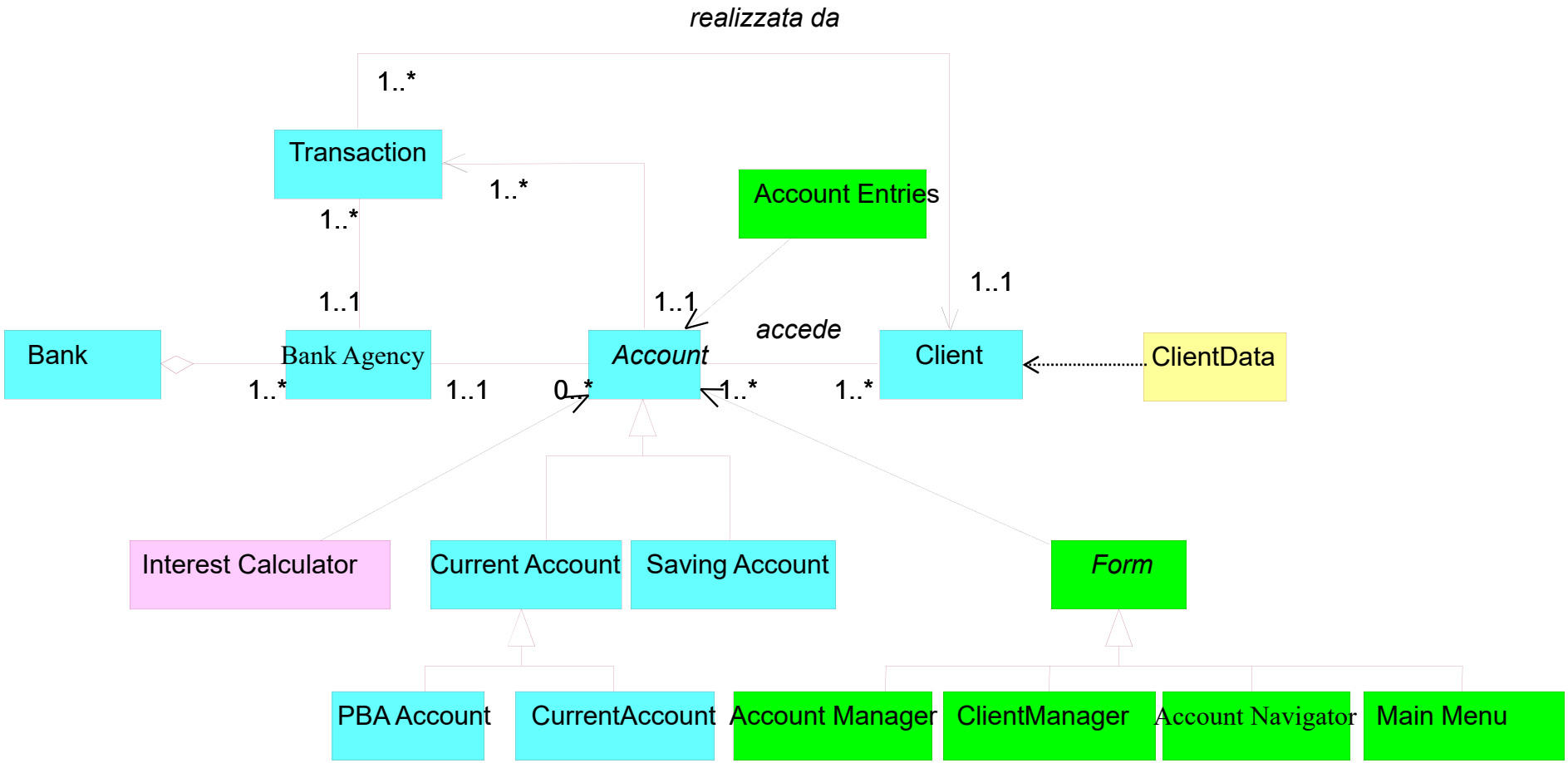
Assigning the responsibility for system operations to objects in the application or domain layer by using the Controller pattern rather than the UI layer can increase reuse potential. If a UI layer object (like the SaleJFrame) handles a system operation that represents part of a business process, then business process logic would be contained in an interface (for example, window-like) object; the opportunity for reuse of the business logic then diminishes because of its coupling to a particular interface and application. Consequently, the design in the next figure is undesirable.



Classes de conception



La diagramme des classes de conception (ou d'application)



FIN DU HUITIÈME CHAPITRE