Cas d'utilisation

Modéliser les vues statiques d'un système

Les cas d'utilisation sont un moyen d'exprimer le besoin des Utilisateurs d'un système informatique vis-à-vis de ce système. Ils sont une vision orientée Utilisateur de ce besoin au contraire d'une vision informatique.

Introduction

La conceptualisation.

Le but de la conceptualisation est de comprendre et structurer les besoins du client. Il ne faut pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins !

Une fois identifiés et structurés, ces besoins :

- définissent le contour du système à modéliser (ils précisent le but à atteindre),
- permettent d'identifier les fonctionnalités principales (critiques) du système.

Le modèle conceptuel doit permettre une meilleure compréhension du système et doit servir d'interface entre tous les acteurs du projet.

Les besoins des clients sont des éléments de traçabilité dans un processus intégrant UML. Le modèle conceptuel joue un rôle central, il est capital de bien le définir !

Cas d'utilisation (use cases)

Il s'agit de la solution UML pour représenter le modèle conceptuel.

Les use cases permettent de structurer les besoins des utilisateurs et les objectifs correspondants d'un système.

Ils centrent l'expression des exigences du système sur ses utilisateurs : ils partent du principe que les objectifs du système sont tous motivés.

Ils se limitent aux préoccupations "réelles" des utilisateurs ; ils ne présentent pas de solutions d'implémentation et ne forment pas un inventaire fonctionnel du système.

Ils identifient les utilisateurs du système (acteurs) et leur interaction avec le système.

Ils permettent de classer les acteurs et structurer les objectifs du système.

Ils servent de base à la traçabilité des exigences d'un système dans un processus de développement intégrant UML.

Les use cases, permettent de modéliser les besoins des clients d'un système et doivent aussi posséder ces caractéristiques. Ils ne doivent pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins !

Une fois identifiés et structurés, ces besoins :

- définissent le contour du système à modéliser (ils précisent le but à atteindre),
- permettent d'identifier les fonctionnalités principales (critiques) du système.

Les use cases ne doivent donc en aucun cas décrire des solutions d'implémentation. Leur but est justement d'éviter de tomber dans la dérive d'une approche fonctionnelle, où l'on liste une litanie de fonctions que le système doit réaliser.

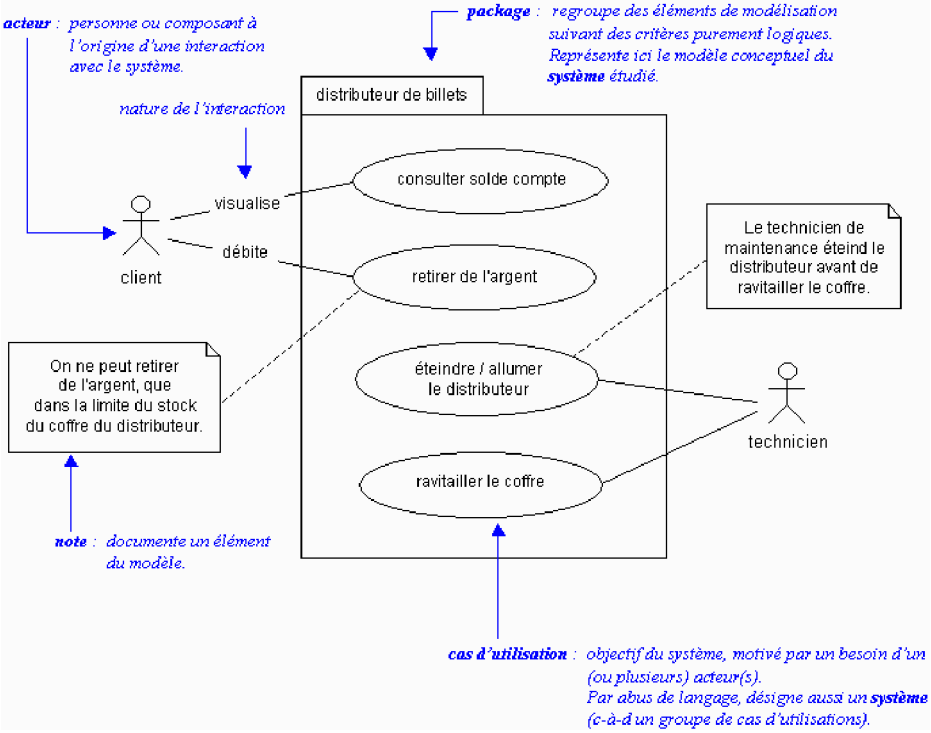
Un modèle conceptuel qui identifie les besoins avec un plus grand niveau d'abstraction reste indispensable. Avec des systèmes complexes, filtrer l'information, la simplifier et mieux l'organiser, c'est rendre l'information exploitable.

Utilisez les use cases tels qu'ils ont été pensé par leurs créateurs ! UML est issu du terrain. Si vous utilisez les use cases sans avoir en tête la démarche sous-jacente, vous n'en tirerez aucun bénéfice.

Eléments de base des cas d'utilisation

- **Acteur** : entité externe qui agit sur le système (opérateur, autre système...).
- **Use case** : ensemble d'actions réalisées par le système, en réponse à une action d'un acteur.

Exemples : **Cas d'utilisation standard :**



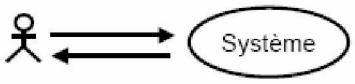
Les Uses Cases permettent :

- De connaître le comportement du système sans spécifier comment ce comportement sera réalisé.
- De définir les limites précises du système
- Au développeur de bien comprendre l'attente des utilisateurs et les experts du domaine.

De plus les Use Cases sont :

- Des instruments de validation et de test du système en cours et en fin de construction.

Les cas d'utilisation représentent les fonctionnalités que le système doit savoir faire. Chaque cas d'utilisation décrit un ensemble d'interactions successives d'une entité en dehors du système (utilisateur) avec le système lui même pour réaliser une fonctionnalité.



Un diagramme de cas d'utilisation définit :

- le système
- les acteurs
- les cas d'utilisations
- les liens entre acteurs et cas d'utilisations

Un modèle de cas d'utilisation se définit par :

- des diagrammes de cas d'utilisation
- une description textuelle des scénarios d'utilisation
- une description de ces scénarios par :
  - les diagrammes de séquences

- les diagrammes de collaboration

Acteurs

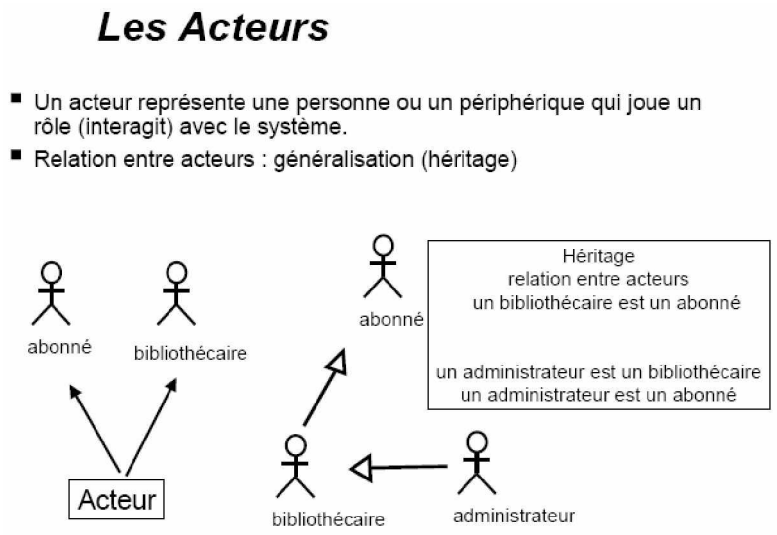
UML n'emploi pas le terme d'Utilisateur mais d'acteur.

Les acteurs d'un système sont les entités externes à ce système qui interagissent avec lui. Quand on dit « qui interagissent », on veut dire qui envoient des évènements comme, entrer une date de naissance, cliquer sur un bouton OK. On veut aussi dire qui reçoivent des informations de la part du système comme, recevoir une facture, mettre à jour un référentiel de données ou encore mettre à jour une application back-office.

Les acteurs sont donc à l'extérieur du système et dialoguent avec lui. Ces acteurs permettent de cerner l'interface que le système va devoir offrir à son environnement. Oublier des acteurs ou en identifier de faux conduit donc nécessairement à se tromper sur l'interface et donc la définition du système à produire. On fera attention à ne pas confondre acteurs et utilisateurs d'un système. Non pas que cela soit faux car tout dépend du sens donné au mot utilisateur mais trop souvent, le mot utilisateur est vu comme un raccourci pour désigner ceux qui vont cliquer dans les fenêtres de l'application. Les acteurs sont plus que les « simples » utilisateurs humains d'un système. D'une part parce que les acteurs inclus les utilisateurs humains mais aussi les autres systèmes informatiques ou hardware qui vont communiquer avec le système.

Pour trouver les acteurs d'un système, on va identifier quels sont les différents rôles que vont devoir jouer ses utilisateurs. Car les acteurs sont en fait les rôles joués par ces différents Utilisateurs (ex : Responsable clientèle, Responsable d'agence, Administrateur, Approbateur,…). On regardera ensuite quels sont les autres systèmes avec lesquels le système va devoir communiquer soit en mode réception soit en mode émission d'évènements (ex : Hardware d'un distributeur de billet, Système d'information partenaire, ERP,…).

Un biais classique dans l'identification des acteurs est dû au fait que les acteurs peuvent aussi être d'autres systèmes. Aussi, il est fréquent de vouloir identifier comme acteur les référentiels de données existant dans le système d'information. Effectivement, le système à produire aura sûrement à récupérer ou à mettre à jour des données issues de ces référentiels mais le risque d'identifier ces systèmes comme acteur est qu'ensuite, lors de la rédaction des cas d'utilisation, on risque de rentrer trop tôt dans la solution et oublier l'expression première du besoin. Je ne dis pas qu'identifier ce type d'acteur est une erreur fondamentale mais l'expérience montre que l'intérêt est minime et que les biais induits sont néfastes : une description des cas d'utilisation plus proche de la solution que du besoin.



Les cas d'utilisation

« Bon, ça y est, on a les acteurs de notre système, qu'attendent-ils de ce système ? »

Les cas d'utilisation vont ici nous aider à décrire ces attendus.

On entend souvent que les cas d'utilisation permettent de décrire les fonctionnalités attendues du système de point de vue des acteurs. Ce n'est pas faux mais attention car « fonctionnalités » se transforme souvent en « fonctions ». On en arrive donc à utiliser les cas d'utilisation pour faire un découpage fonctionnel au sens procédures des langages procéduraux. Et là, on se trompe complètement d'objectif.

Ces fonctionnalités que l'on documente avec les cas d'utilisation doivent avoir un sens pour le métier des acteurs. Pour identifier les cas d'utilisation, il faut donc se poser les questions :

« Mais que va faire cet acteur avec le système en arrivant le matin au boulot ? »

« Pourquoi démarre t-il le système, avec quel objectif métier ? »

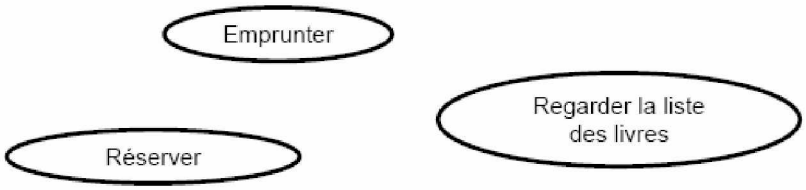
En se posant ce type de question, on verra que généralement des cas d'utilisation comme « Rechercher client » ou « Imprimer facture » ne sont pas des cas d'utilisation mais plutôt des fonctions du système. L'important avec les cas d'utilisation est bien de décrire ce que l'on pourrait désigner savamment par « unité d'intention complète ». C'est-à-dire une série d'envois d'évènements de la part de l'acteur au système et de réponses du système pour atteindre un objectif métier précis. Et non les différentes fonctions du système qui seront en fait déduites des différents cas d'utilisation.

Un cas d'utilisation est donc composé des éléments suivants :

- Un nom : Utiliser un verbe à l'infinitif (Ex : Réceptionner un colis)
- Une description résumée permettant de comprendre l'intention principale du cas d'utilisation. Cette partie est souvent renseignée au début du projet dans la phase de découverte des cas d'utilisation.
- Des acteurs déclencheurs : ceux qui vont réaliser le cas d'utilisation (la relation avec le cas d'utilisation est illustrée par le trait liant le cas d'utilisation et l'acteur dans un diagramme de cas d'utilisation)
- Des acteurs secondaires : ceux qui ne font que recevoir des informations à l'issue de la réalisation du cas d'utilisation (Ex : client ou autre système informatique. La relation avec le cas d'utilisation est illustrée par le trait liant le cas d'utilisation et l'acteur dans un diagramme de cas d'utilisation)
- Des pré-conditions qui décrivent dans quel état doit être le système (l'application) avant que ce cas d'utilisation puisse être déclenché (Ex : un contrat existe avec le client)
- Des scénarii. Ces scénarii sont décrits sous la forme d'échanges d'évènements entre l'acteur et le système. On classe les scénarii en : Un scénario nominal (celui qui est déroulé quand il n'y a pas d'erreur, celui qui est principalement réalisé dans 90% des cas), des scénarii alternatifs qui sont les variantes du scénario nominal et enfin les scénarii d'exception qui décrivent les cas d'erreurs.
- Des post-conditions qui décrivent l'état du système à l'issue des différents scénarii (Ex : un contrat est créé et le système back-office est mis à jour avec le nouveau contrat créé)
- Des informations sur l'utilisation du cas d'utilisation comme : le nombre de personnes exécutant ce cas d'utilisation dans une journée type, le nombre d'objets (métiers !) traités par le cas d'utilisation dans une journée type (Ex : 120 contrats créés entre septembre et novembre, 2000 consultations des contrats par jour)
- Eventuellement une description des besoins en termes d'interface graphique. Ce chapitre étant réservé à des cas simples car généralement traité en dehors de la description même du cas d'utilisation. Dans ce cas une cohérence doit d'ailleurs être assurée entre l'IHM et la description du cas d'utilisation.

Les cas d'utilisation (use-case)

- Un cas d'utilisation est un moyen de représenter les différentes possibilités d'utiliser un système.
- Il exprime toujours une suite d'interactions entre un acteur et l'application.
- Il définit une fonctionnalité utilisable par un acteur.



Les relations entre cas d'utilisation

UML définit 3 grands types de relations entre cas d'utilisation :

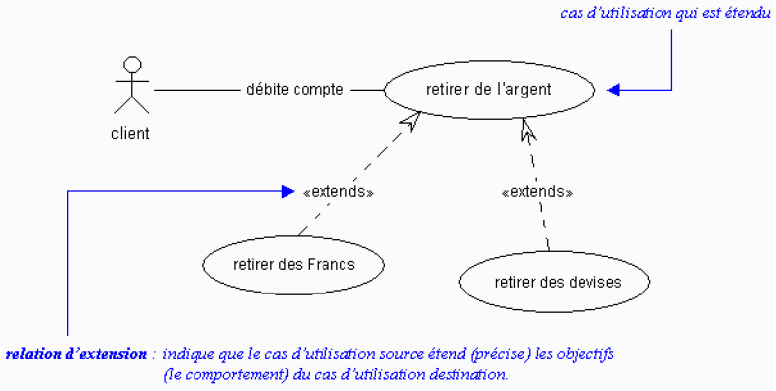
- généralisation/spécialisation,
- include,
- extends.

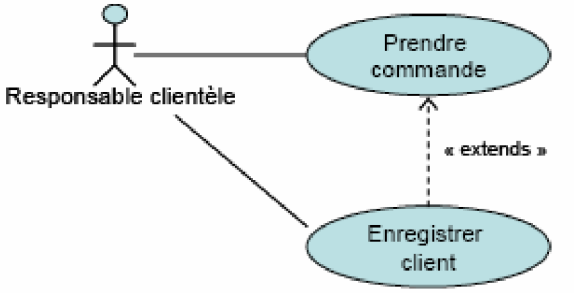
Il est important de noter que l'utilisation de ces relations n'est pas primordiale dans la rédaction des cas d'utilisation et donc dans l'expression du besoin. Ces relations peuvent être utiles dans certains cas mais une trop forte focalisation sur leur usage conduit souvent à une perte de temps ou à un usage faussé, pour une valeur ajoutée, au final, relativement faible.

Extends

La relation « d'extend » est probablement la plus utile car elle a une sémantique qui a un sens du point de vue métier au contraire des 2 autres qui sont plus des artifices d'informaticiens.

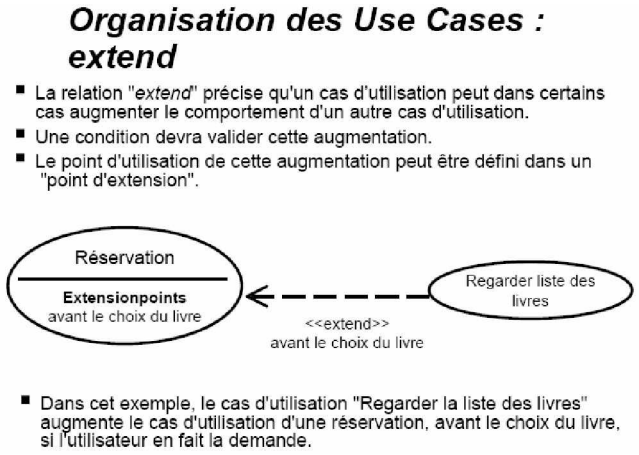
On dit qu'un cas d'utilisation X étend un cas d'utilisation Y lorsque le cas d'utilisation X peut être appelé au cours de l'exécution du cas d'utilisation Y comme :





Ce type de relation est primordial pour l'écriture de l'application. Imaginer dans le cas précédent que l'on n'ait pas mis la relation « extends ». Cela signifierait que lors de la prise de commande pour un nouveau client, le processus de prise de commande devrait être annulé au moment de la saisie des informations client, pour d'abord exécuter « Enregistrer client » afin que le client soit connu puis ensuite, reprendre le processus de prise de commande depuis le début ; pas cool pour notre responsable clientèle et bonjour l'image commerciale donnée au client !

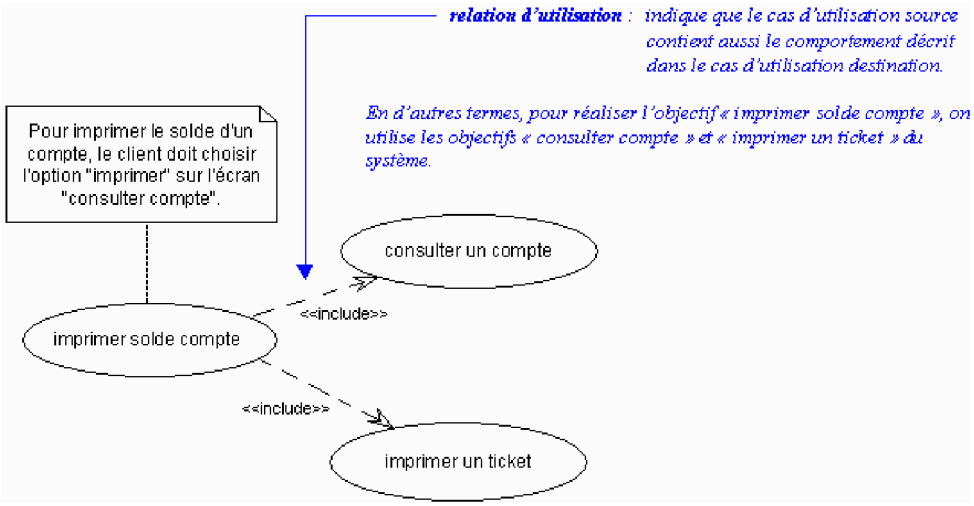
Dans le cas du « extends », le cas d'utilisation qui étend l'autre peut avoir un acteur déclencheur.



Include

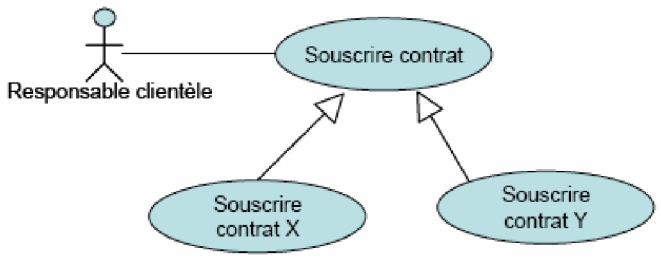
La relation d'include n'a pour seul objectif que de factoriser une partie de la description d'un cas d'utilisation qui serait commune à d'autres cas d'utilisation. Le cas d'utilisation inclus dans les autres cas d'utilisation n'est pas à proprement parlé un vrai cas d'utilisation car il n'a pas d'acteur déclencheur ou receveur d'évènement. Il est juste un artifice pour faire de la réutilisation d'une portion de texte.

Une erreur classique est d'utiliser la relation « d'include » pour faire du découpage fonctionnel d'un cas d'utilisation en plusieurs « sous cas d'utilisation » qui s'enchaînent en fonction de certains critères. On en arrive alors à se demander : Comment documenter l'enchaînement des sous cas d'utilisation avec UML ? Mais cette question n'a en fait pas lieu d'être, tout simplement.



Généralisation / spécialisation (communication)

Cette relation consiste à dire que l'on a un cas d'utilisation dit « de base », générique, qui décrit des séquences d'évènements et d'autres cas d'utilisation qui héritent de ce comportement de base et le spécialise suivant différents critères. On pourra par exemple avoir une situation comme suit :



Bilan sur les relations

Vous l'aurez compris, les relations entre cas d'utilisation ne doivent pas être votre préoccupation première. La relation « extends » est sûrement la plus utile car elle a une réelle signification pour le processus de travail de l'acteur. Notons bien aussi que les cas d'utilisation ne s'enchaînent pas, erreur souvent due à une mauvaise utilisation de la relation « d'include ».

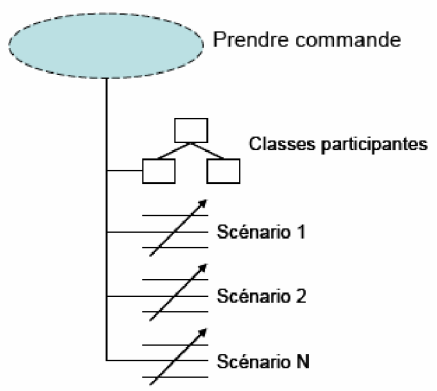
Les « Use case Realization »

Sans vouloir dérouler une méthode complète, je voudrais parler des « use case realization ». Après avoir rédigé les cas d'utilisation, ont fait souvent de l'analyse puis de la conception afin d'identifier des objets, des classes, des données et des traitements qui vont permettre au système de supporter les cas d'utilisation.

Pour documenter un modèle UML d'analyse et de conception et la manière dont sont mis en oeuvre les cas d'utilisation du système, on pourra utiliser le mécanisme des « use case realization ».

Les « use case realization » sont un moyen de regrouper un diagramme de classes et des diagrammes de séquences ou de collaboration. On retrouvera dans le diagramme de classes les classes qui mettent en oeuvre le cas d'utilisation associé au « use case realization » (structure des classes et relations entre classes).

On retrouvera dans les différents diagrammes de séquences (ou de collaboration) une documentation des différents évènements échangés entre les objets afin de réaliser les différents scénarii décrit dans le cas d'utilisation.



Au final, dans le modèle d'analyse et dans le modèle de conception, on aura un « use case realization » par cas d'utilisation et dans chaque « use case realization » on aura autant de diagrammes de séquence (ou de collaboration) que de scénarii décrits dans le cas d'utilisation (scénario nominal, scénarios alternatifs et scénarios d'exception). Plusieurs scénarii pourront cependant être regroupés dans un seul diagramme de séquence.

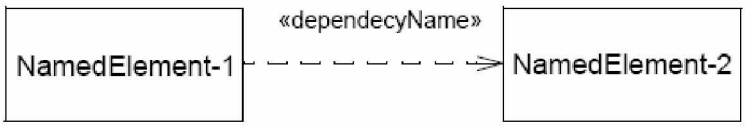
Les « use case realization » permettent dans la pratique d'apporter un élément de réponse à la question : Comment structurer mon modèle UML ?

Ils sont aussi une bonne réponse aux besoins de traçabilité entre les exigences que sont les cas d'utilisation et la solution apportée pour répondre à ces exigences : les classes du modèle. Cette traçabilité est très utile ensuite pour les équipes de maintenance qui peuvent alors plus facilement faire de l'analyse d'impact lors d'un changement sur l'existant.

Autres relations

À côté des principales relations déjà vues comme <<communicate>>, <<include>>, <<extend>>, on peut définir à notre guise d'autres possibilités comme <<use>>, <<associate>>, <<realize>>, <<subscribe>>, <<dependency>>, <<trace>>, etc. selon notre propre besoin.

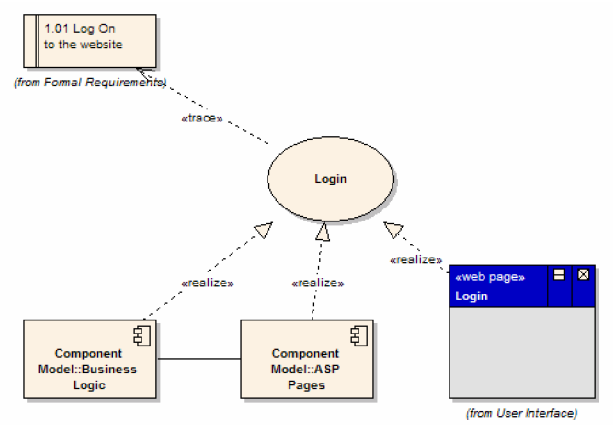
Ci-dessous une dépendance fonctionnelle :



L'élément qui se trouve sur la pointe de la flèche dépend de l'autre élément.

La relation <<realize>> par exemple est plus complexe car elle permet de connecter deux modèles, en général le modèle use case avec le modèle dynamique. Toujours employé au niveau de la spécification, par exemple, si use case est trop complexe à comprendre, on peut utiliser un diagramme dynamique (activités, séquence, communication, charte d'état ou un autre schéma d'analyse, la liste n'est pas limitative a priori). Le modèle

connecté à ce use case est un élément fils. L'objectif est d'apporter toutes les précisions nécessaires pour décrire le use case en cours.



Vous pourrez implémenter votre propre sémantique tout en respectant l'esprit de cette phase qui est une bonne compréhension et estimation du travail à réaliser, sans dériver vers une conception précoce.

Ce schéma prise dans le site de "Enterprise Architect" montre par exemple la réalisation du use case Login avec des composants Business Logic et ASP-Pages.

Caractéristiques de la modélisation par use cases

La modélisation par use cases apparaît comme un document contractuel sur lequel le donneur d'ordre et l'exécutant travaillent. Elle permet de vérifier qualitativement la conformité du produit avec les spécifications.

Ce « mal nécessaire » vient du fait que le donneur d'ordre ignore la façon dont le produit sera conçu avec ses diagrammes techniques par l'exécutant et que le donneur d'ordre pense fondamentalement « fonctionnalités ».

La modélisation par use cases adopte un formalisme vraiment simpliste, ce qui facilite la compréhension par tout le monde, surtout aux non initiés aux pratiques objet. La modélisation par use cases n'est pas une technique OBJET en soi.

L'exécutant a besoin des use cases pour faire l'évaluation de l'ampleur du projet et établir le coût et les autres paramètres d'affaires.

Une bonne modélisation par use cases doit répondre à la question : que fait le système à un haut niveau et jamais comment (WHAT et NOT HOW) ?

Si, accidentellement, vous devez vous adresser au COMMENT, il faudrait tout de même considérer cette partie comme secondaire et « sujette aux changements » et le signaler expressément pour ne pas avoir les mains liés et obliger de quémander des changements des spécifications au donneur d'ordre par la suite.

Doit-on décomposer les use cases et partir dans les techniques de réduction de complexité ?

NON lorsqu'on est capable, avec les use cases en place, d'estimer l'ampleur du projet.

OUI en considérant les sous-systèmes comme des systèmes « indépendants » en soi. Avec cette démarche, on s'assure que la décomposition respecte le principe d'indépendance des sous systèmes. On s'assurera que les sous-systèmes peuvent être considérés comme des objets à part entière.

La modélisation par use cases reste au niveau « donneur d'ordre / exécutant ».

En revanche, si l'exécutant lui-même subdivise le projet et passe les sous-systèmes aux « sous contractants » ou aux divers groupes dans l'entreprise, on rebouclera dans le processus « donneur d'ordre / exécutant ».

Donc, en présence de sous-systèmes avec une répartition des responsabilités, de nouveaux use cases plus élémentaires seront alors nécessaires pour les cahiers de charge sectoriels. Certains use cases peuvent alors devenir des acteurs pour d'autres sous systèmes.

Quelques recettes pour établir les cas d'utilisation

Étape 1: Identifiez les « usagers » du système. Ce sont les acteurs avec le stéréotype « acteur ». Lorsqu'un système est subdivisé en sous-systèmes plus élémentaires, un sous système peut devenir un acteur pour un autre

sous système et n'a rien d'humain. Dans ce cas, servez-vous du stéréotype « classe » (rectangle) pour l'acteur si le sous-système qui agit en tant qu'acteur s'apparente à un objet.

Étape 2: Différenciez les acteurs si possible si la nature du problème l'exige. Cherchez les propriétés communes de ces acteurs. Factorisez éventuellement un acteur commun et dérivez (si possible) les autres acteurs à partir de cet acteur commun.

Étape 3: Choisissez l'acteur type ou l'acteur le plus important. Cherchez ce que cet acteur peut faire avec votre système et identifiez les premiers use cases de base

Étapes 4: Étudiez chaque use case de base. S'il a l'air complexe et ne vous permet pas d'estimer l'ampleur du projet, identifiez les use cases sous-jacents les plus importants sans caractériser à cette étape-ci les relations entre les use cases (include, extend, etc.)

Étape 5: Décrivez en détail chaque use case, aussi bien des use case de base que les use cases sous-jacents. C'est une étape importante car elle permet de clarifier ce que vous voulez confier comme tâches à votre système. Cette description est consultée, d'un côté, par le donneur d'ordre, de l'autre, par les développeurs. Bien que la description soit TEXTUELLE, restez PRÉCIS, CONCIS
Ne pas faire mention de l'interface usager dans cette description car ce faisant, vous ligotez les bras du développeur.

Étape 6 : Supprimer les uses cases qui n'ont pas leur raison d'être. Généralement, un use case est "réalisé" avec une mini diagramme de séquence ou de collaboration, aussi petit soit-il. Si ce n'est pas le cas, peut être votre use case n'a pas sa raison d'être.

Étape 7: Trouvez maintenant les traitements communs dans les descriptions et cherchez les possibilités de factorisation. Dégagez si possible les use cases « factorisés »

Étape 8: Établissez maintenant les liens stéréotypés pour préciser la sémantique de votre analyse. C'est la partie un peu plus technique de la spécification.

Étape 9: Vous devez recommencer les étapes 4 à 9 pour tous les use cases de base.

Étape 10: Identifiez les acteurs secondaires (par exemple, les acteurs de maintenance et recommencez les étapes de 2 à 9 pour les acteurs secondaires)

Nouveau diagramme Processus d'Affaires

Dans sa nouvelle version, UML 2.0 prévoit, spécifiquement dans le paquetage "Profiles", un mécanisme d'extension des méta modèles pour que l'usager puisse adapter et créer d'autres diagrammes adaptés à ses propres besoins.

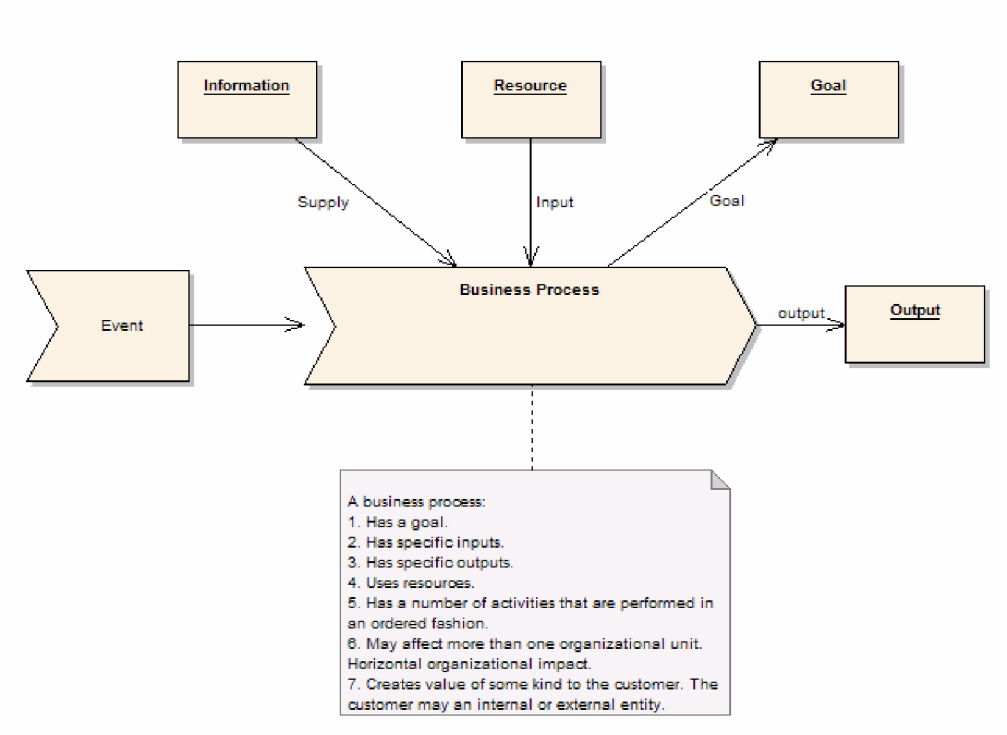
Un diagramme de Business Process permet, au niveau de la modélisation des besoins, de représenter les objets à l'entrée, les objets en sorties, les composants nécessaires au fonctionnement du processus d'affaires. Il s'agit d'un diagramme hybride dérivant du diagramme d'activités qui se trouve dans le paquetage dynamique de UML.

Remarquez que la notion d'objet est absente dans le diagramme des use cases, à l'exception des acteurs humains ou non humains. Par exemple si l'on désire représenter un document telle qu'une facture dans le diagramme des use cases, on aura beaucoup de mal. Le Business Process doit apporter une réponse dans ce contexte.

En observant le diagramme de principe dans la page suivante tirée de Enterprise Architect, on observe, encore une fois, la tendance, le virage vers les modèles fonctionnels du passé.

Soyons vigilant.

Ce type de diagramme peut être conçu comme une "réalisation" d'un use case, donc comme un diagramme fils.


Information
Resource
Goal
Supply
Input
Goal
Event
Business Process
output
Output
A business process:
1. Has a goal.
2. Has specific inputs.
3. Has specific outputs.
4. Uses resources.
5. Has a number of activities that are performed in an ordered fashion.
6. May affect more than one organizational unit. Horizontal organizational impact.
7. Creates value of some kind to the customer. The customer may an internal or external entity.