

A propos de cas d'utilisation

Auteur : Erik Gollot (ego)



http://www.chamonix-aventure.com/g_gkoyo

Quelle drôle d'espèce UMLienne que ces cas d'utilisation ! Avez-vous déjà essayé d'en dompter un ? Difficile à apprivoiser que cette bête là, vous ne trouvez pas ?

En fait, c'est comme tout, comme une passoire ; une fois qu'on a compris que ça ne servait à égoutter et pas comme couvre-chef, on arrive à s'en servir sans avoir l'air trop ridicule (car on est ridicule avec une passoire sur la tête, non ?).

Bon alors, ça sert à quoi les cas d'utilisation ?

En quelques lignes

Les cas d'utilisation sont un moyen d'exprimer le **besoin des Utilisateurs** d'un système informatique¹ vis-à-vis de ce système. Ils sont une vision orientée Utilisateur de ce besoin au contraire d'une vision informatique.

Utilisateurs - Acteurs

UML n'emploie 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 ou encore envoyer un fichier de données ou une trame XML. 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 incluent 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, Approuvateur,...). 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 acteurs des référentiels de données existant dans le système d'information. Effectivement, le

¹ Système, application, logiciel

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 document 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 ? »
ou
- ❖ « 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'événements 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.

Les relations entre cas d'utilisation

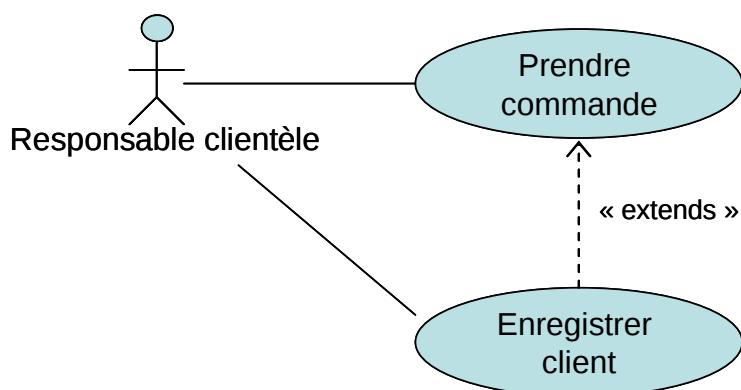
UML définit 3 grands types de relations entre cas d'utilisation : généralisation/spécialisation, include, extends.

Il est important de noter que l'utilisation de ces relations n'est pas primordial 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 !

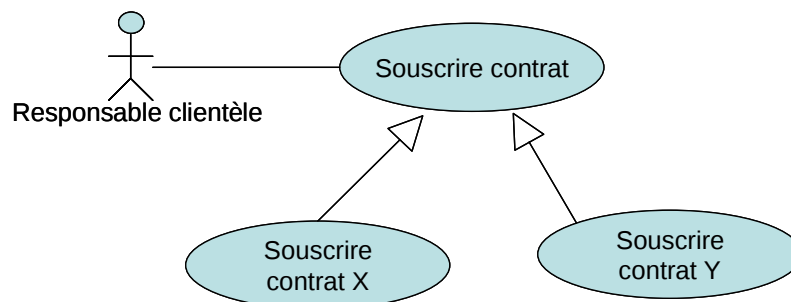
Include

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'évènement². 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 de 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

Cette relation est de mon point de vue la relation la plus discutable du point de vue de son utilité pratique. Elle 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 (le comment de la chose reste nébuleux). On pourra par exemple avoir une situation comme suit :



Il me semble que sauf cas évident, il n'est pas utile de perdre du temps à essayer de rendre générique un quelconque cas d'utilisation pour « s'amuser » ensuite à en faire hériter d'autres.

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 ». Si les cas d'utilisation s'enchaînent, ce n'est que dans le contexte d'une modélisation des processus métier ; et il ne s'agit pas de réaliser ce type de modélisation avec les cas d'utilisation³.

Les « Use case Realization »

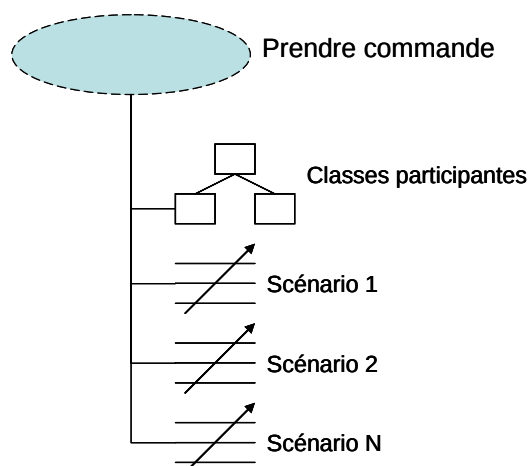
Sans vouloir dérouler une méthode complète, je voudrais parler des « use case realization »⁴ du Unified Process. Après avoir rédigé les cas d'utilisation, on fait souvent de l'analyse puis de la conception afin d'identifier des objets, des classes, des données et

² Dans le cas du « extends », le cas d'utilisation qui étend l'autre peut avoir un acteur déclencheur.

³ Une méthode comme le RUP a d'ailleurs défini le concept de « business use case » pour décrire les processus métier. Et les « business use case » ne sont pas un enchaînement de cas d'utilisation.

⁴ UML2.0 définit un mécanisme plus général avec les « Collaborations »

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 œuvre 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 œuvre 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 événements échangés entre les objets afin de réaliser les différents scénarii décrits 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 diagramme 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. Enfin, ce mode de traçabilité est aussi un bon moyen de répondre à des exigences comme celles du modèle CMMi du SEI⁵ (modèle de maturité de plus en plus adopté par les entreprises) en termes de traçabilité entre exigences et solution.

Conclusion

J'espère vous avoir un peu éclairé sur cette espèce que sont les cas d'utilisation et surtout sur leur bon usage⁶. J'espère aussi que vous aurez compris qu'il est inutile de compliquer leur utilisation en introduisant la notion de relation et que les cas d'utilisation deviennent « simple » si on s'en tient à répondre à la question : *Mais est ce que ce cas d'utilisation à un sens du point de vue métier pour l'acteur, n'est ce pas une vision informaticienne de son besoin ?* Les détails d'informaticiens seront abordés lors des activités d'analyse et de conception.

Ne soyons pas pressé et exprimons d'abord le besoin et ensuite la solution.

⁵ Cf. <http://www.sei.cmu.edu/cmmi/>

⁶ Bien d'autres choses seraient à dire sur le sujet !