

---

2

# **La calculette**

**[www.thierrycros.net](http://www.thierrycros.net)**

# 1

## Le projet calculette

---

La première étude de cas met en œuvre les concepts objets, UML et une démarche très simplifiée mais conforme au Unified Process.

Brièvement, l'objectif de cette première étude de cas est de :

- Parcourir des itérations afin de découvrir les relations entre cas d'utilisation, classes d'analyse, classe de conception

- Pratiquer le prototypage de l'architecture afin de la valider le plus tôt possible

- Tester la traduction des éléments logiques de conception en éléments physiques de programmation

- Mettre en œuvre des principes de base : composants et interfaces qui offrent un couplage plus faible entre implémentation et utilisation.

La mise en œuvre utilise le langage C++. Toutefois, une programmation dans un autre langage objet comme Java ne pose pas de problème particulier. L'environnement technique présenté pour le graphisme est de type Unix. Grâce à l'essor des logiciels libres, une plateforme logicielle sophistiquée de ce type (système multi-utilisateur, multi-tâche, interface graphique, nombreux outillages de développement) est aujourd'hui gratuite.

Cette étude est un cas d'école. Les développements réalisés ont pour but de traiter en détail les activités, en rapport avec le langage UML. De façon générale, le contenu correspond aux préoccupations des développeurs à tel ou tel stade de l'avancement d'un projet. La calculette est un prétexte à la mise en pratique, volontairement simple, sur un cas tout de même réel.

### 1.1 L'étude de cas

#### 1.1.1 Les fonctionnalités de la calculette

Le système à construire est une calculette-convertisseur Franc/Euro. L'étude est constituée d'une version graphique basée sur le standard des systèmes ouverts : Motif de l'Open Software Foundation (OSF).

Les fonctionnalités détaillées seront précisées dans l'étude elle-même, grâce aux cas d'utilisation.

### 1.1.2 Points techniques étudiés

Cette première étude de cas a pour objectif essentiel de traiter un exemple d'approche objet - modélisée grâce à UML – depuis une expression de besoins jusqu'à la programmation du logiciel, dans le cadre d'un cycle simplifié de type Unified Process.

#### Expression de besoins avec les cas d'utilisation

La calculette est un exemple simple qui permet de comprendre la technique d'expression de besoins par les cas d'utilisation. Les besoins fonctionnels et non-fonctionnels de la calculette seront traités sous forme de cas d'utilisation, ce qui est une simplification : généralement, les besoins non fonctionnels sont exprimés dans un dossier spécifique. Les besoins non fonctionnels sont survolés dans cette étude.

#### Relations cas d'utilisation – objets

Les cas d'utilisation correspondent à une approche par fonction du système, bien qu'il existe une différence fondamentale avec des techniques strictement fonctionnelles. La relation de collaboration est la clé de la découverte des objets du à construire.

N La différence essentielle entre approche classique fonctionnelle, telle que mise en œuvre dans la méthode  
 O SADT par exemple – et l'approche par les cas d'utilisation réside dans la *présence des acteurs*. D'un point de  
 T vue strictement étymologique, en effet, les cas d'utilisation sont des *fonctions* du système, mais ces fonctions  
 E sont reliées à des acteurs. Une conséquence bénéfique de la technique cas d'utilisation est le souci de  
 l'utilisateur que le développeur acquiert dès le lancement du projet. De ce fait, l'interface utilisateur, les tests  
 de saisie, les aides en ligne par exemple pourront être plus proches des utilisateurs. La connaissance des  
 acteurs est une condition sine qua non de l'ergonomie des systèmes. Afin que chaque utilisateur affirme : « ce  
 logiciel me va comme un gant ! » (ce qui est la définition même de l'ergonomie) encore faut-il que le  
 développeur apprenne à connaître l'utilisateur. Une véritable étude par les cas d'utilisation suppose que le  
 développeur ne fait pas l'économie de la connaissance de ses utilisateurs en particulier et du domaine en  
 général.

#### Cas d'utilisation, Analyse, Conception

Les cas d'utilisation forment le fil rouge des développements. Le passage aux modèles d'analyse, puis de conception permettra d'exposer les relations qui existent entre ces éléments. Nous utiliserons le stéréotype <<trace>> entre les différents éléments de modélisation, depuis les cas d'utilisation jusqu'aux modèles de conception et de réalisation.

#### Cycle de vie itératif et incrémental

La version proposée a pour but de mettre en pratique l'incrémental du logiciel. Dans cette étude de cas simplifiée, une seule itération de chaque phase sera réalisée. Itérer consiste à effectuer les activités (expression de besoins, analyse... test) plusieurs fois dans le développement d'une version ou release du logiciel.

#### Mise en œuvre en C++

La relation entre modèle de conception (vue statique des classes) et composants <<file>>, est une étape assez délicate de la réalisation. Qui n'a pas été confronté à des fichiers "header" C ou C++ mal conçus ? Par ailleurs, l'objectif n'est pas de mettre en œuvre des mécanismes sophistiqués du langage C++ tels que l'utilisation de classes spécialisées de la bibliothèque standard ou la généricité.

## 1.2 Démarche simplifiée

Le langage UML peut être comparé à une – belle – carrosserie, mais sans moteur. Autrement dit, UML n'est pas une méthode. Il reste donc à définir *au préalable* une démarche de mise en œuvre des concepts et diagrammes proposés par UML. Cet aspect est fondamental, même sur un petit projet. En effet, à quoi sert, par exemple, un modèle "cas d'utilisation" parfait si son intégration dans les développements n'est pas clairement définie ?

Le développement de la calculette est basé sur un cycle de vie simplifié par rapport au cycle complet (création, élaboration, construction, transition) constitué de deux phases : élaboration, construction.

Release	Phase	Itération
Calculette graphique	Elaboration	Itération d'élaboration
	Construction	Itération de construction

Figure 1-1 : Itérations de la calculette

Les phases de création et de transition seront présentées dans la deuxième étude de cas. Le tableau suivant récapitule ces itérations et leurs activités.

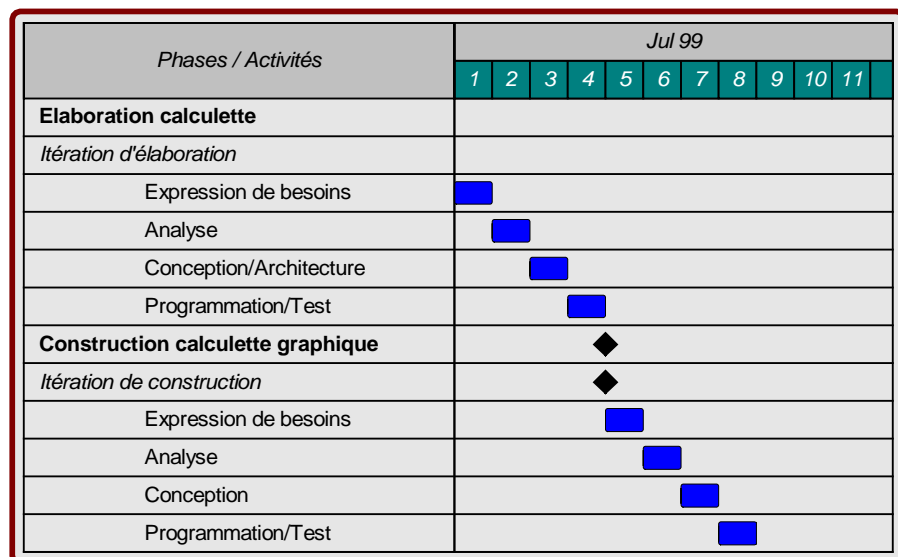


Figure 1-2 : Planning du projet

### 1.2.2.1 Les phases du cycle de vie

La version graphique de la calculette fait l'objet de deux phases, chaque phase étant constituée d'une itération.

#### Élaboration version calculette

Cette phase couvre aussi le démarrage du projet qui est habituellement réalisé lors de la pré-étude (création).

Expression des besoins de type calculette/convertisseur. Le résultat est le modèle des cas d'utilisation. L'expression de besoins n'est pas exhaustive dans la mesure où l'objectif est ici de valider l'architecture de la calculette à partir de cas d'utilisation significatifs.

Analyse du système à partir des stéréotypes proposés dans Unified Process. Le résultat est le modèle d'analyse.

Conception d'une architecture à partir des cas d'utilisation et des exigences en terme d'évolution. Le modèle de conception et le modèle de déploiement constituent le résultat.

Programmation et test des choix de conception, implémentation incomplète qui a pour but de valider l'architecture.

Le modèle des cas d'utilisation correspond à l'ensemble des diagrammes de cas d'utilisation qui expriment les besoins des acteurs. Il comprend également les descriptions textuelles des cas d'utilisation ainsi que des diagrammes de séquence qui précisent les interactions acteur/système lors des scénarios, c'est-à-dire des instances de cas d'utilisation.

Le modèle d'analyse présente les classes, les relations entre objets sous forme de collaborations, les diagrammes d'état des classes pertinentes du point de vue dynamique.

Le modèle de conception est une description des mécanismes de base d'implémentation, par exemple pour vérifier la pertinence et la faisabilité des décisions d'analyse en terme de responsabilités de classes. Il est constitué essentiellement de diagrammes de classes et de sous-systèmes ainsi que de diagrammes d'interactions.

Le modèle de déploiement permet de visualiser l'architecture matérielle de la calculette. Il est basé sur les diagrammes de déploiement et de composants. Les diagrammes d'interaction permettent de modéliser les relations dynamiques entre les nœuds.

A l'issue de cette première itération d'élaboration, nous devons être sûrs de l'architecture. Elle aura été validée par une activité d'implémentation / test.

### **Construction calculette version graphique**

Cette deuxième phase a pour but de fournir une version exploitable.

Expression complète des besoins

Analyse

Conception à partir du prototype réalisé en élaboration et en tenant compte de l'impact de l'aspect événementiel du graphisme

Programmation en C++ avec utilisation de la bibliothèque Motif. Cette itération sera l'occasion de présenter cet environnement graphique particulièrement sophistiqué grâce à UML.

Test à partir des cas d'utilisation.

## **1.3 Expression de besoins**

Ce paragraphe présente l'activité d'expression de besoins de la calculette.

Phases / Activités	Jul 99										
	1	2	3	4	5	6	7	8	9	10	11
Elaboration calculette											
Itération d'élaboration											

Figure 1-3 : Itération dans le cycle de développement

La finalité de cette activité est la description générale des fonctionnalités du système. L'étude des besoins n'est pas l'analyse. La question « quelles sont les fonctions du système ? », devient "quels sont les utilisateurs du système ? Et qu'attendent-ils du système ?".

Phases / Activités	Jul 99										
	1	2	3	4	5	6	7	8	9	10	11
Elaboration calculette											
Itération d'élaboration											
Expression de besoins											

Figure 1-4 : Activité dans l'itération

La démarche que nous proposons ici est très classique et naturelle. Elle est fortement simplifiée dans la mesure où les cas sont par nature très élémentaires et également très peu nombreux.

#### Démarche d'obtention du modèle cas d'utilisation

La démarche simplifiée est la suivante :

Obtenir les cas d'utilisation

- Définir les acteurs
- Lister les cas d'utilisation principaux
- Formaliser par un diagramme de cas d'utilisation
- Préciser par des descriptions textuelles

Vérifier les cas d'utilisation

Approcher le système par des diagrammes d'interaction acteur / système

#### 1.3.1 Définir les acteurs

Les acteurs sont déterminés par catégorie. De ce point de vue, le système est destiné à des utilisateurs de catégorie principale, c'est pour eux que le système est fabriqué. Ils sont la réponse à la question : pour qui est fabriqué le système ?

Le fonctionnement du système implique très souvent des opérations d'exploitation. Les fonctions de sauvegarde, de configuration ne sont pas les fonctions essentielles du système, mais elles sont obligatoires pour assurer son intégrité. Autrement dit, le système n'est pas conçu pour répondre à ces cas d'utilisation, mais ces cas d'utilisation secondaires autorisent l'exploitation du système. Les acteurs concernés sont de catégorie « secondaire ». Ces acteurs sont la réponse à la question : qui est nécessaire pour le bon fonctionnement du système ?

A ce stade, il est nécessaire de distinguer acteurs et utilisateurs humains. L'acteur est le rôle que joue un utilisateur humain, non l'humain en tant que tel. Une personne peut jouer plusieurs rôles. De façon générale, la liste des acteurs doit être très précise car elle induit la liste des cas d'utilisation qui forment le fil rouge des développements. Ici, le fait d'indiquer que la catégorie « matériel » est non applicable montre que l'on s'est vraiment posé la question (et accessoirement que l'on a la réponse). Notons enfin qu'il est possible d'affiner ces catégories de base. Les stéréotypes d'UML sont un excellent moyen d'adapter le concept d'acteur au domaine.

A qui est destinée la calculette ?

Le système - la calculette - est destiné à un utilisateur ou opérateur qui souhaite effectuer des calculs et des conversions. Ce dernier cas – conversion Franc / Euro - suppose la connaissance du taux de conversion. Ainsi, la valeur du taux, qui doit être *fournie* au système, induit la présence d'un autre rôle, donc acteur, qui configure le logiciel. Il s'agit alors d'un acteur secondaire que nous nommons Comptable.

Pourquoi distinguer ces deux acteurs ?

Après tout, c'est probablement le même être humain qui renseignera le taux et qui effectuera les opérations. La réponse est dans la nature des cas d'utilisation. Autant il est cohérent de développer une calculette-convertisseur pour qu'un utilisateur (qui joue un rôle donc en situation d'acteur) effectue des opérations ou des conversions, autant il serait curieux de créer un système dans le but de saisir un taux. C'est là toute la différence entre un cas principal et un cas secondaire. Notons enfin que la distinction entre acteur principal et secondaire permettra de traiter différemment les cas reliés. Typiquement, des cas d'utilisation d'acteurs secondaires pourraient supporter – par exemple – une interface utilisateur moins sophistiquée ou bien faire l'objet d'un exécutable spécifique. Il n'est pas rare de créer un exécutable par acteur voire cas d'utilisation. Le résultat des questions concernant les acteurs est consigné dans un tableau récapitulatif. Généralement, les termes sont précisés dans le dictionnaire ou glossaire associé au projet. Le rôle Utilisateur est retenu car opérateur prête à confusion dans le cadre d'une calculette.

Catégorie	Acteurs
Principal	Utilisateur
Secondaire	Comptable
Matériel	N/A (non/applicable)
Systèmes	N/A

Figure 1-5 : Tableau des acteurs de la calculette

Un acteur principal engendre des cas d'utilisation principaux. Ces cas d'utilisation impliquent la présence d'autres cas, liés à d'autres acteurs. Ainsi, la liste des acteurs est construite au fur et à mesure, en harmonie avec la liste des cas d'utilisation. Autrement dit, il ne faut pas espérer obtenir *d'abord* la liste de tous les acteurs, *ensuite* la liste de tous les cas d'utilisation. Au contraire, ce processus est largement itératif. Il suppose un dialogue très fort avec les utilisateurs du système. Notons au passage que la détermination des acteurs induit naturellement une première liste de cas d'utilisation candidats c'est-à-dire qui restent à valider.

UML permet de décrire les acteurs dans un diagramme de cas d'utilisation simplifié qui ne contient pas de cas. Il reste à juger de la pertinence de ce type de diagramme par rapport à un diagramme classique de contexte système qui visualise éventuellement les propriétés (étiquettes, contraintes...) des acteurs. Un diagramme d'acteur spécifique permet de ne pas surcharger le diagramme de contexte système. Ici, l'étiquette *documentation* permet de définir le rôle. En résumé, ce type de diagramme met l'accent sur les acteurs alors que le diagramme de cas d'utilisation habituel met l'accent sur les cas. Visuellement, les acteurs apparaissent *hors* du système. UML permet de communiquer par l'image. Notez que, par convention « maison », l'acteur principal est à gauche du système : il est vu le premier, alors que l'acteur secondaire est à droite. Ainsi, l'œil le découvre naturellement après. Bien entendu, ce type de convention ne fait pas partie d'UML, mais il est recommandé de faciliter la lecture des diagrammes par de telles conventions d'utilisation du langage.

Les acteurs sont documentés afin de les gérer en configuration (à partir de l'atelier de génie logiciel, AGL). Cette documentation est ainsi facilement et directement accessible depuis le navigateur de l'AGL. La figurine de l'acteur Comptable est volontairement diminuée, ce qui offre un autre moyen visuel de distinction principal / secondaire.

N Les manuels utilisateur des logiciels étaient dans un premier temps constitués de documents papier tels que  
O classeurs ou livres. De fait, ils étaient (très) peu utilisés. Qui ne connaît pas le vieil adage : « quand on a tout  
T essayé, on consulte la documentation » ? L'avènement des interfaces graphiques a généralisé les  
E documentations en ligne hypertexte et l'utilisation de moteurs de recherche à partir de mot clé. Ce type de documentation est parfaitement complémentaire. Sans remettre en cause la nécessité de documentation papier, les programmeurs utilisent l'AGL en tant qu'aide en ligne, l'effet immédiat étant que la documentation est ainsi facilement consultée. D'où la nécessité de renseigner les propriétés « documentation » des éléments de modélisation, ce qui est alors une autre bonne raison de générer la documentation papier à partir de l'AGL.

La génération automatique de documentation est un problème en soi. Au-delà de l'aspect technique de paramétrage des outils, de qualité du tracé après transfert entre AGL et traitement de texte, la question porte sur le fond. Certains générateurs produisent parfois trop de documentation. Les lecteurs ont face à eux des centaines de pages, structurées artificiellement à partir des possibilités de paramétrage de l'outil (possibilités de l'équipe à l'instant t en fonction des connaissances, du temps... et pas uniquement capacités intrinsèques de l'outil). A quoi sert ce type de document ? Autre aspect : la notion même de génération automatique suppose que l'on automatise *a priori* une tâche au préalable manuelle. Lorsqu'une équipe met en œuvre pour la première fois les technologies objet, il est important d'éviter ce piège des générateurs. Une étude préalable consiste à définir une documentation en fonction de la culture de l'entreprise, de besoins client, de procédures en vigueur de type ISO9000, etc. Alors, le paramétrage de l'outil de génération de documentation *automatisera* réellement une tâche parfaitement imposée par l'équipe. Dans certains cas, c'est malheureusement le générateur qui impose et l'équipe qui subit le bavardage textuel. Par ailleurs, la question de la documentation est un véritable projet en elle-même : par exemple synchroniser une documentation externe (client) et une documentation interne (maintenance).

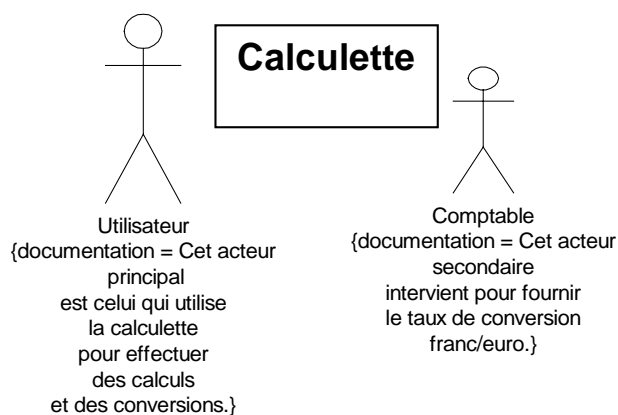


Figure 1-6 : Acteurs de la calculette

Un raffinement de cette taxonomie d'éléments externes consiste à définir un acteur Acheteur qui déclenche les cas de conversion. Dans cet ordre d'idée, l'Acheteur est un Utilisateur particulier. Le diagramme d'acteurs évolue et tient compte de cette caractéristique. Notons au passage que la généralisation entre acteurs est pratiquement la seule relation effectivement modélisée entre acteurs. En effet, visualiser des relations d'interactions entre éléments extérieurs au système remet en cause la frontière même du système. C'est un symptôme qui impose probablement la redéfinition de cette frontière.

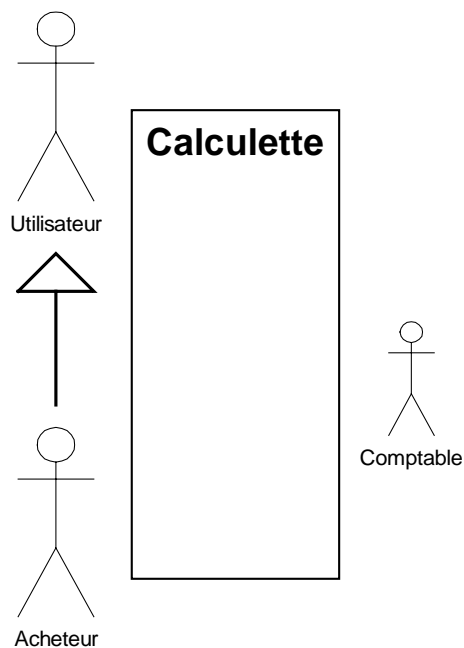


Figure 1-7 : Taxonomie d'acteur plus (trop ?) précise

Cette nouvelle classification sous-entend des situations d'utilisation de la calculette distinctes. Nous ne la retiendrons pas au final car elle n'apporte pas à notre sens de valeur ajoutée significative. Simplement, il est tout à fait normal de manœuvrer, de louvoyer, pour

obtenir la liste des acteurs. C'est vrai pour tous les éléments de modélisation, les acteurs et les cas d'utilisation ont toutefois un impact majeur sur le développement car ils pilotent le développement au moyen de la relation de collaboration. En dernière analyse, tout élément produit par le développement doit être relié à un cas d'utilisation. Du moins, sans le préciser explicitement au travers de relations telles que <<trace>>, un développeur devrait pouvoir répondre à toute question de type : « à quels cas d'utilisation cet élément correspond-il ? ».

### 1.3.2 Liste des cas d'utilisation principaux

L'utilisateur de la calculatrice souhaite effectuer des calculs et des opérations. Une liste de cas d'utilisation telle que : effectuer une addition, effectuer une soustraction, etc est possible. Très rapidement, la description textuelle de ces cas d'utilisation démontrerait leur faible valeur ajoutée. En effet, une description de quelques lignes est suspecte, elle dénote généralement un niveau de détail (granularité) trop fin ou tout simplement une approche fonctionnelle.

Notons que le cas d'utilisation "Taux de conversion" correspond à une synchronisation du système avec son environnement. Ce cas est une réponse à la question : quels sont les changements à l'extérieur du système qui doivent être connus du système ? Ici, nous pouvons considérer que la connaissance du taux de conversion au 1<sup>er</sup> janvier 1999 a créé un changement à l'extérieur du système qu'il convient de répercuter. C'est une question de synchronisation. De même, un système d'exploitation synchronise régulièrement le disque à partir des buffers en mémoire vive.

Acteur	Cas d'Utilisation
Utilisateur	Opération (effectuer une opération) Conversion (convertir une valeur)
Comptable	Taux de conversion

Figure 1-8 : Liste des cas d'utilisation de la calculatrice

L'utilisateur de la calculatrice joue le rôle d'Utilisateur quand il effectue des opérations et des conversions, le rôle de Comptable quand il initialise le taux de conversion Franc / Euro.

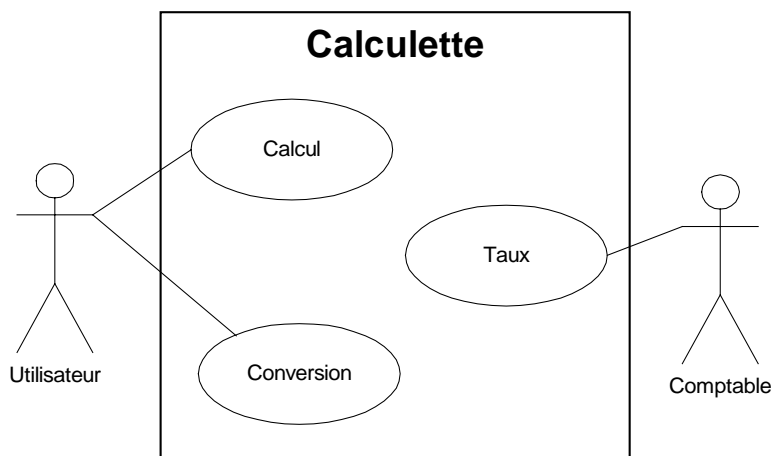


Figure 1-9 : Contexte système calculette (hors répertoire)

Notons que les cas d'utilisation Calcul et Conversion pourraient être car il s'agit de deux cas extrêmement simples et liés fonctionnellement. Par contre, le cas Taux doit être isolé car il s'agit d'un cas d'exploitation.

Pourquoi créer un diagramme de contexte ?

De façon générale, UML permet de visualiser, autrement dit de communiquer, par le dessin. Le rectangle associé au système montre clairement ce qui est à l'intérieur et ce qui est à l'extérieur du système. Bien entendu, le concept même d'acteur est extrêmement clair : un acteur est externe. Simplement, le visualiser ainsi facilite la lecture et la compréhension du diagramme, en particulier lorsque le lecteur n'est pas familiarisé avec UML. Notons aussi que certains projets se caractérisent par une difficulté sérieuse à définir ce qui est – et n'est pas – à l'intérieur du système. Par ailleurs, plusieurs diagrammes de contexte deviennent nécessaires lorsque le système est très complexe et que les éléments à modéliser (acteurs et cas) sortent du format d'impression, en général le format A4.

ICI NOTE SUR LA TAILLE DES DIAGS ET LE NBRE D'ELSTS CONTENUS

### 1.3.3 Descriptions textuelles des cas d'utilisation

#### 1.3.3.1 La fiche-type de description

Les cas d'utilisation de la calculette sont ici décrits au moyen d'une « fiche-type » simplifiée : tous les éléments nécessaires dans un projet réel ne sont pas repris. Le niveau de détail des interactions reste grossier car il s'agit - dans ces descriptions - de spécifier les échanges acteurs / système au travers des cas, non pas de définir précisément l'interface homme-machine (IHM) du système. Une fiche-type de description est indispensable, afin d'harmoniser les descriptions de différents analystes. Notons que la partie description peut couvrir plusieurs pages. Le nombre de pages maximum est aussi une décision de type assurance qualité du projet. Ici, les cas sont trop simples pour que cette question ait un sens. Dans cet ordre d'idée, le terme « fiche » ne doit pas prêter à confusion. Il s'agit en réalité de plan type de paragraphes (un par cas) qui sont injectés dans la documentation du modèle d'expression de besoins.

**N** La longueur moyenne de la description des cas est un sujet de discussion. Certains consultants ou experts  
**O** préconisent des textes de plusieurs pages, par exemple 10 pages au maximum. D'autres préfèrent des  
**T** descriptions beaucoup plus courtes de l'ordre d'une ou deux pages. Le domaine, le projet, l'expérience  
**E** acquise forgent une opinion. Les descriptions courtes ont tout de même un danger, celui de la dispersion. La  
 règle qui consiste à dire « quelques grands cas » assure une bonne synthèse de l'essence du système. Ici, nous  
 évoquons les cas au sens classique, situation dans laquelle un utilisateur est en contact avec le système. Les  
 relations d'organisation de cas (« extend », « include ») limitent la taille réelle des descriptions, ce qui est  
 offre un bon compromis. Nous reviendrons sur ce point précis dans la deuxième étude de cas.

Les références à une éventuelle maquette précisent le style d'échanges. La maquette permet de régler les détails de l'interface. La description des cas d'utilisation et la maquette sont complémentaires et ne font pas double emploi. En effet, une maquette ne permet pas d'énoncer facilement et sans ambiguïté l'enchaînement des interactions. Elle ne fournit pas non plus directement l'ensemble des cas alternatifs qui sont intégrés dans la description textuelle, par opposition aux cas normaux. Sans maquette logicielle, des représentations papier permettent de matérialiser les échanges décrits dans les cas d'utilisation. Toutefois, évoquer une maquette, des noms de boutons, etc dans une description textuelle de cas d'utilisation présente un danger. D'une certaine manière, cela limite l'expression de besoins à la première mise en œuvre de l'interface utilisateur. Le Unified Process propose les descriptions *essentielles* qui excluent toute référence à une quelconque mise en œuvre pour s'en tenir à la signification (l'essence) des échanges entre acteurs et système.

<b>Titre</b>	<Titre du cas d'utilisation>
<b>Acteur(s)</b>	<Ici les acteurs participant au cas d'utilisation (au moins un)>
<b>Description</b>	<Ici description des interactions (et non pas du comportement interne du système). Le langage est généralement naturel. Des mots clés peuvent être insérés pour simplifier les descriptions de choix ou de boucles. Les choix alternatifs, exceptionnels, les cas limites ou d'erreur peuvent être distingués visuellement par des caractères spéciaux, par exemple [annulation]. Ils renvoient au champ "Exceptions". >
<b>Exceptions</b>	<Ici description des interactions alternatives ou exceptionnelles>

Figure 1-10 : Fiche Type simplifiée de description de cas d'utilisation

Le Titre est celui utilisé dans la liste des cas d'utilisation.

Les Acteurs interviennent dans les interactions, qu'ils soient déclencheurs ou non du cas d'utilisation. De façon générale, un cas d'utilisation doit concerner au moins un acteur : le bénéficiaire du cas (« à qui profite le cas ? »). Théoriquement, un cas d'utilisation est déclenché par un acteur.

La Description textuelle doit être compréhensible par un non informaticien, sauf dans le cas particulier de système destiné à des informaticiens. Mais il est rare que *tous* les intervenants d'un projet (client, financier...) soient informaticiens. La description est donc basée sur le langage naturel. En effet, l'une des finalités des cas d'utilisation est de faciliter le dialogue entre développeurs et utilisateurs, afin d'éviter le "syndrome de la balançoire" (célèbre ensemble de dessins qui montre les déviations subies par un logiciel depuis les vœux de l'utilisateur jusqu'à sa réalisation).

ICI NOTE AUTRES POSSIBILITES DE DESCRIPTION DES CAS E/T...

La clause "Exception" correspond à la description des interactions spécifiques aux cas alternatifs ou exceptionnels (cas limite, cas d'erreur ou annulation par exemple).

La description textuelle des cas d'utilisation démarre lorsque la fiche-type et le niveau de détail sont établis. De ce point de vue, un acteur donné devient un objet d'un système englobant au même titre que le système lui-même. Les deux objets (acteur, système) collaborent lors d'un cas d'utilisation. La collaboration entre ces objets suppose l'envoi de messages, *a priori* dans les deux sens. La terminologie consacrée pour ces messages est "interactions". L'association qui relie les deux classificateurs est une relation de communication.

N Les structures de contrôle de flot d'exécution : tant que...fin tant que, boucle... fin boucle, choix parmi...  
 O peuvent être injectées dans les descriptions textuelles. Le terme pseudo-code est suspect car il sous-entend  
 T quelquefois une conception détaillée pour un développeur. Or, ces structures existent dans la réalité des  
 E systèmes, sans aucun lien avec des langages de programmation. Par exemple, un comptable calcule une facture et cumule des prix hors taxes "tant que" des produits sont à traiter, dans son système de facturation. De plus, ces structures de contrôle ne sont absolument pas destinées à se transformer à l'identique en programmation. En effet, les choix de conception peuvent induire des réalisations dans lesquelles les structures sont invisibles. Dans le cas d'interface graphique, un utilisateur utilisera autant de fois que nécessaire un bouton pour réaliser une action, mais le code ne comprendra pas cette structure « autant de fois que nécessaire ». C'est en quelque sorte le scénario imposé par l'acteur qui définit la boucle et non le code. Attention donc à ne pas confondre description de cas d'utilisation et conception détaillée prématurée.

### 1.3.3.2 Les descriptions

#### Cas d'utilisation « Opération »

L'utilisateur n'envisage pas dans une première version l'enchaînement de calculs. Seules des opérations isolées telles que :  $20 + 32$  sont acceptées. L'utilisateur peut annuler une opération à tout moment : c'est un cas alternatif par rapport à l'utilisation normale qui consiste à mener le calcul jusqu'au bout.

Titre	Opération
Acteur(s)	Utilisateur
Description	<p>Ce cas d'utilisation est déclenché par l'Utilisateur lorsqu'il souhaite effectuer une opération et qu'il saisit le premier chiffre.</p> <p>Il saisit le premier opérande.            La calculette affiche cet opérande au fur et à mesure de la saisie des chiffres.</p> <p>L'utilisateur choisit l'opérateur parmi : + - x /            La calculette affiche toujours le premier opérande.</p> <p>L'Utilisateur saisit le deuxième opérande.            La calculette affiche cet opérande au fur et à mesure de la saisie des chiffres.</p> <p>L'Utilisateur demande le résultat.            La calculette affiche le résultat. [Division par zéro]            Le cas d'utilisation est alors terminé.</p> <p>[Annulation]</p>

**Exceptions**

[Annulation]

A tout moment, l'Utilisateur peut annuler l'opération complète en appuyant sur « AC ». Dans ce cas, la calculette affiche « 0 ».

[Division par zéro]

Dans ce cas, la calculette affiche « 0 ».

A ce stade, la maquette de la calculette peut être (sans tenir compte de la virgule pour simplifier) :

0			
1	2	3	/
4	5	6	x
7	8	9	-
AC	0	=	+

Figure 1-11 : Maquette temporaire calculette pour cas Calcul

La mise au point de cette description suppose un dialogue intense entre analyste et utilisateur. L'analyste doit obtenir les besoins fonctionnels – cas d'utilisation – et les spécifications non-fonctionnelles. Ici, la précision de l'affichage, par exemple six chiffres après la virgule, est un exemple de besoin non-fonctionnel. Cette information est alors traitée sous forme de contrainte à associer au cas d'utilisation. Cela permet simplement de faciliter la traçabilité de cette exigence non-fonctionnelle car elle est ainsi partie intégrante du modèle géré dans l'AGL.

**Cas d'utilisation « Conversion »**

Ce cas d'utilisation consiste à effectuer une conversion Franc vers Euro ou l'inverse. Il fait apparaître une pré-condition dans la mesure où le taux de conversion doit être connu pour que ce cas se déroule correctement. Une autre solution est de considérer la situation « taux inconnu » en tant que cas exceptionnel.

<b>Titre</b>	<b>Conversion (convertir une valeur)</b>
<b>Acteur(s)</b>	Utilisateur
<b>Pré-condition</b>	Le taux de conversion doit être connu de la calculette

**Description**

Ce cas d'utilisation est déclenché par l'Utilisateur lorsqu'il souhaite convertir une valeur Franc vers Euro ou l'inverse et qu'il saisit le premier chiffre.

L'Utilisateur saisit la valeur.

La calculette affiche cette valeur au fur et à mesure de la saisie des chiffres.

L'utilisateur choisit l'un des éléments suivants :

Franc→Euro) la calculette affiche la conversion en euros

[Taux inconnu]

Euro→Franc) la calculette affiche la conversion en francs.

[Taux inconnu]

Le cas d'utilisation est alors terminé.

[Annulation]

**Exceptions**

[Annulation]

A tout moment, l'Utilisateur peut annuler la conversion en appuyant sur « AC ». Dans ce cas, la calculette affiche « 0 ».

[Taux inconnu]

Lorsque le taux n'a pas été saisi, la calculette n'autorise pas les choix Franc→Euro ni Euro→Franc. Autrement dit, la connaissance du taux est une pré-condition de ce cas.

A ce stade, la maquette de la calculette devient :

0			
E	F		
1	2	3	/
4	5	6	x
7	8	9	-
AC	0	=	+

Figure 1-12 : Maquette temporaire calculette pour Conversion

E pour convertir la valeur en Euros, F pour la convertir en Francs.

### Cas d'utilisation « Taux »

Le cas d'utilisation « taux de conversion » consiste à initialiser le taux de conversion Franc / Euro connu depuis le 1<sup>er</sup> janvier 1999.

<b>Titre</b>	<b>Taux de conversion</b>
<b>Acteur(s)</b>	Comptable
<b>Description</b>	Le comptable saisit les chiffres qui composent le taux. La calculette les affiche au fur et à mesure. Le comptable sélectionne « Taux ». La calculette mémorise ce taux. Le cas d'utilisation est alors terminé.  [Annulation]
<b>Exceptions</b>	[Annulation] A tout moment, le Comptable peut annuler la saisie en appuyant sur « AC ». Dans ce cas, la calculette affiche « 0 ».

La maquette devient alors

0			
E	F	T	/
1	2	3	x
4	5	6	-
7	8	9	+
AC	0	.	=

Figure 1-13 : Maquette calculette pour Taux

T : pour mémoriser le taux de conversion Franc / Euro.

Ce cas d'utilisation implique la présence du point décimal.

A ce stade, les cas d'utilisation de la première version sont *a priori* terminés. Pourquoi *a priori* ? Car les activités d'analyse, de conception, voire de programmation imposent parfois des modifications de spécification de besoins. De façon générale, nous dirons que l'architecture influence les cas d'utilisation.

#### 1.3.4 Vérification des cas d'utilisation

Les cas d'utilisation doivent être vérifiés à plusieurs niveaux. Les acteurs, la liste des cas, les diagrammes, les descriptions textuelles sont autant d'éléments de modélisation à valider avant de passer à une vue plus intérieure du système.

## Acteurs

Tous les acteurs ont-ils été pris en compte ? La typologie (principal, secondaire...) est-elle correcte ? Chaque acteur correspond-il à un rôle par rapport au système et non à un véritable utilisateur humain (pas d'acteur tels que Pierre, Paul ou Jacques) ?

Ici, « utilisateur » est assez pauvre en terme de description de l'acteur. Toutefois, la calculette en tant que système suppose des rôles d'utilisation très généraux. De façon générale, les termes tels qu'utilisateur, exploitant... dénotent soit des systèmes universels, soit une méconnaissance de l'environnement d'utilisation. Ici, nous opterons pour la première possibilité...

## Liste des cas d'utilisation

Tous les cas d'utilisation sont-ils pris en compte : par acteur, par événement de déclenchement ? La synchronisation système / environnement est-elle assurée ? Les cas d'utilisation secondaires sont-ils exhaustifs (avez-vous pensé à sauvegarder les données) ? Les cas d'utilisation sont-ils majeurs ? Autrement dit, chaque cas est-il une fonctionnalité qui « mériterait » le développement du système ? Souvent les cas définissent plutôt des interactions ou bien des situations particulières de cas plus généraux. De façon générale, la liste des cas d'utilisation peut être diminuée en regroupant des cas logiquement liés et considérés dans un premier temps comme séparés. C'est le très haut niveau sémantique des cas d'utilisation qui explique leur nombre réduit. *A contrario*, un nombre élevé de cas est typique d'un niveau de détail trop fin. Une astuce consiste à imaginer tel ou tel acteur (rôle joué par un utilisateur réel) où vous voulez sauf devant son écran. Il décide soudain de venir à sa station de travail et de lancer l'application. Pourquoi ? Pour créer une instance de cas d'utilisation (scénario). Cette vision de l'utilisation du système permet de conserver un haut niveau d'évaluation des cas d'utilisation et d'éviter la confusion cas / interaction. En dernière analyse, il s'agit d'épurer la liste de cas, afin de synthétiser la finalité du système.

Ici, une ébauche de liste de cas pourrait être : addition, soustraction... ce qui aboutirait à une première liste d'une dizaine de cas. Ce nombre est visiblement trop élevé pour un système de ce type. Imaginons la description textuelle du premier cas. A l'issue de cette première description, il est clair que les différents cas d'opération peuvent être regroupés en un seul texte, donc en un seul cas. Autant il est légitime de démarrer avec une liste imparfaite, autant l'itération sur les cas doit permettre d'obtenir *in fine* une liste correcte.

## Diagramme de cas d'utilisation

Comme pour tous les diagrammes, il faut vérifier la nomenclature : titre du diagramme en particulier. Vérifier aussi le souci de lisibilité : les éléments du diagramme sont-ils dessinés pour faciliter la compréhension du diagramme ? Dans un diagramme de cas d'utilisation, une convention pourrait être : acteurs principaux sur la gauche des ellipses, acteurs secondaires sur la droite.

Ici, un seul diagramme, celui de contexte qui montre tous les acteurs et leurs cas d'utilisation. Simplement, nous avons pris soin de placer l'acteur secondaire sur la droite.

## Description textuelle

L'écueil le plus classique est l'insertion de comportements internes. Or, les cas d'utilisation représentent une vision externe du système. De ce point de vue, le système est un objet opaque avec lequel on interagit lors de cas d'utilisation. Par ailleurs, le langage utilisé est essentiel pour assurer le dialogue entre utilisateurs et développeurs. Des structures de

contrôle telles que « tant que » peuvent être utilisées. L'essentiel est de préserver la valeur d'outil de communication des descriptions. Une description du comportement interne du système est un signe de passage à l'exercice d'analyse. Mais il s'agit de ne pas confondre l'expression de besoins par les cas d'utilisation et l'analyse. La première activité permet de dégrossir les fonctionnalités du système grâce à un formalisme extrêmement et volontairement simple. Un développeur professionnel peut douter de l'efficacité des cas. Il faut les replacer dans leur contexte qui est le dialogue avec la communauté des utilisateurs et ne pas oublier l'aspect essentiel des cas d'utilisation en terme de référence constante lors des développements. Ces derniers sont pilotés par les cas d'utilisation.

### 1.3.5 S'approcher du système

Les cas d'utilisation sont une description externe du système. L'activité suivante (toujours dans l'expression de besoins) consiste à s'approcher du système, tout en restant à l'extérieur. C'est la modélisation des interactions sous forme de diagramme de séquences.

Un premier scénario consiste à effectuer une opération telle que  $20 + 32$  ou même  $1 + 2$ . Le diagramme de séquence qui en découle est extrêmement simple mais il permet de s'habituer aux événements externes du système. Il permet aussi de visualiser plus facilement les interactions décrites textuellement dans les cas.

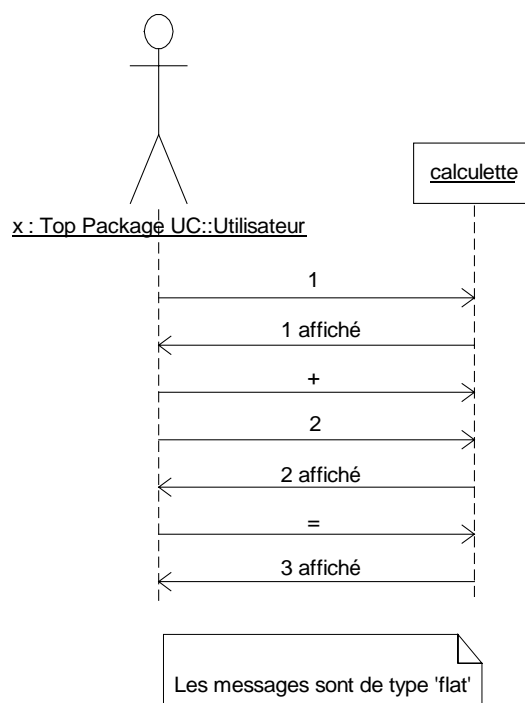


Figure 1-14 : Diagramme de séquence : calcul normal

Le nom de l'acteur est x. Il appartient à la classe d'acteurs « Utilisateur », classe définie dans le paquetage « Top Package UC » imposé par UML. En effet, tout élément de modélisation doit appartenir à un paquetage. Ainsi, chaque modèle est *ipso facto* muni d'un

paquetage racine de tous les autres dans le modèle. Comme tout paquetage celui-ci contient

Des éléments de modélisation, ici acteurs et cas

Des diagrammes, ici diagramme de cas et de séquence essentiellement

Éventuellement des sous-paquetages.

Un deuxième scénario, issu du même cas d'utilisation Calcul permet d'obtenir un nouveau diagramme de séquence. Il s'agit d'un scénario alternatif aussi important que le scénario normal. Quelquefois, la complexité d'un système réside dans ses cas alternatifs essentiellement.

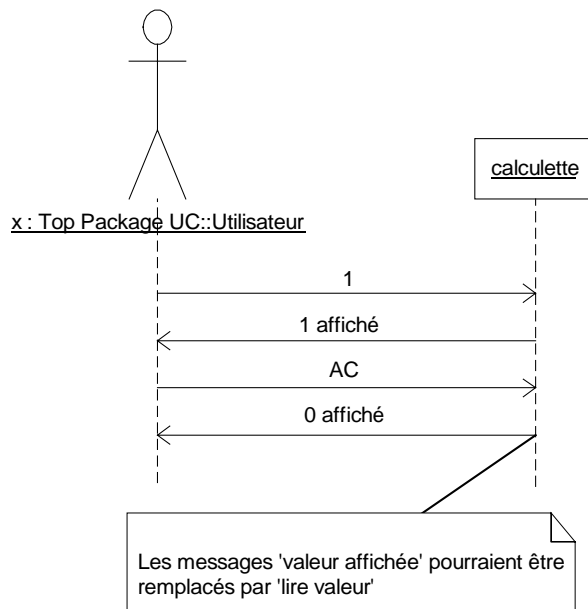


Figure 1-15 : Calcul – cas alternatif d'annulation

Notons qu'une question vient rapidement à l'esprit : combien de diagrammes de séquence dessiner ? Un diagramme quel qu'il soit doit offrir une certaine valeur ajoutée. Si le rédacteur et les lecteurs considèrent que la création puis la lecture de tels diagrammes n'apportent aucune information pertinente alors il est temps d'arrêter et de passer à l'activité suivante. Rien n'est plus désagréable que de créer des diagrammes qui visiblement n'ont aucune valeur ajoutée. De même, il est désagréable de « sauter » des pages entières de document, à la recherche du prochain diagramme qui apportera vraiment une information au lecteur. Si les cas sont correctement rédigés, leur lecture doit pouvoir se traduire directement en diagramme de séquence. Bien entendu, nous évoquons dans ce contexte des cas extrêmement simples de la forme :

L'acteur effectue cela,

Le système répond de telle façon...

N Rappelons tout de même que ces diagrammes sont une aide à l'obtention de diagrammes d'état / transition de  
 O cas d'utilisation ou du système dans son ensemble. En effet chaque transition ne peut survenir que sur un  
 T événement (ou une fin d'activité ou encore une condition qui devient vraie dans le cas de transitions  
 E automatiques). Les diagrammes de séquence visualisent parfaitement les transitions possibles : ce sont les  
 différents segments de la ligne de vie de l'objet considéré. Bien entendu, l'étude de la dynamique retiendra  
 uniquement les transitions jugées pertinentes parmi toutes les transitions au sens changement de valeur  
 d'attribut dans l'objet.

### 1.3.6 Modèle de cas d'utilisation

Le résultat de cette première activité est le modèle de cas d'utilisation, constitué des diagrammes de cas d'utilisation, des descriptions textuelles et des diagrammes de séquence.

Notons que les cas d'utilisation peuvent être visualisés munis de leurs propriétés et contraintes.

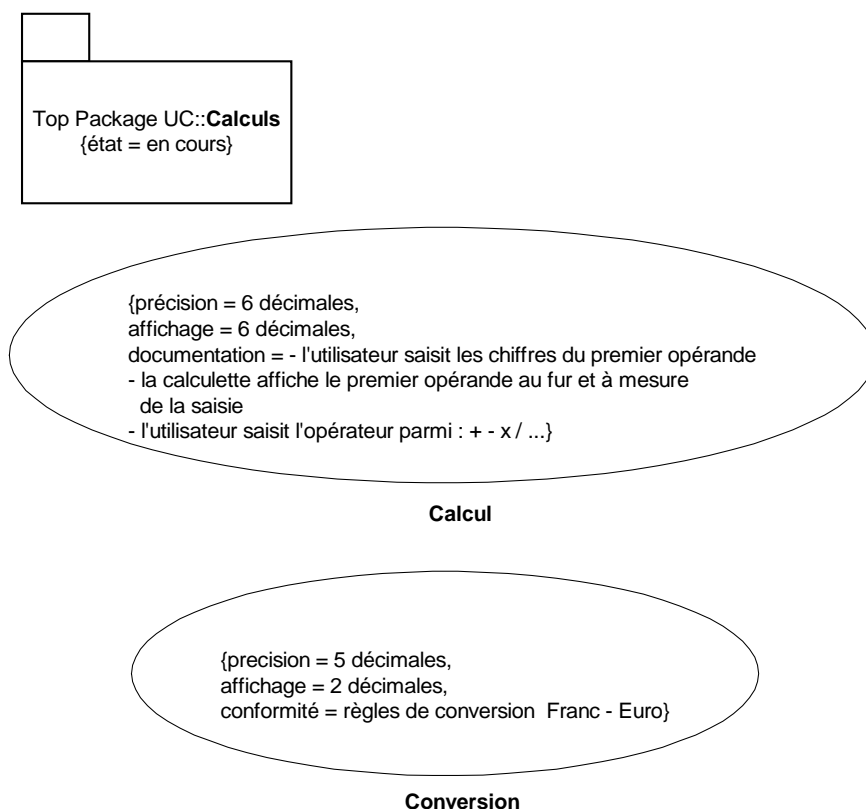


Figure 1-16 : Cas d'utilisation avec propriétés

Ce diagramme visualise de nombreuses informations. Il représente le paquetage de cas d'utilisation nommé « Calculs ». Cela correspond à une certaine organisation des cas d'utilisation en fonctionnalités. Nous le représentons simplement en tant qu'exemple. L'étiquette *état* montre que ce paquetage est en cours de mise au point. Chaque cas d'utilisation possède une étiquette *documentation* utilisée pour renseigner les interactions du cas. Sauf exception, l'ensemble des interactions ne tient matériellement pas dans cet

espace. Il n'est donc pas judicieux de compter sur ce type d'affichage pour documenter facilement le cas. Toutefois, le résumé du cas peut y apparaître. Par contre les étiquettes telles que *précision* ou *affichage* permettent de consigner les besoins non-fonctionnels. L'intérêt étant que ces informations deviennent des éléments de modélisation UML. La relation « trace », correctement annotée, permettra en analyse, conception, programmation, d'associer des éléments de réalisation à ces besoins particuliers.

<b>LE PROJET CALCULETTE.....</b>	<b>2</b>
<b>1.1 L'étude de cas .....</b>	<b>3</b>
1.1.1 Les fonctionnalités de la calculatrice .....	3
1.1.2 Points techniques étudiés .....	3
<b>1.2 Démarche simplifiée.....</b>	<b>4</b>
<b>1.3 Expression de besoins .....</b>	<b>6</b>
1.3.1 Définir les acteurs .....	7
1.3.2 Liste des cas d'utilisation principaux .....	11
1.3.3 Descriptions textuelles des cas d'utilisation.....	12
1.3.4 Vérification des cas d'utilisation.....	18
1.3.5 S'approcher du système .....	19
1.3.6 Modèle de cas d'utilisation .....	21