
2

La calculette

www.thierrycros.net

2

Analyse de la calculette

2.1 Analyse : des cas d'utilisation aux objets

La relation de collaboration entre objets est à la base de cette transformation. En effet, le logiciel est vu comme un système constitué d'objets. Il reste à découvrir ces objets. Chaque cas d'utilisation fournit – via les scénarios - son lot d'objets. La compilation de tous les lots permet d'obtenir le diagramme de classes du système, d'un point de vue analyse. A ce stade, il existe deux familles de stratégies. La première, centrée sur les objets, passe par les diagrammes d'interactions qui permettent autant la représentation que la découverte des objets du système. Nous mettrons en œuvre cette première solution, qui est à notre avis plus proche de « l'orienté objet ». La deuxième, centrée sur les classes, consiste à découvrir les classes du système, par exemple à partir de textes de spécification. Alors, les substantifs sont de bons candidats pour les classes. Ce point particulier est sujet de discussion depuis de nombreuses années.

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

Figure 2-1 : Repérage dans le planning

N
O
T
E

Considérons un exécutable, obtenu à partir d'une programmation C++. Les sources sont les modèles de l'exécutable. Ils contiennent les définitions de classes : types prédéfinis int, float... et types abstraits du programmeur. Or, l'exécutable n'a plus la notion de type au sens du langage. Pour s'en convaincre, il suffit d'étudier les problèmes du langage C dus aux manques de déclaration de fonctions, d'un module à l'autre. Par contre, l'exécutable et donc le process qui en découle possède des objets, instances des classes. Ainsi, les classes sont les supports de la compréhension du système mais n'ont pas de réalité dans le système. Bien entendu, un exécutable contient anecdotiquement des champs de classes. L'approche par les objets est donc plus réaliste, l'approche par les classes plus directement analytique. Avec le risque de ne pas découvrir les classes vraiment nécessaires.

L'analyse précise et structure les besoins. Le résultat est concrétisé par les classes d'analyse, organisées en paquetages. Des diagrammes de la vue dynamique (interaction, état, activité) permettent de préciser les rôles des classes dans le système. Elles sont modélisées par leurs responsabilités et leurs attributs (au sens attribut d'analyse).

2.1.1 Démarche d'obtention des objets

Nous touchons là un aspect essentiel de ce type de développement : la découverte des objets du système. Au plus l'analyste pense objet, au plus il visualise le logiciel comme un écosystème constitué des pseudo « êtres » que sont les objets. Cette étape est particulièrement excitante. L'analyste est explorateur. Des stéréotypes de classes d'analyse, proposés par plusieurs méthodes, sont une aide précieuse. Un peu comme un paléontologue qui recherche sur un terrain des fossiles d'un certain type. Ce périple en *terra incognita* doit amener à comprendre ce qu'est – ou sera - le système à construire. De même que notre paléontologue organise ses recherches, petit à petit, de même nous découvrons les objets que nous regrouperons plus tard en classes puis paquetages. Les cas d'utilisation sont le

point de départ, comme une « carte aux trésors » qui fournit quelques indications et qu'il reste à décrypter.

Pour chaque cas d'utilisation :

1. Identifier les scénarios (normal, exception) pertinents pour la compréhension des interactions acteurs – système, si cela n'a pas été fait dans l'activité précédente.
2. Dessiner le diagramme d'interaction (séquence ou collaboration) qui formalise chaque scénario, ce qui permet de définir au passage les objets d'analyse
3. Enfin, obtenir le diagramme de classes par réunion des collaborations.

Le choix entre diagramme de séquence et diagramme de collaboration est subjectif dans la mesure où ils représentent les mêmes éléments. Le diagramme de collaboration permet toutefois de définir des liens entre objets, même si ce lien n'est pas support d'interaction dans le scénario étudié. Classiquement, les diagrammes de séquence sont utilisés, en activité d'expression de besoins, pour représenter les échanges entre acteurs et système, ce dernier étant vu comme un objet opaque. Les diagrammes de collaboration sont ensuite dessinés pour montrer les objets découverts *dans* le système. Le diagramme de collaboration met en exergue les objets du système. Les diagrammes de séquence sont généralement plus faciles à obtenir dans la mesure où la séquentialité des interactions découle de la description textuelle des cas.

2.1.2 Cas d'utilisation "Calcul"

Ce cas d'utilisation induit une infinité de scénarios normaux : infinité d'additions, de multiplications, etc. Toutefois, un scénario mettant en jeu une opération représente tous les autres. Le cas particulier de la division par zéro pourrait aussi être traité comme scénario « limite » ou « exceptionnel ». Ici, il ne faut pas hésiter à commencer par un scénario très simple. Notre objectif est la découverte des objets, pas la compréhension immédiate et complète du système. Alors pourquoi démarrer par le plus compliqué ?

Premier scénario normal : $1 + 2 = 3$. Il n'est pas interdit d'écrire le scénario. Il est de toute façon nécessaire pour dessiner le diagramme d'interaction, un danger étant de commencer un tel diagramme sans idée précise du scénario à modéliser. Par ailleurs, en analyse (également en conception), le texte du scénario est un guide de lecture et de compréhension du diagramme.

N Le terme « scénario » renvoie au cinéma. De ce point de vue, un analyste est un auteur. Auteur car il doit
 O inventer les personnages, autrement dit les objets en présence dans le système. Ce dernier devient le décor.
 T Les stéréotypes d'analyse sont autant de type de personnages. L'analyste crée ensuite les dialogues entre
 E personnages, c'est ce qui forme peu à peu le rôle des objets.

Par opposition, l'architecte / concepteur est en quelque sorte le metteur en scène. Il adapte les scénarios aux moyens techniques dont il dispose. Les décors imaginaires de l'analyse se transforment alors en plateau de cinéma bien réel. Le metteur en scène doit composer avec les possibilités *réelles* de ses comédiens, de même qu'un concepteur joue avec les bibliothèques, les composants tiers tels que brokers, serveurs, langages.

Cette analogie simpliste a pour but de montrer la différence entre analyse et conception. L'analyste évolue dans un monde idéal, de même qu'un auteur, sur le papier, invente toute sorte de scène. L'analyste crée des objets de même que l'auteur crée des personnages. Leur personnalité correspond aux responsabilités des objets d'analyse. Le concepteur au contraire compose avec la réalité et ses limitations, son objectif étant de restituer l'analyse du système. De même qu'un comédien incarne un personnage, de même un objet de conception implémente un objet idéal d'analyse.

En résumé, les objets d'analyse ne sont pas des objets réels au sens instantiation en Java par exemple. Il s'agit plutôt de cerner les contours de *types* qui seront plus tard réifiés en classes. Par abus de langage, nous employons le terme classe d'analyse alors que « type » est plus approprié. D'ailleurs, cette activité insiste sur les responsabilités des objets, pas sur leurs opérations.

Pour fixer les idées, nous dirons qu'un type au sens utile dans une activité d'analyse correspond d'une part à un ensemble de responsabilités, d'autre part à un ensemble d'attributs de haut niveau sémantique, qui eux-mêmes sont de type d'analyse, à opposer aux types primitifs des langages de programmation. Par exemple, un « type » Document dans une application Traitement de texte possède des responsabilités telles que gestion de sa mise en page, etc. Il possède également des attributs tels que « pied de page ». Il restera par la suite à implémenter cela, par exemple sous forme de classe « pied de page » ou bien sous forme de référence, en fonction des possibilités et contraintes de l'architecture.

Le scénario sur lequel nous travaillons est le suivant.

```
L'utilisateur saisit le premier opérande 1,
La calculette affiche cet opérande au fur et à mesure de la saisie
...
L'utilisateur demande le résultat en appuyant sur « = ».
La calculette affiche 3.
```

Le diagramme de séquence objet_acteur / objet_système ont permis de mieux appréhender le système, sans, pour l'instant, y entrer. Ces diagrammes sont typiques de modèles de cas d'utilisation. Les diagrammes de collaboration posent plus directement la question des objets en présence.

Diagramme de Collaboration du premier cas d'utilisation

Ce premier scénario est visualisé par un diagramme de collaboration qui représente les objets en présence dans le système ainsi que les interactions entre objets.

Il s'agit de déterminer - à partir du scénario - les objets Limite, Contrôle et Entité qui collaborent. A ce stade, nous *analysons* le système. Autrement dit, les objets sont idéaux et n'ont pas à subir de contraintes de mise en œuvre. Nous réservons cela aux phases suivantes.

Limite ou Frontière (Boundary)

Pour que l'utilisateur puisse saisir les chiffres, les caractères tels que "=", il faut un clavier. Pour que la calculatrice affiche les nombres (opérandes, résultats), il faut un afficheur, nous choisissons le nom de *cadran*.

N Les objets de type Limite (stéréotype issu de Unified Process) ne doivent pas être confondus avec les éléments
O « interface » au sens du stéréotype UML. En effet, une interface est une vue d'un classificateur (classe, sous-
T système). C'est un ensemble d'opérations qui précise certains services rendus par le classificateur. Ce
E mécanisme de vue est utilisé par exemple en base de données relationnelle pour interfacier une ou plusieurs tables. Une interface se distingue d'une autre par la qualité, le potentiel des opérations qui la composent. Pour une classe donnée, une interface peut être constituée uniquement d'opérations « sélecteur » (requêtes, accesseurs qui ne modifient pas l'état – les attributs – de l'objet). Une autre interface pourra comporter des opérations de modification. Par ailleurs, un sous-système possède des interfaces qui sont réalisées par des classes. En dernière analyse, les classes de stéréotype Limite sont les interfaces du système. Elles pourraient correspondre à « interface acteur ».

Contrôle

Pour séquencer les échanges entre les objets, nous décidons de confier la supervision à un objet que nous nommons *contrôleCalcul*. En effet, le traitement des saisies suit une séquence bien précise qu'il s'agit d'ordonner, ne serait-ce que pour contrôler la validité des demandes de l'utilisateur. Ceci étant, le contrôle est directement lié aux saisies de l'acteur. Dans ce cas, il pourrait être confié à l'objet limite. Ici, nous considérons que le séquençage est plus une question de domaine.

Ces objets *contrôleur* ne se laissent pas découvrir facilement. En effet, les limites et les entités sont généralement visibles de loin dans notre système. Les limites sont à la frontière du système : entrées et sorties d'informations. A l'opposé les entités correspondent souvent aux substituts du système réel, issus du modèle de domaine quand il existe.

Les contrôleurs ne doivent pas être confondus avec les médiateurs de conception (les objets en charge de la supervision d'interactions entre objets de conception). En effet, nous sommes en activité d'analyse. Ces objets ont donc pour but de modéliser ce que doit être le système, ce qu'il doit automatiser. Prenons l'exemple - très connu - d'un distributeur de billets. Ce système possède visiblement un objet limite qui est le guichet automatique. Le client qui souhaite effectuer un retrait correspond très certainement à un objet entité qui est son compte courant. Mais imaginons le déroulement de l'opération de retrait. Le séquençage des opérations telles que vérification du code secret, du montant autorisé par rapport à la carte, par rapport au compte, constitue en quelque sorte une connaissance métier, pas un algorithme uniquement informatique. Ces éléments particuliers deviennent les contrôleurs. Certains objets sont passifs (les entités). D'autres sont extrêmement actifs : les contrôleurs. Cela n'enlève rien à la qualité de ces objets, en terme d'état, de comportement. Souvent ces éléments sont même typiques de ce qu'est l'approche objet. Ils sont actifs, dynamiques et agissent en fonction de leur état à l'instant t. C'est tout ce que l'on demande à un objet.

Entité

Ici nous ne découvrons pas *a priori* d'entité.

Les objets métier deviennent des contrôleurs ou entités.

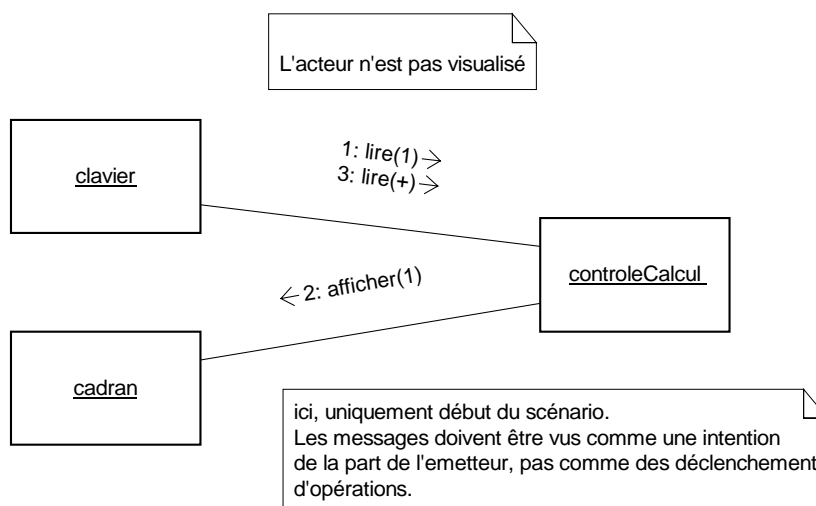


Figure 2-2 : Découverte des objets d'analyse

Ce diagramme formalise uniquement le début du scénario, afin de ne pas le surcharger inutilement. Dans un premier temps, seuls les objets sont représentés. Le dessin est alors un diagramme d'objets, spécifique au scénario étudié.

Les messages envoyés aux objets ont des labels qui représentent ce que l'émetteur souhaite voir effectuer par l'objet destinataire. De façon générale, les messages destinés à un objet destinataire comportent deux types d'informations : le nom du message qui est une demande ou un stimulus envoyé au destinataire et éventuellement un flot de données qui correspond aux échanges entre les objets. A ce stade de l'étude, les noms de messages sont des souhaits ou vœux pieux : ils ne correspondent pas obligatoirement à des noms d'opérations déclenchées dans les objets. Autrement dit, il s'agit de raisonner plutôt en terme de contrat passé avec les objets plutôt qu'en terme de nom de méthode-procédure déclenchée dans l'objet. Ce point de vue permet d'obtenir les responsabilités des objets – les contrats qu'ils doivent respecter – et donc des classes auxquelles ils appartiennent. Plus tard (en conception), les responsabilités ou contrats deviennent des opérations et éventuellement des attributs. Ces responsabilités des objets définissent leurs comportements. La réunion de différents diagrammes de collaboration permet de regrouper les messages reçus par tel ou tel objet. A partir de là, une liste de responsabilités de l'objet peut être établie.

Le premier cas d'utilisation suppose deux scénarios supplémentaires :

- Cas d'erreur division par zéro
- Cas d'annulation.

Ces deux scénarios peuvent être injectés dans un diagramme de collaboration sous plusieurs formes. Il est possible de modifier le diagramme précédent afin qu'il modélise également cette contrainte. Il suffit d'introduire la condition de garde devant le message concerné. Bien entendu, ils peuvent se traduire aussi par des diagrammes spécifiques.

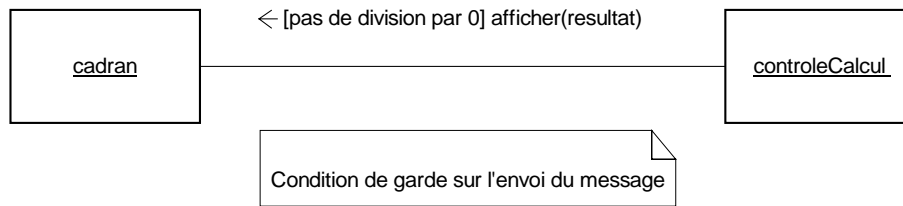


Figure 2-3 : Cas « Calcul » scénario exceptionnel

Dans ce cas, le message `afficher(résultat)` est envoyé uniquement si la condition de garde : *pas de division par 0*, autrement dit, opérateur égal à « / » et opérande 2 différent de 0 est satisfaite.

Le scénario correspondant à l'exception « annulation » met en scène les mêmes objets. Par contre, il fait apparaître des messages spécifiques, tels que `afficher(0)` envoyé depuis `contrôleCalcul`.

Il existe enfin un moyen plus informel mais très simple qui consiste à annoter le diagramme, au moyen du mécanisme général de notes. L'annotation permet d'enrichir les diagrammes sans se préoccuper en avance de phase de détails de mise en œuvre. La règle étant qu'une notation UML spécifique est préférable à une note car elle *visualise* alors que la note commente par du texte.

2.1.3 Cas d'utilisation "Conversion"

Ces deux cas d'utilisation (Calcul et Conversion) se traduisent en scénarios et donc sont modélisés au travers de diagrammes d'interaction. Cela n'est pas une obligation. La question est : quelle est la valeur ajoutée de tel ou tel diagramme ? Nous décidons de traiter uniquement les scénarios normaux dans la mesure où le scénario d'annulation d'un cas d'utilisation précédent correspond probablement à toutes les situations d'annulation. Bien entendu, ce choix est subjectif. La suite de l'analyse du système permettra de valider (ou non) son bien-fondé.

La question de base : quels sont les objets limite, contrôle, entité collaborateurs est la même pour tous les scénarios d'analyse. Au fur et à mesure, l'ensemble d'objets devient plus complet. Ils sont alors réutilisés de diagrammes en diagrammes. Ainsi, certaines interactions permettent de découvrir des objets d'analyse, d'autres uniquement des messages à envoyer et donc recevoir. Un message reçu par un objet correspond à une responsabilité qu'il doit assumer dans le système. Les interactions issues du scénario nous amènent à créer l'objet `contrôleConversion`.

```

L'Utilisateur saisit 1
La calculette affiche 1
L'utilisateur saisit F
La calculette affiche le résultat de la conversion en Francs : 6,56.
  
```

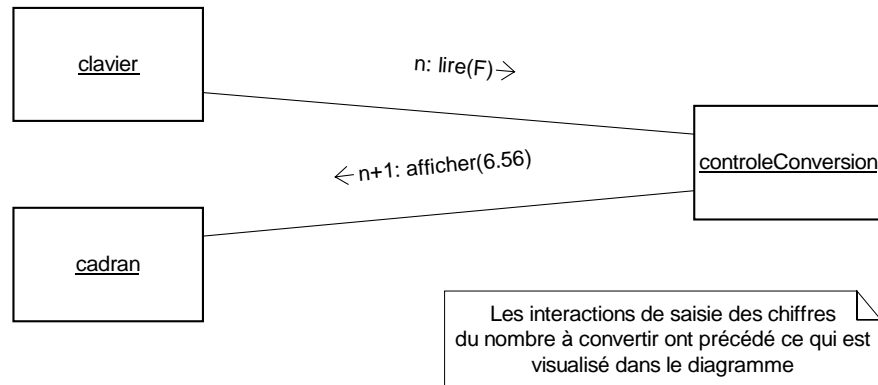


Figure 2-4 : Conversion, scénario normal

Ce diagramme nous apprend que l'objet `contrôleConversion` calcule la conversion, sur réception du message `F` et qu'il demande ensuite à l'objet `cadran` d'afficher ce résultat. Pour simplifier, nous supposons que le taux est correctement initialisé afin d'éviter le cas d'une demande de conversion alors qu'il n'est pas encore saisi. Cette situation pourrait faire l'objet d'un diagramme spécifique, autrement dit d'une étude particulière.

Il est temps de consigner ce que l'on a appris dans un diagramme de classes partiel et provisoire. Partiel car tous les cas d'utilisation n'ont pas été traités, provisoire car une approche objet est un va et vient continu entre la vue dynamique (diagrammes d'interactions dans ce cas) et structurelle (diagramme de classes).

Ici, chaque objet correspond *a priori* à une classe d'analyse car nous découvrons les responsabilités au moyen des objets issus des collaborations. Visiblement il ne s'agit pas de créer par exemple des objets « entité » dans différentes interactions. Dans un premier temps, nous obtenons un diagramme de classes sans relations, le but étant de cerner les responsabilités des objets des classes. Mis à part le cas particulier de la généralisation, les relations entre classes correspondent aux collaborations entre objets.

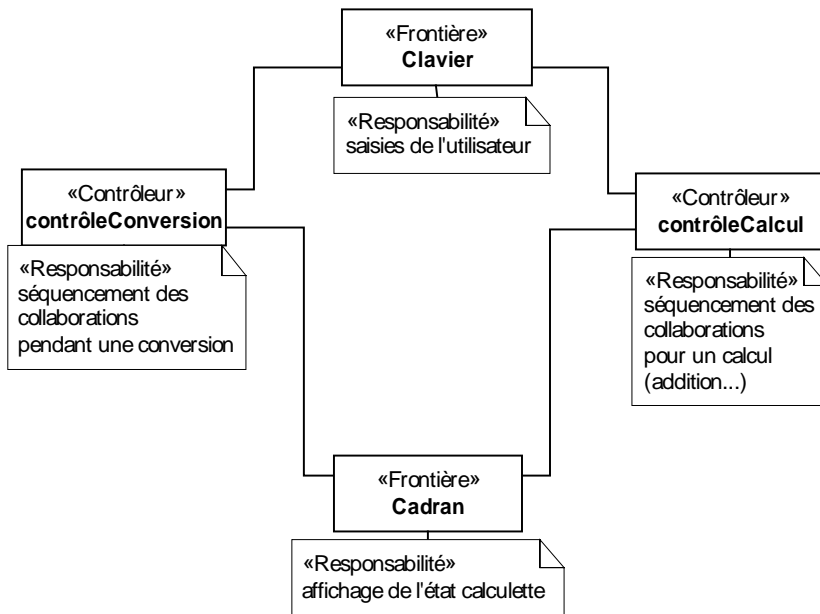


Figure 2-5 : Diagramme de classes d'analyse provisoire

Ce diagramme visualise les classes obtenues à partir des diagrammes de collaboration et leurs responsabilités dans le système. L'AGL utilisé ne supporte pas le compartiment « responsabilité » des classes, apparu dans UML version 1.3. Par chance, ce compartiment est relativement informel : son contenu est du texte dont le format est laissé à l'appréciation de l'analyste. Une note stéréotypée convient donc pour le visualiser. Une autre possibilité est d'utiliser une propriété de la classe.

Examinons de plus près ces responsabilités. Les classes (à prendre au sens analyse, autrement dit type plutôt que classe de réalisation) Clavier et Cadran assument des responsabilités relativement proches. Toutefois, nous préférons les séparer car il s'agit bien d'une part d'un objet miroir (cadran) et d'autre part d'un objet de saisie. Plus tard en conception nous pourrions les rassembler sous forme d'une classe qui assumerait les deux rôles en implémentant deux interfaces (au sens UML) spécifiques. Les classes `contrôleConversion` et `contrôleCalcul` assument également des rôles proches. Étant donné que tous les cas d'utilisation impliquant les conversions (cas dans lesquels les objets `contrôleConversion` et `contrôleCalcul` peuvent a priori collaborer) ne sont pas traités, nous retardons l'étude du rapprochement des deux classes.

Les relations entre cas d'utilisation et classes d'analyse sont traduites sous forme de relation « trace ». Cela permet d'assurer qu'une classe est injectée en activité d'analyse car elle correspond à un besoin effectivement exprimé dans un cas d'utilisation.

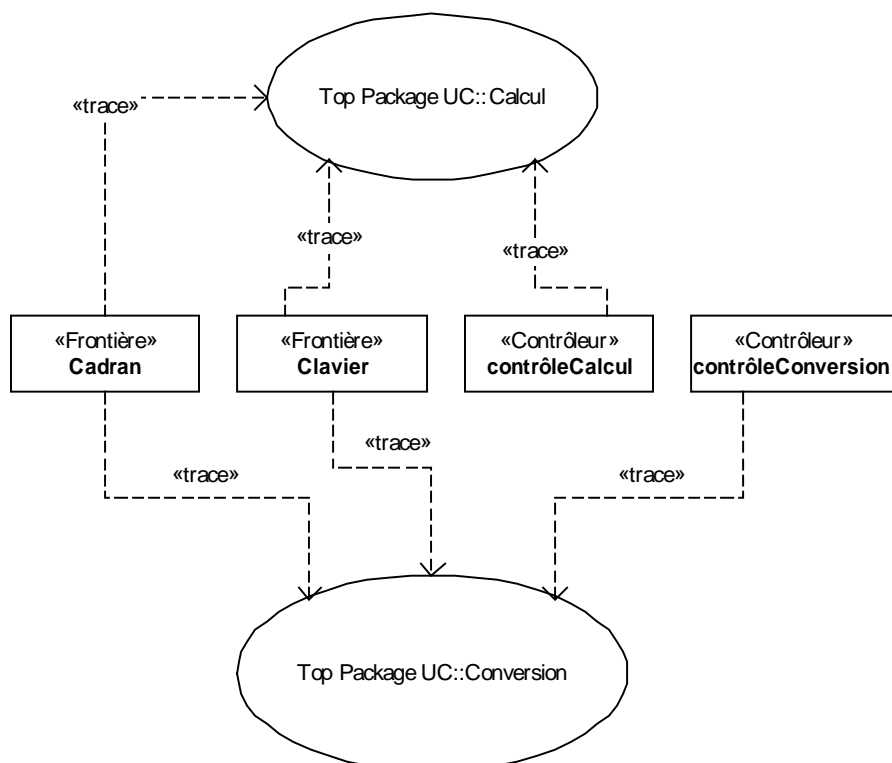


Figure 2-6 : Relation « trace » entre cas d'utilisation et classes d'analyse

En fonction du nombre d'éléments à relier (cas d'utilisation, classes d'analyse), ces informations sont modélisées par quelques diagrammes de traçabilité et des tableaux. En production ou en contrôle (qualité) ces artefacts assurent que les cas d'utilisation pilotent effectivement les développements.

Classe	Calcul	Conversion
Cadran	+	+
Clavier	+	+
ContrôleCalcul	+	
ContrôleConversion		+

Figure 2-7 : Matrice de traçabilité cas / classe

2.1.4 Cas d'utilisation "Taux"

Pour terminer les réalisations de cas d'utilisation en classes d'analyse, nous traitons le cas nommé Taux qui a pour but de fournir le taux de conversion à la calculette. Deux scénarios sont induits par ce cas : scénario normal de saisie du taux et scénario d'annulation.

Le scénario normal de saisie est simplement :

L'utilisateur saisit 6,55957

La calculatrice l'affiche au fur et à mesure de la saisie des chiffres
 L'utilisateur valide le taux en appuyant sur T (d'après la maquette)
 La calculatrice affiche toujours le taux.

En toute rigueur, les objets collaborants sont à nouveau le clavier et le cadran et nous créons un objet `contrôleTaux` pour prendre en charge le stockage du taux, de même que nous avons créé un objet contrôleur pour prendre en charge les interactions de chaque cas d'utilisation, ce qui est une technique de base pour les découvrir. Ces premiers objets devront être refondus lors de la mise au point des paquetages et des classes définitives d'analyse.

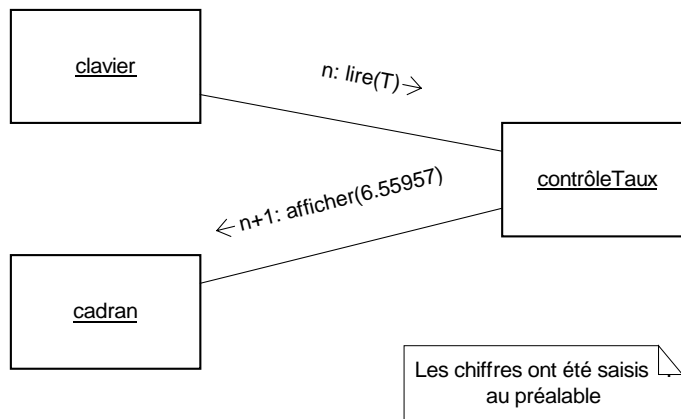


Figure 2-8 : Collaboration : Taux – scénario normal

Le diagramme partiel d'analyse obtenu à partir de ce scénario est le suivant.

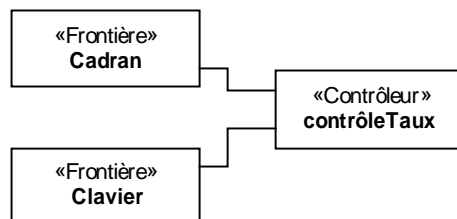


Figure 2-9 : Diagramme de classes partiel issu du cas « Taux »

2.1.5 Diagramme de classes d'analyse

Le diagramme de classes issu de l'étude des réalisations de cas d'utilisation est obtenu par réunion ou compilation des différents diagrammes de classes partiels. C'est le point de départ de la structuration du système en termes de paquetages de classes d'analyse. A ce stade, les considérations telles que évolutivité ou réutilisation entre en jeu pour préciser les besoins non plus en tant que fonctionnalités mais comme contraintes de l'architecture.

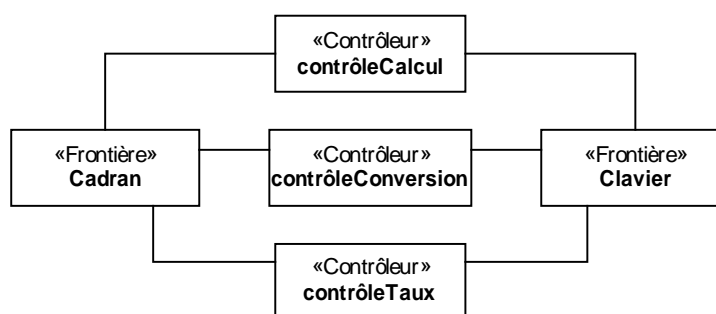


Figure 2–10 : Diagramme de classe d'analyse complet et provisoire

Ce diagramme est complet : il reprend toutes les classes découvertes dans les collaborations. Il est provisoire car il s'agit maintenant d'affiner les responsabilités et éventuellement de créer les paquetages. « Eventuellement » car les classes sont peu nombreuses. Un paquetage doit avoir une finalité : organisation d'éléments ou consigne d'architecture. Ici, le nombre d'éléments est restreint, un paquetage supposera une exigence en terme de découpage en sous-systèmes en activité de conception.

La classe `contrôleTaux` possède une responsabilité qui sous-entend la mémorisation du taux de conversion. Elle est donc modélisée par ses responsabilités et un attribut d'analyse.

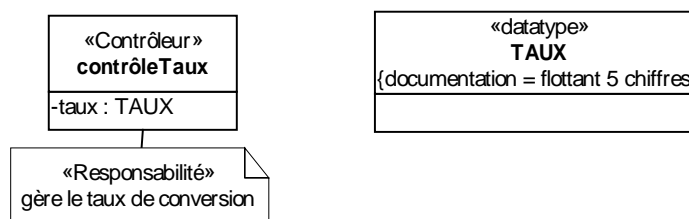


Figure 2–11 : Classe `contrôleTaux`

La classe `TAUX` est en fait un datatype. Cet élément de modélisation permet de définir les types des attributs de classes lorsqu'ils ne sont pas primitifs. C'est un compromis entre type primitif et type abstrait au sens de classe. Les datatypes d'analyse devront être implémentés en conception en fonction des possibilités des environnements. Ils ne figurent pas nécessairement dans les diagrammes de classes d'analyse mais sont rassemblés dans des diagrammes de datatypes communs. D'une certaine manière ces classes particulières créent le dictionnaire des types des attributs. « Datatype » est un élément du méta modèle UML. Un stéréotype permet de le visualiser grâce à la notation de classe.

La classe `contrôleTaux` mémorise désormais le taux en tant qu'attribut de type `TAUX`. Elle pourra être convertie en « entité ». Il est temps désormais d'affiner ce diagramme de classes, c'est-à-dire de préciser ce que l'on attend des classes du système en termes de responsabilités, puis de paquetages.

Les classes `contrôleConversion` et `contrôleTaux` sont liées. Les responsabilités très simples et proches peuvent sans problème être réunies. La classe `contrôleTaux` n'a

strictement aucun intérêt sans la classe `contrôleConversion`. Réunir les deux classes supposent que nous ne prévoyons absolument aucun usage spécifique de `contrôleTaux` hors son lien avec `contrôleConversion`. Visiblement, nous avons affaire à une seule classe que nous appellerons encore `contrôleConversion`. La même question se pose pour les classes `contrôleCalcul` et `contrôleConversion` nouvelle version. En effet, les conversions peuvent être considérées comme des calculs particuliers, si ce n'est qu'ils supposent l'existence du taux de conversion. Il s'agit maintenant de vérifier la cohérence des responsabilités de la classe `contrôleCalcul`.

Nous obtenons *in fine* le diagramme de classes d'analyse suivant.

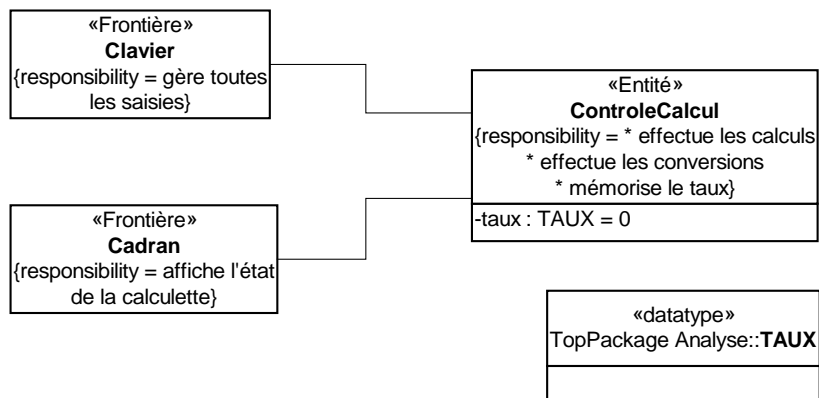


Figure 2-12 : Classes d'analyse, diagramme provisoire

Ce diagramme visualise les responsabilités des classes sous forme de propriétés, ce qui est une variante du compartiment « responsabilité » sous les opérations.

2.1.6 Paquetages d'analyse

2.1.6.1 Obtenir les paquetages

Les relations entre classes n'ont pas été *précisées* dans le diagramme de classes. Les associations actuellement représentées ne sont pas définies clairement. Mais préoccupons-nous des besoins en termes d'utilisation conjointe de classes, d'impacts d'éventuels changements. Cela permettra de créer peut-être un (ou plusieurs) paquetage d'analyse, ce qui peut modifier la nature des relations entre classes, par exemple en imposant plus d'indépendance. Imaginons donc quelques évolutions possibles (même si elles ne sont pas envisagées aujourd'hui) ou bien quelques exemples de réutilisation de tel ou tel élément. Ce type de tableau « matrice des impacts », est une excellente aide à la détermination des paquetages et simplement des responsabilités de classes. Il est décrit plusieurs fois dans l'analyse, soit en amont afin de préciser les besoins de ce type, soit en aval afin de vérifier les impacts du changement ou de la réutilisation sur l'analyse telle qu'elle est réalisée.

Les impacts des évolutions sont notés sous la forme :

- +++ impact fort
- + impact faible

- aucun impact.

Id	Changement	Clavier	Cadran	ContrôleCalcul
A	Changement d'interface utilisateur, alphanumérique (pour tester l'architecture) puis IHM graphique ou bien portage d'Unix à Windows ou inversement	+++	+++	+
B	Changement de bibliothèque de calcul, possibilité d'utiliser des commandes système sous Unix (par exemple liaison par « pipe » avec la commande bc)	-	-	+++
C	Modifier le séquençement, passer d'opération 1+2= à des opérations plus générales telles que 1+2+3=	-	-	+++
D	Ajouter des fonctionnalités telles que mémoire, fonction trigonométrique, type d'affichage (hexadécimal...)	+++	+++	+++

Figure 2-13 : Matrice « impacts des évolutions et réutilisations »

Ce tableau fournit de nombreuses informations essentielles pour l'analyse et donc pour l'architecture du système en terme de sous-systèmes.

Ligne A

Le changement d'IHM impacte aussi bien le clavier que l'écran, ce qui est immédiat. Mais pourquoi ne pas imaginer dans certains projets des changements de miroirs (cadran) et pas de saisies ? Ici, l'impact faible sur `contrôleCalcul` peut être une exigence. En effet, le contrôle des calculs au sens interactions entre objets clavier, cadran... est dans ce cas très lié aux actions de l'utilisateur donc à l'IHM. Ce type d'exigence impose donc un découplage fort dans la mesure où ce contrôle particulier du logiciel sera très lié au style d'interface utilisateur. Ce n'est pas le cas pour toutes les classes de stéréotype « contrôle ». Les IHM graphiques supposent une programmation événementielle alors que les IHM caractère ou alphanumériques supposent une programmation séquentielle. Si cette exigence est conservée, les concepteurs devront en tenir compte dans leurs travaux.

Une conséquence immédiate de cette exigence est la création d'un paquetage IHM. Cette catégorie de classes apparaît habituellement au niveau du découpage « en responsabilités » des classes (ihm, métier...). Ici, le paquetage est rendu nécessaire dans la mesure où ces deux classes sont réutilisées systématiquement ensemble. C'est un critère d'empaquetage au moins aussi important. Des classes réutilisées ensemble appartiennent certainement à la même catégorie ; au contraire des classes cohérentes d'un point de vue fonctionnel ne sont pas toujours regroupées. Les classes d'analyse dans un même paquetage ont une devise :

« Nous sommes réutilisés ensemble, nous changeons ensemble, nous partageons une même fonctionnalité ».

L'organisation provisoire des classes obtenue par étude du changement ligne A est donc basée sur un paquetage IHM. Appelons cette solution : ihm-appli car elle correspond à une couche spécifiquement applicative.

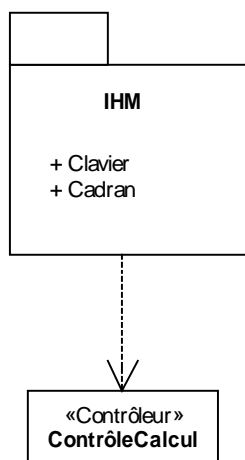


Figure 2-14 : Paquetage d'analyse souhaité et provisoire

Ce diagramme visualise une relation de dépendance depuis le paquetage IHM vers la classe `ContrôleCalcul`. La dépendance inverse n'est pas représentée dans le diagramme de paquetage, ce qui évite un cycle. Ainsi, la relation entre `ContrôleCalcul` et `Cadran`, relation dans laquelle la première envoie un message de type `afficher()` à la deuxième doit subir une évolution. Ce point reste en suspens.

Examinons plus avant les conséquences des évolutions de la matrice des impacts.

Ligne B

Changement de réalisation des calculs. Les impacts des deux premières lignes A et B sont compatibles avec le diagramme ci-dessus. Afin d'équilibrer les éléments, nous injectons la classe isolée dans un paquetage. Cela permet de gérer par la suite des relations inter-paquetages uniquement ou bien intra-paquetages.

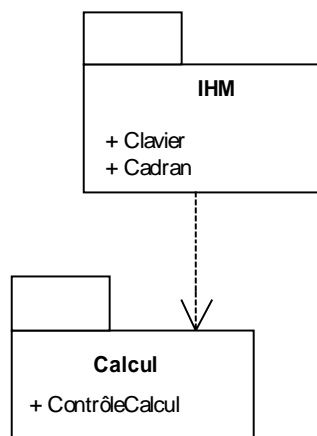


Figure 2-15 : Paquetages d'analyse provisoire

Ligne C

Modifier le séquencement impacte visiblement le contrôle de la calculatrice. Le rapprochement des évolutions possibles dénote des responsabilités de type contrôle d'une part et de type calcul d'autre part. En effet, modifier plus facilement la mise en œuvre des calculs n'a pas d'impact sur le contrôle. D'où l'idée de créer au final deux classes dans le paquetage Calcul : `ContrôleCalcul` et `Calculateur`. La classe `Calculateur` représente essentiellement la responsabilité du calcul lui-même que l'on souhaite implémenter de plusieurs façons, ce qui est conforme aussi à la ligne B.

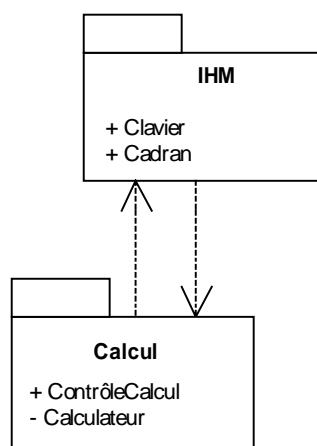


Figure 2-16 : Paquetages d'analyse réels

La classe `Calculateur` est déclarée privée (`-Calculateur`). Elle correspond donc théoriquement à l'implémentation du paquetage. Mais est-ce exact ? La visibilité privée implique une impossibilité d'accès depuis l'extérieur du paquetage. C'est encore une question que nous laissons en suspens. Pourquoi ne pas imaginer une utilisation dans laquelle un nouveau paquetage accède directement aux services de calcul de la classe `Calculateur` ? Par ailleurs, le placement des paquetages correspond à une disposition en

couches, la couche du dessus est cliente de la couche du dessous qui joue alors le rôle de serveur (le cycle entre les deux paquetages restant à éliminer).

Ligne D

Ce type d'évolution impacte l'ensemble des classes : cela doit correspondre à une évolution majeure des fonctionnalités. La question est alors : l'évolution à l'origine est-elle vraiment majeure, est-il normal de compter autant de classes impliquées dans le changement ? Ici, un ajout d'opération suppose naturellement une évolution de l'interface acteur, une évolution également du calcul lui-même.

Il est temps de reprendre les réalisations de cas d'utilisation à partir de ces nouvelles classes pour tenter de mieux cerner les relations entre les catégories qui sont la traduction de relations entre les classes des catégories. Juste un dernier point par rapport à l'aspect structurel. Visualisons le contenu de chaque paquetage au moyen de diagramme de classes.

N L'utilisation d'un AGL doit permettre de déplacer rapidement les classes d'un paquetage à l'autre. Ici, les
 O classes ont été injectées au départ dans le paquetage « Top Package » du modèle d'analyse. Les paquetages
 T d'analyse ont été créés par la suite. Les opérations de type Drag & Drop des outils sont extrêmement
 E pratiques. En effet, elles permettent de transférer rapidement une classe d'un paquetage à l'autre.

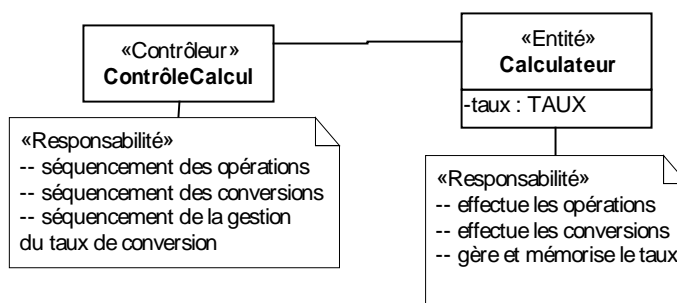


Figure 2-17 : Paquetage Calcul

La classe Calculateur est de type « entité » car elle possède un état constitué d'un attribut à mémoriser.

Le paquetage IHM possède deux classes qui ne sont pas liées a priori.

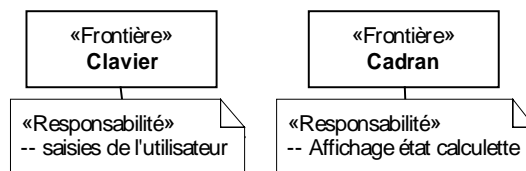


Figure 2-18 : Paquetage IHM

2.1.6.2 Relations inter et intra paquetages

Nous avons étudié le système à partir de sa dynamique, au moyen de collaborations entre objets. L'aspect statique ou structurel est ensuite développé pour aboutir à un ensemble de paquetages. La connaissance du logiciel augmente en spirale : dynamique et structurel sont deux points de vue sur le même système, ils se complètent et s'enrichissent mutuellement.

La relation générale de dépendance entre `IHM` et `Calcul` suppose des interactions entre classes des deux paquetages. Reprenons un scénario afin de visualiser les conséquences des évolutions du modèle via l'aspect structurel. Une traduction directe d'un diagramme de collaboration précédent nous permet de retrouver les messages tels que `afficher()`. Le scénario repris est celui d'une conversion telle que 1 euro en Francs (conversion d'un euro en franc), le résultat étant 6,55957.

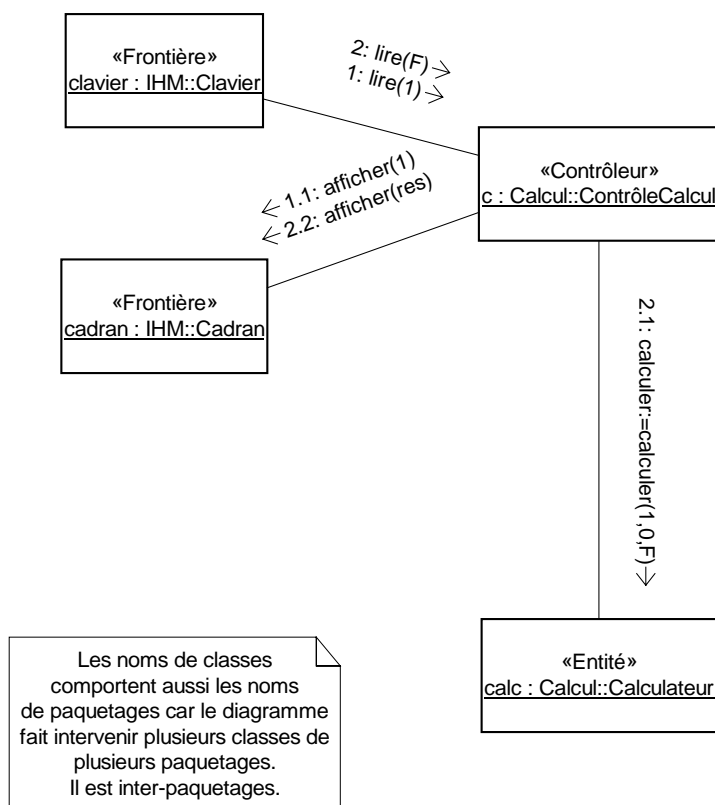


Figure 2–19 : Conversion – scénario normal avec objets d'analyse organisés en paquetage

Un diagramme d'interaction permet de visualiser des objets collaborants issus de paquetages distincts. Le chemin d'accès complet est alors indiqué.

Ce diagramme de collaboration typique des interactions entre les objets montre que

les objets du paquetage `IHM` ne dialoguent pas entre eux

les objets du paquetage `Calcul` interagissent : la classe `ContrôleCalcul` voit la classe `Calculateur`

l'objet `clavier` envoie des messages à l'objet `c:ContrôleCalcul` alors que celui-ci envoie des messages à l'objet `cadran`.

Il nous faut donc étudier ces couplages entre classes d'analyse. Les relations entre classes d'un même paquetage ne posent pas de gros problème en analyse, il faudra les traiter en

conception sous forme d'interface offerte par la classe `Calculateur` par exemple. Par contre, les relations inter paquetages doivent être résolues. La solution consiste à modifier sensiblement l'analyse. Nous allons donc étudier différents moyens pour supprimer le cycle entre les deux paquetages. Il faut savoir que chaque solution correspond à un certain effort de conception et de programmation. La décision à prendre est un compromis entre souhaits d'analyse et possibilités de conception / implémentation.

Les solutions examinées maintenant représentent le travail réel de l'analyste lorsqu'il regroupe les éléments de modélisation (les classes) en paquetages. Les intervenants extérieurs à l'activité, les concepteurs par exemple, ne découvrent que le résultat final de cette étude qui est le modèle d'analyse.

Une première solution consiste à réunir les classes fortement couplées à l'intérieur d'un même paquetage.

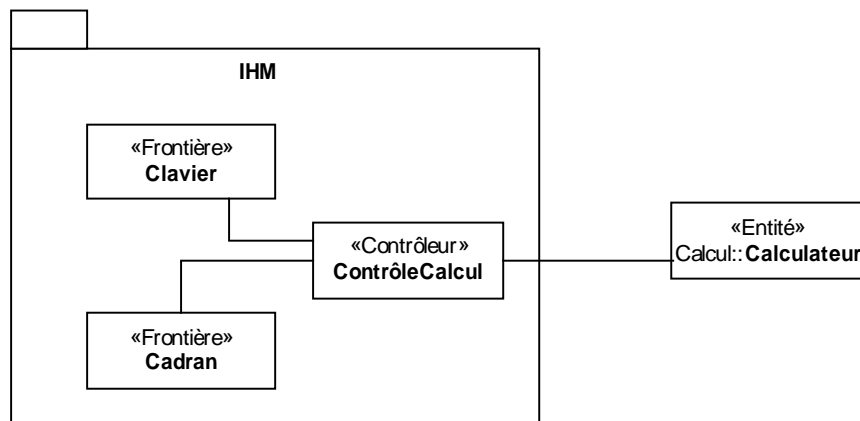


Figure 2–20 : Une solution pour éliminer le cycle entre paquetages

Le changement d'état de la calculette, probablement de l'objet de contrôle, intervient systématiquement après envoi de message par l'objet `clavier`. Ce partitionnement des classes isole la classe `Calculateur`. Effectivement, quel que soit le contrôle, le calculateur est réutilisable, c'est ce que nous avons noté plus haut. Une conséquence de ce partitionnement est le découplage des classes `ContrôleCalcul` et `Calculateur`. Or, ces deux classes peuvent être considérées comme abstraction du « métier ». Cette solution remet donc en cause sensiblement l'approche initiale.

Autre solution, isoler l'élément qui pose problème, en l'occurrence l'objet miroir de la calculette qui est le cadran. Cela impose la création d'une catégorie spécifique pour la classe `Cadran`. Le nom de cette solution est le nom du nouveau paquetage introduit : « Affichage ».

D'autres solutions remettent en cause les relations entre classes (qui n'ont pas changé jusqu'à présent). Ainsi, le clavier prend en charge une responsabilité supplémentaire qui consiste à dialoguer seul avec l'objet de contrôle du calculateur. Une collaboration typique consiste à recevoir en retour de chaque message envoyé l'état du contrôleur, le clavier soustraite alors l'affichage au cadran, qu'il doit désormais connaître. Le nom de cette solution est : « assos-cadran » car elle est basée sur une nouvelle association entre clavier et cadran.

Ce scénario débute alors que les opérandes et l'opérateur ont été saisis. L'utilisateur appuie sur "=". L'objet clavier envoie un message à l'objet contrôleur pour lui demander de traiter. Cet objet propage ce message au calculateur. En retour le calculateur transmet le résultat que le contrôleur renvoie au clavier. Le clavier reçoit ce retour et le transmet au cadran qu'il doit connaître. Ainsi, le clavier joue un rôle plus important qu'auparavant.

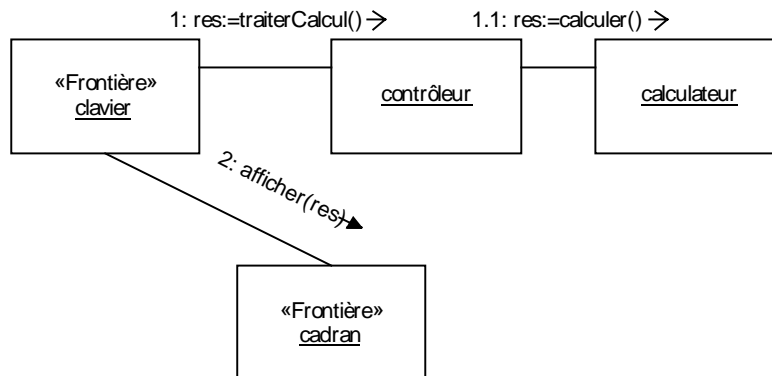


Figure 2–21 : Solution dans laquelle le clavier joue un rôle important

Ainsi, les moyens de supprimer un cycle sont nombreux. De façon générale, *encapsuler ce qui pose problème* est une solution passe-partout. Cette profusion de tactiques a simplement pour but de proposer différentes pistes car les situations de type cycle entre paquetages ne sont pas si rares. Il reste une possibilité qui est basée sur la relation particulière « subscribe to » (diffusion - souscription) entre classes d'analyse et qui met en œuvre un mécanisme de souscription entre une classe souscripteur et une classe à laquelle on souscrit. Le souscripteur souhaite être informé de changements d'état dans l'élément auquel il souscrit.

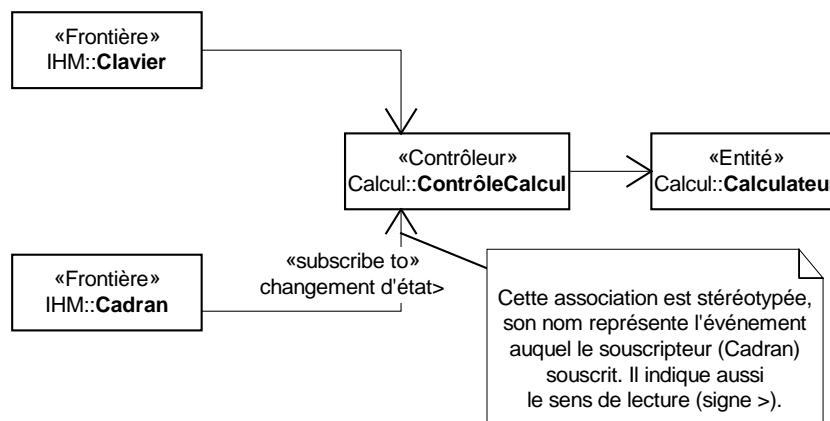


Figure 2–22 : Suppression d'un cycle par la relation « subscribe to »

Les navigations sur les associations ont pour but de montrer les sens d'envoi de messages, donc les relations d'utilisation. Elles permettent de valider le sens des dépendances entre les paquetages. Désormais, le paquetage `IHM` dépend de `Calcul` et non plus l'inverse.

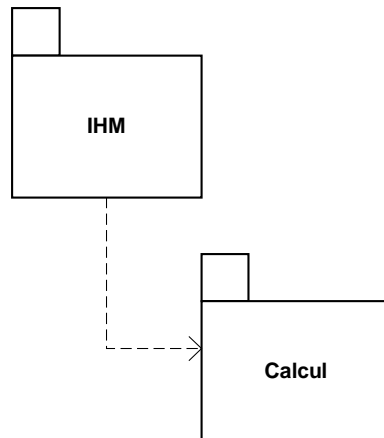


Figure 2-23 : Relations entre paquetages d'analyse

Comment choisir parmi les solutions étudiées ? Il est peut-être temps de voir ce que donne la conception : ne pas tout concevoir mais « simplement » les mécanismes les moins évidents ; ne pas terminer leur conception à tout prix, mais vérifier la faisabilité, le coût, les avantages et inconvénients de telle ou telle solution. Cette stratégie est typique de Unified Process : gérer les risques de mise en œuvre au plus tôt sans attendre un effet dévastateur (sur le planning ou le produit lui-même) quelque part entre les phases de programmation et d'intégration (l'effet « big bang »). L'objectif de cette itération d'élaboration est de définir l'architecture et la majorité des cas d'utilisation. A strictement parler, ces derniers sont (ici) trop détaillés : les questions sur les contraintes d'architecture en terme de définition de paquetage doivent apparaître plus rapidement, ce qui évite des modélisations remises plusieurs fois en cause.

2.2 Conception du prototype

L'objectif de ce paragraphe est de préciser l'impact réel des choix d'analyse. De même que le langage de programmation influence la conception (est-il raisonnable d'implémenter l'héritage dans un langage qui ne le propose pas ?), de même la conception, précisément l'architecture génère un certain feed-back sur l'analyse. Ignorer cela constitue une prise de risque et dans tous les cas une promesse de cauchemars chez les concepteurs. L'analyse structure en paquetages. Cela devient une contrainte pour l'architecte qui vérifie la faisabilité. Un paquetage d'analyse devant être réutilisable (ce n'est pas le cas de tous les paquetages) satisfait à un certain nombre de critères : couplage faible en particulier. Il est essentiel de noter que c'est le *sous-système* correspondant (ou les sous-systèmes) tel qu'il sera obtenu par les concepteurs qui sera *effectivement* réutilisé.

2.2.1 Remarques préliminaires

La conception fait apparaître les sous-systèmes : ils sont la contrepartie des paquetages, du moins pour les couches hautes. Autre point : les sous-systèmes, les classes deviennent des réalisations d'interfaces. Un élément utilise des services déclarés dans une interface et pris en charge par un réalisateur qui lui est inconnu. C'est le principe de base du développement de composants. D'ailleurs les termes « composant » et « interface » jouent un rôle essentiel dans Unified Process. Une classe réalise une ou plusieurs interfaces, de même qu'un sous-système. Les interfaces sont traduites directement en Java (mot clés `interface` et `implements`). Elles sont représentées par des classes abstraites en C++. L'activité de programmation nous permettra de comprendre tout l'intérêt de ce style de développement.

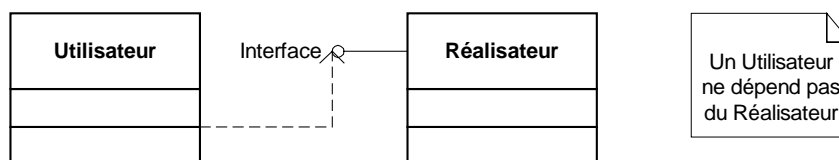


Figure 2-24 : Principe de base de la conception

La maquette (interface utilisateur) de la calculatrice correspond parfaitement à une seule fenêtre graphique qui contient effectivement deux parties : clavier et cadran. En ce qui concerne la conception, séparer véritablement ces deux parties serait probablement une complication inutile.

IHM de type caractère : quelle est la finalité ? La ligne A de la matrice des impacts nous apprend que ce type « d'interface acteur » est simplement un « plastron ». Il simule la véritable interface dans un but de validation de l'architecture, en particulier du paquetage `Calcul`. L'objectif est alors pour nous de limiter les conséquences d'une construction visible jetable en ce qui concerne cet élément. Créer une classe basée sur la bibliothèque standard C++ et qui réalise les types `Cadran` et `Clavier` ne pose pas de problème. Cela permettra de valider les classes `contrôleCalcul` et `Calculateur`, donc le paquetage `Calcul`.

L'architecture doit s'efforcer de préserver la capacité de réutilisation des objets métier. Dans le cas de la calculatrice, le calcul lui-même peut être considéré comme étant le coeur de l'application, basé sur des objets métier indépendants de l'application (et de son interface utilisateur) qui les utilise.

Dans la pratique, le concepteur travaille sur le modèle d'analyse tel qu'il est validé par l'analyste. Dans le cadre de cette étude, les différents choix d'empaquetage sont repris et étudiés afin de montrer l'impact qu'ils ont sur la conception. Dans les faits, l'analyste crée un premier modèle d'analyse qui est repris par l'activité de conception. Si cette activité montre un problème d'empaquetage ou plus généralement d'analyse, cela se traduit par une nouvelle itération : une activité d'analyse est à nouveau déclenchée. Les aspects non fonctionnels des besoins sont repris du modèle d'expression de besoins.

2.2.2 Conception de la solution "ihm-appli"

La classe `Calculateur` réalise une interface `Icalculateur`, idem pour `ContrôleCalcul` et `IcontroleCalcul`. Les types d'analyse `Cadran` et `Clavier` deviennent les interfaces `Icadran`

et `Iclavier`. Ces deux interfaces acteur (`Cadran` et `Clavier`) sont réalisées par une même classe que nous appelons `Fcalc` pour fenêtre calculette. Attention, la classe `Calculateur` est la contrepartie conception de l'entité d'analyse, ce n'est pas le même élément de modélisation.

N Les stéréotypes proposés par Unified Process : « boundary » « control » « entity » sont exclusivement réservés
O à l'activité d'analyse. Les classes de conception sont un raffinement des classes d'analyse. Elles ne
T correspondent pas aux trois stéréotypes. Le concepteur peut créer des stéréotypes propres à son activité. Dans
E ce cas, se pose la question : adaptation / extension de UML contre standard. Les stéréotypes étendent le
 metamodelle UML, ils le personnalisent, ce qui l'éloigne du standard reconnu et utilisé par tous.

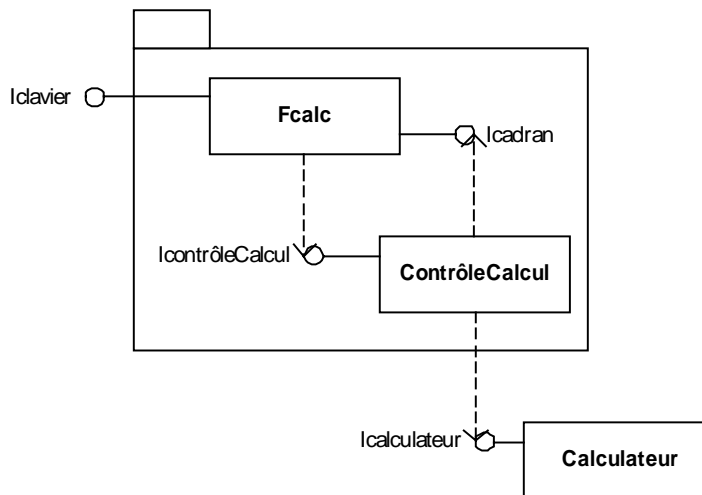


Figure 2-25 : Conception solution « ihm-appli »

La classe `ContrôleCalcul` dépend de l'interface `Icadran`, pas de la classe `Cadran`. Cela signifie que toute mise en œuvre ou réalisation de cette interface est compatible avec cette dépendance.

La solution « Affichage » est basée sur le même principe : la classe de contrôle dépend d'une interface qui joue le rôle de miroir. L'intérêt de la deuxième solution est que le cycle entre sous-système disparaît.

2.2.3 Conception de la solution "assos-cadran"

Cette autre proposition de découpage en paquetage d'analyse est basée sur

Une association entre `clavier` et `cadran` : le `clavier` envoie des messages au `cadran`, ces deux classes sont utilisées systématiquement ensemble, évoluent ensemble, correspondent à la même fonction IHM, les relier ne pose pas de problème particulier

Les messages envoyés au contrôleur sont *toujours* de la forme :
`état_résultant := message()`.

Le diagramme de classes de conception (partiel) visualise précisément l'interface.

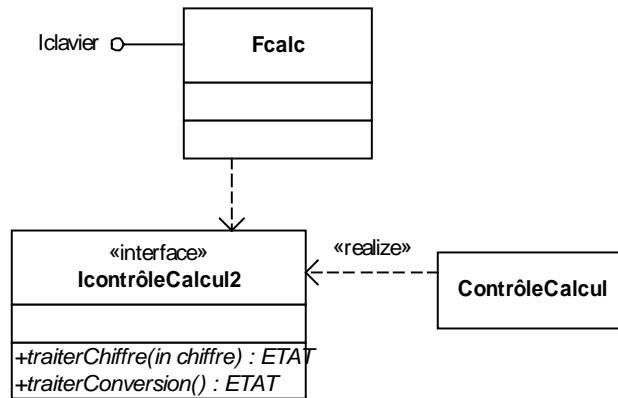


Figure 2–26 : Interface dont les services renvoient systématiquement l'état de la calculette

Les trois solutions proposées en analyse (ihm-appli, paquetage Affichage, assos-cadran) ont un inconvénient majeur : elles ne préservent pas l'empaquetage des objets métier controleCalcul et calculateur. De ce fait, il est nécessaire d'envisager la conception de la solution basée sur le diagramme de paquetages suivant.

La relation de dépendance unidirectionnelle suppose la présence de l'association <<subscribe to>> du Cadran vers le ContrôleCalcul. Il s'agit alors de traduire cette relation en conception. Le pattern Observateur est typique de cette situation.

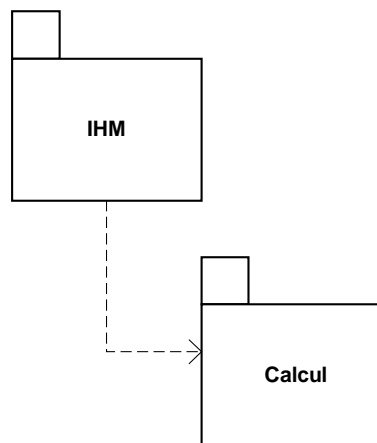


Figure 2–27 : Paquetages d'analyse à concevoir

2.2.4 Pattern Observateur

Ce modèle de conception a pour objectif de découpler observateurs et sujets. Les diagrammes UML suivants visualisent les aspects structurels statiques et dynamiques de ce type de conception, ainsi que la mise en œuvre du pattern sur la calculette.

Aspect structurel du pattern Observateur

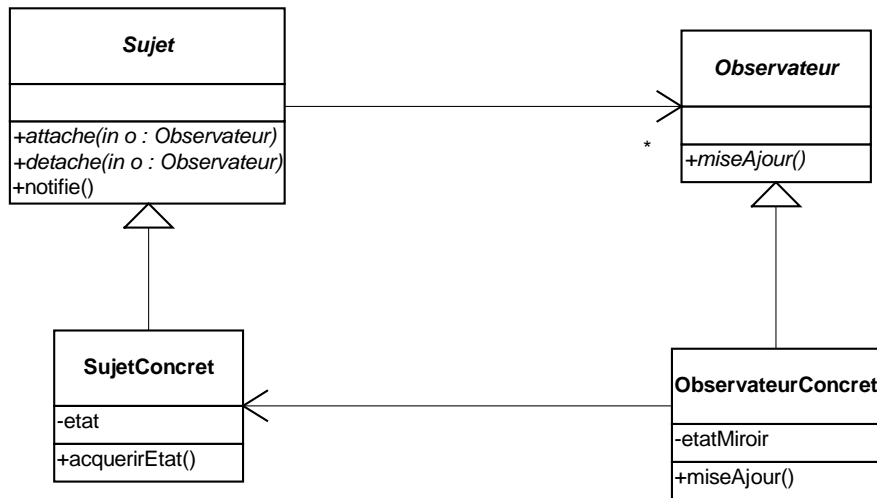


Figure 2-28 : Pattern Observateur : aspect structurel

Sujet est une classe abstraite (`Observable` en Java). Un observateur peut s'attacher à un sujet, autrement dit souscrire à un changement d'état pour en être informé. Il peut également se détacher pour indiquer qu'il ne souscrit plus. La méthode `notifie()` envoie le message `miseAjour` aux observateurs qui ont souscrit.

Appliqué à la calculette, ce pattern résout la question de couplage entre paquets en jouant sur la classe abstraite `Observateur`.

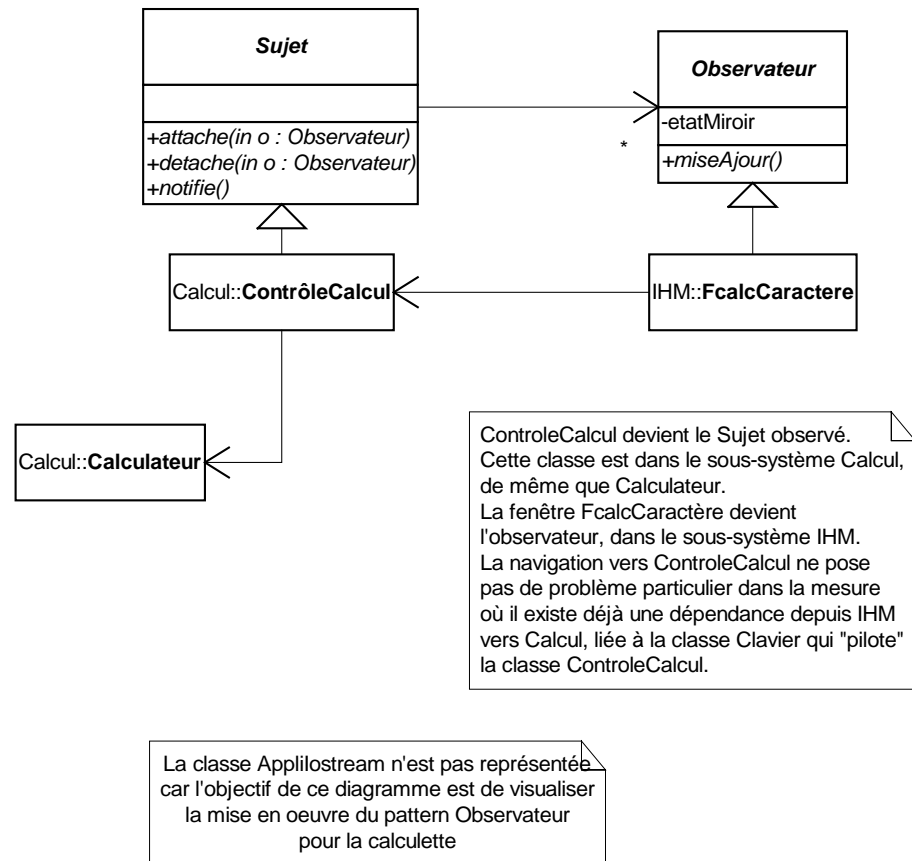


Figure 2–29 : Conception de la calculette basée sur le pattern Observateur

Aspect dynamique du pattern Observateur

La dynamique est visualisée par un diagramme d'interactions.

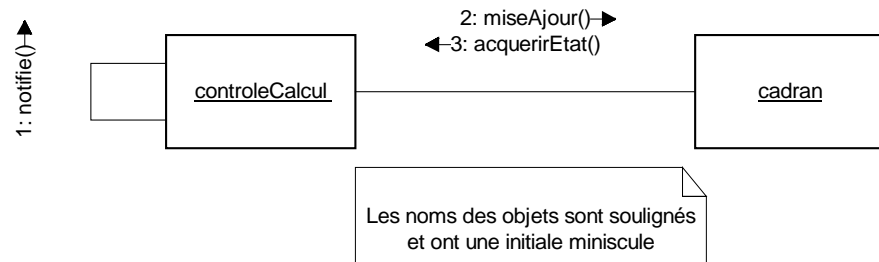


Figure 2–30 : Aspect dynamique du pattern - ici, le diagramme de collaboration est inadap­té

Dans ce cas précis, la collaboration entre les objets est pertinente d'un point de vue *temporel*. Le diagramme de collaboration n'est donc pas particulièrement adapté. Au contraire, le diagramme de séquence se prête bien à cette visualisation.

Le scénario modélisé suppose que l'observateur (l'objet `cadran`) est attaché au sujet (`controleCalcul`).

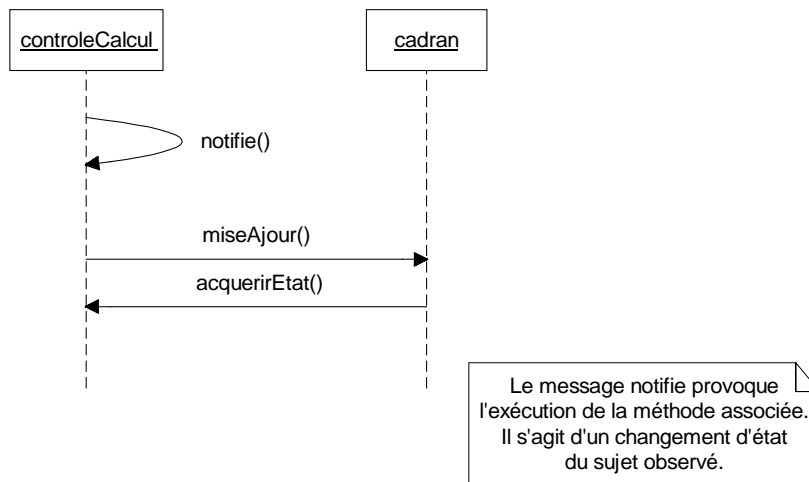


Figure 2-31 : Aspect dynamique du pattern, avec un seul observateur

Désormais, la classe `ContrôleCalcul` dépend (au travers de sa sur-classe `Sujet`) d'une classe abstraite et pas d'une mise en œuvre.

A ce stade, le paquetage `Calcul` est analysé et marqué en tant que paquetage métier à réutiliser. De plus, la conception a permis de valider la faisabilité, *sur papier*.

2.3 Modèle d'analyse

Vue structurelle du système

La vue structurelle statique du système est constituée des paquetages de classes d'analyse.

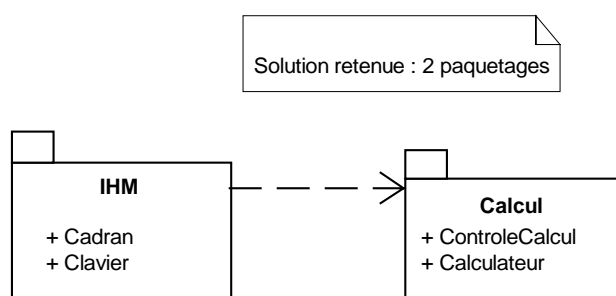


Figure 2–32 : Diagramme de paquetages d'analyse

Les classes d'analyse `Clavier` et `Cadran` sont conservées, même si visiblement la conception prévoit de les réunir pour les réaliser. En effet, elles jouent des rôles distincts : l'une est responsable des saisies des acteurs, l'autre joue le rôle de miroir de la calculatrice.

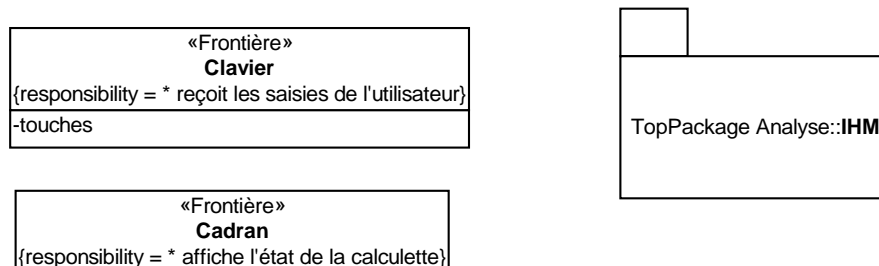


Figure 2–33 : Diagramme de classes du paquetage IHM

Le paquetage `Calcul` constitue le paquetage métier. En terme de réutilisation, il présente le potentiel le plus élevé, par rapport au paquetage `IHM` qui est spécifique à l'application. Ainsi, l'analyste précise les objectifs de réutilisation en jouant sur les paquetages d'analyse.

La vérification du partitionnement en paquetages repose sur les critères de découpage.

Même fonctionnalité : interface utilisateur, objets métier, etc

Utilisation commune : les classes d'un paquetage sont généralement utilisées simultanément

Evolution commune : de même, elles évoluent souvent ensemble.

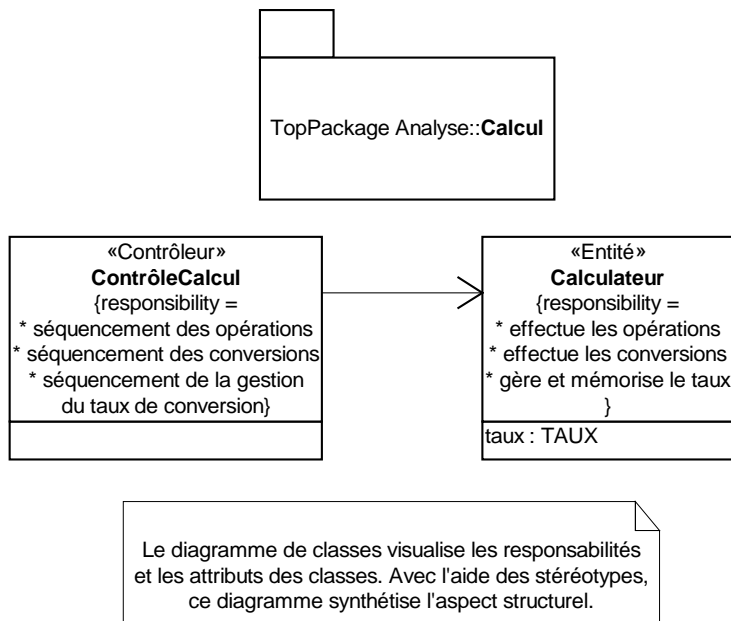


Figure 2-34 : Diagramme de classes du paquetage Calcul

Vue dynamique

Dans l'activité d'analyse, les diagrammes d'interaction permettent de découvrir les objets du système par sa dynamique. De même que les diagrammes de paquetage et de classe synthétisent l'aspect structurel, de même les diagrammes d'état / transition synthétisent l'aspect dynamique.

Le système en tant que tel, un sous-système, différentes classes méritent une étude de leur évolution dans le temps. Quels sont les éléments pertinents de ce point de vue ?

L'état d'un objet est une situation dans laquelle il réagit à certains événements, d'une certaine façon. Dans un diagramme d'état, le non-dit est tout aussi important que ce qui est visualisé. Certaines transitions n'existent pas car l'objet de la classe en question ne peut pas, ou ne doit pas, traiter l'événement correspondant dans tous les états.

Pour déterminer les classes adaptées à une étude de leur dynamique, nous créons un tableau qui indique le degré de pertinence et qui le justifie. Ensuite, nous visualiserons les aspects dynamiques de ces classes.

Classe	Utilité % étude de la dynamique	Justification
Clavier	-	L'aspect graphique pourra justifier l'étude en construction
Cadran	-	Idem Clavier

ContrôleCalcul	+++	Le contrôleur est le cœur de la calculette. Etant donné qu'il a pour responsabilité de traiter les différents messages émis par le clavier, tels que <code>traiterChiffre</code> , sa dynamique est indispensable : les actions déclenchées sur réception de ces événements dépendent de l'état dans lequel se trouve le contrôleur.
Calculateur	++	Le calculateur ne peut pas accepter tous les événements quel que soit son état. Par exemple, une conversion ne peut s'opérer que si le taux est connu, c'est une pré-condition.

Figure 2–35 : Pertinence de l'étude de la dynamique

Les diagrammes de séquence permettent de cerner la dynamique des objets car ils focalisent aussi bien leur réalisation que leur lecture sur le temps. Reprenons quelques scénarii en tenant compte des classes d'analyse définies dans la vue statique.

Le scénario $2+2=4$ est à la fois très simple et très instructif d'un point de vue dynamique. Quels sont les objets collaborants à cette interaction ? Un `clavier`, un `contrôleCalcul`, un `calculateur` et un `cadran`. Le tracé du diagramme de séquence permet de mieux appréhender l'objet `contrôleCalcul` : il reçoit et émet des messages. Ces événements sont potentiellement déclencheurs de transition. A l'inverse, une transition ne peut être déclenchée *ex nihilo*. Ainsi, les diagrammes de séquence nous offrent des « états candidats » qu'il reste à organiser dans les diagrammes d'état.

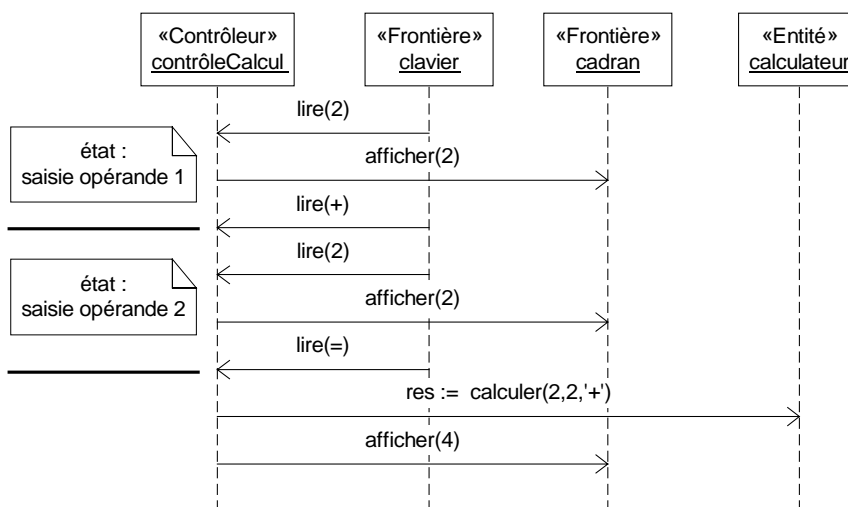


Figure 2–36 : Diagramme de séquence scénario calcul normal

La saisie de l'opérateur provoque un changement d'état : le contrôleur réagit différemment car il traitait l'opérande 1 ; désormais il traite l'opérande 2. Les segments sur la ligne de vie du contrôleur deviennent ainsi des états candidats. Le diagramme de séquence modélise le séquençage des interactions. Par définition, une transition correspond à un événement (la fin d'une opération dans l'objet étant en elle-même un événement qui provoque une transition automatique). Ainsi, chaque message reçu ou émis par l'objet est candidat pour jouer le rôle de déclencheur de transition. De ce fait, les segments sur la ligne de vie sont autant d'états candidats.

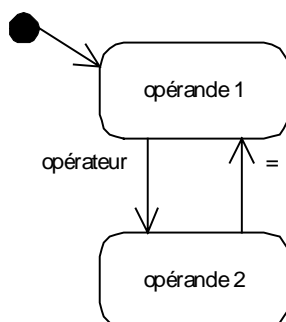


Figure 2-37 : Etat / Transition provisoire de la classe ContrôleCalcul

Ce diagramme doit être maintenant décoré avec les opérations : actions et activités déclenchées par les événements.

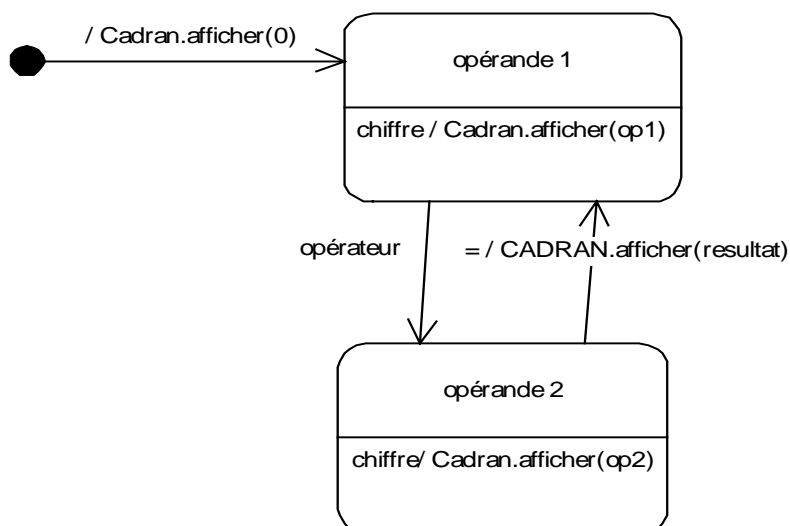


Figure 2-38 : Diagramme Etat / Transition provisoire de la classe ContrôleCalcul

L'événement « = » provoque un changement d'état, une transition, depuis l'état opérande 2 vers l'état opérande 1. Le message afficher(resultat) est envoyé à un objet de la classe Cadran.

Ce diagramme est incomplet. Il ne visualise pas les événements liés aux taux. De même que les diagrammes de collaborations permettent d'obtenir petit à petit le diagramme de classes, de même les diagrammes de séquence construisent le diagramme d'état de la classe étudiée.

Ce diagramme doit être vérifié. Pour cela, reprenons des scénarii et voyons ce que cela donne. Supposons l'enchaînement suivant : $2+2=4$ puis $10+2=12$.

Séquence du scénario et résultat attendu	Modélisé dans le diagramme d'état
La calculatrice démarre et affiche 0	Oui. C'est la transition depuis le pseudo état initial vers <code>opérande 1</code>
L'acteur appuie sur 2	Oui. L'objet est dans l'état <code>opérande 1</code> et reçoit un chiffre, il envoie le message <code>afficher(2)</code>
L'acteur appuie sur '+'	Oui. L'objet subit une transition, il est désormais dans l'état <code>opérande 2</code>
L'acteur appuie sur 2	Oui. Dans l'état <code>opérande 2</code> , l'objet peut recevoir un événement chiffre (en fait <code>traiterChiffre</code>)
L'acteur appuie sur =	L'objet est dans l'état <code>opérande 2</code> , cet événement provoque l'affichage du résultat et le retour dans l'état <code>opérande 1</code> .
Deuxième opération : $10+2$ L'acteur appuie sur 1	L'objet est dans l'état <code>opérande 1</code> , il doit afficher 1. Or, il affiche <code>opérande 1</code> , comment affirmer que cette valeur est correcte ?
...	

La modélisation doit être affinée. Pour cela, nous créons les actions spécifiques `entry` dans chaque état afin d'initialiser correctement l'opérande concerné. Les noms de messages sont affinés, le label `lire` est remplacé par des labels plus précis tels que `traiterConversion`.

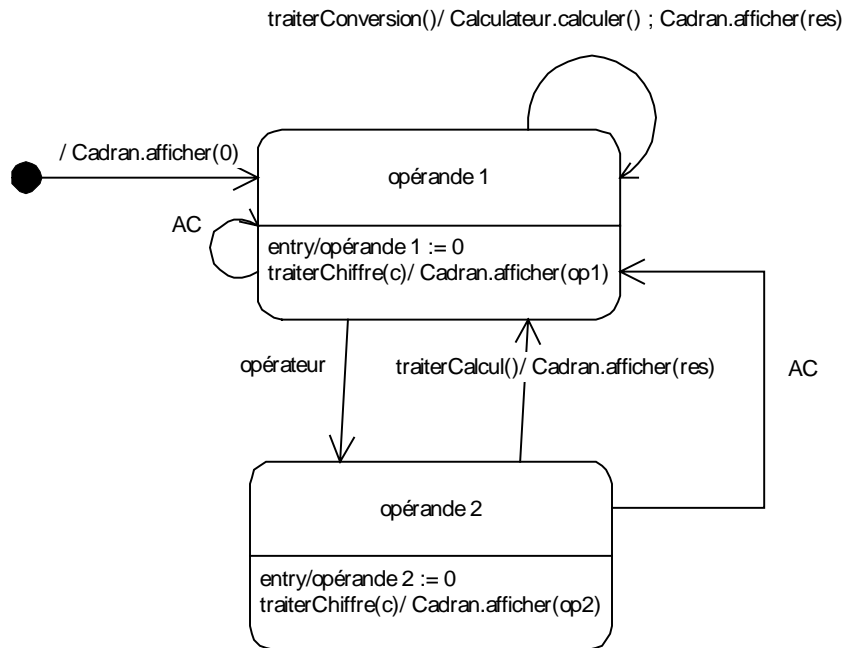


Figure 2–39 : Etat / transition de la classe ContrôleCalcul

La classe `Calculateur` mérite peut-être une modélisation de sa dynamique. Nous pouvons modéliser rapidement cela avec un diagramme d'état, nous serons toujours à temps de le jeter, si le contenu n'est pas de grande valeur ajoutée.

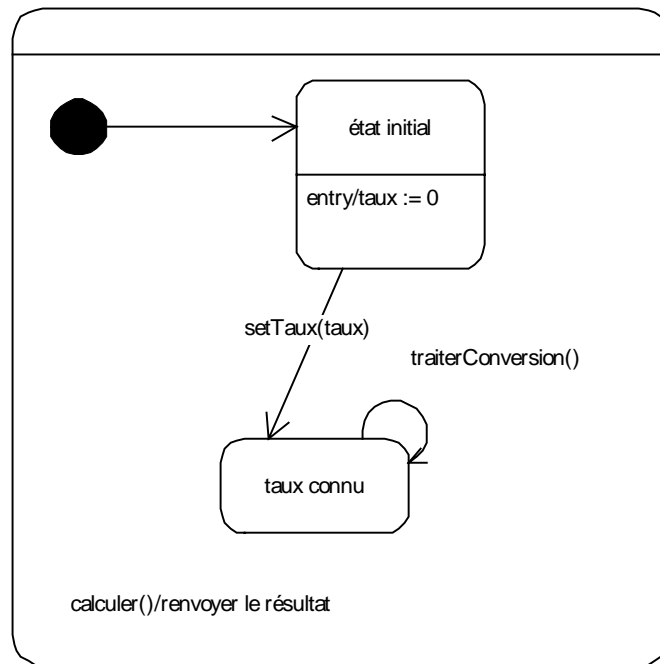


Figure 2–40 : Etat / transition de la classe Calculateur

Le modèle d'analyse est constitué essentiellement du diagramme de paquetages, des diagrammes de classes et des états transition. Les interactions (collaborations et séquences) sont également conservées car elles indiquent la véritable dynamique du système.

La documentation des éléments d'analyse, des classes essentiellement, permet de constituer un dictionnaire ou glossaire des termes du projet.

2.4 Modèle de conception

La solution retenue en conception est constituée d'une classe `Fcalc` qui regroupe les classes `Cadran` et `Clavier` de l'analyse. L'application elle-même correspond à une classe active. C'est un composite constitué des objets instances des classes de conception. La fonction `main()` se réduit alors à :

```

int main(void)
{
    AppliIostream *appli = new AppliIostream ;
    Appli->run() ;
}
  
```

2.4.1 Vue structurelle de la conception

La vue structurelle de l'application est modélisée sous forme de diagramme de classes, en fonction des sous-systèmes et des interfaces.

L'architecture nécessaire au déploiement de la calculette est basée sur l'interface graphique standard des systèmes ouverts : Motif de l'Open Software Foundation (OSF). Cette architecture client / serveur sera présentée en détail dans un chapitre spécifique. Pour l'instant, il importe de valider le paquetage `Calcul` à partir d'un prototype basé sur une interface utilisateur alphanumérique.

Le prototype, dans cette phase, est basé sur un nœud qui est une machine quelconque, pourvu que l'on puisse générer un exécutable à partir du langage C++. En toute rigueur, il faudrait distinguer machine d'exécution du prototype et machine de développement. Sur cet exemple, les deux sont confondues.

L'architecture du prototype est basée sur un composant `<<executable>> calculette`.

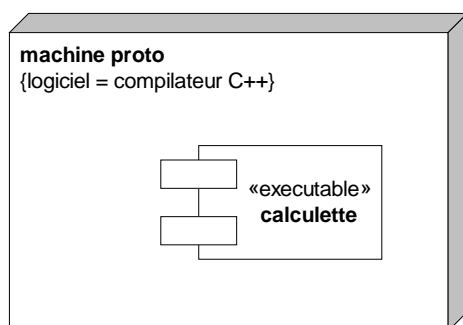


Figure 2-41 : Déploiement prototype

Les sous-systèmes retenus ici sont les contre-parties des paquetages d'analyse.

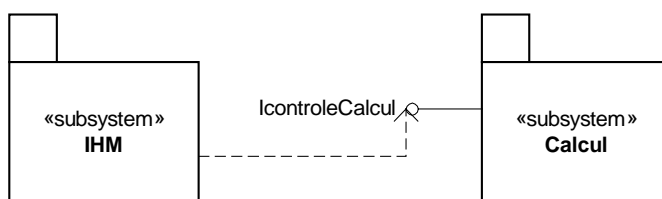


Figure 2-42 : Sous-systèmes du prototype

Le diagramme de sous-systèmes fait apparaître les interfaces. Pour être complet, les classes qui ne font pas partie de ces sous-systèmes doivent malgré tout figurer. Il s'agit des classes

`AppliIostream` : la classe qui représente l'application elle-même, ce qui est en quelque sorte une autre façon d'écrire la fonction `main()`

`Sujet` et `Observateur` : ce sont les classes abstraites nécessaires à la mise en œuvre du pattern Observateur.

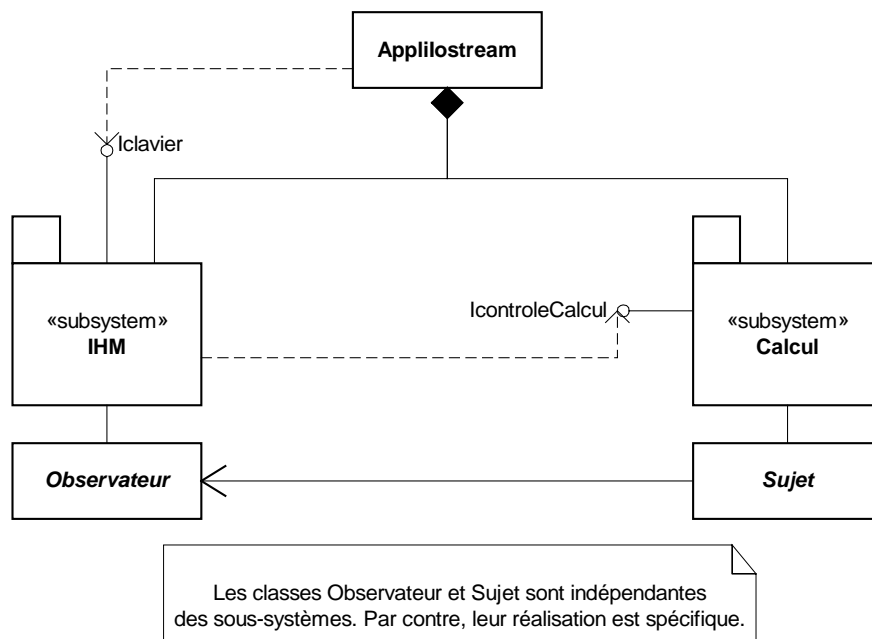


Figure 2-43 : Sous-systèmes et classes du prototype

La mise en œuvre des classes `Observateur` et `Sujet` est spécifique. Un sujet gère ici un seul observateur ce qui est une simplification du pattern. L'application est constituée des sous-systèmes qui eux-mêmes sont en relation avec les classes abstraites du pattern.

2.4.2 Les opérations

La surcharge d'opérateur du C++ nous apprend qu'une expression est constituée d'opérandes et d'opérateurs.

```
| x = a + b; // deux opérateurs : + et =
```

Le résultat de $a + b$ est la somme, le résultat de $x = a + b$ est x .

N Le compilateur gère des symboles typés et des opérateurs. Des instructions telles que :
O $x + y;$
T ne posent pas de problème au compilateur.

E Le résultat d'une assignation est l'objet assigné. C'est ce qui permet d'écrire :
`if (x = y)`
 ou bien
`x = y = z;`

Ainsi, une assignation est une opération au même titre que l'addition ou même l'appel d'une fonction.

Adaptée à la calculatrice, cette caractéristique permet de considérer toute touche hors chiffre comme un code opératoire. Cela permet de modifier l'interface `IcontroleCalcul`.

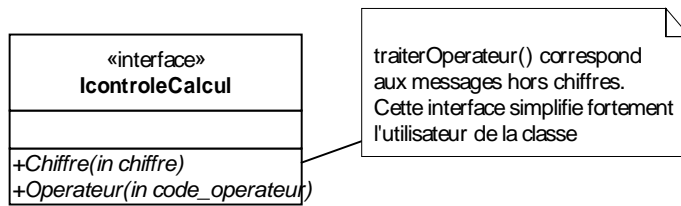


Figure 2-44 : Interface IcontroleCalcul

Le cadran joue le rôle de miroir et à ce titre fournit une interface `Icadran`.

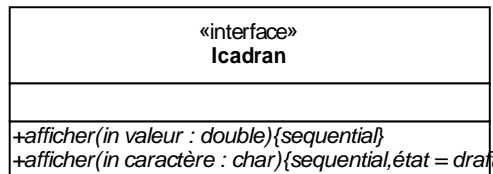


Figure 2-45 : Interface Icadran

Une véritable calculette affiche 'E' en cas d'erreur. La surcharge du service `afficher()` offre cette possibilité. Toutefois, nous l'étiquetons 'draft' car cette demande n'est pas dans les besoins exprimés sous forme de cas d'utilisation. Une rétroaction (feed-back) vers l'expression de besoins est une conséquence naturelle de l'analyse.

Les besoins sont exprimés sous forme de cas d'utilisation, l'analyse a déterminé les classes et paquetages qui correspondent aux spécifications en terme de fonctionnalité, d'exigence de réutilisation. La conception a traduit l'ensemble des besoins à partir des solutions retenues (déploiement, langages, composants tiers...). Cette itération d'élaboration se poursuit par la vérification, la validation de l'architecture par prototype effectif.

Tous les aspects de la conception n'ont pas été traités dans cette activité. En particulier, l'initialisation du système reste à définir. Il est possible de reporter ces travaux à l'activité suivante dans la mesure où l'on suppose que les développeurs ont suffisamment d'informations pour cela.

2.4.3 Etat de la calculette

L'adoption du pattern Observateur implique une connaissance plus précise de l'état dans lequel se trouve le sous-système Calcul. En effet, l'opération `acquérirEtat()` doit renvoyer une information pertinente aux observateurs. Sans que `controleCalcul` est une connaissance précise des observateurs, cet objet doit savoir qu'il peut être observé. A ce titre, il se doit de renvoyer une valeur d'état, la situation dans laquelle il se trouve, adéquate. L'étude de la dynamique montre que les deux états de base « Opérande 1 » et « opérande 2 » ne suffisent plus. Ils doivent être affinés car tout dépend de la saisie de l'acteur. Après la saisie d'un opérateur, la calculette bascule dans l'état « Opérande 2 ». Tant que l'acteur n'a pas saisi de chiffre, l'affichage doit correspondre à la valeur de l'opérande 1. Ces états méritent donc une étude plus poussée.

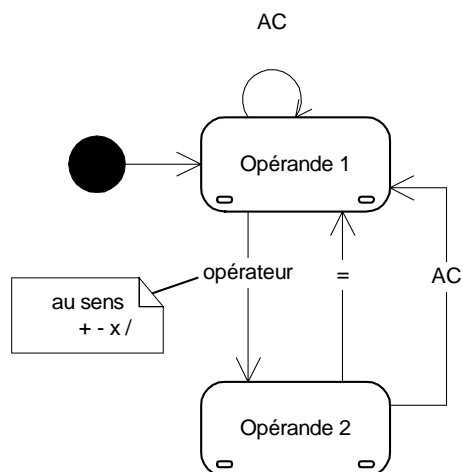


Figure 2-46 : Diagramme d'état de la classe ControleCalcul

Chaque état de ce diagramme correspond à un ensemble d'états de niveau plus fin. Sur réception de AC, quel que soit le sous-état, le système revient dans « OP1 pas de chiffre ».

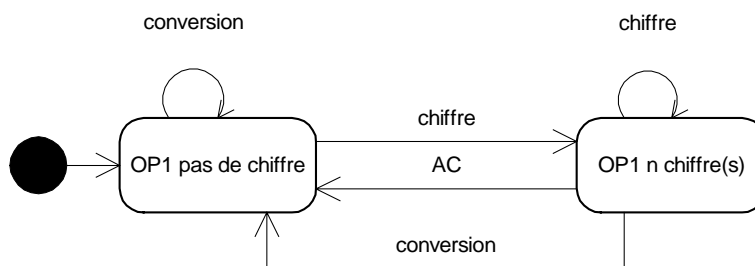


Figure 2-47 : L'état Opérande 1 détaillé

Les transitions sur les événements « conversion » pourraient être remontées au niveau supérieur.

L'état « Opérande 2 » peut être également précisé sous forme de diagramme d'état/transition. Les actions ne sont pas visualisées dans ces diagrammes.

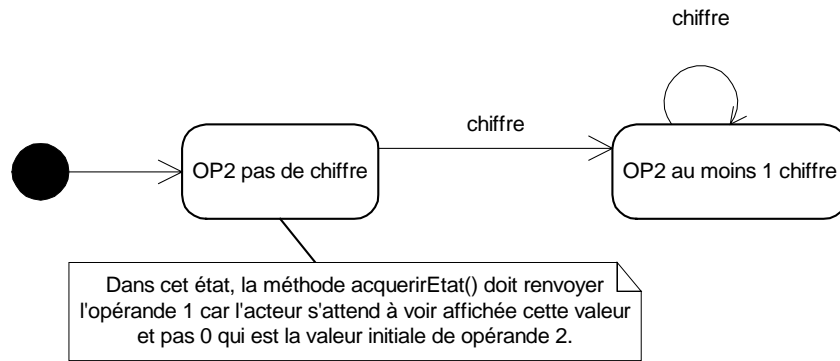


Figure 2-48 : Détails de l'état Opérande 2

La programmation de la classe `ContrôleCalcul` fera apparaître ces quatre sous-états. Ils sont pris en compte

Dans les opérations de traitement des messages reçus (gestion des chiffres...)

Dans l'opération `acquerirEtat()` qui peut alors déterminer quelle est la valeur d'état à renvoyer.

ANALYSE DE LA CALCULETTE.....	1
2.1 Analyse : des cas d'utilisation aux objets	1
2.1.1 Démarche d'obtention des objets	2
2.1.2 Cas d'utilisation "Calcul"	3
2.1.3 Cas d'utilisation "Conversion"	7
2.1.4 Cas d'utilisation "Taux"	9
2.1.5 Diagramme de classes d'analyse.....	10
2.1.6 Paquetages d'analyse	12
2.2 Conception du prototype	20
2.2.1 Remarques préliminaires	21
2.2.2 Conception de la solution "ihm-appli"	21
2.2.3 Conception de la solution "assos-cadran"	22
2.2.4 Pattern Observateur.....	23
2.3 Modèle d'analyse.....	26
2.4 Modèle de conception.....	32
2.4.1 Vue structurelle de la conception	32
2.4.2 Les opérations	34
2.4.3 Etat de la calculette	35