Un modèle pour la séparation et la traçabilité des préoccupations

Dolores Diaz, Lionel Seinturier, Laurence Duchien, Pascal Flament

Un modèle pour la séparation et la traçabilité des préoccupations

Dolores Diaz[*,] — Lionel Seinturier[*] — Laurence Duchien [*] — Pascal Flament [**]**

[*] *Univ. des Sciences et des Technologies de Lille - UMR CNRS 8022*
INRIA Futurs - Projet Jacquard Bat. M3
59655 Villeneuve d'Ascq
CEDEX France

{diaz, seinturi, duchien}@lifl.fr

[**] *Norsys*
1, rue de la cense des raines
ZAC du Moulin
59710 Ennevelin

{ddiaz, pflament}@norsys.fr

RÉSUMÉ. *L'évolutivité des systèmes d'information en entreprise constitue à la fois un enjeu crucial et une propriété logicielle difficile à mettre en œuvre. Des solutions existent pour intégrer cette faculté logicielle mais elles restent propres à une étape du cycle de vie d'un logiciel et ne fournissent finalement que des réponses partielles. Nous proposons de faire le lien entre ces solutions par la définition d'un modèle qui regroupe, structure et trace les éléments de définition d'une application. A l'heure actuelle, notre modèle EUCLIDE relève les éléments logiciels issus de la définition itérative et incrémentale d'une fonctionnalité et établit l'évolution d'une telle définition avec la notion de traçabilité fonctionnelle.*

ABSTRACT. *Nowadays evolubility of applications is a crucial stake and a difficult software property to integrate in applications. Solutions to improve software evolutivity are proposed but each of them is dedicated to one stage of the software lifecycle and provides partial answers. We propose an approach that cross-cuts all these stages with the définition of a link between those partial answers. This solution permits to group, structure and trace units of the definition of functionalities during software lifecycle. Our model, EUCLIDE, emphasizes iterative and incremental definition of a functionality thanks to the notion of functional traceability*

MOTS-CLÉS : *séparation des préoccupations, traçabilité, AOP*

KEYWORDS: *separation of concerns, traceability, AOP*

1re soumission à *l'Atelier sur l'évolution du logiciel - LMO 2005*, le 18 février 2005.

1. Introduction

L'évolutivité des applications est devenue une priorité fondamentale pour les industriels. Indépendamment de la nature des architectures, ces applications doivent savoir évoluer vers de nouveaux contextes, de nouvelles technologies et en fonction des besoins utilisateurs. De plus, ces évolutions se doivent d'être rentables, rapides et efficaces. Or, les outils et les méthodes manquent pour assister ces processus d'évolution et laissent souvent les concepteurs, les développeurs ou les acteurs de la maintenance applicative dans une démarche aléatoire. Deux principes peuvent être retenus pour préparer cette démarche d'évolutivité applicative : la séparation des préoccupations et la traçabilité.

La séparation des préoccupations (également appelée SoC pour *Separation of Concerns*) prône une démarche de structuration d'une application en termes d'entités séparées, appelées préoccupations. Une préoccupation définit la réalisation d'un objectif ou d'un concept. Par la propriété d'encapsulation, une préoccupation regroupe toutes les unités logicielles qui la définissent. Les approches basées sur ce principe s'illustrent dans les domaines de la modélisation et de la programmation et sont généralement dédiées à une étape du cycle de vie d'un logiciel. Le domaine de la programmation orientée aspect (AOP) [KIC 97] est actuellement un domaine de recherche prometteur pour la mise en oeuvre de la séparation des préoccupations. Les frameworks et langages existants, tels que AspectJ, Hyper/J, AspectWerkz, ou JAC [PAW 04] qui est le résultat de nos travaux antérieurs, l'appliquent mais uniquement à l'étape de la programmation. L'introduction des aspects dans les phases amont, comme par exemple les approches "early aspect" [MOR 02, RAS 03] généralisent l'approche et l'appliquent à l'étape de spécification des besoins. Ces approches sont intéressantes mais elles ne proposent pas une répercussion claire et définie du principe de base sur plusieurs étapes de développement d'une application. La définition des préoccupations n'est pas maintenue au travers des étapes de développement et il n'existe aucune description sur la manière de répercuter cette séparation de préoccupations aux étapes suivantes. Elles n'exploitent pas la notion de traçabilité.

Sous le terme de *traçabilité*, nous évoquons le suivi d'une unité logicielle au cours du cycle de développement d'une application et nous nous intéressons par là même, aux différentes transformations ou traductions de l'unité logicielle à travers ces différentes étapes. La nature du suivi ou de la trace de l'unité logicielle caractérise le type de traçabilité. La seule reconnue à ce jour est la traçabilité fonctionnelle, définie par le processus conduit par les cas d'utilisation [JAC 94]. Elle définit l'évolution d'un besoin, modélisé par un cas d'utilisation vers différents diagrammes UML, pour finalement être projeté vers du code. Ce processus établit un type de traçabilité au cours des étapes mais ne maintient pas de structuration en préoccupations durant le développement. La pratique montre qu'elle est perdue à l'étape de conception.

Nous proposons un modèle qui combine ces deux concepts : séparation des préoccupations et traçabilité fonctionnelle afin de préparer la réalisation de mécanismes d'évolution d'une application. Ce modèle appelé EUCLIDE permet la modélisation

d'une application selon une structuration en préoccupations et la modélisation de l'évolution des préoccupations à travers les étapes de développement d'un logiciel par le principe de traçabilité.

La deuxième section de cet article présente EUCLIDE et introduit la modélisation de logiciel dans ce nouveau cadre. La section 3 présente la définition de la traçabilité fonctionnelle dans EUCLIDE. La section 4 illustre notre modèle par un exemple simple. La section 5 confronte notre modèle avec des approches existantes. Enfin, la section 6 conclut cet article et présente les perspectives.

2. Le modèle EUCLIDE

Cette section présente EUCLIDE, notre modèle pour la spécification, la séparation et la traçabilité des préoccupations. A la suite de sa définition, nous introduisons la modélisation euclidienne d'une application.

2.1. *Définition d'EUCLIDE*

Influencé par l'approche de la séparation multidimensionelle des préoccupations, définie par P. Tarr et H. Ossher [OSS 00], EUCLIDE se fonde sur un ensemble d'axiomes qui implantent un contexte particulier de modélisation logicielle. Nous en donnons ici les principaux :

Définition 2.1 *Toute application s'abstrait en un ensemble fini d'unités logicielles. Notons U cet ensemble. Par unité logicielle, nous désignons tout élément de définition participant à la mise en oeuvre de l'application. La nature ainsi que la granularité de l'unité ne sont pas imposées.*

Par exemple, une classe Java, un diagramme de cas d'utilisation ou encore un diagramme de composants sont des exemples d'unités logicielles et leur réunion peut correspondre à la modélisation euclidienne d'une application.

Définition 2.2 *Dans EUCLIDE, toute préoccupation est définie par un ensemble d'unités logicielles. La nature d'une préoccupation n'est pas imposée. Elle peut être de nature fonctionnelle, technique ou encore être complétement indépendante à l'application.*

Le concept de préoccupations [OSS 00] est volontairement large afin de recouvrir par exemple, la réalisation d'un concept ou d'un objectif. Ainsi, la définition d'une préoccupation se base sur un ensemble d'éléments qui définit la réalisation de ce concept. Nous qualifions de fonctionnelle, toute préoccupation qui s'intéresse à la réalisation d'une fonctionnalité métier d'une application. De ce fait, toute définition

d'une préoccupation fonctionnelle se constitue de l'ensemble des unités de modélisation et de programmation qui la caractérise. Par exemple, parmi ces unités nous pouvons retrouver un cas d'utilisation, un diagramme de séquences, un diagramme de classes et des classes Java propres à la fonctionnalité.

La mise en place de l'ensemble de ces définitions a été inspirée par l'approche des hyperespaces. Ces règles imposent un cadre de modélisation particulier et caractérisé par la théorie des ensembles. En nous inspirant de l'approche des hyperespaces, nous avons enrichi ces axiomes par la définition d'EUCLIDE qui permet la structuration d'un sous-ensemble U selon le principe de la SoC.

Définition 2.3 *EUCLIDE est défini par le triplet* (R, P, F) *où :*

– R est un repère, défini par un ensemble d'axes, repris de la géométrie euclidienne. Chaque axe de R définit un domaine de préoccupation qui permet le regroupement des préoccupations de même nature.

*– P est **l'ensemble des préoccupations** identifiées pour la modélisation d'une application.*

*– F est un **ensemble de fonctions caractérisant les préoccupations** identifiées. Chaque fonction est une mesure des unités logicielles qui font partie de la définition d'une préoccupation.*

Suite à nos premières expérimentations, nous avons identifié le domaine des fonctionalités, des composants et des modèles. Le domaine des fonctionnalités regroupe des préoccupations fonctionnelles. Leur réunion donne lieu aux trois axes des fonctionalités, des composants et des modèles. Bien sûr, selon les besoins des concepteurs, le nombre des axes peut varier,

La figure 1 montre une instance d'EUCLIDE et résume les notions introduites. Elle illustre notre intention de répartition des unités logicielles en termes de préoccutions. Ici, trois domaines de préoccupations sont définis et sont représentés par un axe. Par exemple, nous resençons le domaine des fonctionnalités, des modèles et des composants. Chaque domaine se partage la définition d'un ensemble de préoccupations. Ainsi le domaine des modèles a la responsabilité de définir les préoccupations de la spécification des besoins, de l'analyse et de la conception.

Ces définitions étant posées, nous pouvons revenir sur le concept de préoccupation en constatant qu'il repose dans EUCLIDE, sur trois points de vues complémentaires :

– Une vue sémantique : une fonction réalise la sélection des unités logicielles participant à la définition de la préoccupation. Nous qualifions cette définition de sémantique, car nous nous attachons au sens des unités logicielles avant de les intégrer dans la définition d'une préoccupation. Il s'agit de vérifier la cohérence de la définition d'une préoccupation,

– Une vue structurelle : un plan euclidien relève les unités logicielles qui participent à la définition d'une préoccupation et leur donne par là même un contexte. Ce

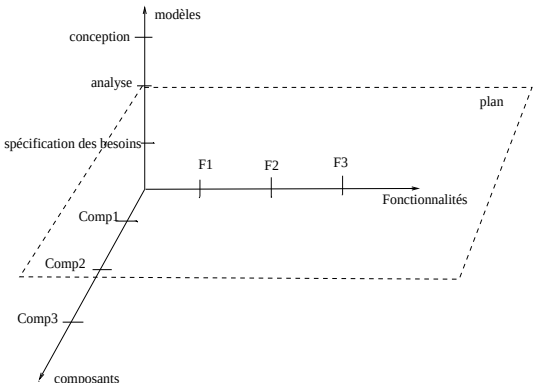

Figure 1. *Une instance d'EUCLIDE*

contexte fournit un ensemble de règles qui permettent de relier les unités logicielles. L'ensemble de règles définit la traçabilité que nous voulons mettre en œuvre,

– La graduation sur un axe de notre repère euclidien permet de localiser la définition de la préoccupation par rapport à une autre. Elle met en valeur le domaine de préoccupation auquel elle appartient.

Nous avons pour l'instant présenté EUCLIDE comme un modèle de spécification et de séparation des préoccupations. Il intègre les outils pour réaliser plusieurs partitions des éléments de définition d'une application. Nous verrons par la suite qu'il ne se réduit pas à ces seuls atouts. La section suivante introduit l'utilisation de ce modèle par la spécification euclidienne d'une application.

2.2. *Modélisation logicielle dans EUCLIDE*

Nous définissons maintenant la modélisation d'une application, en termes d'unités logicielles et de préoccupations au sein EUCLIDE.

Définition 2.4 *Dans EUCLIDE, la modélisation d'une application est donnée par le couple* (U, T)

– U *est* ***l'ensemble des unités logicielles propres à une application****. Comme nous l'avons dit, nous n'imposons aucune règle sur la nature de ces unités logicielles. L'objectif étant de manipuler l'ensemble complet des éléments qui participent de près ou de loin à la définition de notre application.*

– T *est* ***une matrice de structuration logicielle****. Cette matrice détermine la façon dont les unités logicielles sont organisées par rapport aux graduations sur un axe.*

La figure 2 fournit un exemple d'une matrice de structuration logicielle. Elle établit la classification des unités logicielles selon deux domaines de préoccupations :

préoccupations fonctionnelles / préoccupations des modèles	fonctionnalité F1
spécification des besoins	cas d'utilisation scénarios diagramme de séquence système
analyse	diagramme de séquence diagramme d'objets métier diagramme de collaboration
conception	diagramme de classes diagramme de séquence diagramme de composants

Figure 2. *Exemple d'une matrice de structuration logicielle*

le domaine des fonctionnalités et celui des modèles. Ainsi, les éléments logiciels tels que le cas d'utilisation, les scénarii et le diagramme de séquence système modélise la fonctionnalité $F1$ et font également partie du modèle de spécification des besoins. Nous pouvons aussi interpréter cette matrice de la façon suivante : les préoccupations du modèle d'analyse et de la fonctionnalité $F1$ se basent sur les unités logicielles des diagrammes de séquence, d'objet métier et de collaboration.

Grâce à cette modélisation, toutes les unités logicielles inhérentes à une application sont réunies au sein d'un même et unique modèle d'accueil. Les propriétés de ce modèle permettent une structuration claire et visuelle de l'ensemble des unités. Selon le domaine de préoccupations, une application modélisée dans EUCLIDE offre plusieurs compositions logicielles de préoccupations. Par exemple, si le lecteur ne considère que le domaine des fonctionnalités, la réunion des préoccupations définies procure une composition d'ordre fonctionnelle de notre application. De même, si le lecteur s'intéresse au domaine des composants, EUCLIDE offre alors une description de l'application du point de vue des composants. Comme dans l'approche initiale de la séparation multi-dimensionnelle des préoccupations [OSS 00], EUCLIDE met en œuvre une solultion contre *la tyrannie de la composition dominante* [TAR 99].

3. Définition de la traçabilité

Nous sommes conscients que la réalisation d'une évolution applicative intègre une étape de connaissance de l'existant. Or, l'expérience montre que bien souvent cette étape est compromise par le manque de clarté et d'organisation des informations mises à disposition. Par exemple, il est difficile de retrouver les éléments de modélisation associés à un module de code. L'absence de traçabilité entre les unités logicielles nuit à l'évolutivité des applications. La qualité des modifications s'en trouve, par conséquent affectée. EUCLIDE propose de répondre à ce manque en reliant ces unités logicielles par la notion de traçabilité fonctionnnelle qui définit le cycle de réalisation d'un cas

d'utilisation jusqu'à son implémentation. Nous montrons dans la suite de cette section un exemple de traçabilité fonctionnelle et nous fournissons une explication sur son intégration dans EUCLIDE.

3.1. *La traçabilité fonctionnelle*

La traçabilité fonctionnelle est la plus utilisée dans le domaine de l'ingénierie des logiciels. Définie par le processus conduit par les cas d'utilisation [JAC 94], ce type de traçabilité dirige le développement des applications. Elle s'assure que chaque module développé à chaque étape répond aux spécifications de la fonctionnalité exprimée. Ainsi, l'étape de la spécification des besoins définit le cas d'utilisation correspondant et les scénarii associés. L'étape de l'analyse définit les diagrammes de collaboration et de séquence associés au cas d'utilisation ; un diagramme de classe d'ordre métier est également implanté à cette étape d'analyse. Enfin, l'étape de conception permet de rentrer dans les détails du diagramme de classe, du diagramme de séquence et apporte un diagramme des composants. La traçabilité fonctionnelle garde une trace de la réalisation d'une fonctionnalité. Elle s'intéresse aux définitions successives d'un cas d'utilisation au travers des étapes de développement, et basées sur des diagrammes UML. Nous avons intégré la traçabilité fonctionnelle dans EUCLIDE afin de préparer la réalisation d'une évolution applicative.

Nous proposons de modéliser la traçabilité fonctionnelle à l'aide du métamodèle de la figure 3. Ce métamodèle relève les éléments intervenants lors de la réalisation d'un cas d'utilisation. Il les place dans leur contexte de définition, autrement dit par rapport aux modèles qui les intégrent. Ainsi, le modèle de spécification des besoins est un modèle comprenant un modèle de cas d'utilisation, des diagrammes de séquence système et des scénarii. Chaque modèle de cas d'utilisation est composé d'un certain nombre de cas d'utilisation dont la définition s'appuie sur un certain nombre de scénarii. Le complément d'information apporté par un élément de modélisation est matérialisé par la relation "completeS". Le passage d'un niveau d'abstraction vers un autre est matérialisé par la relation "realise". Cette relation modélise à la fois le passage d'une étape du développement logiciel vers une autre et la correspondance directe entre les éléments de modélisation qui permettent ce passage. Par exemple, dans le processus conduit par les cas d'utilisation, il est d'usage de définir un diagramme de collaboration à partir de la définition du cas d'utilisation correspondant. La même remarque est applicable pour le diagramme de classes issu du diagramme de collaboration associé. De manière claire et fidèle, notre métamodèle énonce les éléments de modélisation intervenants lors du processus de développement d'une application. Il définit également les relations existantes entre ces éléments de modélisation et souligne le rôle endossé par chacun de ces éléments lors du développement d'une application.

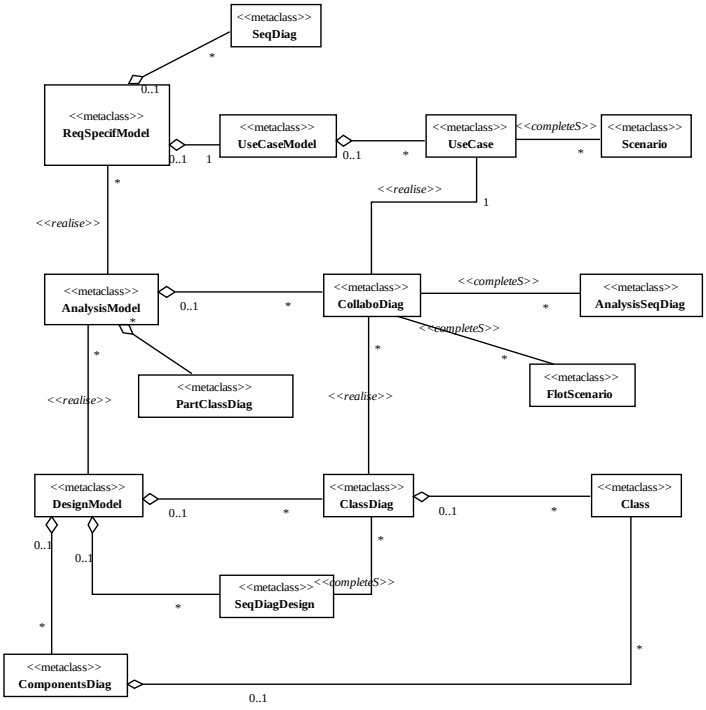

Figure 3. ***Définition de la traçabilité fonctionnelle***

3.2. *Intégration de la traçabilité fonctionnelle dans EUCLIDE*

En combinant les définitions de la traçabilité fonctionnelle et d'une préoccupation fonctionnelle A, la mise en œuvre de la fonctionnalité A peut être décrite étape par étape. Ainsi, il est possible de répertorier l'ensemble des éléments de modélisation, responsables de la définition de la fonctionnalité A et de dire comment ils interviennent au cours du proccessus de développement et de définir la traçabilité sur ces éléments. Pour ce faire, la combinaison des définitions est basée sur la notion de plan de notre modèle euclidien. En effet, jusqu'à présent nous avons vu qu'un plan associé à une préoccupation permet la capture et la mise en valeur des éléments logiciels qui participent à la définition de la préoccupation. Nous associons maintenant à chaque plan, la définition d'un contexte qui implante la définition de la traçabilité fonctionnelle à un ensemble donné d'unités logicielles. Par l'intermédiaire de ce contexte, l'ensemble des unités logicielles devient organisé et agencé selon la définition de la traçabilité fonctionnelle.

La figure 4 est importante car elle présente le métamodèle d'EUCLIDE. Ce métamodèle définit la structure de notre modèle en termes d'axes, de plans et de graduation.

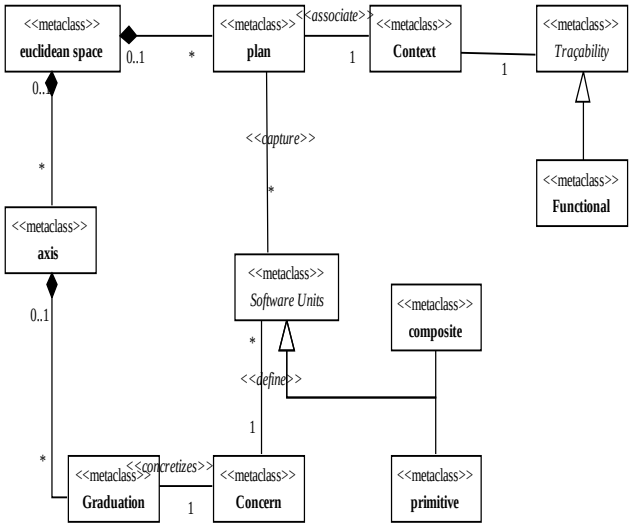

Figure 4. ***Métamodèle d'EUCLIDE***

Chaque graduation est associée à une préoccupation et chaque préoccupation admet un ensemble d'unités logicielles. Parmi les unités logicielles, nous en dénotons deux sortes : les primitives et les composites. Une unité logicielle est primitive lorsqu'elle est non décomposable, autrement dit lorsque toute décomposition lui fait perdre sa complétude. Par exemple, une méthode, un scénario, un cas d'utilisation sont des unités logicielles primitives. A l'inverse, toute unité logicielle dîte composite est décomposable. Ainsi, une classe, un modèle de cas d'utilisation, un modèle de composants sont des unités logicielles composites. Notre métamodèle définit également l'interaction d'un plan sur un ensemble d'unités logicielles, propre à une préoccupation. Chaque plan admet un contexte qui introduit la définition de la traçabilité fonctionnelle. Grâce à ce contexte, les unités logicielles jusqu'ici regroupées dans un simple ensemble de manière libre et désordonnée, deviennent composées et ordonnées selon la définition de la traçabilité fonctionnelle.

4. Application du modèle

Nous introduisons dans cette section une version simplifiée d'un exemple sur un guichet automatique bancaire. Nous nous sommes uniquement intéressés à la fonctionnalité de retrait d'argent. Nous présentons ici la modélisation et la traçabilité de cette fonctionnalité dans EUCLIDE.

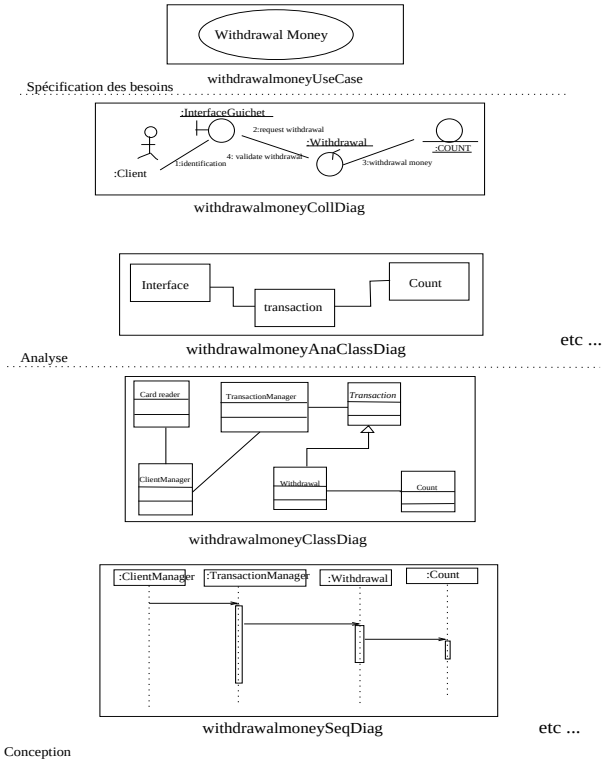

Figure 5. ***Processus traditionnel de développement***

4.1. *Modélisation de la fonctionnalité de retrait d'argent*

Un processus traditionnel de développement d'application fournit un ensemble d'éléments de modélisation.

La figure 5 regroupe une partie de ces éléments de modélisation, produits à travers le développement d'une application. Chaque étape fournit un certain nombre d'éléments de modélisation. L'étape de la spécification des besoins fournit la définition d'un cas d'utilisation, que nous nommerons pour les besoins de notre exemple, *WithdrawalmoneyUseCase*. L'étape de l'analyse définit le diagramme de collaboration, nommé *WithdrawalCollDiag* et un diagramme de classes simplifié, nommé *WithdrawalAnaClassdiag*. Enfin, l'étape de conception définit le diagramme de classes, nommé *WithdrawalClassDiag* et le diagramme de séquence *WithdrawalSeqDiag*. Par manque de place, nous n'avons pas fourni l'ensemble exhaustif des éléments de modélisation fournis à chaque étape du développement mais l'expérience montre que cet ensemble est à la fois conséquent et surtout difficile à gérer.

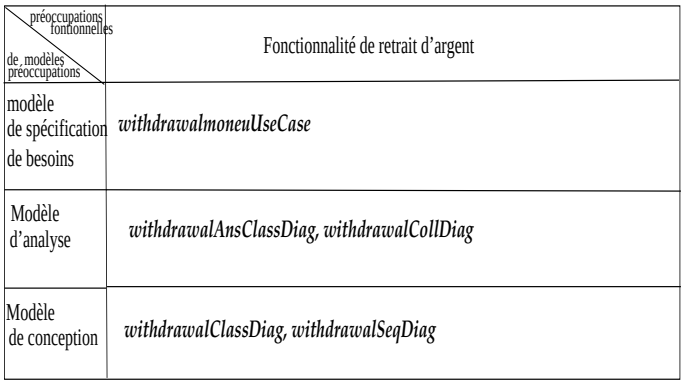

Figure 6. ***Matrice de structuration logicielle***

C'est pourquoi, en complément du processus de développement, EUCLIDE propose de regrouper et de structurer ces éléments de modélisation au sein d'un seul et unique modèle. Partant de cet objectif, la spécification euclidienne de la fonctionnalité de retrait d'argent, se réduit à la définition du couple (U, T) où :

– U est l'ensemble des unités logicielles à structurer dans EUCLIDE.
$U = \{$
$WithdrawalmoneyUseCase$,
$WithdrawalCollDiag$,
$WithdrawalAnaClassdiag$,
$WithdrawalClassDiag$
$WithdrawalSeqDiag\}$

– T est la matrice de structuration des unités logicielles. Elle définit l'organisation des unités en fonction des définitions de préoccupations. La figure 6 illustre cette distribution. Dans le cadre de notre exemple, cette matrice indique que l'organisation des unités logicielles a lieu selon deux domaines de préoccupations : celui des fonctionnalités et celui des modèles. Le domaine des fonctionnalités comprend une seule préoccupation fonctionnelle, la fonctionnalité du retrait d'argent et le domaine des modèles admet trois préoccupations, le modèle de la spécification des besoins, le modèle d'analyse et le modèle de conception. Cette matrice relève essentiellement les unités logicielles qui participent à la définition de certaines préoccupations. Par exemple, les préoccupations du retrait d'argent et du modèle de la spécification des besoins fondent leurs définitions sur l'unité logicielle *WithdrawalmoneyUseCase*.

Précisons aussi que cette matrice n'est pas dédiée à la structuration logicielle d'une unique et seule fonctionnalité. Elle est responsable de la structuration logicielle de l'ensemble complet des fonctionnalités d'une application. En ce qui concerne notre exemple, par manque de place, nous n'avons pas integré les unités relatives aux fonctionnalités de virement d'argent ou de consultation de compte mais elles auraient dû avoir leur place au sein de la matrice de la figure 6.

Venant s'intercaler au côté d'un processus de développement, EUCLIDE assure le regroupement et la structuration des éléments logiciels définis à chaque étape d'un développement logiciel. La définition et la représentation de cette organisation logicielle facilitent la compréhension globale de l'application et préparent tout mécanisme d'évolution. La préparation de tels mécanismes passe aussi par la définition de la traçabilité fonctionnelle du retrait d'argent qui nous rend compte de la mise en œuvre de la fonctionnalité.

4.2. *Traçabilité de la fonctionnalité de retrait d'argent*

Alors que dans un processus d'analyse classique, la définition de la traçabilité fonctionnelle du retrait d'argent est implicite, EUCLIDE permet de l'expliciter. Pour cela, il propose l'utilisation d'un plan introduisant la notion de traçabilité fonctionnelle et appelé *plan de traçabilité fonctionnelle* pour venir capturer et lier les unités logicielles de la fonctionnalité "retrait d'argent". Son rôle est de proposer un contexte de définition de la fonctionnalité de retrait d'argent et d'assurer un assemblage cohérent des unités logicielles.

De manière concrète, la définition de la traçabilité fonctionnelle du retrait d'argent est obtenue par un mécanisme d'instanciation du métamodèle défini précédemment (voir figure 3). Faisant partie de nos perspectives, nous comptons ajouter un mécanisme de validation du modèle afin de vérifier que toutes les instances définies par notre modèle sont bien présentes dans l'ensemble des unités logicielles capturées par un plan euclidien.

5. Travaux connexes

Les applications basées sur le principe de l'AOP [KIC 97] constituent une solution non négligeable pour mettre en œuvre l'évolutivité des applications. Basé sur le principe SoC, ce paradigme de programmation permet la construction d'applications adaptables et donc facilement évolutives. Cependant, bien que cette approche soit prometteuse, elle reste propre à l'étape de programmation et délaisse complétement toutes les unités de définition fonctionnelle du programme. Toute évolution fonctionnelle devient nettement moins évidente à réaliser car il ne reste plus aucune trace de la définition initiale de la fonctionnalité à modifier. La plupart des éléments de modélisation sont regroupés dans des outils tels que Objecteering [SOF] ou Rational Rose [TIG] qui s'avèrent le plus souvent inutiles pour retouver par exemple le cas d'utilisation qui est réalisé par tel ou tel diagramme UML. Dans ces cas, bien qu'un environnement soit proposé pour la visualisation globale des unités de modélisation, les notions manquent pour relier ces unités logicielles. La plupart du temps, elles sont regroupées sans aucun ordre et de façon aléatoire. EUCLIDE exploite la notion de traçabilité fonctionnelle afin de leur donner cet ordre manquant. En plus d'établir une modélisation logicielle sur le principe de la SoC, EUCLIDE assure la répercution de cette organisation à travers les étapes de cycle de vie d'un logiciel.

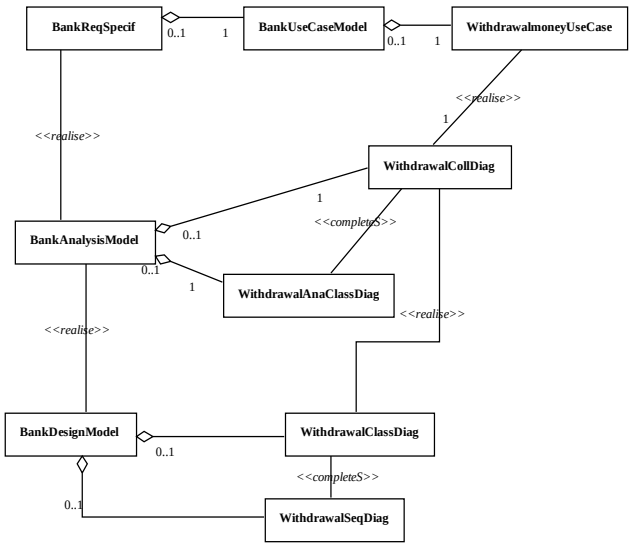

Figure 7. ***Définition de la traçabilité fonctionnelle du retrait d'argent***

Enfin, notre démarche prend sa source dans l'approche de H.Ossher et P. Tarr d'IBM. Développée en 1999, la séparation multidimensionnelle des préoccupations (connu par MDSoC)[TAR 99] est mise en œuvre par l'approche des hyperespaces. Cette approche met en place un vocabulaire et un modèle pour la spécification et la manipulation des préoccupations. Dans la même optique que la nôtre, cette approche propose une modélisation d'une application en termes de préoccupations et une manipulation de cette dernière par recombinaison des préoccupations. Récemment, cette même équipe a fait la présentation du projet CME pour *Concern Manipulation Environment* [TAR 98]. Ce projet développe un support pour le développement d'applications orientées aspect et promet sa maintenance durant tout son cycle de vie.

Notre modèle semble proche de la proposition d'IBM. Mais, bien que nous nous soyons inspiré de l'approche des hyperespaces, nous divergeons de la solution d'IBM, par notre volonté d'offrir un environnement dédié à l'assistance et à la réalisation des évolutions applicatives. Plus précisément, nous visons la description des potentiels de flexibilité de chaque fonctionnalité et de chaque service technique d'une application. Pour ce faire, nous proposons un modèle qui regroupe et organise les unités de définition d'une application selon des domaines de préoccupations et qui fournit la définition complète des fonctionnalités grâce à la notion de traçabilité. L'approche de l'équipe d'IBM vise plutôt la conception, mais surtout le développement d'applications définies par une orientation aspect. Si notre volonté est la connaissance et la description selon une structuration en préoccupations des éléments de définition d'une

application, leur volonté s'attache plutôt à la définition de préoccupations directement dégagées depuis l'architecture logicielle. Tandis que nous voulons manipuler tous les éléments logiciels définis depuis la spécification des besoins jusqu'à la conception, ils préfèrent plutôt s'occuper actuellement de la manipulation de code.

Avec EUCLIDE, nous ne visons pas à remplacer les processus de développement d'applications qui ont prouvé leur force, notre ambition cible l'expression de propriétés, telles que la flexibilité ou le couplage des éléments logiciels liées à l'évolutivité d'une application. Avec CME, l'equipe de chercheurs d'IBM ambitionne un support pour le développement d'une nouvelle génération d'applications.

6. Conclusion et perspectives

Nous avons présenté EUCLIDE, un modèle pour la spécification, la séparation et la traçabilité des préoccupations. EUCLIDE se présente comme la première brique de base d'un environnement centré sur l'évolutivité des applications. Sa définition est basée sur la mise en oeuvre des objectifs suivants :

– La définition d'un modèle qui prenne en compte tous les éléments logiciels d'une application depuis l'étape de la spécification des besoins jusqu'à l'étape de la conception,

– La remodélisation d'une application en termes de préoccupations afin de proposer une struturation claire de ses composants logiciels et favoriser la maintenance des applications grâce à la notion de traçabilité.

Ainsi, EUCLIDE permet la visualisation globale de unités de définition d'une application, une structuration logicielle en termes de préoccupations et le suivi de la mise en œuvre d'une fonctionnalité grâce à la notion de traçabilité.

A terme, EUCLIDE fera partie d'un atelier de génie logiciel : un environnement qui analyse les potentiels d'évolutivité d'une application, qui les simule et les réalise. Pour ce faire, la prochaine étape nous amène à l'intégration d'EUCLIDE au côté d'un processus de développement d'applications. Il s'agira notamment de déterminer les préoccupations d'une application en fonction des besoins du développement. Enfin, une troisième étape de cette démarche consistera à déterminer des modèles de manipulation de notre application afin de simuler ces adaptations pour ensuite en faire des études d'impacts. Pour ce faire, nous pensons nous baser sur la théorie des transformations de modèles. Une quatrième étape verra l'application de notre environnement sur une étude de cas de la SSII Norsys.

7. Bibliographie

[I.S 98] I. SOMMERVILLE P. SAWYER S. V., « Viewpoints for requirements elicitation : a pratical approach », ICRE 98, 1998.

[JAC 94] JACOBSON I., CHRISTERSON M., JONSSON P., OVERGAARD G., *Object-Oriented Software Engineering. A use case driven approach*, Addison - Wesley, 1994.

[KIC 97] KICZALES G., LAMPING J., MENHDHEKAR A., MAEDA C., LOPES C., LOINGTIER J.-M., IRWIN J., « Aspect-Oriented Programming », AKŞIT M., MATSUOKA S., Eds., *Proceedings European Conference on Object-Oriented Programming*, vol. 1241, p. 220–242, Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[MOR 02] MOREIRA A., ARAÚJO J., BRITO I., « Crosscutting quality attributes for requirements engineering », *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, 2002, p. 167–174.

[OSS 00] OSSHER H., TARR P., « Multi-Dimensional Separation of Concerns and The Hyperspace Approach », *Proceedings of the Symposium on Software Architectures and Component Technology : The State of the Art in Software Development*, Kluwer, 2000.

[PAR 72] PARNAS D. L., « On the criteria to be used in decomposing systems into modules », *Commun. ACM*, vol. 15, n° 12, 1972, p. 1053–1058, ACM Press.

[PAW 04] PAWLAK R., SEINTURIER L., DUCHIEN L., FLORIN G., LEGOND-AUBRY F., MARTELLI L., « JAC : An Aspect-Based Distributed Dynamic Framework », SPE. 34(12) :1119-1148., October 2004.

[RAS 03] RASHID A., MOREIRA A., ARAUJO J., « Modularisation and composition of aspectual requirements », *Proceedings of the 2nd international conference on Aspect-oriented software development*, ACM Press, 2003, p. 11–20.

[SOF] SOFTEAM, « Objecteering », http ://www.objecteering.com.

[STE 02] STEIN D., HANENBERG S., UNLAND R., « A UML-based aspect-oriented design notation for AspectJ », *Proceedings of the 1st international conference on Aspect-oriented software development*, ACM Press, 2002, p. 106–112.

[SUZ 99] SUZUKI J., YAMAMOTO Y., « Extending UML with Aspects : Aspect Support in the Design Phase », *ECOOP Workshops*, 1999, p. 299-300.

[TAR 98] TARR P., OSSHER H., HARRISON W., « IBM Research CME : concern manipulation environment », http ://www.research.ibm.com/cme, 1998.

[TAR 99] TARR P. L., OSSHER H., HARRISON W. H., JR. S. M. S., « N Degrees of Separation : Multi-Dimensional Separation of Concerns », *International Conference on Software Engineering*, 1999, p. 107-119.

[TIG] TIGRIS.ORG, « ArgoUML », http ://argouml.tigris.org/.