Les Composants Logiciels et La Séparation Avancée des Préoccupations : Vers une nouvelle Approche de Combinaison

Mehdi Hariati

Laboratoire LRI, Université Badji Mokhtar-Annaba
BP 12, 23000 Annaba
ALGERIE

hamehdi23@yahoo.fr

Djamel Meslati

Laboratoire LRI, Université Badji Mokhtar-Annaba
BP 12, 23000 Annaba
ALGERIE

meslati_djamel@yahoo.com

RÉSUMÉ. L'approche basée composant et la séparation avancée des préoccupations constituent deux paradigmes importants pour le développement des systèmes logiciels. Bien que les deux paradigmes soient complémentaires et que la recherche de leur synergie soit une issue prometteuse, relativement peu de travaux sont actuellement dédiés à leur combinaison. Cet article présente un état de l'art comparatif des principaux travaux qui ciblent la synergie des deux paradigmes et propose une nouvelle approche de combinaison qui part du constat que tous les apports potentiels pouvant être tirés de cette combinaison sont étroitement liés à la bonne manipulation des aspects.

ABSTRACT. The component based approach and advanced separation of concerns are two important paradigms for software systems development. Although the two paradigms are complementary and looking for their synergy is a promising issue, only few research works are currently dedicated to their combination. This paper presents a comparative state of the art of the main research works that aim at the synergy of the two paradigms and proposes a new approach of combination that derives from the fact that all potential contributions that can be drawn, are tightly related with the well manipulation of aspects.

MOTS-CLÉS : Composant logiciel, séparation avancée des préoccupations, combinaison de paradigmes.

KEYWORDS: Software component, Advanced separation of concerns, paradigms' combination.

1. INTRODUCTION

Dans l'ingénierie des logiciels basée composant [7] le développement d'un logiciel se réduit à un assemblage de composants prédéfinis. Chaque composant joue un rôle spécifique dans le système. Il définit clairement les services qu'il offre et les services requis pour accomplir sa fonction. Le développement de logiciel basé composant augmente considérablement la fiabilité des systèmes puisque leur construction est basée sur des composants testés et certifiés. De même, les coûts du développement sont généralement réduits car une partie des composants sont simplement réutilisés. L'étape de maintenance se réduit, quant à elle, à un remplacement de composants, ce qui facilite sa tâche et favorise l'évolution du système.

Selon le paradigme de séparation avancée des préoccupations, un système est un ensemble de préoccupations fonctionnelles et extra-fonctionnelles (aspects). Les préoccupations fonctionnelles sont les fonctionnalités métiers que le système doit assurer, alors que les préoccupations extra-fonctionnelles sont des services dont le système a besoin pour effectuer ses fonctionnalités métiers. Comme exemple de préoccupations extra-fonctionnelles on peut citer la sécurité, la synchronisation, la gestion de la persistance, etc. Dans les approches de développement de logiciels, notamment dans le cas de la programmation orientée objet, il apparaît que les deux types de préoccupations sont enchevêtrés et le code relatif aux préoccupations extra-fonctionnelles est éparpillé dans le code fonctionnel. Cette situation augmente la complexité, empêche la réutilisation et gêne l'évolution des systèmes. La séparation avancée des préoccupations permet de séparer les parties extra-fonctionnelles des parties fonctionnelles d'un système. L'objectif escompté étant d'offrir une meilleure réutilisation, faciliter la maintenance, réduire la complexité des systèmes et augmenter leur évolutivité. Parmi les approches les plus utilisées, on cite : la programmation orientée aspect [8], la composition de filtres [9] et la séparation multidimensionnelle des préoccupations [10].

L'ingénierie des logiciels basée composant fait une séparation des préoccupations du système en des entités clairement définies, appelées composants. Ces composants réutilisables sont définis et composés ensemble. Cependant, cette décomposition entraîne l'éparpillement de certaines préoccupations dans différents composants du système, ces préoccupations qu'on qualifie d'entrecoupantes, empêchent la réutilisation, la maintenance et l'évolution des composants, et par conséquent l'évolution du système [1]. La séparation avancée des préoccupations de son côté, propose des approches offrant des mécanismes qui permettent de séparer les préoccupations entrecoupantes des préoccupations métiers. La combinaison de l'approche basée composant et de la séparation des préoccupations permet un

développement rapide des systèmes, basé sur des composants certifiés et dont les préoccupations non fonctionnelles sont factorisées en dehors de ces composants, ce qui permet d'avoir des systèmes évolutifs, adaptatifs et maintenables. Cependant, certains problèmes peuvent empêcher cette combinaison. Par exemple, dans le cas de la programmation orientée aspect, le fait que l'approche opère au niveau des objets, brise l'encapsulation des composants qui implémentent ces objets. Ainsi leur certification n'est plus garantie. Dans les sections suivantes, nous présentons, dans un premier temps, les approches de combinaison les plus connues et nous les comparons, et dans un second temps, nous présentons notre propre approche qui élimine certains problèmes.

2. CLASSIFICATION DES APPROCHES DE COMBINAISON

Les approches actuelles de combinaison, peuvent être classées en deux catégories : les approches symétriques et les approches asymétriques. Dans la suite, nous présentons trois exemples d'approches dans chacune des deux catégories. Notre choix s'est orienté vers les travaux les plus connus dans le domaine de la combinaison des deux paradigmes. Chacune des approches considérée traite la combinaison d'une manière plus ou moins différente des autres, ce qui permet de voir la combinaison selon des points de vue variés, et d'identifier le maximum de problèmes potentiels liés à cette combinaison.

2.1 Les approches asymétriques

Les approches asymétriques traitent les aspects et les composants du système de façon différente, à cet égard, nous présentons quelques travaux ci-dessous:

A- JAsCo : Dans cette approche, D. Suvée, Wim V et al proposent un nouveau langage d'implémentation orienté aspect appelé JasCo [2]. JasCo est adapté au modèle de composant Java beans [20] et introduit deux concepts : les aspect beans et les connecteurs. Le premier décrit le comportement qui interfère avec l'exécution d'un composant en utilisant des classes internes, appelées hook, dont la spécification est réutilisable indépendamment du contexte. Le second est utilisé pour déployer un ou plusieurs hook dans un contexte spécifique. Dans ce travail, un nouveau modèle de composant a été proposé en se basant sur la notion de trappes intégrées qui permettent d'interférer avec l'exécution normale d'un composant. JasCo est compatible avec le modèle de composant Java beans et permet l'ajout/retrait dynamique des aspects. JasCo résout partiellement le problème d'interaction des aspects conflictuels en permettant de les ordonnancer et de décrire des combinaisons d'aspects explicites et réutilisables.

B- Aspects non fonctionnels dans les applications basées composant : Ce travail a été proposé par F. Duclos et al, du laboratoire LSR-IMAG de Grenoble [3]. Ses objectifs sont la séparation des aspects non fonctionnels des composants eux-mêmes afin d'augmenter la réutilisation du composant et de l'aspect non fonctionnel, ainsi que la réduction de la complexité de programmation pour le fournisseur du composant. Pour cela, la technologie basée composant et la programmation orientée aspect ont été fusionnées, permettant ainsi aux concepteurs d'aspects de définir de nouveaux aspects ou services et aux utilisateurs d'aspects d'appliquer ces aspects ou services sur les composants sans la disponibilité du code du composant. L'approche propose deux langages, l'un pour les concepteurs d'aspects et l'autre pour leurs utilisateurs. L'intérêt étant de pouvoir utiliser des bibliothèques d'aspects pour certains aspects complexes et répétitifs et de développer uniquement ce qui est spécifique au domaine considéré.

C- Adaptations dépendantes du contexte et composants : Partant de la constatation qu'on ne peut anticiper toutes les propriétés non fonctionnelles du composant requises pour la composition et que les propriétés qualitatives ont tendance à être dures à modulariser en dehors des composants. T. Cottenier et E. Tzilla, de Illinois Institute of Technology, ont cherché à prendre la puissance expressive de la programmation orientée aspect pour adapter les composants à leur déploiement, composition, et contexte d'exécution [4]. En spécifiant explicitement les propriétés désirées des composants et des aspects, les auteurs visent à augmenter la capacité du raisonnement sur le tissage d'aspect (l'intégration des aspects et des composants pour produire un code exécutable), afin que la correction du composant raffiné puisse être vérifiée avec la correction du système, ce qui permet d'évaluer et de valider les conséquences du tissage d'aspects sur les composants.

2.2 Les approches symétriques

Contrairement aux approches déjà précitées, dans les approches symétriques, les aspects et les composants sont considérés comme entités uniformes, les principaux travaux entrepris dans cette direction sont les suivants :

A- Séparation avancée des préoccupations et évolution du composant : Le travail proposé dans [1], par Stanley M. et al, du centre de recherche T. J. Watson d'IBM, part du principe qu'une séparation avancée des préoccupations supporte une évolution flexible et bien structurée des systèmes qui justifie un investissement important dans leur modélisation. L'objectif de l'approche est l'application de la séparation avancée des préoccupations pour l'évolution du composant. Dans un premier temps, l'approche utilise un schéma Cosmos pour la modélisation des préoccupations logicielles puis

Hyper/J, un outil pour la composition, pour la combinaison des composants Java. L'approche a servi à modéliser une grande variété de préoccupations et de composer d'une manière flexible des versions différées de systèmes basés sur les préoccupations sélectionnées.

B- Approche symétrique unifiée : L'approche proposée par D. Suvée, De Fraine B et al [5] de l'université libre de Bruxelles, a pour but d'aboutir à une architecture de composants symétrique et unifiée qui traite les aspects et les composants comme des entités uniformes. A cette fin, un nouveau modèle de composant est introduit. Il n'offre pas de constructions spécialisées pour représenter les préoccupations entrecoupantes. En contrepartie, un langage de configuration expressif est fourni qui permet de décrire à la fois les interactions normales et orientées aspect entre les composants. Ce travail présente les recherches en cours par une plateforme nommée FuseJ qui constitue une première expérience pour la réalisation de cette architecture aspect/composant symétrique et unifiée.

C- Programmation orientée aspect sûr pour les composants : Dans ce travail, Nicolas Pessemier et al, du Laboratoire INRIA/LIFL [6]; montrent qu'une programmation orientée aspect (AOP) sûre peut être supportée par la programmation orientée composant (COP) en proposant un mécanisme pour contrôler l'ouverture du composant avec le respect des techniques de l'AOP. La proposition réconcilie la nature envahissante de AOP avec la propriété de boîte noire des composants. Pour cela, l'approche des modules ouverts [12] a été appliquée sur FAC [13], un modèle unifié pour les composants et les aspects. L'approche est capable d'ouvrir un composant pour l'AOP tout en gardant son contenu caché de l'extérieur. Ce compromis ouvre la voie à une intégration sûre pour AOP dans COP. Le terme sûr est pris dans le sens où l'intrusion de AOP est finalement gérée au niveau de chaque composant. De plus, dans le cas de FAC, l'ouverture d'un composant pour AOP peut être gérée dynamiquement et tient compte des besoins imprévus.

3. COMPARAISON ET CRITIQUES DES APPROCHES

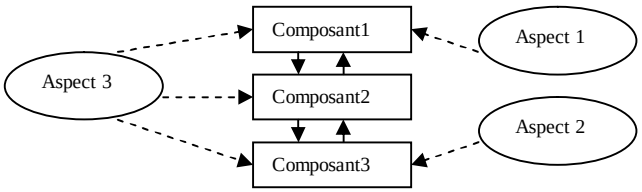

Figure 1. Les composants et les aspects du système.

Les approches symétriques traitent les aspects et les composants comme des entités uniformes, ce qui n'est pas le cas pour les approches asymétriques. D'après [6], l'avantage de l'unification entre les aspects et les composants réside dans la simplification des interactions entre les composants et les aspects ainsi que leurs évolution. Cependant, des insuffisances demeurent dans les travaux de la première ou de la deuxième catégorie. Contrairement à FuseJ [5], l'un des avantages de JasCo est qu'il offre des mécanismes permettant d'ordonnancer les comportements des aspects conflictuels ainsi que des stratégies de combinaison explicites et réutilisables, en vue de remédier aux problèmes d'interactions. De plus, JasCo laisse la priorité des aspects varier selon les applications. Dans certains travaux comme les approches [2] et [3], les caractères de remplacement sont utilisés dans les expressions des points de coupure (formes de prédicats utilisés pour capturer les emplacements de l'application des aspects au niveau des composants), cela peut entraîner l'application des aspects dans des emplacements inappropriés, et par conséquent, le comportement du système peut être altéré. L'approche [4] de son côté, s'attaque à ce problème en associant un profil au composant et à l'aspect, qui décrit leurs propriétés essentielles, ainsi, le raisonnement sur le processus du tissage devient possible et la prévisibilité est largement améliorée. L'une des insuffisances de l'approche [3] est qu'elle ne permet pas un ajout/retrait dynamique des aspects à l'exécution, contrairement à l'approche JasCo. Par conséquent, JasCo a une meilleure adaptation au contexte et aux changements imprévus. Cependant, la plupart de ces changements nécessitent d'écrire et de compiler un nouveau connecteur, ce qui cause un encombrement important et un risque d'erreur. De plus, le caractère dynamique et flexible de JasCo entraîne une dégradation considérable de performances. Un des points forts de l'approche [3] est le fait qu'elle applique le principe de la réutilisation de l'approche basée composant pour les aspects, en offrant des moyens pour définir et utiliser des librairies d'aspects. L'avantage de cela est que l'aspect certifié est défini une seule fois puis utilisé plusieurs fois. De plus, le niveau d'abstraction est plus élevé pour l'utilisateur d'aspect, ce qui facilite son travail. Dans certains cas, il est possible de déplacer la complexité de l'utilisateur au concepteur de l'aspect et vice-versa. Par exemple, pour les aspects très utilisés, il peut être pratique de sophistiquer la description de l'aspect afin de simplifier son utilisation. Similairement à FuseJ, l'approche [3] souffre encore de problèmes d'interaction d'aspects multiples, que les approches [2] et [6] sont parvenues à y remédier par l'introduction de mécanismes permettant de gérer et d'ordonnancer les différents aspects.

Dans l'approche [1], des coûts importants sont générés dans la modélisation des préoccupations. Néanmoins, ces coûts sont amortis dans le temps avec l'évolution du système. De plus, il est possible de composer plusieurs versions d'un même composant en fonction des préoccupations sélectionnées. Comme dans JasCo, l'approche [6] permet l'ajout/retrait des aspects à l'exécution, ce qui lui ouvre la voie à l'adaptation dynamique. Cependant, l'approche a des limitations avec des composants qui ne

supportent pas les techniques de la programmation orientée aspect. Dans une approche de combinaison, lorsque les aspects intra composants se reposent sur les détails d'implémentation internes des composants, ils créent des dépendances entre les composants et les aspects ce qui empêche la réutilisation et l'évolution du système, ce problème est connu comme le paradoxe de l'évolution de AOSD [14]. Par exemple, dans l'approche JasCo le problème n'est pas posé, puisque les aspects opèrent seulement sur les points exposés dans l'interface du composant. Ce n'est pas le cas pour l'approche [6] où les points de coupure du composant sont exposés explicitement dans l'interface du composant. Le tableau suivant récapitule notre comparaison des différentes approches en fonction de six critères que nous avons retenu. Le critère « Ajout/Suppression dynamique **»** reflète les capacités de l'approche en matière d'adaptation dynamique au contexte et d'évolution durant l'exécution. Le critère « variation de la priorité des aspects » indique la possibilité de changer l'ordre d'application des aspects. De même le critère « combinaison des aspects » indique la prise en charge de la gestion les problèmes d'interaction. La « réutilisation des aspects » fait référence à l'application de l'idée des librairies des composants dans le monde des aspects ce qui permet un gain considérable en temps, en efforts et en fiabilité. Notons que la réutilisation des aspects est systématiquement supportée par les approches symétriques, du moment que ces approches considèrent les aspects comme des composants. L' « effet négatif des aspects » permet de voir si l'approche souffre du problème engendré par les caractères de remplacement dans les expressions de points de coupure, pouvant entraîner l'application des aspects dans des emplacements inappropriés.

approches	**Asymétriques**			**Symétriques**		
Propriétés	**[2]**	**[3]**	**[4]**	**[1]**	**[5]**	**[6]**
Aspects et composants uniformes				✓	✓	✓
Ajout / Suppression dynamique	✓		✓		✓	✓
Variation de la priorité des aspects	✓					✓
La combinaison des aspects	✓		✓	✓		
La réutilisation des aspects		✓		✓	✓	✓
L'effet négatif des aspects	✓	✓			✓	

4. Notre approche

Dans le but de combiner l'approche basée composant et la séparation avancée des préoccupations, nous avons été amenés à nous focaliser dans notre proposition, d'un côté sur la programmation orientée aspect, pour sa puissance expressive, d'un autre côté sur le modèle de composant java beans en raison des avantages offerts par l'idée

des conteneurs. En effet, le programmeur ne se soucie que de la partie fonctionnelle lors du développement de son système.

4.1. Principe

Notre approche part du principe qu'un système est formé de *composants*, de *composants aspects* et d'un seul *composant gestionnaire*. Alors que les composants ont pour but d'assurer les fonctionnalités métiers du système, les composants aspects assurent les services extra fonctionnels. Le composant gestionnaire est l'élément central de l'approche, son rôle est la gestion de la combinaison.

Dans ce qui suit, nous détaillerons les concepts sur lesquels est basée notre approche:

a. Composant. C'est un composant Java beans classique constitué d'un ensemble de classes, contenant du code fonctionnel, et qui joue un rôle fonctionnel spécifique dans le système.

Tous les composants du système doivent être liés au composant gestionnaire à travers les interfaces fournies et les interfaces requises.

b. Composant aspect. Tout comme un composant, un composant aspect est un composant java beans lié au composant gestionnaire par les interfaces requises et les interfaces fournies. Cependant, le composant aspect contient du code représentant un aspect dans le système. Par exemple : transaction, sécurité, etc.

c. Composant gestionnaire. C'est un composant java beans où tous les composants et les composants aspects se réfèrent, c'est l'élément central du système, son rôle est primordial, il assure les fonctionnalités suivantes:

- Il gère toutes les liaisons pouvant exister entre les différents éléments du système ; entre un composant aspect et un composant, entre un composant et un autre composant, et entre un composant aspect et un autre composant aspect. Les liaisons entre les composants aspect et le composant gestionnaire peuvent être activées et désactivées durant l'exécution, ce qui permet un ajout et une suppression dynamique des aspects.

- Il gère l'ordre d'exécution des aspects activés, en attribuant une priorité pour chaque aspect, il permet leur exécution selon leur niveau de priorité. Notons que la priorité de chaque aspect varie selon le contexte.

- Il gère les problèmes d'interaction des aspects conflictuels à travers des stratégies de combinaison de ces aspects. Par exemple, dans le cas où deux aspects ne peuvent pas s'exécuter simultanément, le composant gestionnaire gère leur exécution en différé.

Le composant gestionnaire est un composant java beans, contenant d'un côté un ensemble de classes assurant les fonctionnalités citées ci-dessus, d'un autre côté un ensemble d'aspects. Chaque aspect contient une expression d'un point de coupure (pointcut) et une consigne (advice), cependant, la consigne ne contient pas le code de l'aspect, mais plutôt un appel au composant aspect qui contient le code représentant l'aspect. En effet, le tissage de la partie fonctionnelle avec la partie extra fonctionnelle s'effectue totalement au sein du composant gestionnaire, entre les objets représentant les composants du système et les objets aspects représentant les composants aspects.

Notre approche est capable de gérer les aspects inter composants en interceptant les interactions entre les différents composants, ainsi que les aspects intra composant en interceptant les interactions entre les objets d'un même composant, par le biais d'une interface spécifique que possède chaque composant. Cependant, l'interception des appels entre les objets d'un composant peut entraîner la violation de l'encapsulation des composants. Comme solution, nous identifions toutes les propriétés d'un composant au moment du développement du système, ainsi, au moment de l'exécution, à chaque application d'un aspect sur un composant une vérification des propriétés de ce dernier est effectuée, dès qu'une propriété est brisée, une action appropriée sera déclenchée.

Notons que la définition des propriétés des composants est effectuée lors de la composition du système au cœur même du composant gestionnaire à travers ses interfaces. De plus, ce composant gestionnaire est chargé d'assurer la vérification de ces propriétés tout au long de l'exécution du système.

4.2. Architecture de l'approche

La figure 2 illustre l'architecture de notre approche et montre le rôle central que joue le composant gestionnaire.

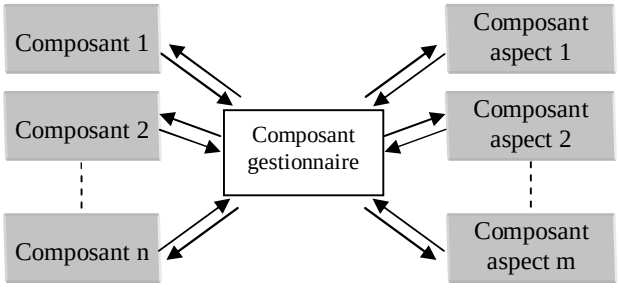

Figure 2. Architecture de l'approche

Force est de constater que tous les éléments du système sont reliés au composant gestionnaire, en effet, ce dernier gère l'ensemble des interactions entre composants et composants, puis entre composants aspects et composants, et enfin celles liant composants aspects et composants aspects.

4.3. Caractéristiques de l'approche

Notre approche est symétrique, en d'autres termes elle considère les aspects et les composants comme des entités uniformes.

Elle permet au système de s'adapter au contexte et à l'environnement, en permettant l'ajout et la suppression des aspects au moment de l'exécution.

L'approche permet d'ordonner l'exécution des aspects suivant leurs niveaux de priorité, cette dernière varie selon le contexte, ce qui offre au système une meilleure adaptation à son environnement.

Elle permet aussi de gérer les problèmes d'interaction des aspects conflictuels en recourant à des politiques de combinaison de ces derniers.

L'approche permet la gestion des aspects inter composant en interceptant les interactions entre les composants du système, ainsi que la gestion des aspects intra composant en interceptant les interactions entre les objets d'un même composant.

L'approche elle-même présente l'avantage d'être évolutive car pouvant recevoir de nouvelles fonctionnalités au niveau du composant gestionnaire.

4.4. Les apports de l'approche

Le fait que l'approche est symétrique lui offre un haut niveau de réutilisation pour les aspects comme pour les composants. En outre, les aspects du système ne perdent pas leurs identités ce qui facilite grandement la maintenance et favorise l'évolution du système.

Bien que l'approche utilise la puissance expressive de la programmation orientée aspect, la nature envahissante de cette dernière est cependant gérée au niveau des composants, ainsi leurs encapsulation est préservée. Par conséquent, la qualité du service demeure garantie.

Contrairement à quelques approches précitées, notre approche n'impose pas son propre modèle de composant, mais utilise un modèle de composants connu (i.e. java beans), ce qui ouvre la voie, parfois (sauf dans le cas du traitement des aspects intra composants, ces composants devant être dotés nécessairement d'une interface spécifique permettant cela), à l'utilisation des composants disponibles sur le marché et

soulage le programmeur de la tâche d'apprendre les spécifications d'un nouveau modèle imposé par une approche ou une autre.

L'approche n'impose pas un nouveau langage, le développement du système est relativement facile, car le développeur se concentre seulement sur la sélection des composants et des aspects, ainsi que sur la configuration du composant gestionnaire.

Pour tout résumer, notre approche de combinaison cherche à rassembler à la fois, les avantages des travaux existants et les solutions portant remède à leurs insuffisances ; en plus, elle n'impose ni un nouveau langage ni un nouveau modèle de composants.

4.5. Exemple

Imaginons un système où la partie métier contient deux fonctionnalités : la première calculant un bulletin de paye et la deuxième assurant la gestion de l'impression. La partie extra fonctionnelle contient un aspect *sécurité* chargé de la vérification des droits d'accès à la fonction impression où seuls les utilisateurs autorisés peuvent imprimer un bulletin de paye.

A un niveau plus concret notre système se compose:

- De deux composants : le premier assure le calcul d'un bulletin de paye, le deuxième gère l'impression.

- D'un composant aspect : implémente la vérification des droits d'accès au composant impression.

- D'un composant gestionnaire.

Notre exemple est illustré par la Figure 3.

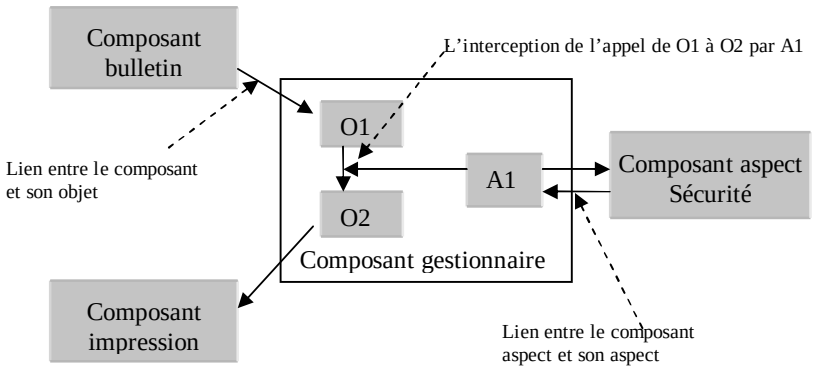

Figure 3. Vérification des droits d'accès pour l'impression.

Cet exemple met en évidence le traitement des aspects inter composants par notre approche, lequel consiste à intercepter les appels entre les composants du système et appliquer le traitement implémenté par un composant aspect.

Le composant gestionnaire contient entre autres une *classe composant* représentant un *composant* ainsi qu'une *classe aspect* représentant un *composant aspect*.

Au moment de l'exécution, au niveau du composant gestionnaire, d'un côté une instance de la *classe composant* est créée pour chaque *composant*. Dans notre exemple l'objet *O1* pour le composant *bulletin* et l'objet *O2* pour le composant *impression*. Chaque objet a pour but de gérer et de représenter un composant durant toute l'exécution du système. D'un autre côté une instance d'une *classe aspect* est créée pour représenter et gérer chaque *composant aspect* durant toute l'exécution du système. Dans notre exemple l'aspect *A1* pour le composant aspect *sécurité*.

Lorsque le composant bulletin invoque le composant impression, il fait appel à l'objet *O1* qui le représente, l'objet *O1* à son tour fait appel à l'objet *O2*, et c'est à ce moment là où l'aspect *A1* intercepte l'appel pour vérifier les droits de l'utilisateur. L'aspect *A1* invoque le composant aspect *sécurité*, ce dernier vérifie les droits d'accès pour autoriser ou rejeter l'appel du composant bulletin au composant impression.

4.6. Démarche d'implantation.

Pour l'implantation de notre approche, nous avons opté pour la plateforme éclipse [17], connue pour ses principes de modularité et d'extensibilité, mis en pratique par le concept de plug-ins [18].

Préalablement à la construction d'un système les plug-ins suivants sont ajoutés à la plateforme :

- Le plug-in AJDT qui permet à la plateforme d'exécuter des programmes orientés aspect [18].

- Le plug-in WTP qui permet de développer éventuellement d'autres java beans (composants ou composants aspects) [19].

- Le plug-in composant gestionnaire, pour intégrer le composant gestionnaire au niveau de la plateforme, permettant ainsi son instanciation au moment de la construction du système.

Ainsi, le développement d'un système se réduit à :

- Une sélection des composants et des composants aspects, préconstruits et disponibles dans des librairies.

- La création d'une instance du composant gestionnaire, ainsi que sa configuration.

Le composant gestionnaire est un composant java beans classique préconstruit, sa configuration est autorisée à travers ses interfaces, elle inclut principalement :

- L'ajout et la connexion de chaque composant au composant gestionnaire.

- L'ajout et la connexion de chaque composant aspect au composant gestionnaire. Notons qu'il faut préciser le lieu et le moment de l'exécution de chaque composant aspect.

- La définition, éventuellement des politiques de combinaison et des contraintes d'exécution pouvant exister entre les aspects.

- La spécification des propriétés des composants qui représentent leurs certifications.

5. TRAVAUX SIMILAIRES

Parmi les travaux ayant déjà abordé la classification des approches de combinaison nous pourrons citer celui présenté dans [4] cite quelques approches de combinaison, les distingue en terme : d'expressivité des aspects, d'oubli des composants (caractérisant le degré avec lequel le développeur prend en compte les aspects lors du développement des composants) et de la transparence des composants (reflétant la transparence avec laquelle un aspect peut être ajouté à un composant). N. Pessemier, propose dans [11] un certain nombre de critères en vue de comparer les approches de combinaison. Deux parmi ces critères présentent une analogie avec les nôtres, la symétrie d'éléments et la symétrie de placement qui signifient respectivement l'unification des aspects avec les composants et la réutilisation des aspects. G. Söldner et R. Kapitza présentent, dans [15], une vue d'ensemble de quelques approches de combinaison en utilisant des critères de comparaison plus ou moins généraux telles que la propriété de boîte grise et l'adaptation. J. Noyé et al. proposent dans [16] une classification des approches et leur comparaison en se basant sur trois catégories de critères : caractéristiques de l'approche, définition d'aspects et mécanisme de tissage. Dans notre travail, les critères de comparaison des différentes approches de combinaison se basent principalement sur les aspects étant donné que tous les apports potentiels sont étroitement liés à la bonne manipulation de ces derniers par l'approche.

En comparant notre approche avec celles citées dans l'état de l'art, on trouve qu'elle traite les aspects et les composants uniformément, par conséquent, elle permet la réutilisation des aspects, ce qui n'est pas le cas pour les approches [2] et [4]. Contrairement aux approches [1] et [3], notre approche permet l'adaptation dynamique au contexte et l'évolution durant l'exécution. De plus, elle autorise le changement de l'ordre de l'application des aspects selon leurs priorités ; notons l'absence de ce mécanisme dans chacune des approches [1], [3], [4] et [5]. La combinaison des aspects

n'est pas fournie dans les travaux [3], [5] et [6], par contre, notre approche permet d'élaborer de telles combinaisons. Quelques approches comme [2], [3] et [5] souffrent de l'effet négatif des aspects, qui signifie leurs applications dans des endroits inappropriés, une insuffisance que notre approche est parvenue à y remédier.

6. CONCLUSION ET TRAVAUX FUTURS

Dans cet article nous avons, dans un premier temps, présenté un état de l'art sur les travaux de combinaison de l'approche basée composant et de la séparation avancée des préoccupations. Nous avons réparti en deux catégories, approches symétriques et approches asymétriques, selon la façon dont ils traitent les aspects vis-à-vis des composants. La recherche d'une synergie des deux paradigmes apporte des avantages pour l'un ou pour l'autre. Cependant, les travaux de combinaison cités souffrent encore d'insuffisances et des travaux restent à faire dans plusieurs directions. Notons aussi que la diversité au sein de chaque paradigme pris isolement est un facteur qui rend leurs combinaison, à la fois difficile, et un centre d'intérêt, en attendant la maturité de la séparation avancée des préoccupations.

Dans un second temps, nous avons présenté une nouvelle approche de combinaison des composants logiciels et de la séparation avancée des préoccupations, on s'est focalisé principalement sur la programmation orienté aspect avec le modèle de composant java beans. Nous avons essayé de fonder toute notre action sur une approche qui consiste à rassembler les avantages des travaux existants tout en tentant de remédier à leurs insuffisances.

L'approche que nous avons proposée repose sur trois concepts principaux : les composants ; représentant la partie métier du système, les composants aspects ; représentant la partie extra fonctionnelle, et le composant gestionnaire qui est le noyau de l'approche, assurant un ensemble de fonctionnalité pour la gestion de la combinaison.

Notre approche offre une bonne adaptation au contexte ainsi qu'un haut niveau de réutilisation. Cependant, étant donné que toutes les interactions entre les composants et les composants aspects passent par le composant gestionnaire, pareillement à l'approche [2], notre approche est inappropriée pour les environnements où les ressources sont limitées. Par conséquent, l'une de nos perspectives de recherche est de rendre l'approche plus optimale en matière de ressources, tout en préservant les avantages qu'elle offre.

L'une de nos perspectives aussi est la réalisation d'une plateforme représentant un environnement complet de développement, supportant les concepts de l'approche, disposant de tous les outils nécessaires à la construction d'un système. A plus long

terme, nous envisageons de généraliser notre approche à d'autres modèles de composant.

7. BIBLIOGRAPHIE

[1] Stanley M., et al, Advanced Separation of Concerns for Component Evolution, Workshop on Engineering Complex Object-Oriented Systems for Evolution—OOPSLA New York, 2001.

[2] Suvée, D., Wim V. and Viviane J., JAsCo: an Aspect-Oriented approach tailored for CBSD, Proc. of the second international conf. on aspect-oriented software development, Boston, march 2003.

[3] Duclos, F., et al, Describing and Using Non Functional Aspects in Component Based Applications. Proc. of the 1st int. conf. on AOSD, Enschede, Netherlands, April 2002.

[4] Cottenier T., Elrad T., Validation of Context-Dependent Aspect-Oriented Adaptations to Components, Ninth Int. Workshop on Component-Oriented Programming, Oslo, June 2004.

[5] Suvée D., De Fraine B., W. Vanderperren, A Symmetric and Unified Approach Towards Combining Aspect-Oriented and CBSD, SSEL Lab., Vrije Universiteit Brussel, http://ssel.vub.ac.be/fusej/

[6] Pessemier N. et al, A Safe Aspect-Oriented Programming Support for Component-Oriented Programming, 11th Int. Workshop on Component-Oriented Programming, Nantes, France, July 2006.

[7] Bachmann F., et al, Technical Concepts of Component-Based Software Engineering, Volume 2, 2nd Edition, May 2000.

[8] Kiczales G., et al. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, ECOOP'97,LNCS, Volume 1241, pp 220–242. Springer-Verlag, June 1997.

[9] Aksit M., Tekinerdogan B., Solving the modeling problems of object-oriented languages by composing multiple aspects using composition filters, LNCS, Vol. 1543, Springer, July 1998.

[10] Tarr P. et al, N degrees of separation: Multi-dimensional separation of concerns. In Proceedings of the 21st Int. Conf. on Software Engineering, pp 107–119, May 1999.

[11] Pessemier N., Unification des approches par aspects et à composants, thèse de Doctorat de l'université des Sciences et Technologies de Lille, France, Juin 2007, pp. 28-29.

[12] J. Aldrich, Open modules: Modular reasoning about advice. LNCS, Vol. 3586, pp 144–168. Springer, 2005.

[13] N. Pessemier, et al, A model for developing component-based and aspect-oriented systems. Proceedings of the 5th Int. Symp. on Software Composition, LNCS. Springer, Mar. 2006.

[14] Tourwe T. et al, On the Existence of the AOSD-Evolution Paradox, In AOSD Workshop on Software-engineering Properties of Languages for Aspect Technologies, 2003

[15] Guido Söldner and Rüdiger Kapitza, AOCI: An Aspect-Oriented Component Infrastructure. pp. 5-6.

[16] Noyé J. et al, Composants et aspects, Projet Obasco EMN – INRIA, Ecole des Mines de Nantes, France, septembre 2004. pp. 9-11.

[17] http://www.eclipse.org/

[18] http://www.eclipse.org/ajdt/

[19] http://www.eclipse.org/webtools/

[20] LEGOND-AUBRY Fabrice, Un modèle d'assemblage de composants par Contrat et Programmation Orientée Aspect, Thèse De Doctorat Du Conservatoire National Des Arts Et Métiers, France, Juillet 2005, pp. 21-29.