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 les approches de combinaison les plus connues et nous les comparons.

2. CLASSIFICATION DES APPROCHES DE COMBINAISON

Les approches actuelles de combinaison, peuvent être classées en deux catégories : les approches symétriques où les aspects et les composants sont traités uniformément, et les approches asymétriques, qui traitent les aspects et les composants différemment. 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

A- JAsCo : Dans cette approche, D. Suvée, et al proposent un nouveau langage d'implémentation orienté aspect appelé JasCo [2]. JasCo est adapté au modèle de composant Java beans 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 résoud 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

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 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 NRIA/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 à dynamiquement et tient compte des besoins imprévus.

3. COMPARAISON ET CRITIQUES DES APPROCHES

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 et 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 / Propriétés	Asymétriques			Symétriques		
	[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. TRAVAUX SIMILAIRES

Le travail 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.

5. CONCLUSION

Dans cet article nous avons 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.

6. 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. et al, 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.