

RAPPORT D'AVANCEMENT DE THÈSE

Programmation par Aspects et Composants

Abdelhakim HANNOUSSE

mai 2009

LABORATOIRE D'INFORMATIQUE DE NANTES-ATLANTIQUE
UMR 6241



ÉCOLE DOCTORALE STIM
« Sciences et technologies de l'information et des mathématiques »
n. 503



IMPORTANT

Ce document de synthèse de l'année écoulée est à transmettre à la Direction du laboratoire, aux encadrants et aux membres du comité de suivi.

Cette fiche sera transmise à l'école doctorale (ED) qu'en cas de difficulté constatée, l'ED étant alors tenue de diligenter une enquête pour prendre une décision quant à l'inscription en année supérieure.

Table des matières

1	Identification	4
1.1	Sujet et financement	4
1.2	Doctorant	4
1.3	Encadrement	4
2	Éléments d'appréciation factuels et personnels	5
2.1	Projet professionnel	5
2.1.1	État qualitatif de la construction du projet professionnel	5
2.2	Aspects scientifiques	5
2.2.1	Participations ? la vie de la recherche	5
2.2.2	État qualitatif de l'avancement du travail de thèse	6
3	Contexte et problématique	8
3.1	Contexte	8
3.2	Présentation de la problématique	9
4	État de l'art	10
4.1	Classification de la bibliographie	10
4.2	Mise en perspective	12
5	Perspectives du travail de thèse	13
	Bibliographie ¹⁴	

Chapitre 1

Identification

1.1 Sujet et financement

Titre : « *Programmation par Aspects et Composants* »

Bourse : Région,Mines

1.2 Doctorant

NOM, prénom : HANNOUSSE, Abdelhakim

Date de naissance : 10/02/1979

Courriel : abdel-hakim.hannousse@emn.fr

Téléphone : 02 51 85 85 25

1.3 Encadrement

Équipe d'accueil : « Ascola, Coloss »

Directeur de thèse : SÜDHOLT, Mario (à 20 %)

LE TAUX D'ENCADREMENT DU DIRECTEUR DE THESE NE PEUT PAS ETRE INFÉRIEUR A 40 %

Co-encadrant : DOUENCE, Rémi (à 40 %)

Co-encadrant : ARDOUREL, Gilles (à 40 %) , équipe « Coloss »

LA SOMME DES TAUX D'ENCADREMENT DOIT ÊTRE ÉGALE A 100 %...

Chapitre 2

Éléments d'appréciation factuels et personnels

2.1 Projet professionnel

Les modules professionnels validés à cette date auprès de l'ED STIM sont indiqués sur le tableau [2.1](#).

2.1.1 État qualitatif de la construction du projet professionnel

Généralement, pas de difficulté rencontrée jusqu'à maintenant, le seul problème qu'on peut signaler est la difficulté à rédiger un document en français. Afin de résoudre ce problème je souhaite suivre des cours de français à partir de l'année prochaine.

2.2 Aspects scientifiques

Les modules scientifiques validés à cette date auprès de l'ED STIM sont indiqués sur le tableau [2.2](#).

2.2.1 Participations ? la vie de la recherche

Il est prévu de participer à deux écoles d'été pour cette année 2009 :

- Ecole des Jeunes Chercheurs en Programmation (EJCP'09) – Juin 2009
- 4th European Summer School on Aspect-oriented Software Development (AOSD'09) – Août 2009

Je suis aussi chargé de la gestion des références bibliographiques de l'équipe Ascola.

2.2.2 État qualitatif de l'avancement du travail de thèse

Le travail qui a été fait depuis la date de commencement de la thèse (01/10/2008) peut être résumé par les points suivants :

1. Une étude bibliographique a été effectuée (une soixantaine de références).
2. Des interprètes prototypes de langages (impératif, orienté-objet et fonctionnel) ont été développés en Haskell afin de maîtriser les différents paradigmes et comment les implémenter.
3. Nous avons commencé la conception d'un modèle mélangeant composants et aspects de manière originale. Ce modèle est basé sur la théorie des Arrows [[Hug00](#), [Hug04](#)]. Un prototype en Haskell sert de base à nos expérimentations.

Enfin, l'avancement de la thèse est conforme aux attentes.

TABLE 2.1 – Modules professionnels validés

Code ED (ou eq.)	Intitulé
MP5	Méthodologie de la recherche
MP11	La recherche et l'entreprise

TABLE 2.2 – Modules scientifiques validés

Code ED (ou eq.)	Intitulé
MSS2	La conception des systèmes informatiques embarqués
MSS13	Introduction au calcul parallèle

Chapitre 3

Contexte et problématique

3.1 Contexte

On assiste actuellement à deux formes de programmation visant à améliorer la réutilisation et l'adaptabilité du logiciel au travers d'une meilleure séparation des préoccupations tout au long du développement d'une application : la programmation par composants [[Szy02](#)] et la programmation par aspects [[KLM⁺97](#)].

L'approche composants consiste à décrire un ensemble de modules réutilisables. Chacun encapsule une ou plusieurs fonctions de base, et l'utilisateur va assembler ces différents modules afin de construire l'architecture globale d'un projet informatique. Ceci facilite le développement des logiciels et permet d'assurer une meilleure lisibilité et une meilleure maintenance.

L'approche aspects vise à séparer les préoccupations techniques ou de contrôle (e.g. synchronisation, persistance, contraintes temps réel, etc.) des préoccupations métier ou fonctionnelles. Elle offre un mécanisme de tissage qui permet de fusionner ces deux types de préoccupations afin de construire le système global. Ceci permet une meilleure séparation du code fonctionnel du code non fonctionnel et d'assurer une meilleure maintenabilité du système.

Ces deux approches sont complémentaires. L'approche composants assure une meilleure modularisation des préoccupations fonctionnelles. L'approche aspects assure une meilleure séparation du code fonctionnel du code non fonctionnel. Un système complet nécessite ces deux approches. Il s'agit dans les deux approches : aspects et composants, de développer des applications par assemblage, de composants dans le premier cas et d'aspects dans le deuxième cas, avec les mêmes problématiques (vérifier la correction des assemblages, autoriser les assemblages dynamiques, etc.). Cet assemblage prend toutefois des formes différentes. Il est symétrique dans le cas des composants, chaque composant fournissant explicitement des services en s'appuyant sur des services fournis par d'autres composants. Il est asymétrique dans le cas des aspects, avec la définition d'un programme de base correspondant à une structure et un comportement principal, sur lequel viennent se greffer des aspects additionnels traitant des préoccupations non couvertes par le programme de

base.

Dans ce contexte, le projet de cette thèse est de proposer un modèle unifié : composants aspects. Ce travail s'effectuera dans le cadre du projet régional Miles. Il correspond au démarrage d'une nouvelle étape dans les travaux communs aux deux équipes (i.e. Coloss et Ascola).

3.2 Présentation de la problématique

Les deux formes d'assemblage utilisées par les deux approches aspects et composants s'avèrent complémentaires. On peut ainsi avoir envie de structurer un aspect complexe sous la forme d'un assemblage de composants et un composant comme un assemblage d'aspects. Ce dernier point de vue est notamment naturel dans le cadre de l'assemblage de composants comme les Entreprise JavaBeans ou les services web. Le code métier du composant constitue alors le programme de base, complété par des aspects techniques gérant des préoccupations comme la distribution, la sécurité, la persistance, sans référence explicite à ces notions dans le code métier. Cette complémentarité est maintenant reconnue et a conduit à la définition de cadres de programmation en Java comme JBoss AOP ou Spring. Ce type d'intégration reste toutefois peu expressif et difficile à maîtriser.

Il s'agira donc d'étudier, à partir de l'expérience des équipes Ascola et Coloss dans le domaine des langages d'aspects et des langages de composants, la possibilité d'une intégration plus fine des composants et des aspects sous la forme d'un langage de programmation unifié. Nous nous intéresserons à la fois à la définition bien fondée d'un tel langage, à sa mise en œuvre et à son utilisation pratique. Le langage ou l'environnement devront supporter l'expression et la vérification de propriétés. Par exemple on veut pouvoir garantir la conformité des interactions entre deux composants après l'ajout d'un aspect, ou un invariant sur l'état d'un composant.

Chapitre 4

État de l'art

4.1 Classification de la bibliographie

Notre étude bibliographique a été divisée en trois parties :

Etude des modèles à composants : Dans cette partie, nous avons étudié différents modèles à composants, en mettant en évidence les différentes caractéristiques des composants que nous avons besoin de considérer dans notre futur modèle. Parmi les modèles étudiés, on peut citer les systèmes : EJB [[Mic](#)], CORBA [[Gro06](#)], Fractal[[BCL⁺06](#)], SOFA[[BHP06](#), [Bur05](#), [MB05](#)], Kmelia [[AAA06a](#), [AAA06b](#), [AAA07a](#), [AAA07b](#)], rCOS[[JLL05](#), [CHLZ07](#), [JLL06](#)].

Chaque modèle parmi ceux qui ont été étudiés a ses propres particularités. Par exemple l'EJB est un modèle à plat qui ne considère pas l'aspect hiérarchique des composants (i.e. il ne supporte que les composants primitifs et pas les composants composites). Il utilise la technologie de conteneur qui prend en charge l'implémentation des services techniques (e.g. persistance), mais ces services sont très limités. Fractal est un modèle plus complet (e.g. hiérarchique, supporte la reconfiguration dynamique à travers la membrane associée à chaque composant, le déploiement, etc.).

Certains modèles considèrent un composant comme étant un objet qui est accessible uniquement à travers des interfaces associées à cet objet, ce qui permet de programmer les composants en utilisant des langages à objets. D'autre comme FuseJ [[SFV06](#), [SFV05](#)] proposent des langages spécifiques aux composants, malheureusement, ces langage n'ont pas de sémantique formelle.

Nous avons aussi étudié le projet CoCoME (i.e. Common Component Modelling Example) [[HKW⁺08](#)]. L'objectif de ce projet, qui a été lancé en 2006, est de comparer les différents modèles à composants à travers une implémentation d'une application de gestion d'un supermarché (i.e. vente de produit, approvisionnement, génération des rapports, collaboration entre les différents magasins, etc.). Une architecture initiale de cette application a été fournie accompagnée par le code source de son implémentation. Chaque équipe a été chargée d'utiliser son propre modèle pour spécifier cet exemple avec toutes

ses contraintes structurelles, comportementales et temporelles. Différents modèles ont été évalués dans ce projet comme : Fractal [BBC⁺08], SOFA [BDH⁺08], CoIn [BvVZ06, ZVB⁺08], GCM [CCH⁺08], rCOS [CHH⁺08], Cowch [SRGPH08], JavaA [BHH⁺06, ASR⁺08], DisCCOMP [Rau07, AHKR08]. Nous avons remarqué que plusieurs architectures (i.e. vues) ont été considérées pour le même système. Nous avons aussi découvert que tous les modèles à composants participants à ce projet ne sont pas suffisant pour modéliser une application concrète comme celle de CoCoME. Par exemple, Fractal et Cowch permettent de spécifier l'architecture de l'application (i.e. les différents composants primitifs et composites et leur interactions) mais ils ne sont pas capables de spécifier des contraintes temps réel ni d'utiliser un modèle formel pour vérifier la cohérence de composition et la compatibilité des interactions entre les différents composants dans des architectures. Par contre, le modèle CoIn a prouvé la compatibilité des différentes interactions entre les composants et prouvé l'absence d'interblocage dans le système. Malheureusement, CoIn est un modèle abstrait de vérification formelle de composition mais pas un modèle à composant complet.

A la fin de cette partie, nous avons étudié différentes taxonomies proposées pour les approches à composants. On peut citer comme exemple la taxonomie de Bunse [BFL06] et la taxonomie de Kung-Ku [LW05, LW07]. La taxonomie de Bunse est abstraite et générale, basée sur des critères qualitatifs qui sont plus des critères d'évaluation que des critères de classification (Par exemple : la modularité, l'interopérabilité, le domaine d'application, etc.). La taxonomie proposée par Kung-Ku se base sur la possibilité d'assembler ou désassembler les composants à différentes phases de développement du système (i.e. phase de conception, phase de stockage des composant dans des repositories, phase de déploiement).

Etude des modèles à aspects : Dans cette partie, comme pour la partie précédente, nous nous sommes intéressés aux différentes caractéristiques des aspects que notre modèle doit comporter. Différents modèles ont été étudiés. Le modèle de référence AspectJ [KHH⁺01] introduit deux notions essentielles : les points de coupures spécifiant l'endroit dans le code où l'effet de l'aspect va être appliqué, et les advices qui spécifient le comportement de l'aspect. AspectJ propose des points de coupures sophistiqués qui dépendent du flot de contrôle et donc de l'historique de l'exécution. D'autres modèles proposent des points de coupures aussi basés sur l'historique de l'exécution mais pas sur le flot de contrôle. En particulier, trace-matches [MPJW00] et EAOP [DFS02, DFS04] définissent des séquences de points d'exécution et CEAOP [DBNS06] étend ces séquences dans un contexte concurrent. Dans ces modèles, la sémantique des points de coupures et le mécanisme de tissage ont été formellement spécifiés et différents opérateurs de compositions ont été définis.

La programmation par aspects a été adaptée aux composants logiciels. Les modèles que nous avons étudiés sont : JASCO [SVJ03], FuseJ [SFV06, SFV05], CaesarJ [AGMO06] et FAC [Pes07, PSCD06, PSDC08, PSD04]. Certains de ces modèles sont des extensions par aspects des modèles à composants existants comme FAC (i.e. extension ou Fractal). D'autres comme CaesarJ, FuseJ et JASCo proposent des extensions d'aspects et de com-

posants à des langages à objets (i.e. généralement Java). Dans le premier cas (i.e. FAC), un aspect a été considéré comme un composant ordinaire, et le mécanisme de tissage comme une sorte de connexion spéciale entre un composant aspect et un composant ordinaire. Dans le deuxième cas, FuseJ et JASCo proposent un langage d'aspects associé aux composants. Mais ces nouveaux langages n'ont pas de sémantique formelle. Le cas de CaesarJ est différent, une sémantique opérationnelle a été spécifiée pour le langage proposé. Nous avons remarqué, que tous ces modèles sont beaucoup plus des modèles à composants que des modèles à aspects. C-à-d la majorité des concepts d'aspects ont été négligés dans ces modèles (e.g. pas de points de coupures basées sur le flot de contrôle ou les séquences, pas de compositions des aspects, etc.).

Etude des Arrows : Le mécanisme appelée Arrows [[Hug00](#), [Hug04](#)] est un nouveau langage qui a été intégré dans le langage fonctionnel Haskell sous forme d'une librairie. Le système Arrows permet de définir des calculs primitifs et de les composer. Il est formellement défini. Il peut donc servir de base formelle pour d'écrire des architectures à base de composants. Actuellement les arrows sont utilisés pour développer une bibliothèque graphique pour Haskell appelée Fruit [[CE01](#)]. Dans ce travail, chaque composant graphique a été défini comme étant un arrows. Les opérateurs de composition des arrows ont été utilisés pour construire des composants graphiques complexes. Ce mécanisme a été aussi utilisé pour la composition parallèle de processus [[HHP07](#), [HCNP03](#)]. Un exemple pratique a été présenté dans ce contexte qui consiste à modéliser un robot qui utilise son bras pour accéder à des objets distants. Après l'analyse de ce nouveau mécanisme qui semble similaire à celui de composants nous avons commencé à étudier ce système en détail.

4.2 Mise en perspective

Notre étude bibliographique nous a permis d'identifier les caractéristiques essentielles des systèmes à composants :

1. composants primitifs boîtes noires,
2. composants composites pour assurer une structure hiérarchique des systèmes,
3. structurations multiples d'un même système.

Les caractéristiques essentielles des systèmes à aspects :

1. les points de jonctions,
2. les points de coupures,
3. les advices.

Les systèmes hybrides à composants et aspects sont généralement des adaptations naïves des langages d'aspects au monde des composants. Par exemple, on perd la notion de points de coupures basés sur le flot de contrôle mais on ne gagne pas de points de coupures basés sur la notion de communication entre composants.

Chapitre 5

Perspectives du travail de thèse

Une première piste à explorer concerne la notion de vue. Dans un système hiérarchique on peut distinguer deux types de composants : les composants primitifs (c.à.d. les boîtes) et les composants composites (c-à-d. les boîtes de boîtes). Un unique système défini par ses composants primitifs peut être structuré de multiples façons à l'aide de composants composites ou de vues. Par exemple, la figure 5.1 montre deux vues différentes d'une même architecture. Une vue peut être considérée comme un composite représentant une préoccupation ou un domaine dont l'enveloppe est calculée et qui permet d'intercepter les communications d'un groupe de composants à l'aide d'un aspect. Des aspects peuvent être appliqués aux différentes vues. Se posent alors les questions de la cohabitation de plusieurs vues d'un même système (contrôle, cohérence, coût) et du recalcul des vues lors d'une reconfiguration dynamique du système.

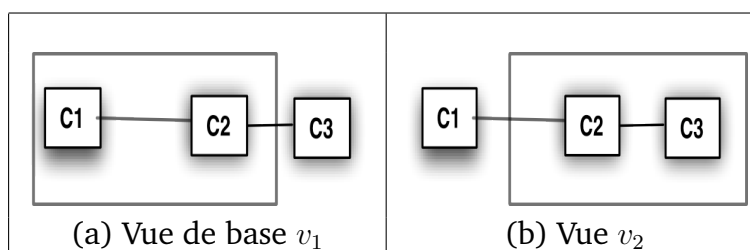


FIGURE 5.1 – Architecture à composants avec différentes vues

Notre modèle doit supporter formellement la modularité et la hiérarchisation des systèmes à composants et utiliser des opérateurs d'assemblage composant-composant et aspect-composant. Il doit aussi supporter les différents concepts pertinents des aspects identifiés dans la section précédente. Pour atteindre cet objectif, notre modèle va considérer les composants comme des Arrows. Ces composants sont composés à l'aide d'opérateurs de composition définis dans Arrows. Un wrapper est une paire de composants before et after qui entoure un composant pour en modifier le comportement. Un système peut être assemblé de différentes manières. Chaque manière est appelée vue. Un wrap-

per dans une vue autre que la vue de base est appelé aspect. La figure 5.2 représente l'application de deux wrappers sur différentes vues.

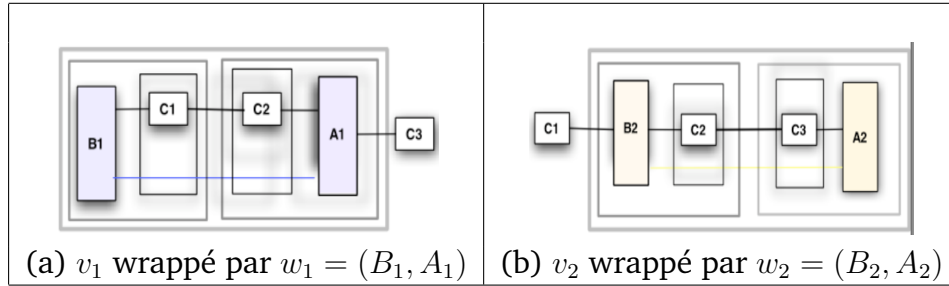


FIGURE 5.2 – Deux vues wrappés différemment

Un algorithme général de spécification d'un système complet consiste à :

1. Définir les composants primitifs.
2. Utiliser les opérateurs d'assemblage pour construire l'architecture désirée (i.e. vue de base).
3. Définir les différents wrappers et les vues sur lesquelles ils s'appliquent.
4. Appliquer les wrappers de base sur la vue de base.
5. Pour chaque autre wrapper :
 - (a) Transformer la vue de base dans la vue correspondante.
 - (b) Appliquer le wrapper.
 - (c) Appliquer la transformation inverse pour revenir à la vue de base.

La figure 5.3 montre l'intégration des deux wrappers de la figure 5.2 à la vue de base, où swap est un composant arrows utilisé pour reordonner les fils.

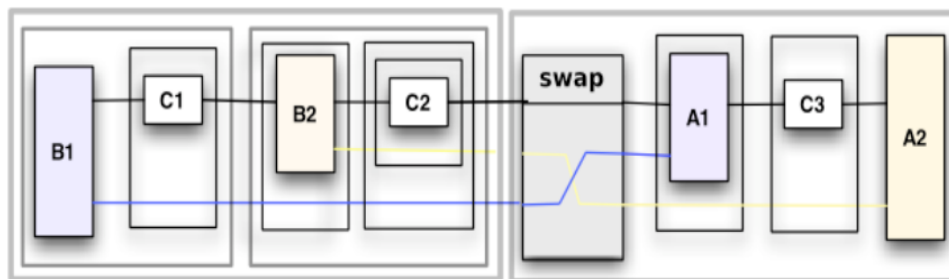


FIGURE 5.3 – Architecture possède un vue v_1 wrappée par w_1 sur v_1 et par w_2 sur v_2

Nous validerons notre modèle sur une application concrète et nous étudierons son adaptation à un système existant comme par exemple Fractal.

Bibliographie

- [AAA06a] Pascal André, Gilles Ardourel, and Christian Attiogbé. Spécification d'architectures logicielles en kmelia : hiérarchie de connexion et composition. In *1ère Conférence Francophone sur les architectures Logicielles*, pages 101–118. Hermès Sciences Publications - Lavoisier, 2006.
- [AAA06b] Christian Attiogbé, Pascal André, and Gilles Ardourel. Checking component composability. In *Software Composition*, volume 4089 of *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin / Heidelberg, August 2006.
- [AAA07a] Pascal André, Gilles Ardourel, and Christian Attiogbé. Adaptation for hierarchical components and services. *Electronic Notes in Theoretical Computer Science*, 189 :5–20, 2007.
- [AAA07b] Pascal André, Gilles Ardourel, and Christian Attiogbé. Defining component protocols with service composition : Illustration with the kmelia model. In *Software Composition*, volume 4829 of *Lecture Notes in Computer Science*, pages 2–17. Springer Berlin / Heidelberg, December 2007.
- [AAA07c] Pascal André, Gilles Ardourel, and Christian Attiogbé. A formal analysis toolbox for the kmelia component model. In *ProVeCS 2007 - Satellite Event of TOOLS Europe*, page to appear, Zurich, Switzerland, 2007. ETH Research Report.
- [AGMO06] Ivica Aracic, Vaidas Gasiunas, Mira Mezini, and Klaus Ostermann. An overview of caesarj. In *Transactions on Aspect-Oriented Software Development I*, volume 3880 of *Lecture Notes in Computer Science*, pages 135–173. Springer Berlin / Heidelberg, February 2006.
- [AHKR08] André Appel, Sebastian Herold, Holger Klus, and Andreas Rausch. Modelling the cocome with disccomp. In *The Common Component Modeling Example : Comparing Software Component Models*, volume 5153 of *Lecture Notes in Computer Science*, pages 267–296. Springer Berlin / Heidelberg, August 2008.
- [ASR⁺08] Knapp Alexander, Janisch Stephan, Hennicker Rolf, Clark Allan, Gilmore Stephen, Hacklinger Florian, Baumeister Hubert, and Wirsing Martin. Modeling the cocome with the java/a component model. In *The Common Component Modeling Example : Comparing Software Component Models*, volume

- 5153 of *Lecture Notes in Computer Science*, pages 207–237. Springer Berlin / Heidelberg, August 2008.
- [BBC⁺08] Lubomír Bulej, Tomas Bures, Thierry Coupaye, Martin Decký, Pavel Jezek, Pavel Parizek, Frantisek Plasil, Tomás Poch, Nicolas Rivierre, Ondrej Sery, and Petr Tuma. Cocome in fractal. In Andreas Rausch, Ralf Reussner, Raffaella Mirandola, and Frantisek Plasil, editors, *The Common Component Modeling Example - Comparing Software Component Models*, volume 5153 of *Lecture Notes in Computer Science*, pages 357–387. Springer Berlin / Heidelberg, August 2008.
 - [BBM04] Tomás Barros, Rabéa Boulifa, and Eric Madelaine. Parameterized models for distributed java objects. In *Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, volume 3235 of *Lecture Notes in Computer Science*, pages 43–60. Springer Berlin / Heidelberg, September 2004.
 - [BCL⁺06] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java. *Software-Practice and Experience*, 36(11-12) :1257–1284, August 2006.
 - [BCM03] Françoise Baude, Denis Caromel, and Matthieu Morel. From distributed objects to hierarchical grid components. In *On The Move to Meaningful Internet Systems 2003 : CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 1226–1242. Springer Berlin / Heidelberg, October 2003.
 - [BCR09] Eric Bodden, Feng Chen, and Grigore Rosu. Dependent advice : a general approach to optimizing history-based aspects. In *AOSD '09 : Proceedings of the 8th ACM international conference on Aspect-oriented software development*, pages 3–14, New York, NY, USA, 2009. ACM.
 - [BDH⁺08] Tomas Bures, Martin Decký, Petr Hnetynka, Jan Kofron, Pavel Parizek, Frantisek Plasil, Tomás Poch, Ondrej Sery, and Petr Tuma. Cocome in sofa. In *The Common Component Modeling Example - Comparing Software Component Models*, volume 5153 of *Lecture Notes in Computer Science*, pages 388–417. Springer Berlin / Heidelberg, August 2008.
 - [BFL06] Christian Bunse, Felix C. Freiling, and Nicole Levy. A taxonomy on component-based software engineering methods. In Ralf H. Reussner, Judith A. Stafford, and Clemens A. Szyperski, editors, *Architecting Systems with Trustworthy Components*, volume 3938 of *Lecture Notes in Computer Science*, pages 103–119. Springer Berlin / Heidelberg, July 2006.
 - [BHH⁺06] Hubert Baumeister, Florian Hacklinger, Rolf Hennicker, Alexander Knapp, and Martin Wirsing. A component model for architectural programming. *Electronic Notes in Theoretical Computer Science*, 160 :57–96, 2006.
 - [BHP06] Tomas Bures, Petr Hnetynka, and Frantisek Plasil. Sofa 2.0 : Balancing advanced features in a hierarchical component model. In *SERA '06 : Pro-*

ceedings of the Fourth International Conference on Software Engineering Research, Management and Applications, pages 40–48, Washington, DC, USA, 2006. IEEE Computer Society.

- [BKR09] Steffen Beckera, Heiko Koziol, and Ralf Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1) :3–22, 2009.
- [BSS08] Manfred Broy, Johannes Siedersleben, and Clemens Szyperski. Cocome jury evaluation and conclusion. In *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in Computer Science*, pages 449–458. Springer Berlin / Heidelberg, August 2008.
- [Bur05] Tomas Bures. Automated synthesis of connectors for heterogeneous deployment. Technical Report 2005/4, Department of SW Engineering, Charles University, Prague, August 2005.
- [BvVZ06] Luboš Brim, Ivana Černá, Pavlína Vařeková, and Barbora Zimmerová. Component-interaction automata as a verification-oriented component-based system specification. *ACM SIGSOFT Software Engineering Notes*, 31(2) :1–8, March 2006.
- [CCH⁺08] Antonio Cansado, Denis Caromel, Ludovic Henrio, Eric Madelaine, Marcela Rivera, and Emil Salageanu. A specification language for distributed components implemented in gcm/proactive. In *The Common Component Modeling Example : Comparing Software Component Models*, volume 5153 of *Lecture Notes in Computer Science*, pages 418–448. Springer Berlin / Heidelberg, 2008.
- [CE01] Antony Courtney and Conal Elliott. Genuinely functional user interfaces. In *Proceedings of the 2001 Haskell Workshop*, pages 41–69. ACM Sigplan, September 2001.
- [CHH⁺08] Zhenbang Chen, Abdel Hakim Hannousse, Dang Van Hung, Istvan Knoll, Xiaoshan Li, Zhiming Liu, Yang Liu, Qu Nan, Joseph C. Okika, Anders P. Ravn, Volker Stolz, Lu Yang, and Naijun Zhan. Modelling with relational calculus of object and component systems - rcos. In *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in Computer Science*, pages 116–145. Springer Berlin / Heidelberg, August 2008.
- [CHLZ07] Xin Chen, Jifeng He, Zhiming Liu, and Naijun Zhan. A model of component-based programming. In *International Symposium on Fundamentals of Software Engineering (FSEN)*, volume 4767 of *Lecture Notes in Computer Science*, pages 191–206. Springer Berlin / Heidelberg, October 2007.
- [CKV98] Denis Caromel, Wilfried Klauser, and Julian Vayssière. Towards seamless computing and metacomputing in java. *Concurrency Practice and Experience*, 10(11-13) :1043–1061, December 1998.

- [DBNS06] Rémi Douence, Didier Le Botlan, Jacques Noyé, and Mario Südholt. Concurrent aspects. In *GPCE'06 : Proceedings of the 5th international conference on Generative programming and component engineering*, pages 79–88, New York, NY, USA, 2006. ACM.
- [DDF08] Simplicé Djoko Djoko, Rémi Douence, and Pascal Fradet. Aspects preserving properties. In *PEPM '08 : Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages 135–145, New York, NY, USA, 2008. ACM.
- [DDFB06] Simplicé Djoko Djoko, Rémi Douence, Pascal Fradet, and Didier Le Botlan. Casb : Common aspect semantics base. Deliverable 41, AOSD-Europe, INRIA, February 2006.
- [DFS02] Rémi Douence, Pascal Fradet, and Mario Südholt. A framework for the detection and the resolution of aspect interaction. In *GPCE'06 : Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative programming and component engineering*, pages 173–188, London, UK, 2002. Springer-Verlag.
- [DFS04] Rémi Douence, Pascal Fradet, and Mario Südholt. Composition, reuse and interaction analysis of stateful aspects. In ACM Press, editor, *AOSD'04 : Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 141–150, Lancaster, UK, March 2004. ACM.
- [EM06] Marcelo Medeiros Eler and Paulo Cesar Masiero. Aspects as components. In *Reuse of Off-the-Shelf Components*, number 4039 in Lecture Notes in Computer Science, pages 411–414. Springer Berlin / Heidelberg, July 2006.
- [EOC06] Erik Ernst, Klaus Ostermann, and William R. Cook. A virtual class calculus. In *POPL'06*, pages 270–282, Charleston, South Carolina, USA, January 2006. ACM.
- [foa09] *Proceedings of the Eighth Workshop on Foundations of Aspect-Oriented Languages (FOAL'09)*, New York, NY, USA, March 2009. ACM.
- [Gro06] Object Managmenet Group. Corba component model specification. Number Version 4.0. Object Managmenet Group, April 2006.
- [HCNP03] Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson. Arrows, robots, and functional reactive programming. In Johan Jeuring and Simon Peyton Jones, editors, *Advanced Functional Programming, 4th International School 2002*, volume 2638 of *Lecture Notes in Computer Science*, pages 159–187. Springer-Verlag, 2003.
- [HHP07] Liwen Huang, Paul Hudak, and John Peterson. Hporter : Using arrows to compose parallel processes. In *Practical Aspects of Declarative Languages*, volume 4354 of *Lecture Notes in Computer Science*, pages 275–289. Springer Berlin / Heidelberg, April 2007.

- [HKW⁺08] Sebastian Herold, Holger Klus, Yannick Welsch, Constanze Deiters, Andreas Rausch, Ralf Reussner, Klaus Krogmann, Heiko Koziolk, Raffaella Mirandola, Benjamin Hummel, Michael Meisinger, and Christian Pfaller. Cocome-the common component modeling example. In Andreas Rausch, Ralf Reussner, Raffaella Mirandola, and Frantisek Plasil, editors, *The Common Component Modeling Example - Comparing Software Component Models*, volume 5153 of *Lecture Notes in Computer Science*, pages 16–53. Springer Berlin / Heidelberg, August 2008.
- [HP04] Petr Hnětynka and František Plášil. Distributed versioning model for mof. In *WISICT '04 : Proceedings of the winter international symposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.
- [HP06] Petr Hnětynka and František Plášil. Dynamic reconfiguration and access to services in hierarchical component models. In *Proceedings of CBSE 2006, Vasteras, Sweden*, volume 4063 of *Lecture Notes in Computer Science*, pages 352–359. Springer-Verlag, 2006.
- [HPB⁺05] Petr Hnětynka, František Plášil, Tomáš Bureš, Vladimír Mencl, and Lucia Kapová. Sofa 2.0 metamodel. Technical Report 11/05, Department of SW Engineering, Charles University, Pargue, December 2005.
- [Hug00] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3) :67–111, 2000.
- [Hug04] John Hughes. Programming with arrows. In *Advanced Functional Programming*, volume 3622 of *Lecture Notes in Computer Science*, pages 73–129. Springer Berlin / Heidelberg, 2004.
- [JLL05] He Jifeng, Xiaoshan Li, and Zhiming Liu. Component-based software engineering : The need to link methods and their theories. In *Theoretical Aspects of Computing – ICTAC 2005*, volume 3722 of *Lecture Notes in Computer Science*, pages 70–95. Springer Berlin / Heidelberg, October 2005.
- [JLL06] He Jifeng, Xiaoshan Li, and Zhiming Liu. rcos : A refinement calculus of object systems. *Theoretical Computer Science*, 365(1-2) :109–142, July 2006.
- [KHH⁺01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *ECOOP '01 : Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 327–353, London, UK, 2001. Springer-Verlag.
- [KLM⁺97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean M. Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Matsuoka Satoshi, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, June 1997.

- [KR08] Klaus Krogmann and Ralf Reussner. Palladio – prediction of performance properties. In *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in Computer Science*, pages 297–326. Springer Berlin / Heidelberg, August 2008.
- [LW05] Kung-Kiu Lau and Zheng Wang. A taxonomy of software component models. In *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 88–95. IEEE Computer Society, August 2005.
- [LW07] Kung-Kiu Lau and Zheng Wang. Software component models. *IEEE Transactions on Software Engineering*, 33(10) :709–724, October 2007.
- [MB05] Vladimir Mencl and Tomas Bures. Microcomponent-based component controllers : A foundation for component aspects. In *APSEC '05 : Proceedings of the 12th Asia-Pacific Software Engineering Conference*, pages 729–737, Washington, DC, USA, 2005. IEEE Computer Society.
- [Mic] Sun Microsystems. Enterprise javabeans technology. [http ://java.sun.com/products/ejb/](http://java.sun.com/products/ejb/).
- [MPJW00] Oege De Moor, Simon Peyton-Jones, and Eric Van Wyk. Aspect-oriented compilers. In *First International Symposium on Generative and Component-based Software Engineering*, volume 1799 of *Lecture Notes in Computer Science*, pages 121–133. Springer Berlin / Heidelberg, January 2000.
- [Pes07] Nicolas Pessemier. *Unification des approches par aspects et à composants*. PhD thesis, Université Lille 1, Laboratoire d’Informatique Fondamentale de Lille, Lille, France, June 2007.
- [POM03] Roman Pichler, Klaus Ostermann, and Mira Mezini. On aspectualizing component models. *Software-Practice and Experience*, 33(10) :957–974, March 2003.
- [PSCD06] Nicolas Pessemier, Lionel Seinturier, Thierry Coupaye, and Laurence Duchien. A model for developing component-based and aspect-oriented systems. In Welf Löwe and Mario Südholt, editors, *Software Composition : 5th International Symposium, SC 2006*, volume 4089 of *Lecture Notes in Computer Science*, pages 259–274. Springer Berlin / Heidelberg, March 2006.
- [PSD04] Nicolas Pessemier, Lionel Seinturier, and Laurence Duchien. Components, adl & aop : Towards a common approach. In *Workshop ECOOP Reflection, AOP, and Meta-Data for Software Evolution (RAM-SE'04)*, pages 61–69. Fakultät für Informatik, Universität Magdeburg, June 2004.
- [PSDC08] Nicolas Pessemier, Lionel Seinturier, Laurence Duchien, and Thierry Coupaye. A component-based and aspect-oriented model for software evolution. *Int. J. Computer Applications in Technology*, 31(1/2) :94–105, 2008.
- [Rau07] Andreas Rausch. Discomp - a formal model for distributed concurrent components. *Electronic Notes in Theoretical Computer Science*, 176(2) :5–23, 2007.

- [SFV05] Davy Suvée, Bruno De Fraine, and Wim Vanderperren. Fusej : An architectural description language for unifying aspects and components. In *Software-engineering Properties of Languages and Aspect Technologies Workshop AOSD*, 2005.
- [SFV06] Davy Suvée, Bruno De Fraine, and Wim Vanderperren. A symmetric and unified approach towards combining aspect-oriented and component-based software development. In *CBSE : Component-Based Software Engineering*, volume 4063 of *Lecture Notes in Computer Science*, pages 114–122. Springer Berlin / Heidelberg, June 2006.
- [SRGPH08] Jan Schäfer, Markus Reitz, Jean-Marie Gaillourdet, and Arnd Poetzsch-Heffter. Linking programs to architectures : An object-oriented hierarchical software model based on boxes. In *The Common Component Modeling Example : Comparing Software Component Models*, volume 5153 of *Lecture Notes in Computer Science*, pages 238–266. Springer Berlin / Heidelberg, August 2008.
- [SVJ03] Davy Suvée, Wim Vanderperren, and Viviane Jonckers. Jasco : an aspect-oriented approach tailored for component based software development. In *AOSD’03 : Proceedings of the 2nd international conference on Aspect-Oriented software development*, pages 21–29, Boston, Massachusetts, 2003. ACM.
- [Szy02] Clemens Szyperski. *Component Software*. Addison-Wesley Professional, November 2002.
- [ZVB⁺08] Barbora Zimmerova, Pavlína Vařeková, Nikola Beneš, Ivana Černá, Luboš Brim, and Jiří Sochor. Component-interaction automata approach (coin). In *The Common Component Modeling Example : Comparing Software Component Models*, volume 5153 of *Lecture Notes in Computer Science*, pages 146–176. Springer Berlin / Heidelberg, 2008.