Tissage d'aspects comportementaux

Jacques Klein — Franck Fleurey

IRISA/INRIA/Université de Rennes 1
Campus de Beaulieu
35042 Rennes Cedex, France
{jacques.klein, franck.fleurey}@irisa.fr

RÉSUMÉ. *La séparation de préoccupations transverses permet au concepteur de logiciel d'avoir un meilleur contrôle sur les variations et les évolutions du logiciel. Dans le domaine de la programmation, cette idée a été popularisée par le langage AspectJ, mais aujourd'hui, la communauté aspect s'intéresse aussi à opérer cette séparation plus tôt dans le cycle de développement : dès les phases d'analyse/conception et même d'expression des besoins. Dans cette optique, nous proposons une technique permettant de tisser des comportements décrits sous forme de scénarios dans un modèle de base contenant un ensemble fini de scénarios. Le processus de tissage se décompose en deux phases. Tout d'abord, une phase de détection permettant d'identifier des parties particulières d'un modèle de base, puis une phase de composition permettant de construire le modèle voulu. Ces deux phases sont implémentées comme des transformations de modèles dans l'environnement Kermeta.*

ABSTRACT. *The notion of aspect looks promising for handling crosscutting concerns earlier in the software life-cycle, up from programming to design, analysis and even requirements. This paper proposes a technique to encapsulate and weave behaviors described as sequence diagrams in base models which contain a finite set of sequence diagrams. The weaving process is two-phased. Firstly the detection phase searches parts in the base model. Secondly the composition phase builds the woven model by composing the advise in the base model for each detected part. The entire weaving process is automated and is implemented as model transformations within the Kermeta environment.*

MOTS-CLÉS : *tissage, aspects, scénarios, diagramme de séquence, transformation de modèles.*

KEYWORDS: *weaving, aspects, scenarios, sequence diagrams, model transformation.*

1. Introduction

L'encapsulation de préoccupations transverses à travers la notion d'aspect apparaît aujourd'hui comme une manière judicieuse de compléter la notion de modules disponible dans la plupart des langages de programmation. La séparation de préoccupations transverses permet au concepteur de logiciel d'avoir un meilleur contrôle sur les variations (dans le contexte des lignes de produits par exemple) et les évolutions du logiciel. A un niveau de programmation, cette idée a été popularisée par le langage AspectJ (Kiczales *et al.*, 2001), mais aujourd'hui la communauté aspect s'intéresse aussi à opérer cette séparation plus tôt dans le cycle de développement : dès les phases d'expression des besoins et d'analyse/conception (Araujo *et al.*, 2004), (Whittle *et al.*, 2004), (Rashid *et al.*, 2003). Outre le fait de séparer les préoccupations transverses tôt dans le cycle de développement, couplée à des mécanismes automatiques de tissage d'aspects, la modélisation orientée aspect pourrait permettre de cibler des plateformes non orientées-aspect ou bien faciliter la mise en place de techniques de validation comme la simulation ou la génération de tests.

Afin de rendre utilisable la séparation des préoccupations au cours des phases de modélisation, les notions d'aspect, de point de coupure et de tissage doivent être précisément définies relativement au formalisme de modélisation utilisé. De plus, le langage de définition des points de coupures (jouant un rôle critique dans l'applicabilité de méthodologies orientées aspects et étant en général uniquement attachée à la syntaxe des programmes ou modèles manipulés) doivent non seulement prendre en compte la structure des modèles utilisés, mais aussi, ce qui est en général plus difficile, la sémantique de ces modèles.

Cet article présente une technique permettant l'utilisation d'aspects pour des modèles de base constitués de scénarios (ou diagrammes de séquence dans le contexte d'UML). Pour cela, nous proposons : (1) Un formalisme permettant de modéliser et encapsuler des aspects. Un aspect est constitué de deux entités, le modèle de la préoccupation transverse (appelé *advice*) et le *point de coupure* faisant le lien entre l'advice et un modèle de base. (2) Un algorithme de détection qui permet, à partir du point de coupure d'un aspect, de trouver l'ensemble des constructions d'un modèle de base avec lesquelles l'advice doit être composé. (3) Un opérateur de composition qui permet de construire le modèle résultant de la composition d'un advice avec une partie détectée dans un modèle de base.

La figure 1 présente le schéma général de l'approche proposée, les scénarios représentés n'ayant qu'un rôle illustratif sur cette figure. La technique proposée est implantée grâce à la plateforme Kermeta (Muller *et al.*, 2005a) développée dans l'équipe Triskell.

Cet article est organisé comme suit. La section 2 introduit le langage utilisé pour décrire les scénarios, Kermeta et les aspects comportementaux. La section 3 présente l'algorithme de détection alors que la section 4 présente l'opérateur de composition. La section 5 discute de l'implantation des techniques proposées grâce à la plateforme

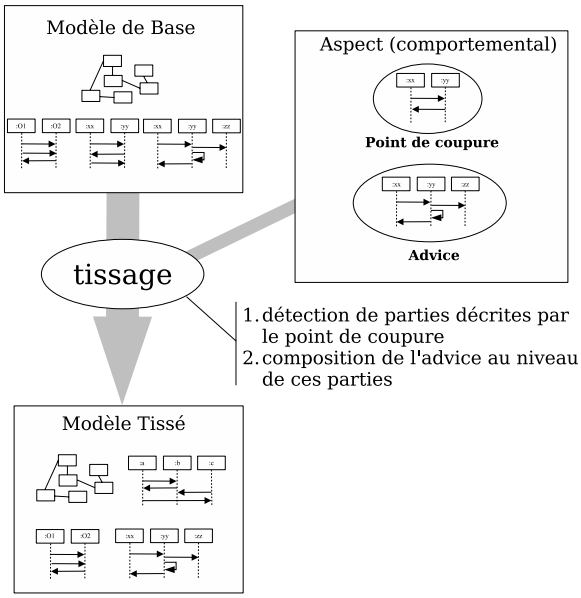

Figure 1 – Schéma global de l'approche proposée

Kermeta. Enfin, la section 6 détaille un ensemble de travaux connexes et la section 7 conclut.

2. Scénarios, Kermeta et aspects comportementaux

2.1. *Scénarios*

Les langages de scénarios sont principalement utilisés pour décrire des comportements de systèmes distribués à un niveau abstrait ou pour capturer des exigences relativement tôt dans des phases de développement avec une représentation claire, graphique et intuitive. Dans ce papier, les scénarios seront exprimés à l'aide des Diagrammes de Séquence (DSs) d'UML2.0 (OMG, 2005) qui proposent deux types de diagrammes : les Diagrammes de Séquence Simples (DSSs) qui décrivent des comportements finis et les Diagrammes de Séquence Composites (DSCs) (*Combined Fragments*) qui composent des DSSs pour engendrer, entre autres, des comportements potentiellement infinis.

Les DSs de UML2.0 ont sensiblement amélioré les versions précédentes de scénarios proposés dans UML1.x. Les DSSs décrivent un nombre fini d'interactions entre un ensemble d'objets. Ces interactions sont maintenant considérées comme une collection d'événements (au lieu d'une collection de messages dans UML1.x) qui introduisent de la concurrence, de l'asynchronisme et permettent la définition de com-

portements plus complexes. De plus, les DSs peuvent maintenant être composés aux moyens d'opérateurs pour obtenir des interactions plus complexes (les DSCs).

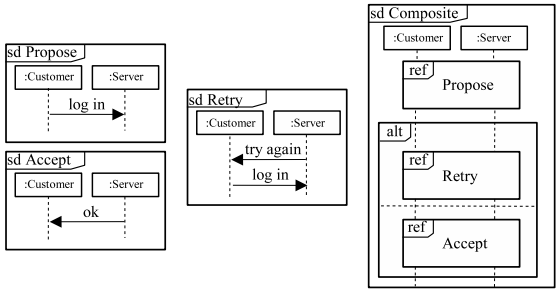

Figure 2 – Un exemple de DSSs et de DS composite

La Figure 2 montre trois DSSs nommés $Propose$, $Accept$ et $Retry$. Dans le DSS $Retry$, les messages "try again" et "log in" sont échangés entre les objets $: Customer$ et $: Server$. La Figure 3 présente le métamodèle des DSSs que nous utilisons, tandis que la Définition 1 en donne une définition formelle.

Définition 1 (Diagramme de Séquence Simple) *Un DSS est un tuple* $M = (I, E, \leq, A, \alpha, \phi, \prec)$*, où* I *est un ensemble d'objets participant aux interactions,* E *est un ensemble d'événements (envoi et réception de messages),* $\leq$ *est un ordre partiel sur* E*,* A *est un ensemble d'actions (nom des messages),* $\alpha : E \rightarrow A$ *associe les événements aux actions,* $\phi : E \rightarrow I$ *associe les événements aux objets,* $\prec \subseteq E \times E$ *est une bijection partielle associant à une émission de message, la réception correspondante, tel que* $\prec \subseteq \leq$*.*

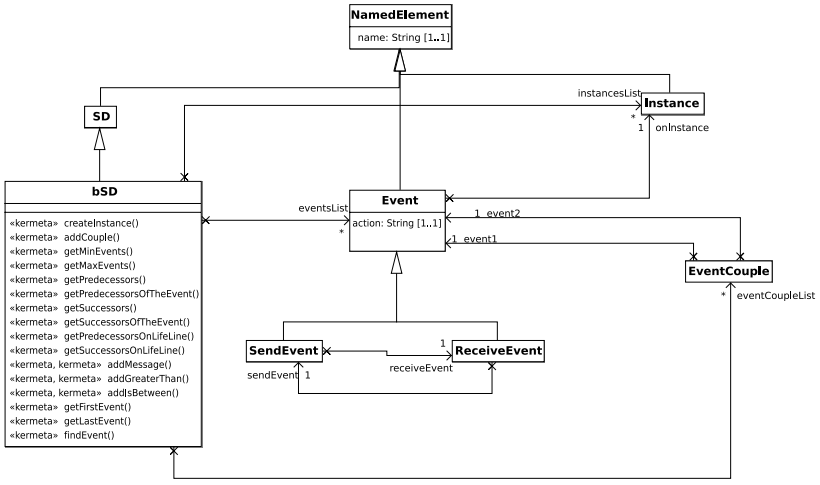

Figure 3 – Métamodèle de diagramme de séquence simple

Le DSS $Retry$ de la Figure 2 contient quatre événements : l'envoi et la réception du message $try\ again$ ainsi que l'envoi et la réception du message $log\ in$. Le DSS contient également deux actions qui sont $try\ again$ et $log\ in$.

Nous noterons par $min(E) = \{e \in E | \forall e' \in E, e' \leq e \Rightarrow e' = e\}$, l'ensemble des événements minimaux, c'est à dire l'ensemble des événements n'ayant d'autres prédécesseurs qu'eux-mêmes. Nous noterons par $\widehat{\downarrow}_{\leq,E}(e) = \{e' \in E | e' \leq e\}$, l'ensemble des prédécesseurs de l'événement e, et par $\widehat{\uparrow}_{\leq,E}(e) = \{e' \in E | e \leq e'\}$, l'ensemble des successeurs de e. Ces deux notations peuvent être utilisées pour un sous-ensemble E' d'un ensemble E : $\widehat{\downarrow}_{\leq,E}E' = \{e \in E | \exists e' \in E', e \leq e'\}$ et $\widehat{\uparrow}_{\leq,E}E' = \{e \in E | \exists e' \in E', e' \leq e\}$. Quand les flèches n'ont pas de chapeau, on utilise l'ordre $<$ au lieu de $\leq$. Par exemple, $\downarrow_{\leq,E}(e) = \{e' \in E | e' < e\}$.

Les DSSs définissent seulement des comportements finis sans réelles alternatives, mais ils peuvent être composés pour produire des comportements plus complexes. Pour cela, les DSs ont été étendus par des DSs composites (DSCs) qui composent des DSSs en utilisant des opérateurs de compositions (*Interaction Operator*). Trois opérateurs fondamentaux sont : seq, alt et $loop$. L'opérateur seq spécifie une composition séquentielle faible entre les comportements de deux opérandes DSs (tous les événements du premier opérande situés sur un objet o doivent être exécutés avant les événements du deuxième opérande situés sur le même objet o). Nous noterons par $M_1 \bullet M_2$, la composition séquentielle faible des deux DSSs M_1 et M_2. Formellement, cette composition est définie par $M_1 \bullet M_2 = (E_1 \uplus E_2, \leq_{1\bullet 2}, \alpha_1 \cup \alpha_2, \phi_1 \cup \phi_2, A_1 \cup A_2, \prec_1 \uplus \prec_2)$, où : $\leq_{1\bullet 2} = \left(\leq_1 \uplus \leq_2 \uplus \{(e, e') \in E_1 \times E_2 | \phi_1(e_1) = \phi_2(e_2)\}\right)^*$. L'opérateur alt définie un choix entre l'ensemble des opérandes. L'opérateur $loop$ spécifie une itération d'une interaction et crée des comportements infinis. La Figure 2 montre un exemple de DS composite nommé "sd composite" où par exemple, les DSSs $Accept$ et $Retry$ sont composés alternativement. Un composite DS peut être vu comme un générateur de DSSs. En effet, le DS composite de la Figure 2 génère les DSSs $Propose \bullet Retry$ et $Propose \bullet Accept$. Dans le cas général, un DS composite peut générer un ensemble infini de DSSs (par exemple, par utilisation de l'opérateur $loop$). Dans ce papier, nous ne considérons que des ensembles finis de DSSs, c'est à dire des modèles ne regroupant que des DSSs ou des DS composites ne générant qu'un ensemble fini de DSSs. Le tissage d'aspects prenant en compte l'ensemble des constructions des DSs fera l'objet de travaux futurs.

2.2. *Kermeta*

Kermeta est une plateforme open-source de métamodélisation développée par l' équipe Triskell. Au coeur de cette plateforme se trouve le langage Kermeta, conçu comme une extension du langage EMOF (Essential Meta-Object Facilities)(OMG, 2004) proposé par l'Object Management Group (OMG). Les langages de métamodélisation tel que EMOF ne permettent de modéliser que des structures (grâce à des concepts tel que les classes, attributs ou associations). Le langage Kermeta ajoute à

cela la possibilité de décrire la sémantique et le comportement de ces structures grâce à un langage d'action. Le langage d'action de Kermeta a été conçu spécialement pour la manipulation de modèles. Cela rend le langage Kermeta particulièrement adapté à la définition à la fois de la structure et de la sémantique de métamodèles ou langages, mais aussi à l'écriture de transformations de modèles ou de contraintes sur des modèles (Muller *et al.*, 2005b).

La structure de Kermeta reste suffisamment proche de EMOF pour être pleinement compatible avec des outils existants tel que le framework de métamodélisation d'Eclipse EMF (Budinsky *et al.*, 2003). La partie action de Kermeta est constituée d'un langage d'actions impératif qui inclut entre autre :

– des concepts propres au domaine de la manipulation de modèles comme par exemple les associations,

– des concepts orientés-objets classiques tels que l'héritage multiple ou la redéfinition avec une politique de liaison dynamique,

– des expressions et fermetures analogues à celles d'OCL (Object-Constraint Language)

Une description détaillée de la construction de Kermeta est présentée dans (Muller *et al.*, 2005a). L'ensemble de la plateforme Kermeta et des documents sur le langage sont disponibles sur le site de Kermeta (www.kermeta.org).

Kermeta a été utilisé avec succès pour implanter une technique de composition de diagrammes de classes (Reddy *et al.*, 2005) mais aussi comme langage de transformations de modèles (Muller *et al.*, 2005b). L'utilisation de Kermeta pour l'implantation des techniques proposées dans cet article a deux intérêts majeurs. Premièrement, cela permet d'encapsuler les algorithmes de détection et de composition directement dans le métamodèle de scénario utilisé (par exemple UML 2.0). Deuxièmement, du fait de sa compatibilité avec les outils Eclipse, tout modèle manipulé en Kermeta peut facilement être édité, stocké ou visualisé avec un outil Eclipse.

2.3. *Aspect comportemental*

Un aspect comportemental est une paire $A = (P, Ad)$ de DSSs. P est le point de coupure, c'est à dire un DSS interprété comme un prédicat sur la sémantique d'un scénario de base satisfait par toutes les parties détectées. Ad est un advice, c'est à dire le comportement désiré là où P est détecté dans le scénario de base. De la même manière qu'AspectJ où un aspect peut être inséré $autour$ (around), $avant$ (before), ou *après* (after) un point de jonction, avec notre approche, un advice peut indifféremment compléter le comportement détecté, le remplacer par un nouveau comportement, ou encore le supprimer entièrement. Le fait d'exprimer des aspects avec des scénarios permet de bénéficier de certains avantages liés aux scénarios. En particulier, il est relativement facile d'exprimer des points de coupure représentant des successions de messages ou des comportements parallèles. De ce fait, un aspect visant, par exemple,

à synchroniser des actions parallèles localisées sur différents objets est relativement facile à exprimer.

La Figure 4 montre un aspect comportemental qui permet de sauvegarder un échec de "log in". Le tissage de cet aspect dans le DS $Composite$, représenté sur la Figure 2, nous donne le DS nommé $Result$. Un exemple plus conséquent nécessiterait une étude de cas plus complète qui n'est pas l'objet de cet article.

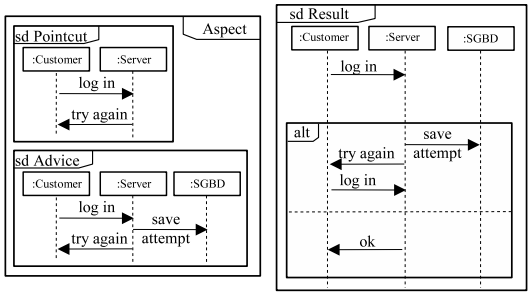

Figure 4 – Un exemple d'aspect comportemental et résultat du tissage

3. Détection

Cette section présente un algorithme qui permet de détecter le DSS point de coupure dans un DSS de base. Cette détection ne sera pas uniquement fondée sur la structure et la syntaxe du modèle de base, mais également sur la sémantique de ce modèle, en l'occurrence sur le respect de l'ordre des événements spécifié sur le point de coupure.

Du fait de la nature sémantique du tissage proposé, plusieurs stratégies de détection peuvent être envisagées pour la détection d'un point de coupure dans un modèle de base. La stratégie choisie détermine si un tissage doit avoir lieu ou pas. Dans notre approche, un point de coupure est détecté dans un DSS de base s'il est isomorphe à une partie du DSS de base. Le choix de la stratégie est alors déterminé par la définition choisie de la notion de $partie$ d'un DSS. Voici trois définitions, de plus en plus restrictives, pouvant qualifier une partie d'un DSS.

Définition 2 (Motif) *Soit $M = (I, E, \leq, A, \alpha, \phi, \prec)$ un DSS. Nous dirons que $M' = (I', E', \leq', A', \alpha', \phi', \prec')$ est un motif de M si :*

– $I' \subseteq I, E' \subseteq E, A' \subseteq A, \alpha' = \alpha_{|E'}, \phi' = \phi_{|E'}$;
– $\leq' = \leq_{|E'}, \prec' = \prec_{|E'}, \forall (e, f) \in \prec, e \in E' \Leftrightarrow f \in E'$.

La Figure 5-a montre plusieurs DSSs. Les messages $m1$ et $m2$ forment un motif dans les DSSs $M1$, $M2$ et $M3$. Notons que dans un motif, l'ordre des événements est le

même que celui du DSS initial restreint aux événements du motif. C'est pour cette raison que les messages $m1$ et $m2$ ne forment pas un motif dans $M4$, car avec les messages $m1$ et $m2$ seuls, la réception du message $m1$ n'est pas ordonnée par rapport à l'envoi du message $m2$, tandis que dans le DSS $M4$, ces deux événements sont ordonnés (par transitivité) à cause du message $m5$. Par contre, un motif n'est pas forcément $clos$, c'est à dire que des événements peuvent être présents entre les événements du motif (par exemple, il y a un message $m4$ entre les messages $m1$ et $m2$ dans le DSS $M3$).

Définition 3 (Partie Close) *Soit $M = (I, E, \leq, A, \alpha, \phi, \prec)$ un DSS. Nous dirons que $M' = (I', E', \leq', A', \alpha', \phi', \prec')$ est une partie close de M si :*

– *M' est un motif de M et $\widehat{\downarrow}_{\leq,E} E' \cap \widehat{\uparrow}_{\leq,E} E' = E'$.*

Dans la Figure 5-a, les messages $m1$ et $m2$ forment une partie close dans les DSSs $M1$ et $M2$. Une partie close est un motif où aucun événement n'appartenant pas à la partie ne peut être intercalé entre les événements de la partie (par ex., les messages $m1$ et $m2$ ne forment plus une partie close de $M3$). Par contre, une partie close peut être "encerclée" par d'autres événements (par ex., dans le DSS $M2$, la partie close est encerclée par le message $m0$).

Définition 4 (Sous Diagramme de Séquence Simple) *Soit M un DSS. Nous dirons que M' est un sous DSS de M s'il existe deux DSSs X et Y tel que $M = X \bullet M \bullet Y$*

Cette dernière définition est la plus restrictive. Dans la Figure 5-a, les messages $m1$ et $m2$ forment un sous DSS uniquement dans le DSS $M1$.

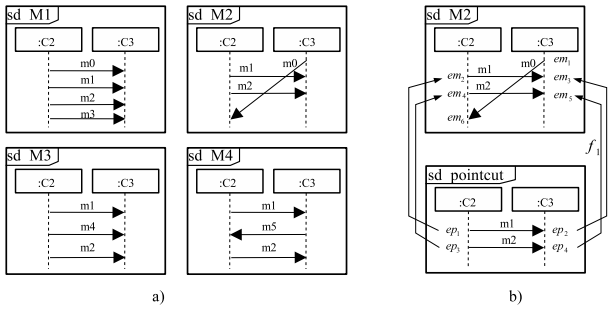

Figure 5 – Illustration des notions de partie et de morphisme

Pour formellement définir la détection d'un point de coupure dans un scénario de base, il nous reste à définir la notion de morphisme et d'isomorphismes entre DSSs.

Définition 5 (Morphisme de DSSs) *Soit deux DSSs* $M = (I, E, \leq, A, \alpha, \phi, \prec)$ *et* $M' = (I', E', \leq', A', \alpha', \phi', \prec')$. *Un morphisme de DSSs, de* M *à* M' *est un triplet* $\mu =< \mu_0, \mu_1, \mu_2 >$ *de morphismes, où* $\mu_0 : I \rightarrow I'$, $\mu_1 : E \rightarrow E'$, $\mu_2 : A \rightarrow A'$ *et :*

$(i)\ \forall (e,f) \in E^2, e \leq f \Rightarrow \mu_1(e) \leq' \mu_1(f)$ $(ii)\ \forall (e,f) \in E^2, e \prec f \Rightarrow \mu_1(e) \prec' \mu_1(f)$
$(iii)\ \mu_0 \circ \phi = \phi' \circ \mu_1$ $(iv)\ \mu_2 \circ \alpha = \alpha' \circ \mu_1$

Notons que les propriétés (i) et (ii) garantissent que par un morphisme de DSSs l'ordre des événements est préservé ainsi que la nature des événements (par exemple, un envoi de message de M sera toujours associé à un envoi de message de M'). La propriété (iii) signifie que tous les événements localisés sur un objet de M sont envoyés sur un seul objet de M'. La Figure 5-b montre un morphisme $f =< f_0, f_1, f_2 >: pointcut \rightarrow M2$ où seul le morphisme f_1 associant les événements est représenté (par ex., l'événement ep_1, qui représente l'envoi du message $m1$, est associé à l'événement em_2).

Définition 6 (Isomorphisme de DSSs) *Un morphisme de DSSs* $\mu = (\mu_0, \mu_1, \mu_2)$ *d'un DSS* M *à un DSS* M' *est un isomorphisme si les trois morphismes* μ_0, μ_1, *et* μ_2 *sont isomorphiques et si le morphisme réciproque* $\mu^{-1} = (\mu_0^{-1}, \mu_1^{-1}, \mu_2^{-1})$ *est également un morphisme de DSSs.*

Nous définissons alors de manière générale la notion de détection :

Définition 7 (Détection) *Nous dirons qu'un point de coupure* P *est détecté dans un DSS* M *si et seulement si il existe un isomorphisme de DSSs* $\mu = (\mu_0, \mu_1, \mu_2)$ *de* P *vers une partie* M' *de* M *où les morphismes* μ_0 *et* μ_2 *sont des morphismes identités (*P *et* M' *ont les mêmes objets et les mêmes noms d'action).*

Dans la Figure 5-b, il est facile de noter que le DSS $pointcut$ est détecté dans le DSS $M2$, car il existe un isomorphisme de DSSs satisfaisant les conditions entre le DSS $pointcut$ et la partie close de $M2$ formée par les messages $m1$ et $m2$.

Par la suite, nous allons toujours considérer qu'une partie d'un DSS est une partie close. Il est cependant important de noter que pour les trois définitions d'une partie d'un DSS proposées, la détection d'un point de coupure est fondée sur la sémantique du langage de scénarios utilisé puisque il n'y a pas seulement le nom des messages de préservé mais également l'ordre partiel définie par le point de coupure.

De notre point de vue, le choix de considérer une partie d'un DSS comme une partie close semble le plus adapté car une partie close présente à nos yeux le meilleur compromis entre les trois définitions de parties proposées (motifs, partie close et sous DSS). La définition d'un sous DSS nous semble trop $restrictive$ car avec cette définition, le comportement recherché peut être présent dans un DSS sans pour autant être détecté s'il est, par exemple, "encerclé" par un message. A l'opposé, la définition de motif nous semble trop $laxiste$ car des messages peuvent être intercalés dans la partie recherchée (par exemple, sur la Figure 5-a, dans le DSS $M3$, les messages $m1$

et $m2$ sont un motif alors que le message $m4$ est présent entre $m1$ et $m2$). Malgré tout, l'intérêt de l'approche proposée est qu'un utilisateur puisse choisir à sa guise la sémantique voulu pour son tissage : il est donc libre de choisir la définition de parties qui lui convient le mieux, en adaptant l'algorithme de détection en fonction de la définition choisie.

L'Algorithme 1 permet de construire un isomorphisme $\mu = (\mu_0, \mu_1, \mu_2)$ d'un point de coupure P vers une partie close M' d'un DSS M, tel que μ_0 et μ_2 soient des morphismes identités (donc M' représente une partie détectée par P). Nous nommerons par $\pi_i(M) \subseteq E_M$ la projection d'un DSS M sur un objet i de M et par $\pi_E(M)$ la restriction d'un DSS M à un sous-ensemble $E \subseteq E_M$. De plus, nous utiliserons une fonction β_E qui pour un événement e de E donne la position de e sur l'objet contenant e. Plus précisément, la position d'un événement sur un objet est défini par le nombre d'événements qui le précèdent sur cet objet : $\forall e \in E, \beta_E(e) = card(\{e' \in E | \phi(e) = \phi(e') \wedge e' \leq e\})$. Enfin, la fonction $\Gamma_{E,o}(n)$ donne l'événement de E situé en nième position sur l'objet o ($\Gamma_{E,o}(n) = e$ tel que $\beta_E(e) = n \wedge \phi(e) = o$).

La première partie de l'algorithme (ligne 1 à 4) permet de construire, pour tous les objets d'un point de coupure P, des ensembles d'événements de M localisés sur le même objet, tels que les actions associées à ces événements soient les mêmes que ceux associés aux événements de P. Dans la deuxième partie de l'algorithme (ligne 5 à 13), nous construisons, quand c'est possible, une partie close M' de M. Après la première partie de l'algorithme, il reste à vérifier que l'ordre des événements de P est le même que l'ordre des événements associés de M'. Pour cela, on vérifie que pour chaque couple d'événements envoi-réception de P, les événements de M' à la même position forment également un couple envoi-réception.

4. Composition

Maintenant que nous savons où se situent les comportements décrits par un point de coupure dans le scénario de base grâce au processus de détection, il nous reste à composer le DSS Advice avec le scénario de base aux endroits détectés. Supposons que la partie détectée soit un sous DSS, noté P, du DS de base, noté B. Par définition, on peut écrire $B = B_1 \bullet P \bullet B_2$ (pour rappel, $\bullet$ est la composition séquentielle faible). Dans ce cas, si l'on note Ad l'advice représentant le comportement voulu, il suffit de remplacer P par Ad pour obtenir $B_1 \bullet Ad \bullet B_2$. Malheureusement, la composition de l'advice avec le DS de base n'est pas aussi simple si la partie détectée est une partie close du DS de base. En effet, une partie close d'un scénario de base, obtenue à partir de la détection d'un point de coupure, pouvant être "encerclée" par des messages (comme la partie close formée par les messages $m1$ et $m2$ dans le DSS $M2$ dans la Figure 5), il n'est pas possible de simplement remplacer la partie détectée par l'advice car le résultat ne peut pas toujours être exprimé avec des opérateurs de compositions classiques (comme avec l'opérateur de composition séquentielle). Il est donc nécessaire de définir un opérateur de composition prenant en compte les parties communes de l'advice et de la partie détectée dans un scénario de base pour

Algorithm 1 Détection (P,M)

entrée : point de coupure $P = (I_P, E_P, \leq_P, , A_P, \alpha_P, \phi_P, \prec_P)$,
DSS $M = (I_M, E_M, \leq_M, A_M, \alpha_M, \phi_M, \prec_M)$
sortie : $\mu = (\mu_0, \mu_1, \mu_2) : P \rightarrow M'$, M' partie close de M

1: Pour tout $i \in I_P$ faire
2: $w_i = \pi_i(M)$ `/* un mot de tous les événements de l'objet i */`
3: $V_i = \{v \in E^* \mid \exists u, w, w_i = u.v.w \wedge \alpha(v) = \alpha(\pi_i(P))\}$
`/*` $\forall v \in V_i$, `les événements de` v `ont le même nom d'action que les événements de` P `sur l'objet` i`*/`
4: Fin Pour
5: Si $\exists v_1, v_2, \ldots, v_{|I_P|} \in V_1 \times V_2 \times \cdots \times V_{|I_P|}$ tel que $\forall (e,f) \in \prec_P$, $\big(\Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e), \Gamma_{v_{\phi(f)},\phi(f)} \circ \beta_{E_P}(f)\big) \in \prec_M$ alors
6: $E_{M'} = min\{v_1, \ldots, v_{|I_P|} \in V_1 \times \cdots \times V_{|I_P|} | \forall (e,f) \in \prec_P, \big(\Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e), \Gamma_{v_{\phi(f)},\phi(f)} \circ \beta_{E_P}(f)\big) \in \prec_M\}$
`/* on vérifie qu'un couple d'événements envoi-reception a bien une correspondance avec un couple envoi-reception dans` M`, et avec la fonction` min `on prend le premier ensemble` $v_1, \ldots, v_{|I_P|}$ `satisfaisant les propriétés*/`
7: μ_0 est l'isomorphisme identité de I_P vers $\phi_M(E_{M'})$,
8: μ_2 est l'isomorphisme identité de A_P vers $\alpha_M(E_{M'})$,
9: μ_1 est l'isomorphisme de E_P vers $E_{M'}$ tel que $\forall e \in E_P$, $\mu_1(e) = \Gamma_{v_{\phi(e)},\phi(e)} \circ \beta_{E_P}(e)$ `/*` μ_1 `est construit en associant, pour chaque objet` o `de` I_p`, à l'événement de` o `en ième position, l'événement appartenant à` $E_{M'}$ `sur` o `en ième position.*/`
10: retour($\mu = (\mu_0, \mu_1, \mu_2)$)
11: Sinon
12: retour($null$)
13: Fin Si

produire un nouveau scénario qui ne contient pas des copies d'éléments similaires des deux opérandes. Nous proposons un opérateur de composition pour DSSs appelé *somme amalgamée gauche* qui s'inspire de l'opérateur somme amalgamée proposé dans (Klein *et al.*, 2004). Nous ajoutons le terme "gauche" car notre opérateur ne sera pas commutatif, mais il impose un rôle différent à chaque opérande.

La Figure 6 montre un exemple de somme amalgamée gauche où plus précisément les deux DSSs $base = (I_b, E_b, \leq_b, A_b, \alpha_b, \phi_b, prec_b)$ et $advice = (I_a, E_a, \leq_a, A_a, \alpha_a, \phi_a, prec_a)$ sont amalgamés. Pour cela, on utilise un troisième DSS que l'on appelle DSS $pointcut = (I_p, E_p, \leq_p, A_p, \alpha_p, \phi_p, prec_p)$ et deux morphismes de DSSs $f : pointcut \rightarrow base$ et $g : pointcut \rightarrow advice$ qui permettent de spécifier les parties communes des deux DSSs $base$ et $advice$. De plus, f doit définir une partie close M' du DSS $base$ tel que f soit un isomorphisme de $pointcut$ à M'. Il faut noter que le morphisme f est obtenu automatiquement grâce au processus de détection décrit dans la section précédente. Le morphisme g, qui indique les éléments communs à l'advice et au point de coupure, doit être spécifié lors de la définition de l'aspect. De cette manière, g permet de définir des advices génériques ou abstraits qui sont "instanciés" grâce à ce morphisme. L'advice ne doit, par exemple, pas obligatoirement contenir des objets portant des noms identiques aux objets présents dans le DSS point de coupure. Dans l'exemple proposé sur la Figure 4, le morphisme g n'est pas spécifié car il reste trivial (on associe les objets et les actions ayant les mêmes noms ainsi que les événements correspondant aux actions portant le même nom).

Sur la Figure 6, les éléments des DSSs $base$ et $advice$ ayant un antécédent commun par f et g seront considérés comme identique dans le DSS $result$, mais ils garderont les noms spécifiés dans le DSS $base$. Par exemple, les objets : X et : Y du DSS $advice$ sont remplacés par les objets : $C1$ et : $C2$. Tous les éléments du DSS $base$ ayant un antécédent γ par f tel que γ n'a pas d'image par g dans le DSS $advice$ seront supprimés (c'est le cas des événements eb_4 et eb_5). Enfin, tous les éléments des DSS $base$ et $advice$ n'ayant pas d'antécédent par f et g seront gardés tels quels dans le DSS $result$, mais les événements du DSS $advice$ formeront toujours un "bloc" autour duquel les événements du DSS $base$ seront ajoutés (par exemple, les événements du message c seront ajoutés après les événements ea_3 et ea_5 du DSS $advice$).

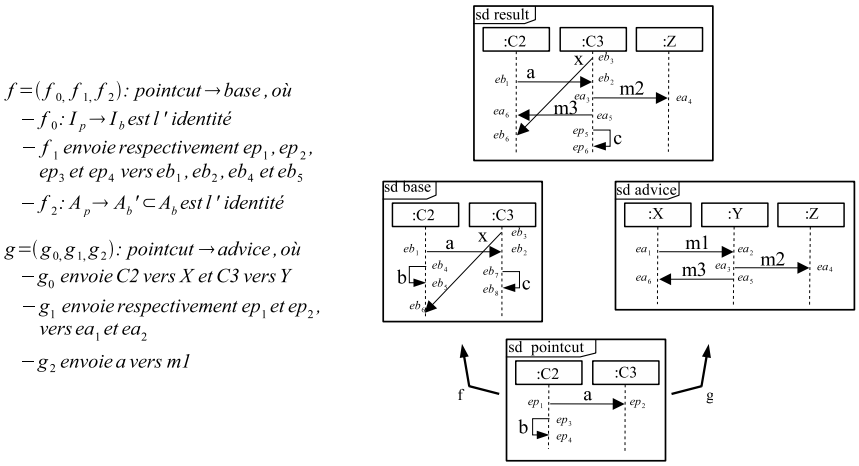

Figure 6 – Un exemple de somme amalgamée gauche

Nous définissons formellement une somme amalgamée gauche par :

Définition 8 (Somme Amalgamée Gauche) *Soit trois DSSs* $M_0 = (I_0, E_0, \leq_0, A_0, \alpha_0, \phi_0, \prec_0)$, $M_1 = (I_1, E_1, \leq_1, A_1, \alpha_1, \phi_1, \prec_1)$ *et* $M_2 = (I_2, E_2, \leq_2, A_2, \alpha_2, \phi_2, \prec_2)$ *ainsi que deux morphismes de DSSs* $f =< f_0, f_1, f_2 >: M_0 \to M_1$ *et* $g =< g_0, g_1, g_2 >: M_0 \to M_2$ *tel que* $f(M_0)$ *défini une partie close* M_1' *de* M_1 *et que* f *soit isomorphique de* M_0 *à* M_1'. *La somme amalgamée gauche de* M_1 *et* M_2 *est le DSS* $M = M_1 +_{f,g} M_2$ *où* $M = (I, E, \leq, A, \alpha, \phi, \prec)$ *est défini par :*

$I = I_1 \cup \{i_2 \in I_2 | \nexists i_0 \in I_0, g_0^{-1}(i_2) = i_0\}$;

$E = \{e_1 \in E_1 | \exists e_0 \in E_0, \exists e_2 \in E_2, f_1^{-1}(e_1) = e_0 \wedge g_1(e_0) = e_2\} \cup \{e_1 \in E_1 | \nexists e_0 \in E_0, f_1^{-1}(e_1) = e_0\} \cup \{e_2 \in E_2 | \nexists e_0 \in E_0, g_1^{-1}(e_2) = e_0\}$;

$$\leq = \left(\begin{array}{l} \{(e_1, e_2) \in (E_1 \cap E)^2 | e_1 \leq_1 e_2\} \\ \cup \{(e_1, e_2) \in (E_2 \cap E)^2 | e_1 \leq_2 e_2\} \\ \cup \{(e_1, e_2), e_1 \in (f_1(E_0) \cap E), e_2 \in (E_2 \cap E) | \exists e_2' \in E_2, e_2' = g_1 \circ f_1^{-1}(e_1) \wedge e_2' \leq_2 e_2\} \\ \cup \{(e_1, e_2), e_1 \in (E_2 \cap E), e_2 \in (f_1(E_0) \cap E) | \exists e_2' \in E_2, e_2' = g_1 \circ f_1^{-1}(e_2) \wedge e_1 \leq_2 e_2'\} \\ \cup \{(e_1, e_2), e_1 \in (\downarrow_{\leq_1, E_1} f_1(E_0) - f_1(E_0)), e_2 \in (E_2 \cap E) | \phi(e_1) = \phi(e_2)\} \\ \cup \{(e_1, e_2), e_1 \in (E_2 \cap E), e_2 \in (\uparrow_{\leq_1, E_1} f_1(E_0) - f_1(E_0)) | \phi(e_1) = \phi(e_2)\} \end{array} \right)^*$$

$$\forall e \in E, \alpha(e) = \begin{cases} \alpha_1(e) \text{ if } e \in E_1 \\ \alpha_2(e) \text{ if } e \in E_2 \end{cases} ;$$

$$\forall e \in E, \phi(e) = \begin{cases} \phi_1(e) \text{ if } e \in E_1 \\ \phi_2(e) \text{ if } e \in E_2 \end{cases} ;$$

$$A = \alpha(E) ;$$

$$\prec = (\prec_1 \cup \prec_2) \cap E^2$$

La première ligne de la définition de $\leq$ signifie que tout couple d'événements de E_1 présent dans E ordonné suivant $\leq_1$ reste ordonné suivant $\leq$. La deuxième ligne est équivalente mais pour les événements de E_2. La troisième ligne signifie qu'un événement e_1 de E_1 présent dans E précède un événement e_2 de E_2 présent dans E, s'il existe un événement e'_2 de E_2 précédant e_2 et ayant le même antécédent que e_1 dans M_0. De même, la quatrième ligne signifie qu'un événement e_2 de E_1 présent dans E succède un événement e_1 de E_2 présent dans E, s'il existe un événement e'_2 de E_2 succédant e_1 et ayant le même antécédent que e_2 dans M_0. Enfin, la cinquième ligne signifie qu'un événement e_1 de E_1 précédant la partie close détectée dans M_1, précédera tout événement e_2 de E_2 si e_1 et e_2 sont situés sur la même instance dans M. La dernière ligne suit le même raisonnement que la cinquième.

5. Implantation en Kermeta

Afin de valider les algorithmes proposés dans cet article, ils ont été implantés dans l'environnement Kermeta. Comme il a été détaillé précédemment, le processus de tissage se divise en deux phases : tout d'abord la détection qui produit un morphisme entre le point de coupure et le modèle de base, puis la composition qui utilise ce morphisme ainsi que le morphisme faisant le lien entre le point de coupure et l'advice pour composer l'advice dans le modèle de base. Chacune de ces phases est une transformation de modèles. La figure 7 détaille les modèles d'entrées et de sorties de ces transformations (les ellipses représentent les modèles manipulés, le rectangle noir en haut à gauche de chaque modèle précise à quel métamodèle il est conforme). L'ensemble des modèles manipulés par les transformations sont des diagrammes de séquence à l'exception des morphismes.

La première étape dans l'implantation de transformations de modèles en Kermeta est la définition des métamodèles d'entrée et sortie des transformations. Les métamodèles ont été définis avec le modeleur Omondo (Omondo, 2006) qui offre, en plus de l'édition de modèles UML, la possibilité d'éditer des métamodèles. La figure 3 présente le métamodèle de diagramme de séquence obtenu. Une fois le métamodèle défini, le framework de métamodélisation d'éclipse (EMF) offre des outils génériques pour créer, éditer et sauvegarder des modèles conformes à ce métamodèle. Kermeta permet d'une part de compléter le métamodèle en spécifiant le corps des opérations contenues dans le métamodèle et d'autre part de manipuler les modèles créés avec EMF. De la même manière, un métamodèle très simple a été défini pour manipuler les

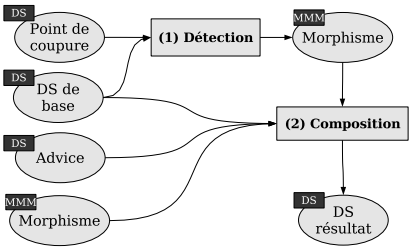

Figure 7 – Transformation de Modèles

morphismes entre diagrammes de séquence. Ce métamodèle est composé d'une classe Morphisme qui encapsule des associations vers deux diagrammes de séquence.

Une fois les métamodèles définis, les transformations elles-mêmes doivent être conçues et implantées. Dans notre cas, deux classes Kermeta (*Detection* et *Composition*) ont été définies pour encapsuler les deux phases du tissage. Différentes variantes de l'algorithme de détection ont été implantées par sous classage de la classe *Detection* (une variante pour chaque type de partie d'un DSS : motif, partie close ou sous DSS). L'implantation du tissage d'aspects comportementaux a permis de mettre au point et de valider les algorithmes proposés dans cet article sur un ensemble d'exemples simples.

6. Travaux existants

Clarke et Walker (Clarke *et al.*, 2001) utilisent des *UML templates* pour définir des aspects. La question abordée par leur travail est davantage liée à la manière de spécifier un aspect qu'à un processus de tissage. Ils composent des propriétés structurelles et statiques d'un aspect dans des modèles de bases, mais ils ne composent pas des propriétés comportementales d'un aspect.

Similairement à notre approche, Whittle et Araujo ((Whittle *et al.*, 2004) et (Araujo *et al.*, 2004)) représentent des aspects comportementaux avec des scénarios. Les scénarios représentant les aspects sont modélisés par des patrons de spécification d'interactions (introduits dans (France *et al.*, 2004)) et sont composés avec des scénarios de base. Le processus de tissage défini dans (Whittle *et al.*, 2004) compose des scénarios et propose divers opérateurs de composition non définis formellement tandis que dans (Araujo *et al.*, 2004), le tissage est accompli en deux étapes. La première étape consiste à générer des machines à états à partir des aspects et des scénarios de base. Le reste du tissage consiste alors à composer ces machines à états. Cependant, dans ces deux approches, la détection des points de jonction n'est pas automatisée : les utilisateurs doivent les spécifier grâce à une relation de jonction ($binding$) entre un aspect et un modèle de base.

Plus généralement, dans (Douence *et al.*, 2002) et (Douence *et al.*, 2001), Douence et al sont intéressés par la détection de patrons d'événements (event pattern matching),

qui est relativement proche de notre approche. Une différence significative est qu'ils utilisent un moniteur pour détecter les patrons d'événements durant le "runtime" de l'exécution d'un programme, tandis que notre tissage est accompli statiquement à un niveau de modélisation.

Les modèles d'aspects et particulièrement les mécanismes pour identifier les points de jonction (les langages de définition des points de coupures) jouent un rôle critique dans l'applicabilité de méthodologies orientées aspects. Selon Kiczales (Kiczales, 2003), les langages de définition des points de coupures ont probablement l'un des rôles les plus importants dans le succès des technologies orientées aspects, mais la plupart des solutions proposées jusqu'ici sont attachées à la syntaxe des programmes manipulés. Ostermann et al (Ostermann *et al.*, 2005) essaient de résoudre ce problème en proposant un modèle de point de jonction statique qui exploite des informations de différents modèles de sémantiques de programmes. Ils montrent que ce modèle de points de jonctions augmente le niveau d'abstraction et de modularité des points de coupures.

7. Conclusion

Ce papier propose une technique permettant de tisser des comportements décrits sous forme de scénarios dans un scénario de base. Le processus de tissage se décompose en deux phases. Tout d'abord, une phase de détection permettant d'identifier, en se fondant sur la sémantique des scénarios, des parties particulières dans le scénario de base, puis une phase de composition, définie formellement, qui construit le scénario voulu. Ces deux phases sont implémentées comme des transformations de modèles dans l'environnement Kermeta.

Ce tissage peut facilement être utilisé avec un modèle de base contenant un ensemble fini de scénarios, l'aspect étant appliqué à chaque scénario de l'ensemble. Grâce à l'utilisation d'un algorithme de composition de diagramme de classes, le diagramme de classes du modèle de base peut-être mis à jour afin d'inclure les classes et opérations utilisées dans les scénarios des aspects.

Afin d'améliorer les techniques proposées, nous travaillons actuellement sur algorithme de détection permettant de supporter l'ensemble des constructions des diagrammes de séquence d'UML 2.0 (en particulier la construction *loop* qui permet de décrire des comportements infinis). De même, notre langage de détection de point de coupure peut être amélioré en permettant, par exemple, un mécanisme de "wildcard" sur les noms des messages (ou noms d'actions) présent dans le DS point de coupure.

8. Remerciements

Ces travaux ont été partiellement supportés par le Réseau d'Excellence AOSD-Europe ainsi que le projet RNTL OpenEmbeDD.

9. Bibliographie

Araujo J., Whittle J., Kim, « Modeling and composing scenario-based requirements with aspects », *Proc. of RE2004*, Kyoto, Japan, September, 2004.

Budinsky F., Steinberg D., Merks E., Ellersick R., Grose T., *Eclipse Modeling Framework*, The Eclipse Series, Addison Wesley Professional, 2003.

Clarke S., Walker R. J., « Composition Patterns : An Approach to Designing Reusable Aspects », *International Conference on Software Engineering*, p. 5-14, 2001.

Douence R., Fradet P., Südholt M., « A Framework for the Detection and Resolution of Aspect Interactions », *Proceedings of GPCE'02*, LNCS, Springer, 2002.

Douence R., Motelet O., Südholt M., « A Formal Definition of Crosscuts. », *Reflection'01*, p. 170-186, 2001.

France R. B., Kim D.-K., Ghosh S., Song E., « A UML-Based Pattern Specification Technique », *IEEE TSE, vol.30(3), 193-206, March 2004*, 2004.

Kiczales G., « The Fun Has Just Begun », , Keynote of Aspect-Oriented Software Development (AOSD), 2003.

Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., Griswold W. G., « An Overview of AspectJ », *LNCS*, vol. 2072, p. 327-355, 2001.

Klein J., Caillaud B., Hélouët L., « Merging scenarios », *9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, vol. 133, ENTCS, Linz, Austria, p. 209-226, sep, 2004.

Muller P.-A., Fleurey F., Jézéquel J.-M., « Weaving Executability into Object-Oriented Meta-Languages », *Proc. of MODELS/UML'2005*, LNCS, Springer, Jamaica, 2005a.

Muller P.-A., Fleurey F., Vojtisek D., Drey Z., Pollet D., Fondement F., Studer P., Jézéquel J.-M., « On Executable Meta-Languages applied to Model Transformations », *Model Transformations In Practice Workshop*, Jamaica, 2005b.

OMG, « Meta Object Facility (MOF) 2.0 Core Specification », , OMG document ptc/04-10-15, 2004. http ://www.omg.org/cgi-bin/doc ?ptc/2004-10-15.

OMG, « UML Superstructure Specification, v2.0 », , OMG Document number formal/05-07-04, 2005.

Omondo, , 2006. http ://www.omondo.com.

Ostermann K., Mezini M., Bockisch C., « Expressive Pointcuts for Increased Modularity », *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*, Springer LNCS, 2005.

Rashid A., Moreira A. M. D., Araújo J., « Modularisation and composition of aspectual requirements. », *AOSD*, p. 11-20, 2003.

Reddy R., France R., Ghosh S., Fleurey F., Baudry B., « Model Composition - A Signature-Based Approach », *Aspect Oriented Modeling (AOM) Workshop*, Montego Bay, Jamaica, October, 2005.

Whittle J., Araújo J., « Scenario modelling with aspects. », *IEE Proceedings - Software*, vol. 151, n° 4, p. 157-172, 2004.

ANNEXE POUR LE SERVICE FABRICATION
A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LES ACTES :
   LMO 2006

2. AUTEURS :
   Jacques Klein — Franck Fleurey

3. TITRE DE L'ARTICLE :
   Tissage d'aspects comportementaux

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   Tissage d'aspects comportementaux

5. DATE DE CETTE VERSION :
   31 janvier 2006

6. COORDONNÉES DES AUTEURS :

– adresse postale :
IRISA/INRIA/Université de Rennes 1
Campus de Beaulieu
35042 Rennes Cedex, France
{jacques.klein, franck.fleurey}@irisa.fr
– téléphone : +33(0)4 92 94 27 48
– télécopie : +33(0)4 92 94 28 96
– e-mail : Roger.Rousseau@unice.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.22 du 04/10/2005.

8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
   `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com