

COOK : Réarchitecturisation des applications industrielles à objets

S. Ducasse, I. Alloui, A. Bergel, S. Cimpan, D. Pollet, H. Verjus

<http://stephane.ducasse.free.fr>

Abstract

Cook s'inscrit dans le contexte de la ré-ingénierie et de l'évolution d'applications industrielles à objets. La ré-ingénierie hérite des problèmes complexes liés à la maintenance : compréhension, analyse, transformation de programmes. Ce projet s'attaque aux problèmes de l'évolution des applications industrielles, d'une part en prenant explicitement en compte la notion d'architecture — notion souvent enfouie dans des millions de lignes de code — et d'autre part en se focalisant sur la modularisation des applications, c'est-à-dire comment repenser la structure de ces applications.

Cook aborde cette problématique par les axes suivants : l'extraction et la modélisation d'architecture, la définition de langages pour l'évolution dynamique d'architecture, l'aide à la compréhension et à la restructuration des applications existantes au niveau de leur structure de paquets, ainsi que la définition de nouveaux constructeurs de langage assurant une meilleure structuration.

Le projet a commencé au sein du laboratoire LISTIC de l'Université de Savoie en décembre 2005. Depuis septembre 2007, il se poursuit par la création d'une équipe, nommée RMoD, à l'INRIA Lille Nord Europe.

1 De l'inéluctable évolution des applications

Bien que les logiciels soient devenus un des tenants de notre industrie, leur développement reste une tâche parsemée d'embûches. Même les projets ayant du succès doivent faire face à ce que Parnas appelle *software aging*, le vieillissement du logiciel [6]. Le caractère chronique des problèmes de développement a amené Pressman à préférer l'expression maladie chronique (*chronique affliction*) au lieu de crise du logiciel (*software crisis*) [7]. Plusieurs facteurs inhérents au développement du logiciel mènent à cette situation : la complexité des domaines et des tâches modélisées, les problèmes liés à la gestion de projets et les relations humaines et les difficultés à comprendre les exigences du client, et le besoin *constant* d'adaptation et de changement.

Quelques études peuvent éclairer le lecteur sur ce besoin d'adaptation, qui est le contexte de ce projet. La maintenance logicielle est le nom donné au processus de modification d'un logiciel après qu'il ait été livré aux clients. Sommerville, en faisant référence à des études conduites dans les années 80 [3, 4], mentionne que de grandes organisations passent au moins 50% de leur développement total dans la maintenance de logiciels existants [8]. McKee mentionne que l'effort de maintenance se situe entre 65% et 75% de l'effort total [4]. Des études plus récentes confirment ces constats [1] et il n'est plus à démontrer que la phase de maintenance est une des phases de développement les plus coûteuses. Cependant, le terme maintenance cache une toute autre réalité que la simple correction de bugs [3, 5].

Sommerville catégorise les activités de maintenance comme suit¹ : *maintenance correctrice* (17%) c.-à-d. corriger des bugs, *maintenance adaptative* (18%) c.-à-d. adapter le logiciel à de nouvelles plates-formes, systèmes d'exploitation, etc. et *maintenance de perfection* (perfective) (65%) c.-à-d. implantation de nouvelles fonctionnalités [8]. De ces données il apparaît clairement que la maintenance, et donc *la plupart du coût de développement, est passée à faire évoluer les logiciels.*

Les lois de l'évolution du logiciel validées empiriquement par Lehman et Belady offrent une interprétation de cette situation : toute application utile pour ses utilisateurs sera forcée d'évoluer pour faire face à de nouvelles contraintes et demandes, et cette évolution entraîne une complexité accrue [2].

De nombreux travaux sont menés pour aider à l'évolution des logiciels. D'une part, des travaux portent sur des niveaux très proches du code, notamment dans la réingénierie du logiciel. D'autre part, dans le domaine des architectures logicielles, des travaux prennent en compte l'évolution à un niveau plus abstrait avec les langages de description d'architectures. Les processus de développement abordés dans ces derniers travaux sont essentiellement déductifs : l'architecture est d'abord représentée à un haut niveau d'abstraction, puis raffinée jusqu'à générer du code. Le projet COOK se propose d'utiliser

¹les pourcentages sont relatifs à l'effort total de maintenance

la notion d'architecture dans le processus de réingénierie, de définir des langages d'évolution et reconfiguration centrés architecture, d'aider à la compréhension des grandes applications et finalement de proposer de nouvelles constructions linguistiques pour obtenir une meilleure modularité dans les applications.

Les problèmes abordés dans le projet sont les suivants :

Modélisation de la notion d'architecture. La première étape est d'introduire cette notion dans Moose, l'environnement de réingénierie développé par le porteur du projet [DGLD05]. Ceci implique de pouvoir modéliser une architecture et les éléments qui la constituent en les reliant aux entités représentant le code de l'application.

Approches pour l'identification des architectures d'applications à objets. Identifier l'architecture à partir du code d'une application ou de son exécution est une information vitale pour comprendre et maintenir une application. Il s'agit d'exprimer et de proposer des outils pour la visualisation, la compréhension des architectures et leur extraction à partir du code. Souvent les logiciels lors de leur évolution détériorent leur architecture (*architectural drift*). Identifier de telles violations est primordial pour éviter des problèmes importants accompagnant l'évolution de telles applications.

Outils pour la remodularisation. Transformer une architecture étape par étape ou remodulariser une application existante est une tâche complexe mais importante. Nous développons des aides à la remodularisation d'une part en proposant des approches pour comprendre les structures existantes des applications puis des analyses qui guideront les modifications.

Langages pour l'évolution dynamique d'applications. Les approches de l'évolution des applications font souvent l'hypothèse d'un besoin de reconfiguration dynamique des composants et de l'architecture des applications. Ceci révèle un besoin en infrastructures et langages pour la reconfiguration et l'évolution des applications et de leur architecture.

Langages pour une meilleure modularisation. Les classes sont les abstractions traditionnelles des langages de programmation à objets ; elles jouent plusieurs rôles souvent en conflit : une classe doit être complète en tant que créatrice d'objets mais en même temps une classe doit être générique pour pouvoir être réutilisée. De nouvelles abstractions telles que les mixins ont émergé pour aider à la structuration des classes et ainsi favoriser la réutilisation au sein des applications. Dans cette lignée, nous avons défini la notion de trait ; les traits permettent de composer des classes à partir d'abstractions composables de plus fine granularité. Dans cette perspective, nous allons continuer (i)

à évaluer comment les traits peuvent servir à remodulariser des applications et (ii) étendre le modèle des traits pour gérer l'état des objets en plus de leur comportement.

Il est à noter que certains points proposés dans la description originale du projet ont été modifiés suivant une dynamique normale de recherche. Ces modifications n'ont pas perturbé la cohérence globale du projet mais renforcé les résultats en accentuant les points pertinents.

2 Résultats

En résumé, les résultats couvrent un spectre large du projet :

Élaboration d'un *état de l'art* extrêmement complet sur les approches d'extraction d'architecture : deux articles acceptés, dont le prix du meilleur article à CSMR [PDP⁺07], et un article dans IEEE Transactions of Software Engineering [DPP08], une des deux meilleures revues en génie logiciel).

Extraction de vues architecturales et définition d'un outil d'aide à l'extraction d'architectures nommé Uranus [PDP⁺07, DPP08].

Définition de *plusieurs visualisations* pour l'analyse fine des paquetages d'une application. Ces vues ont été intégrées à la plate-forme open-source de réingénierie Moose. Deux articles ont été publiés à International Conference on Software Maintenance [DGK06, DPS⁺07], un article à European Conference on Software Maintenance [AAD⁺08] and Reengineering. Plus récemment, un article de journal a été soumis [ADPA08] et deux articles sur l'amélioration des matrices de dépendances sont en préparation [BDLP08].

Support et langage pour l'évolution dynamique des architectures logicielles. Prototypage de Nimrod, un environnement de développement centré architecture. Nimrod fournit un langage de description architecturale (Nimrod ADL) qui décrit des architectures logicielles pouvant évoluer dynamiquement [Ver06a, Ver07]. Des travaux et expérimentations ont été aussi conduits pour exprimer des architectures dynamiquement évolutives. Un article a été publié dans les actes d'une conférence internationale [CVA07], un autre article a été accepté pour publications dans une revue francophone (TSI) [CVA09].

Identification des composants "vitaux" d'une architecture logicielle : les composants d'une architecture logicielle n'ont pas tous la même importance au sein de cette architecture ; aussi, identifier les composants qui ont un rôle prépondérant est une tâche importante avant toute

démarche de réarchitecturisation. Un article sur ce sujet a été publié dans les actes de la plus importante des conférences internationales portant sur les architectures logicielles : WICSA [ACV08].

Définition de *nouveaux modèles* de traits avec prise en compte des états et des conflits de noms lors de compositions. *Validation* du modèle original des traits par la réécriture d'une bibliothèque de flots dans le Smalltalk open-source Squeak. Un article a été publié dans OOPSLA, la meilleure conférence de la programmation objet [DWBN07], deux articles dans des conférences internationales [BDNW07, CDW07] ainsi que deux articles de journaux dans Computer Languages Elsevier [CDW08, BDNW07] et TOPLAS [DNS⁺06]).

Les architectures à base de services font l'objet de larges études ces dernières années. Un environnement de modélisation, d'analyse et d'exécution des architectures à base de services dynamiquement évolutives a été proposé et implémenté. Ces travaux ont fait l'objet de plusieurs rapports et publications (conférences francophones et internationales, chapitre de livre) : [PVO06, PV06, PV07, PV07a, PV07b, PV08, VP07, VP07, VP08]. Dans ce cadre, deux langages formellement fondés ont été définis : π -Diapason et Diapason*.

2.1 Uranus : un outil d'aide à l'extraction d'architectures

Après avoir mené un état de l'art conséquent sur les approches utilisées pour l'extraction d'architectures [PDP⁺07, DPP08] nous avons abordé la définition de vues architecturales. En effet, une architecture est souvent une vue ou un ensemble de vues sur un système. Ainsi il est important de pouvoir définir des vues, de les composer et que le réingénieur puisse émettre des hypothèses qu'il validera par la suite. Nous avons développé l'outil Uranus au sein de l'environnement Moose.

2.2 Outils pour la compréhension et l'analyse de grandes applications

Comprendre une application industrielle au niveau de sa structure à gros grain est important pour la compréhension de son architecture ou structure générale. De plus, il est aussi important de comprendre comment certaines propriétés se distribuent au sein des éléments qui constituent une application. Nous avons développé plusieurs visualisations pour répondre à ces problèmes. Ces visualisations sont intégrées à l'environnement Moose.

Distribution Map [DGK06] met en perspective comment une propriété est présente sur un ensemble d'éléments.

Ainsi nous avons pu voir comment les développeurs travaillent sur des parties spécifiques d'une application. Nous avons aussi étudié l'utilisation des symboles dans une application. Package Surface Blueprints [DPS⁺07] est une vue condensée des références entrantes et sortantes d'un package, triées par des utilisateurs ou fournisseurs de services. Package Fingerprint [AAD⁺08] donne une vue de l'utilisation de propriétés données par un package.

Nous avons enrichi les matrices de structure de dépendances afin d'offrir plus d'information à un réingénieur lors de l'analyse et l'identification de couches architecturales [BDLP08]. Les analyses faites sur Squeak montrent le potentiel d'amélioration du code et surtout l'efficacité de l'approche sur des cas réels.

Infrastructure pour les outils d'analyse. Orthogonalement à la problématique du projet, nous contribuons au développement d'une infrastructure dirigée par les modèles au sein de l'environnement open-source de réingénierie Moose [DG06, DGKR08]. Ceci nous permet d'obtenir des outils de réingénierie plus génériques et plus intégrés.

2.3 Identification d'aspects à partir du code source des applications à objets

La programmation par aspects promeut la séparation des préoccupations transverses. Identifier ces préoccupations dans du code développé selon une méthodologie objet est difficile mais il l'est encore plus dans du code objet montrant des signes de programmation procédurale. Nous avons utilisé l'analyse formelle de treillis de Galois (FCA) pour aider à l'identification d'objets et d'aspects dans du code objet procédural [BDR08, BDH08].

2.4 Nimrod : un langage pour l'évolution dynamique d'architecture

Élaboration de travaux sur l'évolution des architectures logicielles, en particulier Nimrod un environnement support à l'évolution dynamique des architectures et Pi-Diapason, un environnement pour la réarchitecturisation d'architectures orientées services [PV07, PV07a, PV08]. Des langages ont également été définis permettant l'expression de l'évolution dynamique des architectures logicielles [CVA07, ACV08, CVA09].

L'environnement Nimrod permet d'une part de formaliser des architectures, de les exécuter (une description architecturale en Nimrod ADL peut être exécutée) et, d'autre part, de faire évoluer les architectures en cours d'exécution en modifiant la description architecturale "au vol" [Ver06b]. Les premières expérimentations de ce nouvel environnement portent sur des architectures simples. D'autres expériences sur des architectures plus complexes

doivent être menées tout en poursuivant le développement de l'environnement Nimrod. Nimrod ADL combine les points forts des ADL que l'on trouve dans l'état de l'art (permettant de décrire la structure et le comportement les architectures, ainsi que la mobilité) et les points forts des langages orientés objet dynamiques (liaison tardive, interprétation du langage par une machine virtuelle).

2.5 Diapason : une approche formelle pour l'expression, l'analyse et l'exécution d'architectures à base de services dynamiquement évolutives

Un environnement a été défini et implémenté pour l'expression, l'analyse et l'exécution d'architectures à base de services dynamiquement évolutives. Cet environnement repose sur deux langages en couches, formellement fondés permettant d'une part la description d'architectures à base de services dynamiquement évolutives et d'autre part l'expression de propriétés. Définition du langage π -Diapason, un langage pour l'expression d'orchestrations de services Web (Architectures Orientées Services ou SOA). Ce langage fondé sur le π -calcul fournit un DSL (Domain Specific Language) pour l'expression d'orchestrations de services Web. Il intègre des mécanismes permettant aux orchestrations de services Web d'évoluer dynamiquement, en cours d'exécution. Un modèleur graphique a également été développé permettant au concepteur de SOA de définir graphiquement une orchestration de services Web. Une fois formalisée en π -Diapason, une orchestration est déployée comme un service Web à part entière et peut être réutilisée dans d'autres orchestrations. Ce nouveau service Web intègre, en plus de la définition π -Diapason, une machine virtuelle permettant l'exécution dynamique du processus d'orchestration. Toute orchestration formellement définie peut également être vérifiée en fonction de quelques propriétés intéressantes dans le cadre des Architectures Orientées Services. L'environnement appelé Diapason, met en œuvre l'approche proposée avec le langage π -Diapason. Dans le cadre des SOA, ces travaux expérimentaux ont pour objectif de transformer une architecture logicielle classique, en une architecture orientée services.

Dans le cadre de l'étude de l'évolution des architectures à base de services (SOA), et pour compléter le langage π -Diapason, un langage de description de propriétés des SOA a été défini ; il s'agit du langage Diapason* qui fait partie des langages de la classe de la logique temporelle arborescente basée sur actions et qui permet l'analyse de propriétés de sûreté et de vivacité sur toute orchestration décrite en avec le langage π -Diapason. L'analyse d'une architecture à base de services et donc ses propriétés est un aspect important lorsqu'on étudie les architectures sous l'angle de l'évolution, leur réarchitecturisation. Par exemple, pour

toute évolution, s'accompagne-t-elle de pertes sémantiques et de propriétés qui ne seraient plus vérifiées, voire de nouvelles propriétés qu'il faudrait vérifier ? Un outil d'analyse de traces d'exécution a été développé, permettant de vérifier les propriétés de vivacité et de sûreté sur une orchestration π -Diapason.

2.6 Les Traits pour une meilleure modularisation des applications

Lors d'un projet de recherche précédent nous avons défini les traits [DNS⁺06]. Les traits sont des ensembles de méthodes qui servent d'entité de réutilisation pour les classes. Une classe déclare utiliser un ensemble de traits qui lui apportent du comportement supplémentaire. En plus de définir du comportement, les traits déclarent des méthodes requises. Ces méthodes sont nécessaires à l'utilisation du trait par une classe. Les traits ne définissent pas d'état, à la place, ils peuvent nécessiter des accesseurs. Lors de conflits, l'entité composante a le contrôle de la résolution du conflit.

Les traits ont reçu une attention particulière de la communauté scientifique : ainsi Fortress, un nouveau langage de calcul scientifique développé par SUN Microsystems est basé sur les traits. Le langage Scala de l'EPFL intègre un mécanisme similaire. Les langages Squeak, AmbientTalk, Slate ont également intégré les traits. Plusieurs systèmes de types ont été définis pour permettre la validation de programmes utilisant des traits.

Dans le cadre du présent projet, nous avons d'une part validé expérimentalement le modèle original des traits, et d'autre part raffiné ce modèle.

Validation. Pour évaluer l'efficacité des traits, des bibliothèques ont été refactorisées, montrant une réutilisation importante du code. Cependant, bien que ces travaux soient intéressants, ils ne permettent pas de rencontrer tous les problèmes d'utilisation des traits ; ceci parce que les bibliothèques d'origine étaient réalisées et pensées avec les contraintes de l'héritage simple. Nous avons donc évalué l'expressivité des traits lors de la réalisation d'un projet complet, en se servant des traits comme unité de réutilisation de comportement dès le départ. Le cas d'étude était la conception d'une nouvelle bibliothèque de flots appelée Nile [CDW07, CDW08].

Nouveau modèle de traits. Nous avons travaillé à l'introduction de l'état dans les traits [BDNW07, BDNW07], ainsi qu'à la résolution des conflits de noms non-anticipés [DWBN07]. Ces deux modèles ont été définis formellement et implémentés. Bien qu'ils soient prometteurs, nous souhaitons maintenant prendre de la distance et mener une nouvelle validation empirique afin de voir par la pratique et en grandeur réelle leur adéquation avec des problèmes fréquents.

2.7 Interactions avec des laboratoires étrangers

Nous avons eu la visite des chercheurs suivants : Dr. T. Girba, A. Kuhn, L. Renggli, Dr. M. Denker, Prof. O. Nierstrasz du Software Composition Group de l'Université de Berne (Suisse), Prof. R. Wuyts (un mois) de l'Université Libre de Bruxelles (Belgique), Prof. K. Mens et J. Bri-chau de l'Université de Louvain la neuve (Belgique), K. Gybels (un mois) de Prog de la Vrije Univeriteit Brussels, Dr. A. Bergel, Trinity-College-Dublin, Prof. A. Black, Portland University, Prof. Brian Henderson-Sellers, University of Sydney, Dr. T. van Cutsem de la VUB, Dr. G. Bracha de Cadence, Dr. Blay-Fornarino de l'I3S de l'Université de Nice Sophia Antipolis.

Séminaires et réseaux. Nous avons participé aux séminaires Sattose (Belgique), Moose Workshops à Berne, Séminaires autour de Moose (février 2006), Kermeta Workshop à l'IRISA. Nous participons au réseau du ERCIM working group EVOL.

Publications

Articles de journaux internationaux

- [ADPA08] Hani Abdeen, Stéphane Ducasse, Damien Pollet, and Ilham Alloui. Package fingerprint : a visual summary of package interfaces and relationships. *Under submission at International Journal of Software Maintenance*, 2008.
- [CDW08] Damien Cassou, Stéphane Ducasse, and Roel Wuyts. Traits at work : the design of a new trait-based stream library. *Journal of Computer Languages, Systems and Structures*, pages 00–00, 2008. To appear.
- [CVA09] Sorana Cîmpan, Hervé Verjus, and Ilham Alloui. Gestion de l'évolution dans le cadre d'une approche d'ingénierie logicielle centrée architecture. *Technique et Science Informatiques (TSI)*, 2009.
- [DGKR08] Stéphane Ducasse, Tudor Gîrba, Adrian Kuhn, and Lukas Renggli. Meta-environment and executable meta-language using smalltalk : an experience report. *Journal of Software and Systems Modeling (SOSYM)*, 2008. To appear.
- [DNS⁺06] Stéphane Ducasse, Oscar Nierstrasz, Nathanael Schärli, Roel Wuyts, and Andrew Black. Traits : A mechanism for fine-grained reuse. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(2) :331–388, March 2006.
- [DPP08] Stéphane Ducasse, Damien Pollet, and Loïc Poyet. Software architecture reconstruction : A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 2008.

Chapitres de livres

- [DGLD05] Stéphane Ducasse, Tudor Gîrba, Michele Lanza, and Serge Demeyer. Moose : a collaborative and extensible reengineering environment. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 55–71. Franco Angeli, Milano, 2005.
- [PV08] F. Pourraz and H. Verjus. *Managing Service-Based EAI Architectures Evolution Using a Formal Architecture-Centric Approach*, volume 3 of *Lecture Notes in Business Information Processing*, pages 269–280. Springer Berlin Heidelberg, February 2008.

Articles de conférences internationales

- [AAD⁺08] Hani Abdeen, Ilham Alloui, Stéphane Ducasse, Damien Pollet, and Mathieu Suen. Package reference fingerprint : a rich and compact visualization to understand package relationships. In *European Conference on Software Maintenance and Reengineering (CSMR)*, pages 213–222. IEEE Computer Society Press, 2008.
- [ACV08] I. Alloui, S. Cimpan, and H. Verjus. Towards software architecture physiology : Identifying vital components. In IEEE Computer Society, editor, *Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pages 293–296, Vancouver, Canada, February 2008.
- [BDH08] Muhammad Usman Bhatti, Stéphane Ducasse, and Marianne Huchard. Reconsidering classes in procedural object-oriented code. In *International Conference on Reverse Engineering (WCRE)*, 2008.
- [BDLP08] Alexandre Bergel, Stéphane Ducasse, Jannik Laval, and Romain Peirs. Enhanced dependency structure matrix for moose. In *FAMOOSt, 2nd Workshop on FAMIX and Moose in Reengineering*, 2008.
- [BDNW07] Alexandre Bergel, Stéphane Ducasse, Oscar Nierstrasz, and Roel Wuyts. Stateful traits. In *Advances in Smalltalk — Proceedings of 14th International Smalltalk Conference (ISC 2006)*, volume 4406 of *LNCS*, pages 66–90. Springer, August 2007.
- [BDR08] Muhammad Usman Bhatti, Stéphane Ducasse, and Awais Rashid. Aspect mining in procedural object-oriented code. In *International Conference on Program Comprehension (ICPC 2008)*, 2008.
- [CDW07] Damien Cassou, Stéphane Ducasse, and Roel Wuyts. Redesigning with traits : the Nile stream trait-based library. In *Proceedings of the 2007 International Conference on Dynamic Languages (ICDL 2007)*, pages 50–75. ACM Digital Library, 2007.
- [CVA07] Sorana Cîmpan, Hervé Verjus, and Ilham Alloui. Dynamic architecture based evolution of enterprise information systems. In *International Conference on Enterprise Information Systems (ICEIS'07)*, pages 221–229, Madeira, Portugal, June 2007.
- [DG06] Stéphane Ducasse and Tudor Gîrba. Using Smalltalk as a reflective executable meta-language. In *International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006)*, volume 4199 of *LNCS*, pages 604–618, Berlin, Germany, 2006. Springer-Verlag.
- [DGK06] Stéphane Ducasse, Tudor Gîrba, and Adrian Kuhn. Distribution map. In *Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, pages 203–212, Los Alamitos CA, 2006. IEEE Computer Society.
- [DPS⁺07] Stéphane Ducasse, Damien Pollet, Mathieu Suen, Hani Abdeen, and Ilham Alloui. Package surface blueprints : Visually supporting the understanding of package relationships. In *ICSM '07 : Proceedings of the IEEE International Conference on Software Maintenance*, pages 94–103, 2007.
- [DWBN07] Stéphane Ducasse, Roel Wuyts, Alexandre Bergel, and Oscar Nierstrasz. User-changeable visibility : Resolving unanticipated name clashes in traits. In *Proceedings of 22nd International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'07)*, pages 171–190, New York, NY, USA, October 2007. ACM Press.
- [PDP⁺07] Damien Pollet, Stéphane Ducasse, Loïc Poyet, Ilham Alloui, Sorana Cîmpan, and Hervé Verjus. Towards a process-oriented software architecture reconstruction taxonomy. In

René Krikhaar, Chris Verhoef, and Giuseppe Di Lucca, editors, *Proceedings of 11th European Conference on Software Maintenance and Reengineering (CSMR'07)*. IEEE Computer Society, March 2007. Best Paper Award.

- [PV07] F. Pourraz and H. Verjus. Diapason : an engineering environment for designing, enacting and evolving service-oriented architectures. In *International Conference on Software Engineering Advances (ICSEA 2007)*, pages 23–30, France, August 2007. IEEE Computer Society.
- [PVO06] F. Pourraz, H. Verjus, and F. Oquendo. An architecture-centric approach for managing the evolution of eai services-oriented architecture. In *Eighth International Conference on Enterprise Information Systems (ICEIS 2006)*, pages 234–241, Paphos, Cyprus, May 2006.
- [VP07] H. Verjus and F. Pourraz. A formal framework for building, checking and evolving service oriented architectures. In *IEEE European Conference on Web Services (ECOWS 2007)*, pages 245–254, Halle, Germany, November 2007. IEEE Computer Society.
- [VP08] H. Verjus and F. Pourraz. Diapason : A formal approach for supporting agile and evolvable information system service-based architectures. In *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS'08)*, volume ISAS-2, pages 76–81, Barcelona, Spain, June 2008. INSTICC-WfMC-AAAI.

Articles de conférences francophones

- [PV06] F. Pourraz and H. Verjus. π -diapason : un langage pour la formalisation des architectures orientées services web. In *1ère Conférence francophone sur les Architectures Logicielles (CAL 2006)*, pages 119–127, Nantes, September 2006.
- [VCAO06] H. Verjus, S. Cimpan, I. Alloui, and F. Oquendo. Gestion des architectures évolutives dans archware. In *1ère Conférence francophone sur les Architectures Logicielles (CAL 2006)*, pages 41–57, Nantes, September 2006.

Autres

- [Cas07] Damien Cassou. Remodularisation à base de traits. Master's thesis, Université Bordeaux I, 2007.
- [PV07a] F. Pourraz and H. Verjus. Diapason : An engineering environment for designing, implementing and evolving service orchestrations. *ERCIM News Magazine - Special : Service-Oriented Computing*, 70 :41–43, July 2007.
- [PV07b] F. Pourraz and H. Verjus. π -diapason : A π -calculus based formal language for expressing evolvable web services orchestrations. Research Report LISTIC 07/06, University of Savoie - LISTIC, November 2007.
- [Sue07] Mathieu Suen. Package blueprints. Master's thesis, Université de Savoie, 2007.
- [Ver06a] H. Verjus. Nimrod : A software architecture engineering environment. technical Report LISTIC No 06/03, University of Savoie - LISTIC, December 2006.
- [Ver06b] Herve Verjus. Nimrod : A software architecture engineering environment. Technical report, Technical Report 06/03, LISTIC — Université de Savoie, 2006.
- [Ver07] H. Verjus. Nimrod : A software architecture-centric engineering environment - revision 2 - nimrod release 1.4.3. Technical Report LISTIC No 07/03, University of Savoie - LISTIC, Annecy, France, June 2007.
- [VP07] H. Verjus and F. Pourraz. Maintaining and evolving service oriented architectures using a π -calculus based approach. Research Report LISTIC No 07/04, University of Savoie - LISTIC, Annecy, France, June 2007.

Références

- [1] P. Grubb and A. A. Takang. *Software Maintenance Concepts and Practices*. World Scientific, second edition edition, 2005.
- [2] M. Lehman and L. Belady. *Program Evolution : Processes of Software Change*. London Academic Press, London, 1985.
- [3] B. Lientz and B. Swanson. *Software Maintenance Management*. Addison Wesley, Boston, MA, 1980.
- [4] J. R. McKee. Maintenance as a function of design. In *Proceedings of AFIPS National Computer Conference*, pages 187–193, 1984.
- [5] J. T. Nosek and P. Palvia. Software maintenance management : changes in the last decade. *Software Maintenance : Research and Practice*, 2(3) :157–174, 1990.
- [6] D. L. Parnas. Software aging. In *Proceedings 16th International Conference on Software Engineering (ICSE '94)*, pages 279–287, Los Alamitos CA, 1994. IEEE Computer Society.
- [7] R. S. Pressman. *Software Engineering : A Practitioner's Approach*. McGraw-Hill, 1994.
- [8] I. Sommerville. *Software Engineering*. Addison Wesley, fifth edition, 1996.