Résumé des activités de recherche

Philippe Lahire

Préambule

Ce résumé a pour objectif de relater les diverses propositions effectuées dans le cadre des thèmes de recherche abordés. Le fil conducteur de ces activités de recherche concerne le développement d'applications à objets. En particulier, je m'intéresse à l'étude des différentes techniques qui d'une part, pourraient améliorer la robustesse, la lisibilité, la documentation, les capacités d'adaptation et d'évolution des applications et d'autre part, réduiraient le temps nécessaire à leur mise en œuvre sur des plates-formes logicielles. Nous promouvons l'idée qu'il faut réduire le fossé entre les phases de conception et de programmation jusqu'à avoir un processus continu de développement, par exemple par transformation de modèles.

Notre contribution peut se découper en quatre phases qui se sont déroulées tantôt successivement, tantôt en parallèle au gré des réflexions et des opportunités de collaboration.

Il s'agit d'une part de propositions pour l'intégration de services dans les applications (voir section 1). La première proposition concerne l'ajout d'une gestion transparente des objets persistants pour les applications. La seconde est plus générale et a débuté il y a trois ans ; elle traite d'une approche générique pour l'intégration de fonctionnalités par séparation des préoccupations dans le but d'augmenter les capacités de réutilisation et d'adaptation des éléments qui constituent une application.

D'autres travaux (projet K2) se sont déroulés un certain moment en parallèle avec les travaux sur la persistance ; ils ont permis de faire un point sur les apports et les limitations des langages à objets par rapport aux critères de développement d'applications. Un certain nombre de conclusions, notamment concernant l'héritage, nous ont amenés à proposer une approche (voir section 2), pour mieux capturer la sémantique opérationnelle des classes et des relations entre classes. L'objectif de ce travail est en particulier d'utiliser le surplus d'information ainsi obtenu pendant la phase de conception afin d'automatiser le plus possible la phase de programmation et de faciliter les tâches de rétro-conception. À partir de cette étude plusieurs autres réflexions ont été menées ou sont en cours d'élaboration et concernent notamment l'ajout de l'héritage inverse, d'une relation de réutilisation de code ou l'annotation des relations et des classes.

Plus récemment, nous avons voulu élargir notre champ d'investigation. L'effort de modélisation, mentionné ci-dessus, pour capturer la sémantique des classifieurs et des relations entre classes correspond à une première tentative de modèle métier pour la description des langages à objets. Nous désirons l'étendre et proposer une approche générique pour la description de modèles métiers (voir section 3). En effet nous croyons que, de plus en plus, l'écriture d'applications se fera à travers la description d'un modèle (données et sémantique) qui sera instrumenté, au fur et à mesure des besoins, par les traitements à appliquer à ce modèle et qui constituent les applications. Cette approche qui s'intègre dans les travaux dédiés aux développements (d'application) guidés par le métier, utilisera naturellement les techniques de séparation des préoccupations sur lesquelles nous travaillons déjà depuis un certain temps.

1 Intégration de préoccupations dans les applications à objets

Le développement d'applications nécessite l'utilisation de plus en plus de services, indépendants de l'application elle-même. On peut citer par exemple la persistance, la mobilité ou la distribution des objets, l'intégration *a posteriori* de patrons de conception, de métriques, etc. Les techniques pour réaliser ces intégrations ont évolué au fil du temps pour prendre en compte le changement

d'échelle : d'un ou deux services dont il faut équiper une application, on se dirige vers un vaste ensemble de services. Nous proposons donc deux approches qui adressent respectivement le premier et le second cas.

1.1 Par un couplage fort et des bibliothèques

Ces travaux ont été effectués entre janvier 1990 et décembre 1995 ; ils ont débuté pendant ma thèse dont ils ont constitué une partie importante [36]. Ils se sont déroulés majoritairement dans le cadre d'un projet ESPRIT II (appelé *Business Class*) et ont donné lieu à un prototype de recherche nommé *FLOO* [1, 4].

L'objectif de ces travaux était de proposer une approche pour la gestion des objets persistants qui soit orthogonale (c'est-à-dire qui s'applique à tous les types manipulés par une application donnée), et transparente pour le programmeur (aucune modification du code source). C'est un service dont toute application à objets doit pouvoir disposer « gratuitement ». L'approche que nous proposons ici est basée sur un couplage fort et sur l'introduction d'une bibliothèque de services de persistance. Plus précisément cette approche repose sur :

- la modification de l'exécutif notamment pour la gestion transparente du va et vient et la mise à jour des objets entre les mémoires volatiles et persistantes [12] ;
- la possibilité de pouvoir considérer une méthode comme un objet de première classe [13] ;
- la gestion automatique de l'extension d'un type en tenant compte de la relation d'héritage [14] ;
- la propagation automatique et statique de la description du type d'une application (données et traitements) dans la mémoire persistante afin de bénéficier de la puissance du serveur d'objets pour réaliser des requêtes évoluées sur les objets [39] ;
- des classes permettant la gestion de transaction et la définition de requêtes dans un style conforme à la programmation à objets [13].

L'approche décrite ci-dessus a été validée par la réalisation d'un prototype (*FLOO*) qui étend le langage *Eiffel* pour la gestion des objets persistants et délègue la gestion de la mémoire persistante au SGBD *O2*. Ce prototype se compose d'une bibliothèque de classes, d'un support d'exécution *Eiffel* spécifique et d'un traducteur. Ce dernier propage (avec certaines optimisations), les classes *Eiffel* susceptibles d'adresser des objets persistants ou d'être la cible de requêtes, dans le Système de gestion de bases de données (SGBD) *O2*. Le système *O2* n'est utilisé que comme serveur d'objets persistants et dialogue avec le support d'exécution d'*Eiffel* par un protocole client/serveur. L'originalité de ce prototype est de permettre la manipulation d'instances volatiles et persistantes et d'optimiser les requêtes au serveur *O2*[1].

Ce prototype a été développé pour *Eiffel 2* et les premières versions du système *O2* ; il n'a malheureusement pas été distribué, ni par les sociétés distribuant *Eiffel*, ni par celles distribuant *O2*, car ces sociétés avaient d'autres priorités telles que le changement complet des environnements et du langage *Eiffel 3*, l'interfaçage graphique et la norme ODMG (*Object Data Management Group*) pour *O2*.

Le prototype *FLOO* a fait l'objet d'évaluations très positives dans le cadre du projet ESPRIT 5311 (*Business Class*) et de plusieurs démonstrations en 1994, dont une à la conférence ECOOP à Bologne [39]. Au cours de la période de référence, plusieurs autres articles ou rapports ont été publiés (voir ma bibliographie page 8).

1.2 Par une approche plus générique basée sur la séparation des préoccupations

Par rapport au service de persistance mentionné dans la section précédente, ce travail représente une généralisation de l'approche et permet de considérer d'autres services. Il a été entrepris bien plus tard, après que nous nous soyons intéressés à la modélisation par séparation des préoccupations (voir section 2.1).

[1] Certaines sont plus performantes que celles que produirait un utilisateur du système *O2* [1].

Plus précisément l'objectif est d'introduire des mécanismes relatifs à la séparation des préoccupations dans les langages à objets ou à composants, dans le but d'améliorer la réutilisation des classes ou des composants. La programmation par séparation des préoccupations repose sur le constat que la « classe » n'est pas toujours une unité de réutilisation adaptée :

- soit parce que sa granularité est trop faible : on réutilise plutôt des groupes de classes collaborantes,
- soit au contraire parce que des fonctionnalités de nature très différente sont réparties entre plusieurs classes et qu'une même classe peut regrouper des éléments participant à plusieurs fonctionnalités ; ceci induit un découpage transversal de l'ensemble des classes.

L'état de l'art dans ce domaine a permis de dégager de nombreux concepts (rôles, aspects, sujets, points de vue, etc.), qui ont à la fois proposé des schémas architecturaux de représentation originaux et développé des modèles et des outils associés. On peut citer par exemple, la métaprogrammation, la composition de méthodes, le « tissage » des aspects, ou encore la composition de sujets. Même si les techniques utilisées ou la motivation des concepts proposés sont différentes, les points communs sont absolument patents. En outre, les liens entre ces concepts et d'autres éléments classiques de la conception par objets sont loin d'être parfaitement maîtrisés.

Le modèle que nous proposons [77, 5] repose sur les concepts objets et sur deux approches pour la séparation des préoccupations : la programmation par aspects (AOP), la programmation par sujets (SOP) [55]. Ce modèle permet d'adapter le contenu des classes existantes de façon à pouvoir intégrer de nouvelles préoccupations, qu'elles soient fonctionnelles, non fonctionnelles ou hybrides [77]. Parmi les avancées proposées, on pourra retenir en particulier la définition d'une entité (appelée *adaptateur*) pour encapsuler une préoccupation. Elle permet de *i)* regrouper un ensemble de classes, *ii)* réaliser l'intégration *in-situ* ou *ex-situ* (à l'intérieur ou en dehors des classes existantes), *iii)* décrire un protocole de composition indépendant du contexte d'utilisation et par la même occasion de limiter les déclarations à décrire et guider le programmeur, lors de l'utilisation de préoccupations déjà définies.

2 Modélisation des concepts objets

Le point de départ de ce travail repose sur plusieurs constatations :

- L'intégration de la persistance dans les langages à objets [36] fait apparaître tout un ensemble de problématiques qui ne cessent de se diversifier (gestion de l'évolution, distribution sur le réseau, mobilité, interopérabilité, etc.). Elles sont de plus en plus nécessaires à l'écriture et à la maintenance d'applications dites modernes. Il semble donc intéressant d'étendre les langages de telle manière à pouvoir les inclure tout en ne polluant pas le code (métier) de l'application.
- La sémantique des concepts de classes (*classifieurs* en UML) et de relations (notamment la relation d'héritage) est souvent spécifique à un langage donné et de relativement bas niveau du point de vue conceptuel. De son côté, UML ne propose pas une sémantique claire mais offre des mécanismes d'extension pour spécifier des relations adaptées au contexte. Cependant, cette information conceptuelle est le plus souvent perdue lors de la projection vers les langages de programmation. Il semble donc intéressant d'une part, de réduire le fossé qui existe entre les phases de conception et de programmation et, d'autre part, de pouvoir comparer la sémantique associée aux différents concepts présents dans des langages à objets de l'état de l'art.
- Les mécanismes associés aux classes et aux relations dans les langages de programmation ont souvent une expressivité qui n'est pas adaptée aux besoins : elle peut être soit trop grande, soit au contraire trop faible. Dans les deux cas, cela nuit à la réutilisabilité. Pour dire cela, nous nous appuyons sur une expérience de développement important réalisé pour le compte de la *Food Agriculture Organization* (projet K2). Ce projet a permis à la fois de tester en vraie grandeur le bien-fondé des techniques à objets soutenues par Bertrand Meyer, dans un projet où les contraintes de coût, d'évolutivité et de portabilité étaient particulièrement

sévères[2], et de montrer certaines limites des concepts mis en œuvre. Quelques-unes de ces limitations sont décrites dans [11].
– Il peut être intéressant d'offrir un support pour expérimenter de nouvelles relations ou de nouvelles sortes de classe et pour adapter l'expressivité de ces concepts au contexte d'utilisation.

Pour répondre à ces constatations nous proposons un modèle paramétrable pour décrire la sémantique opérationnelle des concepts objets que l'on peut trouver à la fois dans les méthodes de conception et les langages de programmation. Dans un second temps, nous présentons un certain nombre de travaux qui dérivent des utilisations possibles de ce modèle ou des idées sous-jacentes.

2.1 Le modèle OFL

Dès que l'on évoque la volonté de modéliser des concepts, il est naturel de regarder vers les standards existants et particulièrement ceux de l'OMG (*Object Management Group*) ou du W3C (*World Wide Web Consortium*). Le modèle MOF (*Meta-Object Facility*) est spécialement dédié à la description de modèles. Il offre en effet la possibilité de réifier tout type d'entité et donc de ce fait, celle de modéliser, l'ensemble du modèle OFL[3].

Dans OFL (*Open Flexible Languages*), nous voulons décrire la structure des concepts mais surtout leur sémantique. En particulier, il nous a semblé important de pouvoir définir des concepts paramétrés afin de capturer les points communs (la sémantique commune) des entités prépondérantes (idée proche de la notion de ligne de produits). Le choix de paramétrer les concepts et de décrire leur sémantique par rapport à ces paramètres à un niveau méta nous a semblé une approche séduisante. Nous la préférons d'une part aux systèmes réflexifs existants qui rendent implicite un usage intensif de la métaprogrammation et, d'autre part, aux utilisations combinées de l'héritage et de variables et méthodes de classes[4].

Le modèle repose d'abord sur le concept de relation sémantique entre classifieurs, sur l'incidence de son paramétrage sur celui les classifieurs eux-mêmes et sur les implications qui en résultent au niveau d'un langage. Ce paramétrage structuré permet d'adresser certains problèmes liés à la persistance des objets [28, 24] et des composants, et plus généralement, la modélisation des langages. En particulier, son expressivité permet pour un langage donné (représenté dans OFL par un *concept-langage*), de définir une nouvelle relation sémantique (*concept-relation* dans OFL), ou une nouvelle notion de classifieur (*concept-description* dans OFL), particulièrement adaptée à l'écriture d'une catégorie d'applications [23, 26]. La capture de la sémantique opérationnelle d'un langage nécessite aussi la présence d'autres entités, qui réifient les différentes parties d'une application (attribut, méthode, paramètre de méthode, appel de routine, etc.).

Concrètement, il est fréquent d'utiliser le mécanisme d'héritage d'un langage à objets donné pour mettre en œuvre les différentes relations qui existent entre les classes d'une application. On n'hésite pas, par exemple, à utiliser l'héritage pour réaliser un sous-typage strict, une simple opération de réutilisation de code source ou même une généralisation [26, 27]. Cette diversité dans les usages de l'héritage démontre l'intérêt très général de ce mécanisme mais laisse aussi apparaître un défaut immédiat : il est très difficile pour un programmeur de spécifier quel usage il souhaite faire de ce mécanisme avec toutes les conséquences que cela peut avoir sur le contrôle, la lisibilité, la documentation, la maintenabilité et l'évolution des programmes.

Notre démarche, à travers la proposition du modèle OFL, adresse ce problème particulier et a pour objectif d'une part, d'améliorer l'expressivité de l'héritage en mettant à la disposition du métaprogrammeur les paramètres nécessaires à une meilleure modélisation de ses usages et d'autre part, de proposer des combinaisons de ces paramètres pour les cas les plus courants d'utilisation (relations de spécialisation, de généralisation, de réutilisation de code, de version, de vue, etc.) [27].

[2] Un haut niveau de lisibilité du code, de sa structure et de sa documentation interne étaient donc requis.
[3] Nous avons pu le faire avec le langage *Java*, même si l'absence de niveau méta chez ce dernier nous a obligé à faire quelques concessions en terme de modélisation.
[4] Leur complexité d'utilisation ou leur manque de flexibilité et de lisibilité sont à notre sens préjudiciables.

Par exemple, le langage Java est associé à un concept-langage qui contient entre autres les concepts-relations *« implements »* et *« extends »* et les concepts-descriptions, *« classe »*, *« interface »*, et *« tableaux »* [26]. Ces concepts s'imbriquent les uns dans les autres. Ainsi, le concept-langage fédère plusieurs concepts-descriptions et plusieurs concepts-relations, chaque concept-description coordonne lui-même les interactions de plusieurs concepts-relations vers le même ou d'autres concepts-descriptions, tandis que chaque concept-relation décrit la sémantique d'une relation entre une description « source » et n descriptions « cible ». L'étude des langages à objets existants (*C++*, *Java*, *Eiffel*, *Smalltalk*, etc.) et des techniques de compilation nous a permis d'abord, de dégager les paramètres qui différencient deux concepts-relations, deux concepts-descriptions ou deux concepts-langages, et ensuite, de mettre en évidence un ensemble d'opérateurs qui sont influencés par les valeurs de ces paramètres [49].

Le besoin de structuration du niveau méta (modélisation de la sémantique opérationnelle) est rendu nécessaire par les contraintes associées à notre approche. D'une part, le mécanisme de paramétrage doit être adapté à la définition de bibliothèques de concepts-relations et de concepts-descriptions réutilisables aussi bien qu'à la définition de ces concepts pour des langages dont la définition est fixée au départ. D'autre part, il est fondamental que la tâche du métaprogrammeur soit proportionnelle aux modifications de sémantique qui sont à réaliser. En particulier, à moins qu'il ne veuille modifier profondément la sémantique d'un ou plusieurs concepts, il ne devrait avoir pratiquement à modifier que la valeur des paramètres [19, 20, 23].

Une constante de notre démarche est de proposer des approches toujours plus simples pour modifier la sémantique associée à un classifieur ou à une relation. Nous avons souhaité appliquer ce principe à l'ajout de préoccupations orthogonales à la sémantique de base (on rejoint ici les motivations décrites à la page 1). Ainsi, nous nous sommes intéressés à la définition d'une extension du modèle OFL afin qu'il soit possible de spécifier un protocole pour l'intégration de paramètres et d'actions, spécifiques à un domaine d'application (exemple : gestion des aspects multimédia, gestion de l'évolution, etc.) [21]. Dans un premier temps, nous nous sommes intéressés plus particulièrement aux services relatifs à la persistance des objets [10, 24].

Par ailleurs, nous nous sommes aperçus qu'une partie de la sémantique associée aux classifieurs et aux relations entre classifieurs était implicitement mémorisée dans les qualifieurs (par exemple : *private*, *public*, *abstract*, etc.) et qu'en général ces derniers étaient très spécifiques à un langage donné et donc peu modélisables par des paramètres généraux. Nous nous sommes donc intéressés à cet aspect en introduisant le concept de *modifier*, qui s'appuie en particulier sur des contraintes OCL [18, 45].

La description d'une plate-forme objet en Java (*OFL/J*) intégrant un MOP[5] fournit une première implantation du modèle. L'idée sous-jacente est généralisée par l'approche et le prototype proposé dans la section 3 qui permet de générer la majeure partie des classes d'*OFL/J*.

2.2 Applications du modèle

Dans le prolongement des travaux de modélisation menés dans le cadre des thèses de Pierre Crescenzo pour ce qui concerne le cœur du modèle [80], d'Adeline Capouillez pour la mise en œuvre d'un protocole d'extension basé sur la séparation des préoccupations [21] et de Dan Pescaru pour l'introduction de nouvelles capacités de description [45, 78], nous avons initié plusieurs travaux qui ont déjà produit des résultats. En voici un rapide descriptif :

– La définition d'une approche pour annoter l'héritage [17, 79]. Cette approche propose d'associer des informations aux relations d'héritage qui existent entre des classes en vue de réaliser des contrôles, d'assister l'édition des classes, de documenter le code, ou plus généralement d'enrichir les environnements de programmation pour faciliter, par exemple, la maintenance des applications. Les informations recensées reposent sur une classification adaptée aux contextes d'utilisation et une classification plus technique qui décrit les adaptations qu'il est possible de réaliser dans une classe déclarant une ou plusieurs relations d'héritage.

[5] Protocole Méta-Objet.

– La définition d'une relation d'héritage inverse dans le but de fournir un support original pour la description des adaptations que l'on désire réaliser sur une hiérarchie de classes. Ces adaptations ont pour objectif de faciliter l'adaptation des hiérarchies de classes en fonction du contexte d'utilisation et dans une certaine mesure de simplifier leur évolution. Une application au langage Java est en cours de définition [43, 42, 7, 15].
– Dans le but de réduire le fossé entre les concepts définis dans les langages de programmation et des méthodes de conception comme UML, nous proposons un mécanisme de génération de *profils UML* à partir des informations modélisées à l'aide du modèle OFL [44, 6]. L'intérêt est double, d'une part, on veut pouvoir décrire précisément la sémantique opérationnelle des différents types de classifieurs ou de relations associés à un langage à objets pour ensuite les utiliser pendant la phase de conception, d'autre part, il s'agit de définir de manière précise, dans UML, des concepts de classes ou de relations issus des langages de programmation existants et par ce biais, de faciliter la projection de schémas de conception vers les langages de programmation grâce aux informations mémorisées pour chacun des concepts.

3 Une approche générique pour la description de modèles métiers exécutables

Ce travail est une prolongation de notre activité de recherche autour du modèle OFL décrit dans la section précédente. Il correspond à une volonté de proposer avec Didier Parigot (Chargé de Recherche à l'INRIA) une démarche commune qui s'inspire de ses travaux autour de SmartTools et des nôtres autour d'OFL [40, 41, 16, 8]. C'est une des directions de recherche pour les prochaines années [40].

Plus précisément, nous constatons que les approches du développement d'application sont en train de fortement évoluer autour de trois idées clés :

– La séparation des préoccupations ; cet aspect a été évoqué assez longuement dans la section 1 ;
– La programmation par composants. Une application est réalisée à partir de composants logiciels indépendants et exécutables qui sont vus les uns par les autres comme des boîtes noires offrant des points d'entrées pour se composer les uns aux autres, chaque composant pouvant être développé avec le paradigme objet ou avec tout autre paradigme.
– L'approche dirigée par les modèles (MDA). Le savoir-faire métier est décrit par des modèles indépendants du contexte d'utilisation (*Platform Independent Model* - PIM) c'est-à-dire indépendants de la plate-forme logicielle ou des traitements à réaliser. Les applications sont ensuite réalisées par transformation du modèle métier vers des plates-formes logicielles spécifiques (*Platform Specific Model* - PSM). Les deux principales idées sous-jacentes sont la capitalisation du savoir-faire et la prise en compte rapide de l'évolution de la technologie.

Par rapport aux approches mentionnées ci-dessus, les travaux autour des langages à objets et plus généralement l'approche classique de développement se situent clairement dans la programmation du ou des composants. Cet aspect de la programmation nous intéresse bien sûr toujours (et nous comptons continuer à la développer), mais nous désirons étendre notre spectre de recherche et appliquer le savoir-faire que nous avons acquis pour décrire la sémantique opérationnelle des concepts objets à la description de modèles métiers exécutables. Désormais le modèle OFL (modélisation des concepts objets) est un modèle métier particulier (parmi d'autres) qui permet de valider notre approche et d'offrir des exemples d'utilisation.

Il est maintenant important de poser les fondements de notre approche qui s'intègre parfaitement dans l'axe de recherche *Domain Driven Development* (DDD). Celui-ci s'appuie en particulier sur *i)* les langages à objets et à composants, *ii)* l'approche MDA, *iii)* la séparation des préoccupations, et *iv)* la programmation générative. Notre objectif est de promouvoir une nouvelle approche de la programmation que nous appelons la *Programmation Orientée-Modèles*[6].

Cette approche s'appuie d'une part, sur un méta niveau, pour identifier clairement la sémantique des concepts utilisés dans la modélisation d'un métier particulier, et d'autre part, sur les

[6] En anglais *Model-Oriented Programming.*

approches par séparation de préoccupations et sur la programmation générative pour permettre une instrumentation modulaire des applications associées au métier (ou domaine) considéré. Cette approche est originale et se différencie des autres approches sur plusieurs points : *i)* elle associe aux entités qui structurent un modèle (entités de réification) un niveau sémantique qui permet de préciser et de factoriser les fonctionnalités de base du domaine, *ii)* elle permet une instrumentation (réalisation rapide des applications associées au modèle) qui s'appuie fortement sur les deux niveaux du modèle (niveau structurant et sémantique), *iii)* elle assure une séparation nette entre la modélisation (du métier) et les technologies utilisées pour rendre le modèle exécutable (c'est-à-dire mettre en œuvre des applications vers ce modèle). Dans [40], nous avons proposé plusieurs règles (elles sont actuellement au nombre de neuf) qu'il nous semble essentiel de respecter ; certaines sont conceptuelles et d'autres plus orientées vers l'implémentation.

L'intérêt de notre approche (appelée SmartModels) réside dans le fait qu'elle propose un moyen de décrire la sémantique des entités utilisées dans un modèle qui est indépendant d'une application donnée. En général, la sémantique d'un modèle est dispersée dans les diverses applications qui le manipulent. Notre approche ne fait pas de différence entre la modélisation du métier et son instrumentation[7]. En effet, puisque la sémantique du modèle est associée aux entités elles-mêmes, les diverses applications peuvent utiliser directement cette connaissance sans passer par une phase d'implémentation[8]. Les apports de la programmation générative et par séparation des préoccupations sont utilisés pour disposer dans la partie « instrumentation du modèle », d'une expressivité, d'une capacité d'évolution et d'une modularité accrues.

Parmi les aspects importants de SmartModels issus de nos travaux sur OFL, on trouve en particulier :

- la capacité de décrire n'importe quel élément ou entité d'un modèle de données[9],
- la possibilité de décrire des paramètres métiers qui permettent de capturer une partie de la sémantique du modèle,
- la modélisation de concepts génériques (la généricité est décrite par les paramètres métiers), permettant par exemple de représenter des « lignes de produits »,
- la spécification d'actions sémantiques associées aux concepts et aux entités du modèle, dont l'exécution est dirigée par des paramètres métiers,
- un support pour la définition de méta-assertions,
- un protocole méta objets permettant de mettre en œuvre tous les aspects importants du modèle (actions, paramètres, préoccupations, assertions, etc.), comme des entités de première classe du modèle et d'offrir automatiquement à chaque entité un ensemble de services comme par exemple la gestion des instances.

La mise en œuvre de SmartModels dans un premier prototype de « fabrique logicielle » que nous avons baptisé SmartFactory[10], s'inspire naturellement des idées mises en œuvre dans SmartTools[11] comme par exemple *i)* les techniques de génération de code permettant d'implanter les modèles décrits avec SmartModels sur des plates-formes logicielles, *ii)* la description de traitements par visiteurs et par aspects, et *iii)* les techniques de transformations de modèles, notamment vers les technologies issues du *W3C*.

Pour conclure, il est important de citer les principaux travaux que nous désirons mener à court terme pour enrichir à la fois SmartModels et sa mise en œuvre SmartFactory.

Il s'agit en particulier d'approfondir certains aspects et de les modéliser eux-mêmes comme des modèles métiers (des instances de SmartModels)[12]. Cela concerne par exemple la séparation des préoccupations, la description des plates-formes logicielles (*Platform Dependent Model* - PDM) et la production de composants à partir de la spécification des applications. Il s'agit aussi d'améliorer les techniques proposées pour décrire la sémantique de manière à pouvoir la faire évoluer en même

[7] Les fonctionnalités ou la glue qui rendent le modèle exécutable.

[8] Ce sont les générateurs qui, en proposant une implémentation générique des concepts du modèle, prendront en compte cette phase d'implémentation.

[9] La description de ces entités s'inspire (et même peut se décrire) en MOF.

[10] Un premier prototype partiel est déjà opérationnel. Il permet, entre autres, de générer la plupart des classes de la plate-forme *OFL/J* qui était au préalable écrites à la main.

[11] Il a été conçu et réalisé par Didier Parigot et son équipe.

[12] C'est une manière élégante d'enrichir le modèle avec lui-même.

temps que le modèle de données du modèle métier lorsque celui-ci subit des transformations. Plus généralement, nous désirons améliorer la qualité et le volume de code généré automatiquement et surtout proposer de nouvelles techniques pour aider le développeur à obtenir plus rapidement des applications robustes et évolutives.

4 Liste des publications classées par catégorie

Préambule

Les publications sont classées par projet puis par catégorie. La plupart des publications citées ci-après sont disponibles sur mon site Internet à l'adresse suivante : ***http ://www.i3s.unice.fr/~lahire/.*** Elles sont aussi pour la plupart, disponibles comme rapports de recherche du Laboratoire I3S. Il est à noter que dans notre équipe, nous avons choisi que les auteurs soient classés par ordre alphabétique à de très rares exceptions près.

Ouvrages ou contributions à un ouvrage

[1] Robert Chignoli, Jacques Farré, Philippe Lahire & Roger Rousseau. FLOO : a strong coupling between Eiffel language and O2 DBMS. Object-Oriented Technology for Database and Software Systems. World Scientific Publishing Co. , ALAGAR V. S., MISSAOUI R. (eds), 1995, pages 206-223.

[2] Ph. Lahire, G. Arévalo, H. Astudillo, A.P. Black, E. Ernst, M. Huchard, M. Sakkinen, P. Valtchev (eds.) Proceedings of The 3rd International Workshop on MechAnisms for SPEcialization, Generalization and inHerItance - MASPEGHI'04. Oslo, Norway, 15 June 2004 at ECOOP 2004, 14-18 June 2004. Publié comme rapport de recherche n° I3S/RR-2004-15-FR du laboratoire I3S, Université de Nice-Sophia Antipolis, juin 2004, 88 pages.

[3] Ph. Lahire, G. Arévalo, H. Astudillo, A.P. Black, E. Ernst, M. Huchard, T. Oplustil, M. Sakkinen, P. Valtchev, Report from the ECOOP 2004 Workshop on Mechanisms for Specialization, Generalization and Inheritance (MASPEGHI 2004), à paraître dans "Object-Oriented Technology : ECOOP 2004 Workshop Reader", J. Malenfant and Bjarte M. Østvold, Springer Verlag, LNCS series N°3344, pages 101-117.

Revues et journaux

[4] Robert Chignoli, Jacques Farré, Philippe Lahire & Roger Rousseau. FLOO : un environnement pour la programmation persistante en Eiffel. Technique et science informatiques, Editions Hermès, Paris, Numéro spécial Systèmes à objets : tendances actuelles et évolution, A. Napoli & J.-F. Perrot (eds), volume 15, n°6 (Juin 1996), pages 735-763. Editions Hermès. juin 1996, Sophia-Antipolis, France.

[5] Philippe Lahire & Laurent Quintian. New Perspective To Improve Reusability in Object-Oriented Languages, 20 pages, accepté sous réserve de légères modifications *à Journal of Object Technology*, à paraitre fin 2004 - début 2005.

Conférences internationales avec sélection

[6] Ciprian-Bogdan Chirila, Philippe Lahire, Dan Pescaru, & Emanuel Tundrea. A Better Representation for Class Relationships in UML using OFL meta-information. AQTR 2004, conférence internationale sur "Automation, Quality & Testing, Robotics", Cluj-Napoca, Roumanie, 13-15 mai 2004. 6 pages.

[7] Ciprian-Bogdan Chirila, Pierre Crescenzo, Philippe Lahire, Dan Pescaru & Emanuel Tundrea. Factoring Mechanism of Reverse Inheritance. CONTI'2004, 6ème conférence internationale sur "Technical Informatics", Timisoara Roumanie, 27-28 mai 2004, pages 131-136.

[8] Philippe Lahire, Didier Parigot, Carine Courbis, Pierre Crescenzo & Emanuel Tundrea. An Attempt to set the framework of Model-Oriented Programming. CONTI'2004, 6ème conférence internationale sur , Timisoara Roumanie, 27-28 mai 2004, pages 71-76.

[9] Philippe Lahire & Dan Pescaru. OpenIDL : an open modelling language based on IDL and OFL. CONTI'2000, 4ème conférence internationale sur "Technical Informatics", Timisoara Roumanie, 12-13 octobre 2000, pages 145-152.

[10] Adeline Capouillez, Robert Chignoli, Pierre Crescenzo & Philippe Lahire. How to Improve Persistent-Object Management using Relationship Information ? WOON'2000, 4ème conférence internationale "The White Object Oriented Nights". editeurs : Martinus Nijhoff Eastern Europe ; Alexander V. Smolyaninov et Alexei S. Shestialtynov. juin 2000, 14 pages, Saint-Pétersbourg, Russie.

[11] Jean-Marc Jugant & Philippe Lahire. Lessons Learned with Eiffel 3 : the K2 Project dans TOOLS 17, USA'95, 17 ème conférence internationale sur "Technology of Object-Oriented Languages and Systems". Ege R., Singh M., Meyer B. (eds),31 juillet - 4 août 1995. Prentice Hall Inc. (Englewood Cliffs, NJ), 1995, pages 207-215.

[12] Robert Chignoli, Jacques Farré, Philippe Lahire & Roger Rousseau. FLOO : basic principles and illustration of automatic persistency mechanism for Eiffel. 6ème conférence internationale "Software engineering & its applications", Paris - La Défense, 15-19 novembre. 1993. EC2 (Paris), pages 181-190.

[13] Philippe Brissi, Philippe Lahire & Roger Rousseau. An integrated query language for handling persistent objects in Eiffel. 4ème Conférence Internationale TOOLS "Technology of Object-Oriented Languages and Systems", Paris - La défense, 4-8 mars 1991. Prentice Hall, J. Bezivin, B. Meyer, J-M. Nerson (eds), pages 101-114.

[14] Philippe Lahire, Nhan Le Thanh & Serge Miranda. Persistency for the Eiffel language. 2ème Conférence Internationale TOOLS "Technology of Object-Oriented Languages and Systems", Paris - La défense, 25-29 juin 1990. Prentice Hall, J. Bezivin, B. Meyer, J-M. Nerson (eds), pages 299-314.

Workshops internationaux et Posters avec sélection

[15] Ciprian-Bogdan Chirila, Pierre Crescenzo and Philippe Lahire. A reverse inheritance relationship for improving reusability and evolution : the point of view of feature factorization (position paper). Actes du 3ème workshop MASPEGHI de la conférence ECOOP 2004, juin 2004, Oslo, Norvège, 6 pages. Ph. Lahire, G. Arévalo, H. Astudillo, A.P. Black, E. Ernst, M. Huchard, M. Sakkinen, P. Valtchev (eds), Laboratoire I3S, Université de Nice-Sophia Antipolis.

[16] Philippe Lahire, Didier Parigot & Emanuel Tundrea. SmartFactory - an Implementation of the Domain Driven Development Approach. SACI'2004, 1er symposium roumain-hongrois sur "Applied Computational Intelligence", 25-26 mai 2004, 6 pages.

[17] Pierre Crescenzo, Christophe Jalady & Philippe Lahire. Annotations of classes and inheritance relationships : an unified mechanism in order to improve skills of library of classes. Actes du 2ème workshop sur "Managing Specialization/Generalization Hierarchies" de la conférence ASE 2003 (conférence internationale sur "Automated Software Engineering"), Octobre 2003, 10 pages, Montréal, Canada.

[18] Philippe Lahire & Dan Pescaru. Modifiers in OFL - An approach for access control customization. Actes du workshop sur "Encapsulation and Access Rights" (WEAR'2003) de la conférence OOIS 2003 (9ème conférence internationale sur "Object-Oriented Information Systems"), septembre 2003, 10 pages, Genève, Suisse.

[19] Pierre Crescenzo & Philippe Lahire. Using both Specialisation and Generalisation in a Programming Language : Why and How ? 1er workshop sur "Managing Specialization/Generalization Hierarchies" de la conférence OOIS 2002 (8ème conférence internationale sur "Object-Oriented Information Systems", Springer-Verlag, LNCS series, N°2426, septembre 2002, pages 64-73, Montpellier, France. Le rapport de recherche I3S/RR–2002-19–FR est une version longue de cet article.

[20] Pierre Crescenzo & Philippe Lahire. Customisation of Inheritance. Actes du workshop "The Inheritance" de la conférence ECOOP 2002 (16ème "European Conference on Object-Oriented Programming"), juin 2002, Malaga, Espagne. Andrew P. Black, Erick Ernst, Peter Grogono et Markku Sakkinen (eds) ; ISSN : 1236-1615 ; ISBN : 951-39-1252-3 ; University of Jyväskylä, Finlande, 7 pages. Synthèse dans « Object-Oriented Technology : ECOOP 2002 Workshop Reader », Springer Verlag, LNCS series N°2548.

[21] Adeline Capouillez, Pierre Crescenzo & Philippe Lahire. Separation of Concerns in OFL. Dans workshop "Advanced Separation of Concerns" de la conférence ECOOP 2001 (15ème "European Conference on Object-Oriented Programming"). Synthèse dans "Object-Oriented Technology : ECOOP 2001 Workshop Reader" ; Ákos Frohner ; ISSN : 0302-9743 ; ISBN : 3-540-43675-8 ; Springer Verlag, LNCS series 2323, juin 2001, 6 pages, Budapest, Hongrie.

[22] Adeline Capouillez, Robert Chignoli, Pierre Crescenzo & Philippe Lahire. Modeling Hypergeneric Relationships between Types in XML. Invité pour le symposium ETc 2000 (4th Edition of "Symposium of Electronics and Telecommunications") . Scientific Bulletin of the "Politehnica" University of

Timisoara, tome 45(59), fasc. 1, ISSN 1224-6034 ; éditeurs : "Politehnica" University of Timisoara, Faculty of Electronics and Telecommunications, Institute of Electrical and Electronics Engineers [IEEE], and Association of Electronic Engineers, novembre 2000, 6 pages, Timisoara, Roumanie. Egalement dans workshop “Reflection and Metalevel Architectures” de la conférence ECOOP 2000 (14ème “European Conference on Object-Oriented Programming”). Synthèse dans "Object-Oriented Technology : ECOOP 2000 Workshop Reader" ; Jacques Malenfant, Sabine Moisan et Ana Moreira ; ISSN : 0302-9743 ; ISBN : 3-540-41513-0 ; Springer Verlag, LNCS series 1964, juin 2000, Sophia Antipolis et Cannes, France.

[23] Adeline Capouillez, Robert Chignoli, Pierre Crescenzo & Philippe Lahire. Customization of Relationships between Types. Posters de la conférence ECOOP 2000 (14ème European Conference on Object-Oriented Programming). Synthèse dans "Object-Oriented Technology : ECOOP 2000 Workshop Reader" ; Jacques Malenfant, Sabine Moisan et Ana Moreira ; ISSN : 0302-9743 ; ISBN : 3-540-41513-0 ; Springer Verlag, LNCS series 1964, juin 2000, 19 pages Sophia Antipolis et Cannes, France.

[24] Adeline Capouillez, Robert Chignoli, Pierre Crescenzo & Philippe Lahire. Towards a More Suitable Class Hierarchy for Persistent Object Management. Dans workshop “Objects and Classification : a Natural Convergence” de la conférence ECOOP 2000 (14ème “European Conference on Object-Oriented Programming”). Synthèse dans "Object-Oriented Technology : ECOOP 2000 Workshop Reader" ; Jacques Malenfant, Sabine Moisan et Ana Moreira ; ISSN : 0302-9743 ; ISBN : 3-540-41513-0 ; Springer Verlag, LNCS series 1964, juin 2000, 6 pages, Sophia Antipolis et Cannes, France.

Conférences francophones avec sélection

[25] Gilles Ardourel, Pierre Crescenzo & Philippe Lahire. LAMP : vers un langage de définition de mécanismes de protection pour les langages de programmation à objets . LMO’2003, conférence nationale sur les Langages et Modèles à Objets. Publié dans la revue L’objet : logiciels, bases de données, réseaux", volume X, numéro 1-2/2003 ; Jean-Pierre Briot et Jacques Malenfant. Editions Hermès Science Publications, janvier 2003, 13 pages, Vannes, France.

[26] Adeline Capouillez, Pierre Crescenzo & Philippe Lahire. Le modèle OFL au service du métaprogrammeur - Application à Java. LMO’2002, conférence nationale sur les Langages et Modèles à Objets. Publié dans la revue L’objet : logiciels, bases de données, réseaux", volume 8, numéro 1-2/2002 ; Michel Dao et Marianne Huchard ; ISSN : 1262-1137 ; ISBN : 2-7462-0403-7. Editions Hermès Science Publications, janvier 2002, 14 pages, Montpellier, France.

[27] Adeline Capouillez, Robert Chignoli, Pierre Crescenzo & Philippe Lahire. Hyper-généricité pour les langages à objets : le modèle OFL. LMO 2001, conférence nationale sur les Langages et Modèles à Objets. Publié dans la revue L’objet : logiciels, bases de données, réseaux", volume 7, numéro 1-2/2001 ; Robert Godin et Isabelle Borne ; ISSN : 1262-1137 ; ISBN : 2-7462-0211-5 ; éditions Hermès Science Publications, janvier 2001, 13 pages, Le Croisic, France.

[28] Adeline Capouillez, Robert Chignoli, Pierre Crescenzo & Philippe Lahire. Gestion des objets persistants grâce aux liens entre classes. OCM 2000, conférence nationale sur les Objets, Composants, Modèles « Passé, Présent, Futur ». Editeurs : Université de Nantes, Soft-Maint Groupe Sodifrance et École des Mines de Nantes, mai 2000, 10 pages, Nantes.

[29] Robert Chignoli, Jacques Farré, Philippe Lahire & Roger Rousseau. FLOO : un couplage fort entre le langage Eiffel et le SGBD O2. Dans le 62ème Congrés de l’Association Canadienne Française pour l’Avancement des Sciences - ACFAS, Montréal. ACFAS, 16-17 Mai 1994, pages 107-123. Version anglaise corrigée publiée dans Object-Oriented Technology for Database and Software Systems. World Scientific Publishing Co. , ALAGAR V. S., MISSAOUI R. (eds), 1995, pages 206-223.

[30] Robert Chignoli, Jacques Farré, Philippe Lahire & Roger Rousseau. Une implémentation de la persistance pour le langage Eiffel. Dans RPO’93 - Représentation par Objets , La Grande Motte, 22-23 Juin 1993, pages 131-142.

Séminaires et autres manifestations

[31] Philippe Lahire. Extensibility and Reusability : AOP vs OOP - Case studies in Eiffel and AspectJ. Présentation à la demande de Bertrand Meyer titulaire de la chaire de Génie Logiciel à l’ETH, Zurich, 16 Avril 2003.

[32] Laurent Quintian & Philippe Lahire. Vers une meilleure intégration des composants sur l'étagère. Dans la journée Objects Composants et Modèles (OCM) du GDR ALP, Vannes, 5 pages, février 2003.

[33] Philippe Lahire. OFL : How to capture and customise the operational semantics of object languages based on classes. Séminaires Microsoft, Cambridge research center, Février 2002.

[34] Philippe Lahire. Un modèle de persistance pour l'environnement Eiffel. Dans un séminaire de génie informatique, Université de Nantes. Décembre 1991.

[35] Philippe Lahire. Une approche pour introduire la persistance en Eiffel. Dans un séminaire pour I3S CNRS/UNSA et un autre pour le groupe PRC-BD3 "dynamique", Nice et Paris. Mars 1990.

Mémoires

[36] Philipe Lahire. Conception et réalisation d'un modèle de persistance pour le langage Eiffel. Thèse de Doctorat d'Informatique - Université de Nice-Sophia Antipolis, soutenue le 11 Mai 1992.

[37] Philippe Lahire. De l'extension des langages à objets à la réalisation de modèles métiers : une évolution du développement logiciel. Habilitation à diriger des recherches en sciences - Université de Nice-Sophia Antipolis, soutenue le 10 décembre 2004.

Logiciels

[38] Philippe Lahire & Jean-Marc Jugant K2 system : Modules Demand and SUA. Décembre 1995. Produit livré à la FAO et comprenant plus de 100 000 lignes de code, plus de 800 classes et 2 500 pages de documentation interne.

[39] Robert Chignoli, Jacques Farré, Philippe Lahire & Roger Rousseau. FLOO : a strong coupling between Eiffel Language and O2 DBMS. Dans ECOOP'94, "European Conference on Object-Oriented Programming". 4-8 Juillet 1994, Bologna. Démonstration sélectionnée.

Rapports de recherche

[40] Philippe Lahire, Didier Parigot, Carine Courbis, Pierre Crescenzo, emanuel Tundrea. Toward a New Approach for the Development of Software : the Model-Oriented Programming. Rapport de recherche I3S/RR–2003-33–FR (25 pages), laboratoire Informatique, Signaux et Systèmes de Sophia Antipolis (UNSA / CNRS UMR 6070) décembre 2003, Sophia-Antipolis, France.

[41] Didier Parigot, Carine Courbis & Philippe Lahire. Towards Domain-Driven Development : Approach and Implementation. Rapport de recherche I3S/RR–2003-14–FR (20 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070) Avril 2004, Sophia Antipolis, France.

[42] Ciprian-Bogdan Chirila, Pierre Crescenzo & Philippe Lahire. Reverse Inheritance : an Approach for Modeling Adaptation et Evolution of applications. Rapport de recherche I3S/RR–2003-XX–FR (13 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070) octobre 2003, Sophia Antipolis, France.

[43] Ciprian-Bogdan Chirila, Pierre Crescenzo & Philippe Lahire. Towards Reingineering : an Approach Based on Reverse Inheritance - Application to Java. Rapport de recherche I3S/RR–2003-XX–FR (70 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070) novembre 2003, Sophia Antipolis, France.

[44] Pierre Crescenzo, Philippe Lahire & Dan Pescaru. Automatic Profile Generation for OFL-Languages. Rapport de recherche I3S/RR–2003-32–FR (67 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070) Novembre 2003, Sophia Antipolis, France.

[45] Pierre Crescenzo, Philippe Lahire & Dan Pescaru. An Extension of OFL Model through Modifiers. Rapport de recherche I3S/RR–2003-31–FR (36 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070) Novembre 2003, Sophia Antipolis, France.

[46] Pierre Crescenzo & Philippe Lahire. Using both Specialisation and Generalisation in a Programming Language : Why and How ? Rapport de Recherche I3S/RR–2002-19–FR (14 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070) mai 2002, Sophia Antipolis, France. Ce rapport de recherche est une version longue de l'article publié dans le workshop Managing Specialization/Generalization Hierarchies en septembre 2002.

[47] Adeline Capouillez, Pierre Crescenzo & Philippe Lahire OFL : Hyper-Genericity for Meta-Programming - An Application to Java. Rapport de Recherche I3S/RR–2002-16–FR (11 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070), avril 2002, Sophia Antipolis, France.

[48] Robert Chignoli, Pierre Crescenzo & Philippe Lahire. Customization of Links between Classes. Rapport de Recherche 99-18 (16 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070), novembre 1999, Sophia-Antipolis, France

[49] Robert Chignoli, Pierre Crescenzo & Philippe Lahire. OFL : An Open Object Model Based on Class and Link Semantics Customization. Rapport de Recherche 99-08 (20 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070) mars 1999, Sophia Antipolis, France

[50] Robert Chignoli, Pierre Crescenzo & Philippe Lahire An Extensible Environment for Views in Eiffel Rapport de Recherche 96-53 (16 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070), novembre 1996, Sophia Antipolis, France.

[51] Robert Chignoli, Pierre Crescenzo & Philippe Lahire. Un modèle de paramétrage des liens entre classes Rapport de Recherche 99-13 (13 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070) août 1999, Sophia Antipolis, France.

[52] Robert Chignoli, Pierre Crescenzo & Philippe Lahire. OFL : une machine virtuelle objet flexible pour la gestion d'objets persistants et mobiles. Rapport de Recherche 99-09 (71 pages), laboratoire Informatique, Signaux et Systèmes de Sophia Antipolis (UNSA / CNRS UMR 6070), mars 1999, Sophia-Antipolis, France

[53] Robert Chignoli, Pierre Crescenzo & Philippe Lahire. Liens entre classes dans les langages à objets. Rapport de Recherche 97-22 (28 pages), laboratoire Informatique, Signaux et Systèmes de Sophia-Antipolis (UNSA / CNRS UMR 6070), juillet 1997, Sophia Antipolis, France.

[54] Philippe Lahire & Jean-Claude Boussard Etude et conception du logiciel de CAO économétrique "K2" Rapport RR-1993-16, I3S - URA CNRS/UNSA, Février 1993, 28 pages.

[55] Laurent Quintian & Philippe Lahire. Vers une meilleure réutilisation des patrons de conception. Rapport de Recherche I3S/RR–2001-04–FR, Laboratoire Informatique, Signaux et Systèmes de Sophia Antipolis (UNSA / CNRS UMR 6070), 13 pages.

[56] Philippe Lahire & Philippe Brissi & Jacques Farré. How to optimize queries on Eiffel persistent objects when object manager does not support routines. Rapport RR-1992-68, I3S CNRS/UNSA, décembre 1992, 15 pages.

[57] Philippe Lahire & Roger Rousseau. Integration of selective queries in the Eiffel language. Rapport RR-1992-67, I3S CNRS/UNSA, décembre 1992, 23 pages.

[58] Philippe Lahire & Philippe Brissi. Optimisation de l'accès aux objets persistants en Eiffel. Rapport RR-1991-37, I3S CNRS/UNSA, décembre 1990, 17 pages.

Rapports de contrat

[59] Philippe Lahire & Jean-Marc Jugant K2 Internal Documentation : listing of source code. Dans le projet FAO/ONU - Food Agriculture Organization of the United Nations, rapport, release 1, Décembre 1995, 1203 pages. Nettoyage et édition des codes sources par Roger Rousseau

[60] Lahire & Jean-Marc Jugant K2 developper's Manual - Global Architecture and Design. Dans le projet FAO/ONU - Food Agriculture Organization of the United Nations), rapport, release 1, Décembre 1995, 1045 pages. édition par Philippe Collet

[61] Philippe Lahire & Philippe Collet Demand Module developper's Manual. Dans le projet FAO/ONU - Food Agriculture Organization of the United Nations, rapport, release 1, Décembre 1995, 238 pages.

[62] Philippe Lahire & Roger Rousseau & Robert Chignoli & Jacques Farré. FLOO : A Prototype for an Eiffel Object Repository with O2 DBMS - User's Guide. Dans le projet Esprit II N° 5311 "Business class", rapport BC.R.LIS.W4.1.3, Release 2.1, 1994, 130 pages.

[63] Philippe Lahire & Roger Rousseau & Robert Chignoli & Jacques Farré. Technical Review of Basic Object-Oriented TOOLS and Recommendations for Further Work. Dans le projet Esprit II N° 5311 "Business class" , rapport BC.R.SOL.W4, Release 1.0 (french), Release 1.1, Janvier 1994, 20 pages.

[64] Philippe Lahire & Robert Chignoli & Jacques Farré & Nhan Le Thanh & Roger Rousseau. Specification and design of a model for an Eiffel object repository. Dans le projet Esprit II N° 5311 "Business class", Rapport BC.R.LIS.W4.1, Release 1.1, Août 1991, 167 pages.

[65] Philippe Lahire & Roger Rousseau. Six-month activity report Number 6, T0+30 -> T0+36. Dans le projet Esprit II N 5311 "Business class", rapport BC.R.TS.PR.6 (contribution), Janvier 1994, 3 pages.

[66] Philippe Lahire & Roger Rousseau. Six-month activity report Number 5, T0+24 -> T0+30. Dans le projet Esprit II N 5311 "Business class", rapport BC.R.TS.PR.5 (contribution), Août 1993, 2 pages.

[67] Philippe Lahire & Roger Rousseau. Six-month activity report Number 4, T0+18 -> T0+24. Dans le projet Esprit II N 5311 "Business class", rapport BC.R.TS.PR.4 (contribution), Janvier 1993, 4 pages.

[68] Philippe Lahire & Roger Rousseau. Six-month activity report Number 3, T0+12 -> T0+18. Dans le projet Esprit II N 5311 "Business class", rapport BC.R.TS.PR.3 (contribution), Août 1992, 3 pages.

[69] Philippe Lahire & Roger Rousseau. Six-month activity report Number 2, T0+6 -> T0+12. Dans le projet Esprit II N 5311 "Business class", rapport BC.R.TS.PR.2 (contribution), Janvier 1992, 3 pages.

[70] Philippe Lahire & Roger Rousseau. Six-month activity report Number 1, T0 -> T0+6. Dans le projet Esprit II N 5311 "Business class", rapport BC.R.TS.PR.1 (contribution), Août 1991, 3 pages.

Séminaires de projet

[71] Philippe Lahire & Materne Maetz Object Oriented Design of the K2 System. Dans un "review meeting", projet FAO/ONU - Food Agriculture Organization of the United Nations, Nicosi, Chypre, 15-18 Mai 1993.

[72] Robert Chignoli, Jacques Farré & Philippe Lahire. FLOO project : Evaluation and Perspectives. Dans le "Review Meeting" final, projet Esprit II N° 5311 "Business class", Paris, 7-11 Mars 1994.

[73] Robert Chignoli, Jacques Farré & Philippe Lahire. An example of object oriented inter-operability : Eiffel->O2. Dans le 4ème "Review Meeting", projet Esprit II N° 5311 "Business class", Paris, 4-5 Mai 1993.

[74] Robert Chignoli, Jacques Farré & Philippe Lahire. A prototype for an Eiffel Object repository. Dans le 3ème "Review Meeting", projet Esprit II N° 5311 "Business class", Milan, 28-29 Septembre 1992.

[75] Robert Chignoli, Jacques Farré, Philippe Lahire, Nhan Le Thanh & Roger Rousseau. Implementation of an Eiffel Object Repository with O2 OODBMS. Dans le 2ème "Review Meeting", projet Esprit II N° 5311 "Business class" , Paris, 24-25 Février 1992.

[76] Philippe Lahire & Robert Chignoli & Jacques Farré & Nhan Le Thanh & Roger Rousseau. Design of a model for an Eiffel Object Repository. Dans le 1er "review meeting" , projet Esprit II N° 5311 "Business class" , Bruxelles, 23-24 Septembre 1991.

Publications liées à l'encadrement de thèses

[77] Laurent Quintian. JAdaptor : Un modèle pour améliorer la réutilisation des préoccupations dans le paradigme objet. Mémoire de Thèse (160 pages) de Doctorat en Informatique de l'Université de Nice-Sophia Antipolis, juillet 2004, Sophia-Antipolis, France.

[78] Dan Pescaru. Bridging the Gap between Object Oriented Modeling and Implementation Languages Using a Meta-Language Approach. Mémoire de Thèse (121 pages) de Doctorat en Informatique de l'Université polytechnique de Timisoara décembre 2003, Timisoara, Roumanie.

[79] Laurent Jalady Vers une relation d'héritage générique. Rapport de DEA, 20 pages, Laboratoire I3S - Université de Nice-Sophia Antipolis, juin 2003.

[80] Pierre Crescenzo. OFL : un modèle pour paramétrer la sémantique opérationnelle des langages à objets - Application aux relations inter-classes. Mémoire de Thèse (190 pages) de Doctorat en Informatique de l'Université de Nice-Sophia Antipolis, décembre 2001, Sophia-Antipolis, France.

[81] Pierre Crescenzo. The Model OFL. Dans "12th Workshop for PhD. Students in Object-Oriented Systems" de la conférence ECOOP 2002 (16th European Conference on Object-Oriented Programming). Synthèse (Springer Verlag, LNCS series, juin 2002, 1 page, Malaga, Espagne.

[82] Pierre Crescenzo. The OFL Model to Customise Operational Semantics of Object-Oriented Languages - Application to Inter-Classes Relationships. Dans "11th Workshop for PhD. Students in

Object-Oriented Systems" de la conférence ECOOP 2001 (15th European Conference on Object-Oriented Programming). Synthèse "Object-Oriented Technology : ECOOP 2001 Workshop Reader" ; Ákos Frohner ; ISSN : 0302-9743 ; ISBN : 3-540-43675-8 ; Springer Verlag, LNCS series 2323, juin 2001, 1 page, Budapest, Hongrie.

[83] Adeline Capouillez. Improving the OFL Model Extendibility using Aspect-Oriented-Programming Techniques. Dans "11th Workshop for PhD. Students in Object-Oriented Systems" de la conférence ECOOP 2001 (15th European Conference on Object-Oriented Programming). Synthèse "Object-Oriented Technology : ECOOP 2001 Workshop Reader" ; Ákos Frohner ; ISSN : 0302-9743 ; ISBN : 3-540-43675-8 ; Springer Verlag, LNCS series 2323, juin 2001, 1 page, Budapest, Hongrie.