INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet TRISKELL

Construction fiable et efficace d'applications par assemblage de composants logiciels

Rennes

THÈME 1C

Table des matières

1 Composition de l'équipe

Responsable scientifique

Jean-Marc Jézéquel [professeur, Université de Rennes 1]

Assistante de projet

Marie-Noëlle Georgeault [TR Inria]

Christèle Soulas [IE Inria]

Personnel Inria

Jacques Malenfant [professeur Université de Bretagne Sud, en délégation Inria jusq'au 31 août 2001]

Loïc Helouët [CR, Inria, depuis le 1er novembre 2001]

Personnel Université de Rennes 1

Yves Le Traon [maître de conférences]

Noël Plouzeau [maître de conférences]

Post-doctorants

Gerson Sunyé [post-doc Inria, jusqu'au 31 août 2001]

Angeles Manjarrés [post-doc Université de Madrid, Espagne, depuis le 1er novembre 2001]

Ingénieurs-experts

Simon Pickin [ingénieur-expert CNRS (projet Cote) ; partagé avec VERTECS]

Laurent Monestel [ingénieur-expert Inria (projet CAFE), depuis le 1er novembre 2001]

Ingénieur associé

Katia Vinceller [poste d'accueil Inria]

Chercheurs doctorants

Benoît Baudry [bourse MENRT]

Éric Cariou [bourse MENRT]

Wai Ming Ho [bourse Inria jusqu'en septembre 2001]

Alain Le Guennec [bourse MENRT jusqu'en juin 2001]

Karine Macedo [bourse Inria]

Clémentine Nébut [bourse Inria région depuis le 1er octobre 2001]

François Pennaneac'h [bourse BDI/Inria-CNRS]

Damien Pollet [bourse MENRT depuis le 1er septembre 2001]

Hanh Vu Le [bourse Inria]

Tewfik Ziadi [bourse Inria depuis le 1er octobre 2001]

Collaborateur extérieur

Daniel Deveaux [maître de conférences, université Bretagne Sud]

2 Présentation et objectifs généraux

Le projet a pour objectif général d'aider à la construction fiable et économiquement efficace de lignes de produits par assemblage de composants logiciels modélisés en UML (Unified Modeling Language), en particulier dans le domaine des systèmes répartis et réactifs ayant des temps de réponse statistiquement contraints (« temps réel mou »).

Triskell a pour ambition de construire des modèles, des outils et des bibliothèques de composants spécifiques pour donner à un concepteur de logiciel un certain niveau de confiance dans la fiabilité d'assemblages de composants pouvant provenir de sources tierces. Il s'agit notamment d'étudier des modèles permettant la spécification de propriétés à la fois fonctionnelles et non-fonctionnelles de composants devant être déployés sur des systèmes répartis, et de bâtir un continuum d'outils tirant parti de ces éléments de spécification, allant de vérificateurs hors-ligne à des moniteurs supervisant en ligne le comportement des composants d'une application répartie.

Le projet a également pour ambition de connecter de manière explicite les résultats de ses recherches aux problématiques industrielles au travers d'actions de transfert technologique, ce qui implique en particulier de prendre en compte dès le départ les standards industriels du domaine que sont UML, Corba Component Model (CCM), .Net et Enterprise JavaBeans.

Triskell veut se placer à la frontière de deux domaines du logiciel : d'une part le domaine de la spécification et de la preuve formelle de logiciel et, d'autre part, le domaine de la conception informelle mais structurée par composants banalisés. Les techniques que nous développons permettront d'améliorer et de fiabiliser le passage entre formel et informel, et contribueront à la fluidité des processus de conception, mise en œuvre et test de logiciels.

3 Fondements scientifiques

3.1 Panorama

Résumé : *Le projet élabore de nouvelles technologies logicielles permettant d'aider à la construction fiable et économiquement efficace de lignes de produits logiciels en particulier dans le domaine des systèmes répartis et réactifs. Les problèmes centraux sont la modélisation des composants et de leurs comportements, et le développement d'outils de manipulation de modèles pour raffiner la conception, générer du code ou des tests. Les techniques de validation utilisées s'appuient sur des simulations complexes des modèles considérés, dans le cadre des standards du domaine.*

Glossaire :

Logiciel réparti désigne un programme informatique dont l'exécution met en jeu un ensemble de calculateurs travaillant en réseau. Chaque calculateur évolue à sa vitesse propre. Nous considérons généralement que l'interaction entre ces calculateurs est asynchrone et s'effectue par échange de messages.

Asynchrone signifie qu'un message peut rester en transit un temps non déterminé, découplant ainsi fortement l'activité des processus s'exécutant sur ces calculateurs. En général, ce type de logiciel est aussi réactif dans le sens où chacun des processus doit réagir aux sollicitations de son environnement et émettre des réponses à ces sollicitations.

Composant logiciel désigne au sens large un élément de logiciel jouant un rôle dans une composition. Nous l'utilisons ici dans le sens plus spécifique d'*unité de déploiement*, qui est celui le plus généralement admis dans l'industrie du logiciel [Szy98].

[Szy98] C. SZYPERSKI, *Component Software: Beyond Object-Oriented Programming*, ACM Press and Addison-Wesley, New York, N.Y., 1998.

*Canevas d'application (*framework *en anglais)* désigne un ensemble de composants logiciels qui fournit un ensemble intégré de fonctionnalités spécifiques à un domaine. Ces composants sont liés entre eux par de multiples motifs (*patterns*) de collaboration statiques et dynamiques. Il fournit un modèle d'interaction entre les différents objets instances des classes définies (ou seulement spécifiées pour les classes abstraites) dans le framework. Celui-ci présente en général une inversion du contrôle à l'exécution : alors qu'une application utilisant une bibliothèque s'appuie sur celle-ci, dans le cas d'une application utilisant un framework, c'est le framework qui effectue l'essentiel du travail et appelle "de temps en temps" un composant spécifique réalisé par l'implanteur de l'application. Un framework peut donc être vu comme une application semi-complète. Des applications complètes sont développées en héritant et en instantiant des composants paramétrés de frameworks. Il suffit donc en quelque sorte d'enficher dans un framework les composants spécifiques de son application pour obtenir une application complète.

Dans un contexte de programmation par objets, on s'appuie sur le mécanisme de la *liaison dynamique* pour dissocier la spécification d'une opération (donnée dans une classe du framework) de son implantation dans une sous-classe, qui fait partie du code applicatif fourni par l'utilisateur du framework.

3.2 Fondements mathématiques des systèmes réactifs et répartis

Mots clés : système de transitions étiqueté, ensembles partiellement ordonnés.

Résumé : *La structure mathématique qui caractérise le mieux les fondements des travaux de recherche sur les modèles de logiciels répartis sont les systèmes de transitions étiquetés (*labelled transition systems *en anglais, abréviation* LTS*)* [Arn92]*. Cette structure, développée il y a près de cinquante ans est l'un des fondements de l'informatique. Cependant, pour des modèles de systèmes de taille réelle son explicitation est hors de portée, il est nécessaire de ne la construire que de manière paresseuse (ce qu'on appelle «au vol»). L'autre aspect fondamental est la notion de causalité entre événements dans les exécutions réparties. C'est le concept central qui permet de parler de l'analyse des comportements des systèmes distribués* [Jar94]*.*

Systèmes de transitions Un LTS est un graphe orienté dont les arêtes, appelées transitions, sont étiquetées par une lettre prise dans un alphabet d'événements. Les sommets de ce graphe sont appelés états.

$M = (Q^M, A, T^M \subset Q^M \times A \times Q^M, q^M_{init})$

avec : Q^M ensemble des états, q^M_{init} l'état initial, A l'ensemble des événements, T^M la relation de transition.

Il est usuel de parler d'automate d'états finis pour désigner un système de transitions étiqueté dont l'ensemble des états et celui des événements sont finis. Il s'agit en fait du modèle de machine le plus simple que l'on puisse imaginer. Nous employons les LTS pour modéliser

[Arn92] A. Arnold, *Systèmes de transitions finis et sémantiques de processus communicants, Études et recherches en informatique*, Masson, 1992, 196 p.

[Jar94] C. Jard, *Vérification dynamique des protocoles*, Habilitation à diriger les recherches de l'université de Rennes 1, décembre 1994.

des systèmes réactifs le plus souvent répartis. Dans ce cadre les événements représentent les interactions (entrées ou sorties) du système avec son environnement. On parle alors de système de transitions *entrées-sorties* ou de IOLTS (*input-output* LTS).

Ces systèmes de transitions sont obtenus à partir de spécifications de systèmes réactifs répartis décrits dans des langages de haut niveau comme UML. L'association d'un LTS à un programme se fait par l'intermédiaire d'une définition opérationnelle de la sémantique du langage et est en général formalisée sous la forme d'un système de déductions. Pour un langage aussi simple qu'une algèbre de processus (CCS par exemple), la définition de sa sémantique opérationnelle tient en moins de dix axiomes et règles d'inférences , alors que pour un langage aussi complexe que UML, cela est plutôt l'affaire d'un document de plus de cent pages.

Pour des raisons de performance, ces sémantiques opérationnelles ne sont jamais mises en œuvre directement , mais font l'objet de transformations diverses. En particulier, la compacité du codage des états est un facteur déterminant de l'efficacité de la génération des LTS.

Les calculs et transformations opérés sur les LTS se résument à des parcours et calculs de points fixes sur les graphes. L'originalité réside dans la façon de les effectuer : par calcul explicite du LTS ou bien implicitement, sans calcul ou stockage exhaustif du LTS.

Les algorithmes classiques de théorie des langages construisent explicitement des automates d'états finis. Ils sont le plus souvent intégralement stockés en mémoire. Cependant, pour les problèmes qui nous intéressent, la construction (ou la mémorisation) exhaustive des LTS n'est pas toujours nécessaire. Une construction partielle suffit et des stratégies analogues aux évaluations paresseuses des programmes fonctionnels peuvent être employées : seule la partie nécessaire à l'algorithme est calculée.

Dans le même esprit il est possible d'oublier certaines parties précédemment calculées du LTS , et par recyclage judicieux de la mémoire, il est possible d'économiser l'espace mémoire utilisé par nos algorithmes.

La combinaison de ces stratégies de calcul sur des LTS implicites permet de traiter des systèmes de taille réelle même en utilisant des moyens de calcul tout à fait ordinaires.

Ensembles Partiellement Ordonnés On considère qu'une exécution répartie sur un réseau de processus I est faite d'événements atomiques E, certains étant observables, d'autre ne l'étant pas. Chaque événement est l'occurrence d'une action ou opération ; on considère habituellement qu'une action a lieu sur un seul et même processus du réseau. Si on note Σ l'alphabet des actions, nous avons donc :

$$\begin{cases} I & \text{ensemble fini de processus} \\ \Sigma & \text{ensemble d'actions} \\ \pi : \Sigma \to I & \text{placement des actions} \\ E & \text{ensemble d'événements} \\ \phi : E \to \Sigma & \text{étiquetage des événements} \end{cases}$$

La relation de causalité décrit le plus petit ordonnancement partiel sur les événements que l'on peut déduire du modèle de fonctionnement du réseau de processus que l'on s'est donné. Il a été présenté sous cette forme pour la première fois dans [Lam78] avec comme hypothèses sur le fonctionnement de l'architecture répartie que :

[Lam78] L. LAMPORT, « Time, clocks and the ordering of events in a distributed system », *Communications*

1. Les processus sont séquentiels. Deux événements ayant eu lieu sur le même processus sont ordonnés.
2. Les communications sont asynchrones et points à points. L'émission d'un message précède causalement sa réception.

Ces deux axiomes nous permettent de définir une relation d'ordre $\leq$ sur E : c'est la relation de causalité.

Tout ce que l'on peut dire est que tout ordonnancement aurait respecté l'ordonnancement causal ; autrement dit, le comportement réel du système est une *extension linéaire* de l'ordre de causalité.

Il n'est cependant ni réaliste ni même utile de chercher à savoir quelle extension linéaire s'est réellement produite. Cela n'est pas réaliste car les architectures réparties existantes n'offrent pas les moyens de synchroniser des horloges locales à chaque processus avec une précision suffisante. Cela n'est pas utile car cet ordonnancement dépend de conditions d'exécution qui ne sont pas contrôlables ou répétables. Il faut donc considérer que toute extension linéaire de la relation de causalité est un ordonnancement plausible.

La difficulté est dans la combinatoire en général exponentielle dans le nombre d'événements des extensions linéaires d'une relation d'ordre. Il existe cependant une structure intéressante pour représenter l'ensemble les états dans lequel le système a pu se trouver : c'est le treillis des antichaînes de la relation d'ordre [DP90]. En terme d'exécution répartie ceci correspond à désigner pour chaque processus quel a été le dernier événement qui s'est produit ; cela définit sans ambiguïté un état possible du système. Ce treillis (distributif) peut être représenté sous la forme d'un LTS, avec comme relation de transition, la relation de couverture. Il s'agit du treillis des états globaux.

3.3 Technologie objet dans un contexte de génie logiciel réparti

Mots clés : objets, composant logiciel, motifs de conception, framework.

L'approche objet pour le génie logiciel

L'approche à objets s'est aujourd'hui généralisée pour l'analyse, la conception et la réalisation des grands systèmes d'information, sans doute parce qu'elle permet de prendre en compte la nature fondamentalement incrémentale, itérative et évolutive du développement de logiciel [Jac85,Boo94]. Les diverses phases d'analyse, de conception et de réalisation utilisent le même cadre conceptuel (fondé sur la notion d'objet) et n'ont pas entre elles de frontières rigides, ce qui fait que le processus de développement par objets est parfois qualifié de *continu*. La notion de continuité est ici empruntée aux mathématiques : il s'agit de la propriété attribuée à la transformation menant du domaine du problème vers l'espace des solutions : informellement

of the ACM 21, 7, July 1978, p. 558–565.

[DP90] B. Davey, H. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, 1990.

[Jac85] M. Jackson, *System Development*, Prentice-Hall International, Series in Computer Science, 1985.

[Boo94] G. Booch, *Object-Oriented Analysis and Design with Applications*, édition 2nd, Benjamin Cummings, 1994.

une « petite » modification dans la définition des besoins produit une « petite » altération du logiciel.

Cet aspect *continuité* est particulièrement important dans le contexte de la construction des grands systèmes d'information car l'effort principal en termes de logiciel (parfois jusqu'à 80% ou plus [Mey88]) est consacré à sa maintenance. La modélisation du domaine du problème alliée à cette continuité du processus de développement facilite en effet la flexibilité du logiciel ainsi que la traçabilité des besoins (chemin allant du besoin à sa satisfaction par le logiciel). Ces propriétés se traduisent par des systèmes logiciels de meilleure qualité (moins de défauts et de retards), plus faciles à maintenir et à faire évoluer.

La notion d'objet fournit le substrat nécessaire au développement du concept de *composant logiciel*, vu ici comme unité de déploiement [Szy98]. Elle offre en effet des possibilités d'encapsulation et de masquage d'information qui permettent d'établir une analogie avec les composants matériels, avec en plus la notion d'adaptabilité qui permet de les adapter souplement. En effet, au-delà de la réalisation d'économies d'échelle dans le développement de logiciels en réutilisant des canevas d'application et des composants plutôt qu'en redéveloppant les applications à partir de zéro (approche *ligne de produits*), il s'agit aussi aujourd'hui d'offrir la possibilité de modifier radicalement le comportement et les services d'une application par substitution ou ajout de composants, même longtemps après son déploiement. Ceci a un impact majeur sur le cycle de vie du logiciel, qui doit maintenant intégrer des activités de :

- conception d'infrastructures d'accueil de composants ;
- conception de composants utilisables comme unités de déploiement ;
- validation et assemblage de composants (d'origines diverses) ;
- gestion des composants (maintenance).

Or il est clair que les approches empiriques, sans réel modèle de composition de composant, qui ont présidé à l'émergence d'une véritable industrie du composant (au moins dans le monde Windows) posent des problèmes insurmontables d'intégration et de validation, et ne peuvent donc facilement être transposées à des systèmes plus critiques, comme en témoigne par exemple la destruction accidentelle de la fusée Ariane 501 [JM97].

Parmi les défis à relever, le besoin de modèles d'assemblage de composants fondés formellement, ainsi que celui d'une qualité vérifiable se signalent tout particulièrement. Il ne faut toutefois pas non plus négliger l'impact méthodologique d'un développement fondé sur les composants, dans le cadre par exemple du modèle de maturité défini par le SEI (modèle CMM).

Si certains éléments de réponses à ces problèmes existent déjà dans des prototypes de laboratoire, l'adoption massive d'UML dans de nombreux secteurs du monde industriel ouvre des perspectives nouvelles pour faire évoluer, passer à l'échelle et ainsi rendre économiquement rentables leurs idées sous-jacentes. En effet, au contraire de ses prédécesseurs (OMT, Booch, etc.) qui ne définissaient qu'une syntaxe graphique, UML est partiellement formalisé au travers d'un méta-modèle (lui-même exprimé dans un sous-ensemble de UML) et contient un langage sophistiqué de description de contraintes appelé OCL (*Object Constraint Language*), utilisable

[Mey88] B. MEYER, *Object-Oriented Software Construction*, Prentice-Hall, 1988.

[Szy98] C. SZYPERSKI, *Component Software: Beyond Object-Oriented Programming*, ACM Press and Addison-Wesley, New York, N.Y., 1998.

[JM97] J.-M. JÉZÉQUEL, B. MEYER, « Design by Contract: The Lessons of Ariane », *Computer 30*, 1, janvier 1997, p. 129–130.

aussi bien au niveau du modèle que du méta-modèle. De plus, une extension d'UML appelée AS (*Action Semantics*) est en cours de normalisation afin de permettre de décrire précisément la sémantique des actions en s'appuyant sur des modèles à base d'ordres partiels. Tout ceci permet d'envisager de manipuler formellement non plus seulement des programmes, mais aussi des modèles UML capturant de nombreux aspects du logiciel, à la fois sur le plan technique (avec les quatre dimensions : données, fonctionnelle, dynamique, et déploiement) et sur le plan processus, depuis l'expression des besoins et l'analyse jusqu'à l'implantation et les tests, en passant par la conception (modèles de frameworks et de patrons de conceptions).

4 Domaines d'applications

4.1 Logiciels pour les télécommunications

Mots clés : télécommunication, génie logiciel, test, UML.

Résumé : *le secteur des télécommunications (au sens large des systèmes répartis et réactifs ayant des temps de réponse statistiquement contraints) est en grande expansion, avec la mise en place d'infrastructures mondiales interconnectant de multiples composantes, l'explosion des télécommunications mobiles et le développement de nouveaux services. Le contexte industriel français et européen est par ailleurs assez favorable. Du point de vue du logiciel, il n'est plus possible de concevoir une nouvelle application à partir de zéro pour répondre à chaque nouveau besoin. La pression est grande pour trouver des solutions flexibles répondant à des* gammes de besoins *(aspect méthode et modélisation de lignes de produits), tout en raccourcissant les délais de mise sur le marché (aspect outils de dérivation et de validation). Le projet Triskell, spécialiste de l'ingénierie des modèles, trouve là un terrain privilégié d'applications. L'augmentation de la complexité et les exigences de fiabilité et de réutilisation justifient pleinement les méthodes développées dans le projet. Les sujets abordés sont la composition fiable de composants logiciels, la validation de conceptions* UML*, la génération de tests à partir de modèles* UML.

L'activité de recherche du projet, centrée sur la maîtrise à la fois de l'efficacité du développement et de sa fiabilité, est en rapport avec deux types d'applications dans le domaine des télécommunications : la conception fiable de logiciels communicants et le test et diagnostic de systèmes communicants.

Conception fiable de logiciels communicants

L'exigence de fiabilité des logiciels est facile à comprendre dans un contexte où ils sont présents à de très nombreux exemplaires et dans différentes versions sur un grand réseau de télécommunication. L'accent est porté sur la faculté d'interopérer. Le coût d'apparition d'une faute majeure est considérable dans ces systèmes et la réparation longue et difficile. Il est à noter aussi une exigence d'évolutivité importante liée à la mise en œuvre rapide de nouveaux services. Nous encourageons l'utilisation de méthodes formelles pour faciliter la résolution de ces problèmes.

Mais il ne faut pas oublier que les méthodes formelles doivent de plus en plus s'intégrer à une "approche système" permettant aux ingénieurs de concevoir globalement les systèmes pour prendre en compte tout un ensemble de contraintes et d'objectifs liés aux besoins des utilisateurs. Ces méthodologies formalisées n'en sont qu'à leurs débuts : un bon exemple est l'approche objet qui s'étend rapidement au contexte des télécommunications.

Test des systèmes à objets

Le test est une autre facette du développement fiable ; il consiste à s'assurer que le système, une fois réalisé, est conforme à ses spécifications. Quel que soit le soin apporté à la conception, cette phase reste de première importance pour vérifier le bon fonctionnement du système dans des environnements complexes et évolutifs.

Les travaux du projet Triskell s'attaquent actuellement à résoudre deux grandes difficultés :
- qualification d'un composant
- intégration de composants

5 Logiciels

5.1 UMLAUT : un outil de manipulation de modèles UML

Participants : Jean-Marc Jézéquel, Wai Ming Ho, Alain Le Guennec, François Pennaneac'h, Gerson Sunye, Katia Vinceller.

Mots clés : UML, composant, patterns, validation.

Résumé :

De nombreux Ateliers de Génie Logiciel (AGL) pour UML permettent aux ingénieurs logiciels de tracer des diagrammes et de générer des squelettes de code à partir de ceux-ci. Mais souvent les utilisateurs avertis voudraient faire plus de chose avec leurs modèles UML, comme par exemple appliquer des design patterns spécifiques, générer du code pour des systèmes embarqués, simuler des aspects fonctionnels ou non fonctionnels du système, ou encore passer des outils de validation sur le modèle , activités difficiles à mener en utilisant les facilités de script offertes par la plupart des AGL.

UMLAUT (UML Automatic Universal Transformations) est un framework de transformation de modèles (dont le cœur est développé dans le cadre de la CTI FT R&D MetaFOR) permettant d'appliquer des manipulations complexes à un modèle UML. Ces manipulations sont exprimées comme des compositions algébriques de transformations élémentaires réifiées. Elles sont donc extensibles au travers des mécanismes classiques d'héritage et d'agrégation. UMLAUT est utilisé en particulier dans le cadre du projet RNRT Oural pour transformer le modèle UML d'une application répartie en un système de transitions étiquetées , dans l'objectif de le valider avec des outils avancés de validation de protocoles (e.g. TGV). UMLAUT est aussi utilisé, dans le cadre du projet RNRT Convergence, pour prototyper de futures évolutions d'UML telles que discutées actuellement à l'OMG.

UMLAUT *a été déposé par l'Inria à l'Agence de Protection des Programmes en 1999. Des versions de démonstration pour Linux, Solaris et Windows* NT *sont librement disponibles depuis le site web de l'équipe (http ://www.irisa.fr/UMLAUT).*

La notation UML (*Unified Modeling Language*) est issue des travaux de Booch, Rumbaugh et Jacobson, auteurs des méthodes de développement à objets parmi les plus utilisées (les méthodes Booch, OMT et OOSE). UML est le successeur commun de ces trois méthodes, dont elle reprend la plupart des concepts et notations, dans un souci d'unification. Le manque d'uniformisation, les difficultés à communiquer entre outils constituaient en effet un frein au déploiement des méthodes de développement à objets. Il était donc souhaitable de créer un langage commun convenant à la modélisation des systèmes informatiques, mais également suffisamment générique pour prendre en compte des problèmes d'autres domaines. Contrairement à ces prédécesseurs, UML impose un langage (en cours de normalisation par l'OMG), mais laisse libre le choix du processus de développement associé.

La principale particularité de la notation UML réside dans sa base formelle appelée *méta-modèle* : la sémantique des concepts manipulés par la notation UML est décrite en utilisant la notation elle-même. Cela a pour conséquence l'amélioration de la correction des modèles (suppression des ambiguïtés et des incohérences), et permet d'envisager l'utilisation de techniques formelles à des fins de vérification/validation, les éléments d'un modèle devant respecter les propriétés et contraintes définies dans le méta-modèle.

L'outil UMLAUT vise à montrer la validité d'une telle approche. Cet outil intègre le méta-modèle UML et fournit un ensemble de facilités permettant de raisonner sur des modélisations UML. Il s'agit en fait d'un véritable framework de transformation de modèles permettant d'appliquer des manipulations complexes à un modèle UML. Ces manipulations sont exprimées comme des compositions algébriques de transformations élémentaires réifiées. Elles sont donc extensibles au travers des mécanismes classiques d'héritage et d'aggrégation.

Les modèles sont fournis à l'outil soit à l'aide d'une interface graphique (implantée en Java avec Swing), soit dans un format d'échange comme CDIF (Case Data Interchange Format) ou XMI (le format normalisé par l'OMG d'échange de modèles UML sur la base de XML), ou encore un format "propriétaire" comme celui de Rational Rose (en cours de développement).

L'outil accepte également en entrée des sources de programmes Java ou Eiffel, afin de reconstruire automatiquement les modèles UML correspondants (*reverse-engineering*).

UMLAUT intégrant le méta-modèle UML, il est possible de réaliser simplement des transformations de modèles définies par un ensemble de règles fournies par l'utilisateur. Dans le cadre du projet RNRT Oural, nous avons commencé à appliquer cette idée pour transformer le modèle UML d'une application répartie en un système de transitions étiquetées , dans l'objectif de le valider avec des outils avancés de validation de protocoles.

UMLAUT est aussi utilisé, dans le cadre du projet RNRT Convergence, pour prototyper de futures évolutions d'UML en ce qui concerne un langage d'actions (ASL) telles que discutées actuellement à l'OMG.

6 Résultats nouveaux

6.1 Modèles d'assemblages de composants

Participants : Jean-Marc Jézéquel, Yves Le Traon, Noël Plouzeau, Jacques Malenfant, Karine Macedo, Éric Cariou.

Mots clés : télécommunication, objets, composant, ordres partiels, concurrence, UML, MSC.

Résumé : *Un modèle d'assemblage de composants est fondé sur la spécification précise des responsabilités réciproques des composants d'une part, et des composants vis-à-vis de leur environnement d'accueil d'autre part (aujourd'hui principalement Com+/.Net,* EJB*, Corba Component Model et ses dérivés).*

La notion de contrat vient de l'engagement entre le client et le fournisseur : si le composant client fournit des données respectant les préconditions, le composant fournisseur s'engage à produire des résultats qui respectent les postconditions. L'utilisation de contrats force les concepteurs à spécifier avant de mettre en œuvre, fournit une documentation plus formelle, apporte une redondance entre le code et la spécification permettant un contrôle croisé automatique et, enfin, permet une détection rapide des erreurs et de leur origine.

Dans la pratique, cette notion classique de contrat ne suffit pas pour la conception d'applications actuelles, qui sont souvent réparties, interactives, dépendantes de ressources aux performances très variables. Dans ces conditions, il est intéressant d'étendre la notion classique de contrat pour prendre en compte des aspects non fonctionnels d'une interface de service : par exemple des contraintes de synchronisation (gestion du parallélisme d'accès au service, par exemple), ou encore des contraintes de qualité de service (durée, délai, bande passante réseau, etc.) [?].

Les travaux réalisés en 2001 par le projet Triskell portent essentiellement sur la définition de méta-architecture UML intégrant les notions de qualité de service et compatibles avec le méta-modèle standard d'UML. Les travaux s'appuient sur un modèle UML définissant des dimensions de qualité de service à la QML. Dans le contexte de la spécification et de la conception de composants, il est nécessaire de distinguer plusieurs types d'activités :

1. la conception de composants élémentaires capables d'offrir une abstraction à partir de services qui ne sont pas fondés sur des modèles « propres » des aspects quantitatifs,
2. la conception de composants par assemblage et encapsulation de composants existants, eux-mêmes capables de définir les aspects quantitatifs des services qu'ils savent rendre,
3. la conception d'application classique où les aspects quantitatifs sont fournis aux utilisateurs, non sous la forme d'interface logicielle du type composant mais sous forme de paramètres physiques directement perçus (et éventuellement mesurés, pour contrôle).

[?] *** ERROR: citation '**Beugnard99**' undefined ***

Le projet Triskell a conçu une extension du méta-modèle UML qui permet d'ajouter des contrats (au sens de [?]) à un modèle de composant qui est lui-même une extension du modèle prédéfini dans le standard UML. Cette extension a été présenté en octobre 2001 à la conférence UML 2001 [28].

6.2 Validation d'assemblages de composants

Participants : Jean-Marc Jézéquel, Yves Le Traon, Daniel Deveaux, Hanh Vu Le, François Pennaneac'h, Alain Le Guennec, Benoit Baudry, Clémentine Nebut.

Mots clés : Testabilité, conception testable, test d'intégration, qualification des composants, UML, MSC.

Résumé : *La vérification et la validation d'assemblages de composants posent des défis nouveaux quant à l'estimation de la qualité et la fiabilité de l'édifice obtenu* [Wey98] *; ces défis devront impérativement être relevés pour permettre au génie logiciel fondé sur l'assemblage de composants de s'épanouir dans des domaines exigeant un haut niveau de fiabiloité comme celui des télécommunications. À cet égard, les certifications de type* ISO9000*, qui procèdent du principe selon lequel un processus de développement de qualité doit avoir pour résultat un logiciel fiable et de qualité, arrivent vite à leurs limites.*

Un premier angle d'attaque concerne l'amélioration des techniques de validation du logiciel à tous les niveaux (composant [13], intégration, système, maintenance), en particulier dans le contexte des lignes de produits qui ont pour but de réutiliser le plus grand nombre d'éléments logiciels au sein d'une famille de produits. Mais dans le cas de tels assemblages de composants, il semble impératif d'intervenir dès en amont, au niveau du processus de développement, en cherchant à améliorer la démarche de spécification et de conception pour élaborer par construction un produit logiciel plus testable, c'est-à-dire plus « facile » à tester. On parle alors de conception testable. Cet axe de recherche vise ainsi à établir les règles et modèles applicables au plus tôt permettant :

- *d'estimer l'effort et la qualité du test ;*
- *de déterminer les faiblesses d'une architecture logicielle ;*
- *de planifier les étapes de test et le contrôle de qualité.*

Intégration de composants

Le but de cette activité de recherche est de planifier les tests depuis un modèle de conception objet, aussi bien dans un cadre d'intégration du système que dans un cadre de non-régression lors des évolutions.

Le modèle de test pour être utilisable doit pouvoir évoluer avec la conception, et produire des plans de test d'autant plus précis et détaillés que la conception est avancée et complète.

[?] *** ERROR: citation '**Beugnard99**' undefined ***

[Wey98] E. J. Weyuker, « Testing Component-Based Software: A Cautionary Tale », *IEEE Software 1*, 5, septembre 1998, p. 54–59.

Enfin ce modèle, pour être utile lors des évolutions du système, doit différencier les parties contractuelles du système pour lesquelles les tests sont " figés " et donc réutilisables facilement, des parties correspondant à des choix d'implantation et susceptibles d'être remplacées.

À l'heure actuelle, le modèle de test proposé, le Graphe de Dépendances de Test, permet d'abstraire depuis UML ces informations utiles pour planifier les tests d'intégration et de non-régression [TJJM00].

Concernant la planification du test d'intégration, l'ordre d'intégration des composants influe beaucoup sur le coût de l'intégration, sur le nombre de bouchons ("stubs") qu'il faudra produire. Les bouchons ou bouchons de test ont pour rôle de simuler le comportement d'un composant non encore intégré au système. L'enjeu principal consiste alors minimiser la création de bouchons, l'un des problèmes étant les interdépendances entre composants, les cycles de dépendances. Le Graphe de Dépendances de Test, en adaptant des algorithmes de graphes permet de traiter ce problème de manière efficace et permet de produire un plan de test d'intégration proche de l'optimal, trop coûteux à calculer en général. Enfin, ce modèle permet de planifier l'utilisation des ressources de test et de réduire le coût et les délais de l'intégration du système [TJJM00].

Les travaux accomplis cette année portent :

1. Sur un état de l'art permettant la comparaison expérimentale des algorithmes existants avec notre approche. Cet état de l'art révèle tout d'abord qu'aucun travail scientifique n'a cherché à modéliser et à planifier les tests d'intégration des composants à partir de UML. Il existe par contre des travaux non spécifiques à UML qui portent sur le test d'intégration dans le domaine OO. Les algorithmes d'intégration étudiés servent de base de comparaison lors de l'évaluation de l'efficacité de nos propres algorithmes [21]. Les résultats obtenus sur 6 logiciels réels montrent que notre stratégie est la plus efficace en terme de minimisation des coûts et des délais d'intégration.

2. Sur l'étude d'une intégration mixte big-bang/incrémentale. Intégrer de manière strictement incrémentale des composants fortements dépendants est une solution assez artificielle et coûteuse car elle implique d'introduire des *bouchons de test* (stubs) ou des simulateurs des composants non encore testés. Le gain est important car le nombre de "stubs" nécessaires s'en trouve considérablement réduit, d'après les premières études que nous avons menées. Dans cette approche, la première difficulté consiste à identifier les parties du logiciel fortement dépendantes fonctionnellement. La seconde difficulté est algorithmique et se ramène à déterminer une heuristique dans une composante fortement connexe du graphe de dépendance. Étant donné un nombre maximum de composants intégrables en une étape, le problème se ramène à déterminer une coupe maximale en taille dans une composante fortement connexe qui minimise le nombre de stubs.

[TJJM00] Y. L. Traon, T. Jéron, J.-M. Jézéquel, P. Morel, « Efficient OO Integration and Regression Testing », *IEEE Trans. on Reliability 49*, 1, mars 2000, p. 12–25.

Testabilité d'une architecture objets décrite avec UML : application aux lignes de produits et aux patrons de conception

Pour toute estimation de la testabilité, la question qui se pose est la détermination de critères de test pouvant être capturés à partir de la spécification, ici une architecture décrite avec UML.

Un premier travail a permis de modéliser la notion *d'interactions de test* comme représentative de la testabilité d'une architecture. Cette clé d'interprétation d'un diagramme de classes a été appliquée aux patrons de conception.

Un catalogue des principaux patrons de conceptions avec leur testabilité potentielle et les règles pour lever les ambiguïtés de la spécification et améliorer la testabilité est présenté en [18].

Dans le prolongement de ce travail nous commençons à appliquer notre modèle pour la testabilité de familles de systèmes. Les lignes de produits induisent en effet un type d'architecture orienté-objet très particulier, qui comporte des éléments importants de réutilisabilité et des points d'évolution prédéterminés. Aussi, la validation de tels systèmes porte sur ces deux points :

- validation des éléments réutilisables du cœur de la ligne de produits,
- intégration des nouvelles versions à ligne de produit (problématique à mi-chemin entre le test d'intégration et la non-régression).

Ces points spécifiques au problème des familles de systèmes visent à assurer que chaque ajout d'une nouvelle version du système assure à la fois la bonne utilisation du cœur et ne provoque pas de régression du système. Les architectures étant spécifiques et permettant d'identifier les façades des points d'évolution, on devra produire un modèle qui capture automatiquement les dépendances de la version/évolution vis à vis du cœur : un test d'intégration pourra être alors produit et les interactions avec le cœur de l'architecture testées et validées.

Globalement, l'ensemble des modèles de test proposés devront exploiter au maximum les spécificités de la notion de ligne de produits pour produire des tests minimaux (car portant sur les points de variabilité), réutilisables (car les fonctionnalités sont en grande partie conservées) et extensibles,pouvant se spécialiser à chaque instanciation en un produit particulier.

Mesures de testabilité, diagnosabilité et robustesse pour des architectures orientées-objet

Dans le cas d'un assemblage de composants, plutôt que d'intervenir au niveau de la phase effective de test, il est préférable d'intervenir en amont du développement pour produire un logiciel testable. Pour y parvenir, il faut pouvoir identifier les attributs d'une architecture qui contribuent :

- à sa testabilité (sa capacité à " révéler" ses défauts lors des tests pour un effort donné),
- à sa diagnosabilité (la propriété du logiciel à permettre la localisation des fautes détectées),
- à sa robustesse (capacité à fonctionner même dans des conditions anormales).

Ces attributs permettent d'élaborer d'une part des mesures de ces facteurs qualité, d'autre part, des critères de modification d'une architecture pour la rendre plus testable/diagnosticable/robuste.

Les modèles sous-jacents à ces mesures et critères ont été définis pour être conformes à l'intuition, ce qui a été obtenu par axiomatisation de la mesure, et défini rigoureusement dans le cadre de la théorie de la mesure du logiciel, domaine qui atteint un début de maturité depuis quelques années ([17]). Pour obtenir la qualibration du modèle en vue de la mesure, nous avons utilisé une analyse de mutation sur plusieurs études de cas. L'analyse de mutation est une méthode de mesure de la qualité des tests procédant par injection systématique de fautes dans un programme. Si les tests détectent toutes les fautes injectables dans le programme, alors les tests sont de bonne qualité. La proportion des fautes détectées par les tests par injection de faute (analyse de mutation) est donc un indicateur révélant la confiance que l'on peut mettre dans la fiabilité du logiciel, sachant que les tests sont efficaces. La détermination des fautes significatives pour des programmes objet est en soi un problème de recherche [19]. Pour notre problème l'analyse de mutation a permis d'abord de construire un ensemble de test efficace pour les systèmes étudiés, puis d'estimer la robustesse des composants d'un système selon la qualité des contrats embarqués.

6.3 Manipulations de modèles UML

Participants : Jean-Marc Jézéquel, Yves Le Traon, Noël Plouzeau, Jacques Malenfant, Gerson Sunye, Katia Vinceller, François Pennaneac'h, Alain Le Guennec, Wai Ming Ho, Damien Pollet, Tewfik Ziadi.

Mots clés : télécommunication, objets, composant, ordres partiels, UML.

Résumé : *Les deux précédents axes de recherche nécessitent la manipulation formelle de modèles* UML *de composants et de leur infra-structure d'accueil, ce qui est en soi un problème de recherche. Les particularités d'*UML*, et en particulier le fait qu'il soit fondé sur l'utilisation de multiples points de vue d'une part et d'un méta-modèle défini plus ou moins formellement d'autre part, permettent d'envisager l'application de techniques classiques de transformations de programmes, non plus à des programmes, mais à des modèles d'analyse et de conception. Se posent alors bien sûr des problèmes fort intéressants d'équivalence entre raffinements successifs de modèles* UML*, depuis des modèles d'analyse jusqu'aux modèles d'implantation.*

*Sur le plan de la formalisation des aspects dynamiques d'*UML *(en cours de normalisation à l'*OMG [Con00] *et dans lequel nous sommes fortement impliqués), nous sommes motivés par l'utilisation de modèles à base d'ordres partiels. Ils permettent en effet de capturer de façon efficace les phénomènes de causalité et de concurrence dans les logiciels répartis. Pour l'instant, leur développement a été contenu dans le domaine de l'informatique théorique. Notre ambition est d'une part d'arriver à transférer les idées sous-jacentes aux modèles à base d'ordres partiels dans la sémantique d'*UML *et d'autre part à leur trouver des applications, par exemple dans le domaine de la simulation de scénarios, du diagnostic, de la génération de tests parallèles.*

[Con00] T. A. S. Consortium, « Updated Joint Initial Submission against the Action Semantics for UML RFP », 2000, `http://cgi.omg.org/cgi-bin/doc?ad/00-08-03`.

*La maîtrise de la sémantique d'*UML *nous ouvre aussi des perspectives nouvelles quant à l'application automatique ou guidée de* design patterns *(vus alors comme des règles de transformation de modèles sur lesquels des relations d'équivalences peuvent être définies), ainsi que sur la conception par aspects (cf. la programmation par aspects) dont l'application, manuelle ou automatique, peut-être vue comme l'exécution d'un méta-programme. De même que chaque étape d'un programme agit sur les données qu'il manipule, chaque étape d'exécution d'un méta-programme peut-être vue comme un composant qui transforme un modèle* UML *en un autre modèle* UML. *Les technologies sous-jacentes utilisées ici résultent du mariage de celles de la programmation par objets et composants avec celles de la méta-modélisation.*

Comme pour tout autre composant, on doit se poser la question de sa validité intrinsèque (c'est-à-dire de la transformation de modèle qui y est implantée), ainsi que celle de son intégration avec les autres composants de l'environnement de manipulation de modèles UML *: c'est ce défi de la méta-programmation fiable qu'on se propose de relever ici en reportant au niveau du méta-modèle* UML *les techniques élaborées dans les deux axes précédemments mentionnés, en s'appuyant sur notre outil Umlaut de manipulation de modèles* UML *(cf. §5.1).*

Langages d'actions pour modèles et méta-modèles UML

La principale particularité de la notation UML réside dans sa base formelle appelée *méta-modèle* : la sémantique des concepts manipulés par la notation UML est décrite en utilisant la notation elle-même. Nous avons beaucoup investi dans la formalisation de cette sémantique, en particulier au travers de notre participation à la fois au groupe de travail *p*UML (precise UML) et aux actions de normalisations d'UML entreprises à l'OMG. Ceci s'est concrétisé par la soumission *Response to OMG RFP ad/98-11-01 : Action Semantics for the UML* soumise conjointement entre : Concept Five Technologies, I-Logix, INRIA, Kennedy-Carter, Motorola, Kabira Technologies, Project Technology, Rational Software Corporation, et Telelogic. Au-delà de cette soumission, nous avons prolongé ces travaux en donnant à UML dans son ensemble une sémantique réflexive [23, 26, 11].

Cette approche permet de développer des outils manipulant des modèles UML, qui ont la particularité de pouvoir aussi manipuler le méta-modèle (perspectives de «bootstrapping» complet d'un outil). Avec l'outil UMLAUT, qui fournit un ensemble de facilités permettant de raisonner sur des modélisations UML, nous avons démontré la possibilité d'utiliser de techniques formelles à des fins de vérification/validation, les éléments d'un modèle devant respecter les propriétés et contraintes définies dans le méta-modèle.

Nous l'avons conçu comme un véritable framework de transformation de modèles permettant d'appliquer des manipulations complexes à un modèle UML. Ces manipulations sont exprimées comme des compositions algébriques de transformations élémentaires réifiées [10]. Elles sont donc extensibles au travers des mécanismes classiques d'héritage et d'agrégation. Ce travail a été supporté par un contrat avec FT R&D (voir 7.1).

Nous avons montré en particulier comment le modèle UML d'une application répartie peut être automatiquement transformé en un système de transitions étiquetées, ce qui permet de proposer une sémantique opérationnelle à de tels modèles UML [9]. Une des applications princi-

pales de ces travaux est la connexion à outils avancés de synthèse de tests (e.g. TGV) [24, 20, 25].

Refactorings et UML : ré-architecturation automatique d'un système au niveau de sa conception en UML

Initialement, les refactorings sont des opérations de transformation de code source écrit dans des langages à objets. Elles permettent la restructuration d'applications, tout en préservant leur comportement. Grâce aux refactorings, les développeurs peuvent faire évoluer la conception de leurs applications, en les rendant plus lisibles et surtout plus réutilisables, et de préserver leur fonctionnement initial.

L'objectif de ce travail est d'inventer et de prototyper un ensemble d'opérations de restructuration de modèles UML pertinentes vis-à-vis de la problématique d'une conception incrémentale. Le répertoire d'opérations s'appuiera sur une étude des opérations de restructuration de code existantes (et qui sont applicables à un modèle UML). Le prototypage devra permettre l'utilisation effective des opération répertoriées dans l'environnement open-source UMLAUT développé à l'Irisa, ainsi que de les combiner en des transformations complexes.

Les transformations étudiées à ce jour portent sur les vues suivantes du modèle UML :

1. Diagrammes de classes : renommage, ajout d'une super-classe, transfert d'une méthode partagée par les sous-classes à une super-classe...
2. Diagramme d'état : création d'un état composite, suppression d'un état composite, ajout d'un état initial dans un état composite...

   Ces transformations peuvent avoir un impact sur les autres vues du logiciel modélisé avec UML. Alors que la plupart de ces transformations —à savoir le renommage et l'ajout d'une super-classe— n'ont pas d'effet sur les autres vues d'UML, la fusion d'associations peut induire des changements dans les collaborations et les diagrammes d'objets. Les refactorings peuvent aussi être utilisés pour améliorer le design des diagrammes d'états.

Cependant, comme ceux-ci ne modélisent plus la structure du système mais son comportement, leur transformation soulève des difficultés au sujet de l'équivalence de modèles [27]. En perspective de ce travail, nous prévoyons une utilisation poussée de l'Action Semantics [23] pour rendre les modèles plus précis, ce qui pourrait ouvrir la voie à des refactoring plus sûrs (prouvés), et plus indépendants du langage de programmation.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 Méta-Framework pour Objets Répartis

Participants : Jean-Marc Jézéquel, Wai Ming Ho.

Mots clés : framework, design patterns, composant, système distribué.

Résumé : *Contrat* CTI- FT R&D *ref. 98 1B 070, mars 1998 - mars 2001*

L'objectif de cette étude est d'étudier, concevoir et valider des modèles et des méthodes pour la conception et le développement d'architectures d'objets distribués.

Le modèle Metafor intègre plusieurs patrons de conception et s'appuie sur le langage UML *pour mettre en pratique les techniques de construction et de réutilisation de Frameworks. Les prototypes sont développés sur la base d'*UMLAUT *l'outil de manipulation de modèles* UML *en cours de développement dans l'équipe. Une étude de cas représentative, permettant d'expérimenter et de valider l'approche, est fournie par* FT R&D.

Les résultats principaux des travaux réalisés sont décrits dans le document de thèse de Wai Ming Ho [10], qui propose un framework ayant pour objectif d'aider le concepteur à se forger un outil spécialisé pour transformer son modèle de conception général en un modèle de conception détaillé (ou modèle d'implantation) prenant en compte les contraintes de la plate-forme logicielle cible (par exemple, framework de persistance particulier).

Le modèle de conception générale de départ peut être vu comme un modèle UML orienté "métier" et "annoté" par l'occurrence de "design patterns" ou encore par des contraintes non fonctionnelles dans l'esprit de la programmation par aspect.

L'idée est que la réalisation d'un outil générique est probablement illusoire vue la variété des situations rencontrées, et donc, il semble opportun de définir plutôt un framework permettant de fabriquer rapidement un outil ad hoc pour chaque modèle de conception particulier (voire une famille d'outils si on veut offrir une variété de solutions d'implantations pour le même modèle de conception). Il s'agit donc d'utiliser l'idée de framework non plus seulement au niveau du modèle (dans le cas ou la plate-forme logicielle ciblée se présente comme un framework), mais aussi du méta-modèle pour permettre au concepteur de construire conjointement sa solution conceptuelle et l'outil qui va la transformer en solution opérationnelle. Le pari est ici que les avantages de l'idée de framework vont pouvoir se retrouver au niveau méta, c'est-à-dire de la manipulation de modèle.

Le framework proposé par Wai Ming Ho se présente comme un moteur de transformation qui permet de définir et d'exécuter des transformations de modèles construites par composition algébrique de transformations élémentaires (map, filter, reduce, ifelse, tuple) dans la lignée du Bird-Meertens Formalism, sans toutefois résoudre le problème de la non-commutativité des transformations.

Ce framework a été implanté en Eiffel, en relation avec la translation dans le même langage du méta-modèle UML 1.3 (constitué d'environ 150 classes et 500 relations), et constitue le cœur de l'outil UMLAUT (version Gauvain). Pour en valider l'intérêt, ce framework a été utilisé dans trois études de cas :

- la réalisation d'un outil de réingéniérie d'un sous-ensemble modèle du système d'information de France Télécom (Mercure), dans le cadre du contrat de recherche avec France Télécom R&D qui a servi de support à cette thèse,
- la réalisation de l'outil UMLAUT lui-même, qui dans sa version Lancelot a donc été (partiellement) bootstrappé à partir de la définition officielle de UML 1.3 au format MDL utilisé par Rational Rose
- la réalisation d'un outil de compilation d'UML vers les systèmes de transitions étiquetées à entrées-sorties (au format requis par Open Caesar) qui a implanté par Alain Le Guennec [9] (UMLAUT/Simulator).

Pour mener à bien cette tâche, il a fallut implanter un certain nombre de modules péri-

phériques, comme des lecteurs ou générateurs de formats d'échanges de modèles (CDIF, XMI, MDL) et de code, ce qui représente au total un travail de développement conséquent (de l'ordre de 100 000 lignes de code).

Les différents outils issus de ce framework sont pleinement opérationnels et dans le cas UMLAUT/Simulator, commence à être largement utilisé hors de l'IRISA.

7.2 Convergence SDL-UML (RNRT)

Participants : Jean-Marc Jézéquel, François Pennaneach.

Mots clés : UML, SDL, objets, protocole.

Résumé : *Projet exploratoire* RNRT *(Contrat 298C4990031318011, 1/12/98 - 1/12/2001).*

SDL *est un standard utilisé dans les télécommunications pour la conception de systèmes temps réel.* UML *est un nouveau standard pour l'analyse et la conception orientées objet d'applications de tous types. Des sociétés américaines étudient des évolutions de* UML *pour le temps réel en réinventant de nombreux concepts sans tenir compte du fait que* SDL *les normalise déjà.*

Ce projet étudie et prototype des extensions temps-réel de UML *s'appuyant sur les solutions* SDL. *Il va également enrichir* SDL *avec des constructions* UML. *Il aboutit sur des propositions d'évolutions de ces normes à l'*ITU *et à l'*OMG *pour arriver à une approche unifiée permettant à* UML *d'être plus puissant et aux dizaines de milliers d'utilisateurs* SDL *dans les télécommunications de préserver leur acquis et de bénéficier des avantages de* UML.

UML s'impose aujourd'hui comme notation universelle pour l'analyse et la conception. Certains des concepts présents dans UML visent plus particulièrement les systèmes temps réel, réactifs ou distribués : StateCharts, notion de classe active, de thread, etc.

La diffusion large et rapide d'UML fait qu'il est souvent considéré comme une alternative crédible à SDL et MSC, en dépit de quelques faiblesses dans le domaine du temps réel.

Les objectifs de ce projet étaient :

1. Prise en compte dans UML des concepts nécessaires au développement de systèmes temps réel ou distribués, dans un standard reconnu qui garantisse l'indépendance du client vis-à-vis du vendeur.
2. Définition d'une correspondance formelle entre UML et SDL, permettant par exemple :
   - de faciliter la migration de l'existant SDL vers UML,
   - de clarifier, voire de définir formellement les concepts flous d'UML en bénéficiant du travail important effectué sur SDL,
   - de faire cohabiter des descriptions SDL et UML.
3. Intégration dans SDL de descriptions UML.

Le projet s'est déroulé sur une durée de trois ans et a effectivement influencé les standards UML et SDL pendant cette période. Nos partenaires étaient Telelogic et Objectif Technologie, ainsi que Alcatel jusqu'en juin 2000.

7.3 Oural (RNRT)

Participants : Jean-Marc Jézéquel, Alain Le Guennec, Séverine Simon.

Mots clés : UML, TGV, validation, objets.

Résumé : *Projet pré-compétitif* RNRT. *Le projet s'est déroulé sur une durée de deux ans, à partir de février 1999. Nos partenaires étaient Softeam, Alcatel et Thomson LCR.*

*L'objectif de ce projet était de fournir un ensemble d'outils destinés à accélérer et améliorer le processus industriel de développement de produit par le biais de réutilisation de composants logiciels. Dans ce processus, nous nous sommes focalisés essentiellement sur les phases "amont" que sont : la définition et analyse d'architecture, la validation de l'architecture, l'exploitation des informations en vue de la génération de scénarios exécutables et la génération de squelettes de code. Sur une échéance de deux ans, le projet avait pour objectif la conception et le développement d'un environnement autour d'*UML *permettant à la fois :*

- *de décrire une application du point de vue de son architecture dans le formalisme standard* UML,
- *de valider fonctionnellement cette architecture,*
- *de valider, notamment par la simulation, des aspects non fonctionnels de l'architecture,*
- *de dériver cette description sur une plate-forme générique basée sur l'architecture de Corba.*

Le projet a réussi à fédérer autour du langage UML *les différents outils des partenaires, et à expérimenter un processus de développement industriel basé sur une modélisation unique de l'application.*

Le projet est articulé autour de l'outil d'édition UML Objecteering fourni par Softeam. Thomson-CSF apporte son expérience en validation non fonctionnelle d'architecture. Le projet Triskell apporte ses compétences en validation fonctionnelle du logiciel, et Alcatel-CIT réalise le lien entre la modélisation sous UML et la génération de code exécutable sur une plate-forme générique Corba.

Le sous-projet outil de description a pour but de définir autour de l'outil Objecteering les extensions requises pour permettre l'utilisation d'UML, tant du point de vue de l'ingénierie système que de l'ingénierie logicielle. Dans ce sens, une évolution de la sémantique d'UML a été prototypée (UML/T) et les outils exploitant cette sémantique ont été intégrés autour de l'éditeur d'Objecteering.

Le sous-projet outils d'ingéniérie fonctionnelle a pour but de définir et fournir un outillage destiné a faire une validation formelle d'une architecture logicielle décrite en UML/T. Ces travaux s'articulent autour des extensions d'UML destinées à enrichir la description fonctionnelle du comportement d'une classe. L'environnement propose un simulateur de comportement et un générateur de tests. L'utilisation de tels outils nécessite une modélisation fonctionnelle de la plate-forme d'exécution générique.

Le sous-projet outils d'ingénierie non fonctionnelle a pour but de fournir un environnement

de validation par la simulation d'une architecture de composants. Ces composants possèdent des propriétés aussi diverses que les performances temps-réel, l'utilisation de ressources, le degrés de fiabilité... qui sont extraites des descriptions UML/T. L'environnement de validation comprend un simulateur de comportement non fonctionnel et un générateur de tests. Afin de mener à bien les simulations, une modélisation non fonctionnelle de la plate-forme générique a été étudiée.

Le sous-projet outils de déploiement a pour but de fournir les outils destinés à déployer une architecture de composants sur la plate-forme générique Alcatel/Thomson pour en faire une architecture exécutable. Cet outil exploite les informations de déploiement décrites dans UML pour générer le code destiné à piloter les services de la plate-forme (ORB, tolérance au défaillances, autres middleware).

7.4 ITR-Softeam

Participants : Yves Le Traon, Jean-Marc Jézéquel, Hanh Vu Le.

Mots clés : tests d'intégration, UML.

Résumé :

Il s'agit d'une collaboration avec la société Softeam se déroulant sur une durée de trois ans à partir de octobre 1999 et visant à étudier la problématique du test d'intégration dans un cycle de vie objet fondé sur l'utilisation de UML (programme ITR de la région Bretagne).

Le but de l'ensemble du projet ITR en collaboration avec SOFTEAM et Triskell est :
- de fournir des moyens de planifier les tests dès la conception du logiciel (à partir de modèles UML, en particulier des diagrammes de classes),
- de fournir des stratégies en amont du développement les plus efficaces possibles pour minimiser l'effort de test,
- et de produire le plus automatiquement une synthèse de ces tests intégrée à l'outil Objecteering.

Les travaux accomplis portent sur :

1. Un état de l'art permettant de situer l'intérêt d'un tel projet et de servir de base de comparaison expérimentale pour mesurer les gains obtenus par les algorithmes proposés. Cet état de l'art révèle qu'aucun travail scientifique n'a cherché à modéliser et à planifier les tests d'intégration des composants à partir de UML. Il existe par contre des travaux non spécifiques à UML qui portent sur le test d'intégration dans le domaine OO. Les algorithmes d'intégration étudiés servent de base de comparaison lors de l'évaluation de l'efficacité de nos propres algorithmes.
2. La mise en place d'une méthodologie basée sur la notion classe-spécification-test comme canevas et la modélisation de la notion de dépendances de test à partir de UML. Les algorithmes développés visent à minimiser les coûts et délais de la phase de test d'intégration (minimisation des stubs et optimisation de la répartition des tâches de test). Ils traitent de manière efficace le problème des cycles de dépendances et produisent un plan d'intégration dès la conception.

3. Des études de cas sur des modèles UML correspondant à des logiciels réels et comparaison de notre approche avec les autres approches existantes.

7.5 COTE

Participants : Claude Jard, Jean-Marc Jézéquel, Simon Pickin.

Mots clés : Tests en UML, TGV, test symbolique.

Contrat RNTL/COTE CNRS. Novembre 2000 - Novembre 2002.

Ce projet fait collaborer deux laboratoires de recherche (Irisa avec les projets Triskell, Vertecs et Lande, et le LSR/Imag), deux grands industriels (France Telecom et Gemplus) ayant un intérêt stratégique pour le test de composants, et un industriel outilleur Softeam, développant un atelier UML. COTE a pour objectif de définir des techniques de modélisation de test en UML, pour pouvoir modéliser les modes d'exploitation d'un composant, et en dériver des moyens de test automatique, ainsi que des moyens de "labellisation". Triskell, en liaison avec Vertecs amène ses compétences UMLAUT/TGV et va participer à une intégration dans l'atelier Objecteering, tout en effectuant de la recherche en collaboration avec le projet Lande et le LSR sur le mixage des aspects contrôle et données dans le processus de génération de tests.

8 Actions régionales, nationales et internationales

8.1 Actions nationales

8.1.1 Groupe de travail test et objets

Participants : Jean-Marc Jézéquel, Yves Le Traon, Alain Le Guennec.

L'équipe Triskell anime un groupe de travail national sur le thème du test et des objets. Ce groupe est affilié au PRC/GDR ALP. Yves Le Traon en est le secrétaire. Une dizaine d'équipes en France y participent. La page Web `http://www.irisa.fr/testobjets` peut être consultée.

8.2 Actions financées par la Commission Européenne

8.2.1 IST QCCS (11/2000–05/2003)

Participants : Jean-Marc Jézéquel, Noël Plouzeau, Karine Macedo.

L'objectif du projet IST QCCS (*Quality Controlled Component-based Software development*) est de développer une méthodologie et des outils pour la conception et la mise en œuvre de composants distribués et à contrats, en s'appuyant sur la conception et la réalisation par aspects (*aspect-oriented programming* ou AOP).

La méthode de travail choisie consiste à :

- développer une méthodologie de conception de composants à contrats ;
- étudier plus précisément les propriétés de qualité de service (QoS), validité et sûreté ;
- concevoir une infrastructure de développement et d'outils ;

– valider l'approche au moyen de deux applications *user-driven*.

Sur ce projet d'une durée de 30 mois, démarré au 1er novembre 2000, les partenaires sont :

– la société Sema Espagne (*prime*), fournisseur de la première application ;
– l'Irisa, pour la méthodologie et le système de composition d'aspects (60 hommes mois) ;
– l'université de Francfort, pour l'infrastructure globale de la plate-forme ;
– la société KD software, PME tchèque, fournisseur de la seconde application.

Les travaux réalisés en 2001 ont conduit à la production d'une publication en commun, présentée à la conférence UML 2001 [28].

8.2.2 ITEA CAFE (07/2001–07/2003)

Participants : Jean-Marc Jézéquel, Yves Le Traon, Benoît Baudry, Clémentine Nebut, Tewfik Ziadi, Laurent Monestel.

L'objectif du projet ITEA CAFE (*from Concept to Application in System-Family Engineering*) est de développer autour d'UML une infrastructure permettant de définir et de gérer des familles d'architectures de logiciels correspondant à des lignes de produits. Ces familles d'architectures doivent exprimer les caractéristiques communes à un ensemble de logiciels et les moyens de les spécialiser pour des plates-formes ou des applications particulières.

Le rôle de plus en plus important des produits à base de logiciels dans les télécommunications, les systèmes de supervision, les systèmes de contrôle et commande soulève en effet une question majeure : celle du développement rapide et de l'exploitation efficace de la grande masse de logiciels qui constitue la partie riche en fonctionnalités des produits. Comme les marchés hautement concurrentiels augmentent les contraintes sur les coûts de développement et les délais de livraison, l'efficacité des processus, méthodes et techniques de production de logiciels est devenue un facteur essentiel vis-à-vis de la concurrence. Ce sont ces considérations, entre autres, qui font du développement de produits logiciels réalisés par familles d'applications et assemblage de composants, une réponse particulièrement pertinente aux exigences techniques et commerciales des marchés de haute technologie.

Dans ce contexte, le projet CAFE a pour objectif d'établir et de valider les technologies pour pouvoir rendre systématique l'approche nouvelle de développement en ligne de produits du logiciel de familles de systèmes. L'approche *ligne de produits* a pour but de réutiliser le plus grand nombre d'éléments logiciels au sein d'une famille de produits. Exprimés le plus souvent à l'aide d'UML, ces éléments logiciels sont produits à différents niveaux du cycle de vie logiciel et comprennent des exigences logicielles, des schémas de conception élémentaires *(design patterns)*, du code, des suites de tests, etc. Une ligne de produits a pour but la mise en commun des travaux de développement, de tests et de maintenance de ces éléments logiciels communs de façon à :

– réduire les coûts de production et de maintenance des affaires ;
– réduire le temps de production d'une affaire (ou *time-to-market*) ;
– améliorer la qualité des affaires par la réutilisation d'éléments logiciels déjà validés.

Labellisé en janvier 2001, le projet CAFE regroupe une nombre important d'industriels européens (notamment Philips, Siemens, Thales, Alcatel, ESI, Bosch Gmbh, etc.) dans le cadre du programme européen ITEA. L'intervention de Triskell dans ce projet porte sur deux points :

l'application de nos techniques de manipulations de modèles UML à des familles d'architectures (en particulier l'expression et la vérification de contraintes d'architectures exprimées au niveau méta-modèle en OCL), et dans ce cadre UML, la problématique du test des lignes de produits.

8.3 Réseaux et groupes de travail internationaux

Jean-Marc Jézéquel est membre de Nice (*Non-Profit International Consortium for Eiffel*), qui a en charge la standardisation du langage Eiffel et de ses bibliothèques. Il participe au groupe de travail international `\emTrustedComponents`, qui a pour objectif l'étude et le développement de technologies permettant l'utilisation de composants logiciels dans des systèmes critiques.

Par l'intermédiaire de J. Malenfant, qui fut professeur associé au DIRO de l'université de Montréal, et de D. Deveaux qui a effectué une partie de sa demi-année sabbatique à l'université de Montréal, nous entretenons des contacts avec l'équipe du Pr. Keller sur le thème de la manipulation de *design patterns* en UML. D'autre part, sur le plan du test objet, Triskell a développé des contacts avec l'université de Pise (A. Bertolino).

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

Jean-Marc Jézéquel a organisé et présidé la manifestation OCM 2001 (Objets, Composants et Modèles, Rennes, mai 2001), qui dans la lignée de ses incarnations précédentes (annuelles depuis 1995) avait pour objectif d'associer des laboratoires de recherche (Irisa, École des Mines de Nantes, Télécom Bretagne, etc.) et des industriels du grand ouest (Sodifrance, Softeam, TNI, Atlantide, France Télécom R&D) dans un pôle de compétences et un forum de discussion sur les technologies objets.

Jean-Marc Jézéquel a fait partie des jurys de thèse suivants :

- Rita Ramakrishnan, PhD Thesis, mars 2001, université Monash (Australie) (rapporteur) ;
- Hugues Martin, mars 2001, université de Lille (rapporteur) ;
- Ileana Ober, avril 2001, université de Toulouse (rapporteur) ;
- Alain Le Guennec, juin 2001, université de Rennes (encadrant) ;
- Marjorie Russo, juillet 2001, université de Nice (examinateur) ;
- Wai Ming Ho, septembre 2001, université de Rennes (encadrant) ;
- Frédéric Fondement, novembre 2001, université de Mulhouse (rapporteur) ;
- Xavier Blanc, novembre 2001, université de Paris VI (rapporteur) ;
- Gautier Koscielny, décembre 2001, université de Rennes (président) ;
- Francois Pennaneac'h, décembre 2001, université de Rennes (encadrant) ;
- Philippe Boinot, décembre 2001, université de Rennes (examinateur) ;

Jean-Marc Jézéquel a fait partie des comités de programmes des conférences :

- UML'2001 : 3rd International Conference on UML. Toronto, octobre 2001.
- OCM'2001 (Objets, Composants et Modélisation). Rennes, mai 2001.

- LMO'2001 (Langages et Modèles à Objets), Le Croisic, janvier 2001.
- AFADL'2001 : Nancy, juin 2001.
- ICSSEA 2001 : Génie logiciel, Paris, décembre 2001.
- AICCSA 2001 : Beyrouth, juin 2001.
- TOOLS Pacific 2002, Sydney, mars 2002.

Yves le Traon a fait partie des comités de programmes des conférences :

- ISSRE 2001 (Software Reliability Engineering), Hong-Kong, novembre 2001.
- Software Metrics Symposium 2002, Ottawa, juin 2002.

Jacques Malenfant a fait partie des comités de programmes des conférences :

- REFLECTION 2001 (Third International Conference on Meta-Level Architectures and Separation of Crosscutting ConcernsSoftware Reliability Engineering), Kyoto, Japon, septembre 2001.
- LMO'2001 (Langages et Modèles à Objets), Le Croisic,

et a été co-responsable de l'organisation des ateliers (workshops) pour la conférence internationale ECOOP 2001 tenue à Budapest, en juin 2001.

9.2 Enseignement universitaire

Jean-Marc Jézéquel anime un cours de méthodologie de conception par objets de logiciels en Diic 3$^{\text{ème}}$ année à l'Ifsic, et à l'ENSTB (Rennes) ; il assure les mêmes cours plus un cours d'architectures logicielles pour réseaux à haut débit à Supélec (Rennes).

Yves Le Traon a créé la filière Génie Logiciel du DESS-Isa.

Noël Plouzeau est responsable du cours d'analyse et conception par objets dans la filière *Génie logiciel* du DESS Isa de l'Ifsic.

L'équipe Triskell a accueilli en 2001 deux stagiaires de DEA de l'Ifsic, et un mini-projet Diic.

10 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] A. BEUGNARD, J.-M. JÉZÉQUEL, N. PLOUZEAU, D. WATKINS, « Making Components Contract Aware », *IEEE Computer 13*, 7, juillet 1999.

[2] C. JARD, J.-M. JÉZÉQUEL, A. L. GUENNEC, B. CAILLAUD, « Protocol Engineering using UML », *Annales des Telecoms 54*, 11–12, novembre 1999, p. 526–538.

[3] C. JARD, J.-M. JÉZÉQUEL, « ECHIDNA, an Estelle-compiler to prototype protocols on distributed computers », *Concurrency Practice and Experience 4*, 5, août 1992, p. 377–397.

[4] J.-M. JÉZÉQUEL, J.-L. PACHERIE, *Object-Oriented Application Frameworks*, John Wiley & Sons, New York, 1999, ch. EPEE : A Framework for Supercomputing.

[5] J.-M. JÉZÉQUEL, M. TRAIN, C. MINGINS, *Design Patterns and Contracts*, Addison-Wesley, octobre 1999, ISBN 1-201-30959-9.

[6] J.-M. Jézéquel, *Object Oriented Software Engineering with Eiffel*, Addison-Wesley, mars 1996, ISBN 1-201-63381-7.

[7] J.-M. Jézéquel, « Reifying Variants in Configuration Management », *ACM Transaction on Software Engineering and Methodology 8*, 3, juillet 1999, p. 284–295.

[8] Y. L. Traon, T. Jéron, J.-M. Jézéquel, P. Morel, « Efficient OO Integration and Regression Testing », *IEEE Trans. on Reliability 49*, 1, mars 2000, p. 12–25.

Thèses et habilitations à diriger des recherches

[9] A. L. Guennec, *Génie Logiciel et Méthodes Formelles avec UML : Spécification, Validation et Génération de tests*, thèse de doctorat, École doctorale MATISSE, Universite de Rennes 1, juin 2001.

[10] W. M. Ho, *Contribution à la Réification d'un Processus de Conception*, thèse de doctorat, Ecole doctorale MATISSE, Universite de Rennes 1, septembre 2001.

[11] F. Pennaneac'h, *UML : de l'action à la réflexion*, thèse de doctorat, Ecole doctorale MATISSE, Universite de Rennes 1, décembre 2001.

Articles et chapitres de livre

[12] L. Hélouët, C. Jard, B. Caillaud, « An Event Structure Semantics for Message Sequence Charts », *Special issue on graph transformation in Mathematical Structures in Computer Science (MSCS) journal*, 2001.

[13] J.-M. Jézéquel, D. Deveaux, Y. LeTraon, « Reliable Objects : a Lightweight Approach Applied to Java », *IEEE Software 18*, 4, July/August 2001, p. 76–83.

[14] J.-M. Jézéquel, *Traité IC2 : Langages à objets*, Hermès Science Publications, Paris, To be published 2001, ch. Eiffel.

[15] G. Texier, N. Plouzeau, « Automatic Management of Sessions in Shared Spaces », *Journal of Supercomputing*, 2001.

[16] Y. L. Traon, F. Ouabdesselam, C. Robach, B. Baudry, « From diagnosis to diagnosability : axiomatization, measurement and application », *Journal of Systems and Software*, 2002.

Communications à des congrès, colloques, etc.

[17] B. Baudry, Y. LeTraon, J.-M. Jézéquel, « Robustness and Diagnosability of OO Systems Designed by Contracts », *in : Proceedings of Metrics'01*, London, UK, April 2001.

[18] B. Baudry, Y. L. Traon, G. Sunyé, J.-M. Jézéquel, « Towards a 'Safe' Use of Design Patterns to Improve OO Software Testability », *in : Proceedings of ISSRE 2001*, November 2001.

[19] D. Deveaux, P. Frison, J.-M. Jézéquel, « Increase Software Trustability with Self-Testable Classes in Java », *in : Proceedings of ASWEC 2001*, IEEE CS Press, p. 3–11, Canberra, Australia, 2001.

[20] L. Du Bousquet, M. H., J.-M. Jézéquel, « Conformance Testing from UML Specifications », *in : Proceedings UML2001 wkshp : Practical UML-Based Rigorous Development Methods*, *GI-Edition - Lecture Notes in Informatics (LNI)*, Bonner Köllen Verlag, October 2001.

[21] H. V. Le, A. Kamel, Y. L. Traon, J.-M. Jézéquel, « Selecting an Efficient OO Integration Testing Strategy : An Experimental Comparison of Actual Strategies », *in : Proceedings of ECOOP2001*, J. L. Knudsen (éditeur), *LNCS*, *2072*, Springer, p. 381–400, Budapest, Hungary, June 2001.

[22] J. Malenfant, M.-T. Segarra, F. André, « Dynamic Adaptability : the MolèNE Experiment », *in : Proceedings of Reflection 2001*, 2001.

[23] F. Pennaneac'h, J.-M. Jézéquel, J. Malenfant, G. Sunyé, « UML Reflections », *in : Proc. of Reflection 2001*, septembre 2001.

[24] S. Pickin, C. Jard, T. Heuillard, J.-M. Jézéquel, P. Desfray, « A UML-integrated test description language for component testing », *in : Proceedings UML2001 wkshp : Practical UML-Based Rigorous Development Methods*, *GI-Edition - Lecture Notes in Informatics (LNI)*, Bonner Köllen Verlag, October 2001.

[25] S. Ramakrishnan, C. Mingins, J.-M. Jézéquel, A. Ulrich, « Panel : A Framework for Distributed Component Test Certification Facility - Conformity and Compliance Testing », *in : Proceedings ISSRE2001*, 2001.

[26] G. Sunyé, F. Pennaneac'h, W.-M. Ho, A. L. Guennec, J.-M. Jézéquel, « Using UML Action Semantics for Executable Modeling and Beyond », *in : Advanced Information Systems Engineering — CAiSE 2001*, K. R. Dittrich, A. Geppert, M. C. Norrie (éditeurs), *LNCS*, *2068*, Springer, p. 433–447, Interlaken, Switzerland, June 2001.

[27] G. Sunyé, D. Pollet, Y. LeTraon, J.-M. Jézéquel, « Refactoring UML Models », *in : Proceedings of UML 2001*, *LNCS*, *2185*, Springer Verlag, p. 134–148, 2001.

[28] T. Weis, C. Becker, K. Geihs, N. Plouzeau, « An UML Meta Model for Contract Aware Components », *in : Proceedings of UML 2001*, *LNCS*, *2185*, Springer Verlag, p. 442–456, 2001.