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]

Personnel Inria
Didier Vojtisek [IR Inria]
Loïc Helouët [CR Inria]

Personnel CNRS
Claude Jard [DR, CNRS, détaché professeur à l'ENS Cachan/Bretagne depuis le 1er septembre 2002]

Personnel Université de Rennes 1
Yves Le Traon [maître de conférences]
Noël Plouzeau [maître de conférences]

Post-doctorante
Angeles Manjarres [de janvier à juin 2002]

Ingénieurs-experts
Simon Pickin [ingénieur-expert CNRS (projet COTE) ; partagé avec VERTECS jusqu'au 30 octobre 2002]
Laurent Monestel [ingénieur-expert Inria (projet CAFE), jusqu'au 30 octobre 2002]
Jean-Philippe Thibault [ingénieur expert Inria (projet RNTL ACCORD), depuis le 1er octobre 2002]
Julien Thomas [ingénieur expert Inria(Projet MAGDA2), depuis le 15 mars 2002]
Romain Sagnimorte [ingénieur expert Inria (Projet QCCS), depuis le 1er Décembre 2001]

Ingénieur associé
Katia Vinceller [ingénieur associé jusqu'au 1er octobre]

Chercheurs doctorants
Benoît Baudry [bourse MENRT]
Éric Cariou [bourse MENRT]
Karine Macedo [bourse Inria]
Clémentine Nébut [bourse Inria région]
Damien Pollet [bourse MENRT]
Hanh Vu Le [bourse Inria]
Tewfik Ziadi [bourse Inria]

Collaborateur extérieur
Jacques Malenfant [professeur Université de Bretagne Sud]

2. Présentation et objectifs généraux

2.1. Introduction

Le développement des réseaux d'ordinateurs permettant l'interconnexion de machines se poursuit. Aussi les questions posées par la construction du logiciel pour ces systèmes sont d'une grande actualité. Même si des progrès spectaculaires ont été accomplis dans les méthodes de génie logiciel, le caractère intrinsèquement parallèle et réparti des logiciels mis en œuvre continue à poser des problèmes ardus de programmation. La question la plus sensible à nos yeux est celle de la maîtrise de la fiabilité du logiciel, c'est-à-dire le contrôle des conditions de son bon fonctionnement. La maîtrise du développement passe par le renforcement des activités de conception, validation et test.

Du point de vue de la conception, la priorité est donnée aux environnements de conception objet et à l'invention de « frameworks » spécialisés pour les systèmes communicants et intégrant des outils de validation.

Du point de vue de la validation, l'idée est de renforcer l'impact des méthodes formelles et des outils d'analyse pour permettre la mise au point des spécifications et la génération de tests pour les codes répartis.

Le projet Pampa contribue à l'élaboration de nouvelles technologies logicielles par l'étude de modèles formels des protocoles et l'invention d'outils informatiques associés. Nous privilégions la conception d'outils automatiques permettant d'aider aux tâches de conception, vérification, génération de code et test de programmes réels. Ces outils ont vocation à être diffusés dans le milieu académique et/ou industriel. La mise au point des modèles et outils s'effectue dans le cadre d'applications réparties situées principalement dans le domaine du logiciel pour les télécommunications.

Le fonds scientifique du projet est constitué des méthodes formelles en logiciel de télécommunication, des techniques du type « model-checking » par des parcours de systèmes de transitions, des méthodes de conception objet et des techniques de distribution de code.

L'activité scientifique du projet peut être structurée en deux thèmes de recherche :

- test des programmes répartis,
- conception objet et validation.

2.2. Le test des programmes répartis :

Nous nous concentrons sur les techniques par modèles (dites « model-checking »), et particulièrement sur les aspects algorithmiques (algorithmique à la volée). Nous avons étendu ces techniques pour être capables de générer automatiquement des séquences de test de conformité à partir de spécifications dans des langages comme SDL ou Lotos. Cela a conduit à un outil original par son algorithmique et son architecture (appelé TGV), que nous valorisons auprès de la société Telelogic. Un autre aspect du test est l'évaluation des comportements d'un code réparti à partir des traces qu'il produit. Pour modéliser les phénomènes de causalité et de concurrence, la théorie de l'ordre est notre outil mathématique de base. Nous avons plusieurs applications comme le test d'interopérabilité et le diagnostic de pannes dans les réseaux. Enfin, pour qu'elle ait un véritable impact, la validation (vérification/test) doit s'intégrer dans des environnements et méthodes de conception existantes. Pour cela, le paradigme objet et la notation UML sont maintenant incontournables.

2.3. Conception objet et validation :

L'objectif général est la construction fiable et efficace d'applications réparties par assemblage de composants logiciels. Le cadre objet et la notation UML largement diffusée servent de support. L'approche est de fonder les différentes vues d'UML sur des modèles sémantiques formels, de proposer des méthodes d'assemblage et de validation s'appuyant sur l'expression en UML des propriétés des composants, et de prototyper des outils afférents. Les activités du projet concernent la manipulation formelle de modèles UML (l'outil UMLaut), l'étude d'un modèle sémantique pivot mixte synchrone-asynchrone (BDL), la synthèse d'automates à partir de scénarios et l'intégration de techniques de génération de tests (utilisation de TGV, algorithmes pour le test d'intégration).

3. Fondements scientifiques

3.1. Panorama

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.

3.2. Technologie objet pour le génie logiciel réparti

Mots clés : *objets, composant logiciel, motifs de conception, cannevas d'application.*

3.2.1. 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 du logiciel [66][55]. 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 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 [70]) 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 les bases nécessaires au développement du concept de *composant logiciel*, vu ici comme unité de déploiement [74]. 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 d'assemblage 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 [68].

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 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).

3.3. Fondements mathématiques des systèmes réactifs et répartis

Mots clés : *système de transitions étiqueté, ordres partiels, structures d'événements.*

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) [49]. 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 [67].

3.3.1. 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. Un LTS peut se définir par un nuplet $M = (Q^M, A, T^M \subset Q^M \times A \times Q^M, q^M_{init})$, dans lequel : Q^M est un ensemble d'états, q^M_{init} est un état initial, A est un ensemble d'événements, T^M une 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 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 d'une 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, 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.

3.3.2. Modèles non entrelacés

Un des inconvénients bien connus des systèmes de transitions [56] est de représenter la concurrence au moyen d'entrelacements de comportements. Il en résulte ainsi une perte de place en mémoire, mais également une complexité accrue des modèles. Les modèles à base d'ordres pariels résolvent en partie ce problème.

Un *ordre partiel* est un nuplet $(E, \leq, \Pi, \phi, \Sigma, I)$ dans lequel :

- E représente un ensemble d'événements atomiques, certains étant observables, d'autres 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.
- $\leq$ est une relation d'ordre partiel décrivant une relation de précédence entre les événements. Cette relation d'ordre est obtenue en posant comme hypothèses que :

  i. Les processus sont séquentiels. Deux événements ayant eu lieu sur le même processus sont ordonnés.
  ii. Les communications sont asynchrones et points à points. L'émission d'un message précède causalement sa réception.
- Σ est un alphabet d'actions.
- I est un ensemble de noms de processus
- $\Pi : \Sigma \longrightarrow I$ est un placement des actions
- $\phi : E \longrightarrow \Sigma$ est un étiquetage des événements

Un ordre partiel peut représenter un ensemble d'exécutions d'un système, de manière plus « compacte » qu'un système de transitions. Un autre avantage des ordres partiels est d'identifier clairement la concurrence : deux événements qui ne sont pas causalement dépendants peuvent s'exécuter concurremment. Dans un système de transitions, cette situation aurait été représentée par un entrelacement.

On appelle *linéarisation* d'un ordre partiel un ordre total respectant l'ordre causal. Toute linéarisation d'un ordre partiel est une exécution potentielle du système représenté. Un des inconvénients des ordres partiels est l'impossibilité de représenter des alternatives autres que celles liées à la concurrence.

Une *structure d'événements* première [79] est un ordre partiel auquel on adjoint une relation binaire dite de conflit. On décrit une Structure d'événements par un nuplet $(E, \leq, \sharp, \Pi, \phi, \Sigma, I)$ où :

- $E, \leq, \Pi, \phi, \Sigma, I$ ont la même signification que précédemment,
- $\sharp \subseteq E \times E$ est une relation binaire et symétrique et héritée par causalité ($\forall e \sharp e', e \leq e'' \Longrightarrow e'' \sharp e'$).

La relation de conflit dans une structure d'événements décrit les événements qui ne peuvent pas apparaître dans une même exécution. Les structures d'événements permettent ainsi d'introduire des choix dans un modèle d'ordre partiel. Les exécutions potentielles d'un système représenté par une structure d'événements sont des linéarisations de sous ensembles sans conflit de la structure d'événements. Le grand avantage d'une structure d'événements est de représenter concurrence et alternative au sein d'un unique modèle d'ordre partiel, plus proche de la compréhension habituelle de la concurrence qu'un modèle entrelacé.

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 [74].

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.

UML (Unified Modeling Language) est un langage de modélisation orienté objet qui permet de définir un système suivant plusieurs points de vue : utilisateur, statique, dynamique, et implantation. UML est la synthèse de plusieurs approches de la modélisation précédentes (OMT, Statecharts, Merise, ...) et est devenu une notation largement acceptée. Le *Meta-Modèle* d'UML est un modèle décrivant la structure de modèles écrits en UML.

Contrainte Une contrainte est une restriction d'un ensemble de valeurs d'un modèle ou d'un système orienté objet. Parmi les contraintes que l'on souhaite le plus souvent exprimer, on distingue des invariants (des propriétés qui doivent être vraies dans tout état stable du système), des préconditions (des propriétés qui doivent être satifaites par un système avant l'exécution de certaines actions), et des postcondition (des propriétés qui doivent être satifaites par un système avant l'exécution de certaines actions). *OCL* (Object Constraint Language) est un langage associé à UML permettant de définir des contraintes sur un modèle orienté objet.

Ligne de Produits Logiciels une ligne de produit logiciels est une architecture regroupant un ensemble d'éléments nécessaires à la construction d'une famille de logiciels. Ces éléments sont généralement appelés assets, et peuvent être des exigences, des morceaux de modèle, de code, ... Ce concept vise à faciliter la réutilisation de composants en la prenant en compte dès la phase de conception (on concoit dès lors *pour réutiliser*). Une fois une ligne de produits créée, la conception d'un logiciel de la même famille consiste à sélectionner un ensemble de composants, à les configurer, et à les assembler.

Scénario dans son sens le plus large, un scénario est un enchainement d'actions d'un système. Dans le contexte de TRISKELL, le terme scénario désigne un chronogramme, modélisé par un ordre partiel. Un scénario peut servir à la définition de jeux de test, pour documenter un système, pour capturer des exigences... Il existe différents langages permettant de définir des scénarios : les Message Sequence Charts (MSC), les Live Sequence Charts (LSC), et les diagrammes de séquences d'UML.

Programmation par aspects La programmation par aspect [69] cherche à apporter une réponse à une préoccupation courante en génie logiciel : la séparation des préoccupations. En effet, les méthodes de développement classiques doivent souvent faire face à un entrelacement de sous-problèmes techniques, difficile à maîtriser. La programmation par aspects consiste à décomposer un problème en un ensemble de composants et d'aspects transversaux. Un programme exécutable est ensuite obtenu en « tissant » les aspects, c'est à dire par des méthodes de composition (cela va de la définition de points de synchronisation explicites

permettant d'intégrer un aspect à un programme aux méthodes de superimposition de programmes).

4. Domaines d'application

4.1. Logiciels pour les télécommunications

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

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. 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 TRISKELL, 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.

4.1.1. 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.

4.1.2. 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, Simon Pickin, Damien Pollet, Romain Sagnimorte, Katia Vinceller, Didier Vojtisek.

Mots clés : *UML*, *MDA*, *composant*, *patterns*, *validation*.

UMLAUT (UML Automatic Universal Transformations) est un framework de transformation de modèles UML développé dans le projet Triskell depuis 1998 pour démontrer la validité d'une telle approche. Il permet d'appliquer des manipulations complexes à un modèle UML pour par exemple appliquer des design patterns spécifiques, utiliser la conception par contrats, tisser ensemble différents aspects d'une modélisation (au sens du *weaving* de la programmation par aspect), 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.

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.

De nombreux Ateliers de Génie Logiciel (AGL) pour UML permettent aujourd'hui 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 choses avec leurs modèles UML, comme par exemple appliquer des design patterns spécifiques, utiliser la conception par contrats, tisser ensemble différents aspects d'une modélisation (au sens du *weaving* de la programmation par aspect), 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.

Or, 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 pour les manipuler.

UMLAUT (UML Automatic Universal Transformations) est un framework de transformation de modèles développé dans le projet Triskell depuis 1998 pour démontrer la validité d'une telle approche. Il permet d'appliquer des manipulations complexes à un modèle UML pour répondre à l'ensemble de des besoins évoqués ci-dessus. 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.

Récemment, l'OMG a entrepris de définir un processus de développement centré sur le modèle pour suivre ce que l'on appelle une approche MDA (Model driven Architecture). Cette approche décrit les étapes de conception comme des raffinements successifs de modèles. Au cours de leur cycle de vie, ces derniers passent d'une forme indépendante de la plateforme (PIM ou Plateform Independent Model), vers une forme dépendente de la plateforme (PSM ou Plateform Specific Model). Une des technologies clé de cette approche réside donc dans les techniques de transformation de modèle qui permettent ce raffinement. UMLAUT se positionne donc aussi comme un des moyens de le réaliser.

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).

UMLAUT a été utilisé dans les projets suivant :

Contrat FT R&D Metafor développement du cœur de l'outil, application de patterns
RNRT Oural avec Softeam, Alcatel, Thalès sur la validation de modèles UML ;
RNRT Convergence avec Telelogic, Alcatel, Q-Labs sur la convergence SDL-UML ;
RNTL COTE avec Softeam, FT, Gemplus, LSR sur la synthèse de tests de composants ;

RNTL ACCORD avec Softeam, FT, EDF, CNAM, ENST, ENSTBr, LIFL sur l'Assemblage de Composants par Contrats en environnement Ouvert et Réparti ;

IST QCCS avec SchlumbergerSema, KD-soft, et TU Berlin sur la conception par contrats et par aspects avec UML ;

ITEA CAFE avec (en France) Softeam, Thalès, sur la manipulation de lignes de produits.

5.2. sofat : une boîte à outils pour l'analyse de scénarios.

Participant : Loïc Hélouët.

Mots clés : *Scénarios, diagrammes de séquence, analyse, simulation, validation.*

Il est habituel de représenter des comportements de systèmes distribués au moyen de chronogrammes informels ou scénarios. Cette représentation peut être utilisée dans des phases de définition d'exigences lors de la conception de logiciels, pour visualiser des traces de sortie lors du debogage d'un système, ou pour définir des jeux de tests. La plupart des ateliers de génie logiciel propose une telle notation. On trouve ainsi un langage appelé Message Sequence Charts dans la plupart des environnements SDL, et un autre langage appelé Diagrammes de Séquences dans les environnements UML. Bien qu'apparemment intuitifs, les scénarios atteignent un pouvoir d'expression élevé dès que les chronogrammes de base peuvent être composés. Il devient alors difficile de déduire des propriétés d'un scénario décrit sans un outillage approprié. SOFAT (Scenario Oracle and Formal Analysis Toolbox) est une boîte à outils permettant la manipulation de scénarios.

Les scénarios sont souvent utilisés pour décrire de manière informelle le comportement de systèmes distribués. Ils peuvent être utilisés à des fins de documentation, de capture d'exigences, pour l'expression d'objectifs de tests, etc. Les Message Sequence Charts et les futur Diagrammes de Séquences de la norme UML 2.0 permettent de composer des scénarios de base au moyen d'opérateurs de choix, de composition parallèle, de séquence, etc... Il en resulte un pouvoir d'expression accru, mais en contrepartie, de nombreuses propriétés deviennent indécidables sur le langage. Il est par exemple impossible de faire du model-checking d'un Message Sequence Charts dans le cas général. De telles limites n'en sont plus dès lors que l'on sait distinguer certaines sous-classes du langage permettant de décider de telle ou telle propriété.

La boîte à outils SOFAT permet d'effectuer une classification des scénarios en fonction de critères formels. Il est ansi possible de décider si un scénario est transformable en système de transitions, s'il est simulable, si les choix qu'il contient sont tous effectués par le même processus, etc. Lorsqu'un scénario est effectivement transformable en systèmes de transitions, SOFAT réalise cette transformation. Il devient alors possible d'effectuer toutes les opérations réalisables sur les automates : model-checking, abstraction, composition, ... Certains scénarios contiennent des ambiguités empêchant même leur simulation. SOFAT permet de détecter ces ambiguités, et d'effectuer une simulation interactive le cas échéant. Cette simulation est basée sur une sémantique des scénarios à partir de structures d'événements [63].

5.3. Mutator : une famille de logiciels de test par mutation.

Participants : Yves Le Traon, Daniel Deveaux, Benoit Baudry, Franck Fleurey.

Mots clés : *Test, test par mutation, Java, .Net.*

L'outil MUTATOR permet d'évaluer la qualité des tests embarqués dans un composant. Cette qualité est calculée grâce à une analyse de mutation. Pour effectuer cette analyse, des erreurs simples sont injectées dans le programme à tester, ce qui donne un ensemble de programmes erronés appelés mutants. Les tests sont ensuite exécutés sur chaque mutant et ainsi un score de mutation, correspondant à la proportion de mutants détectés par un test, est calculé pour chaque test. Un score de mutation de 50% signifie que les tests ne sont capables de détecter que la moitié des mutants, i.e. la moitié des fautes injectées.

La famille *MUTATOR se dérive actuellement en deux produits correspondants aux langages cibles : JMUTATOR pour Java et NMUTATOR pour $C\sharp$ (le langage spécifique de développement de la plate-forme .Net). Les deux produits offrent les même fonctionnalités de base : génération d'un ensemble de mutants à partir de l'ensemble de classes qui constituent le logiciel sous test, lancement des tests sur tout ou partie des

mutants générés, production d'un rapport de diagnostic pour visualiser les mutants encore vivants et définir les équivalents, et enfin détermination du score de mutation. Les deux outils permettent de faire du test par mutation de manière incrémentale, ce qui accélère la convergence vers un score de mutation de 100%. Enfin, ils sont compatibles avec les frameworks de test Junit pour Java et Nunit pour .Net. L'outil NMutator, dans l'état actuel, offre toutefois un ensemble d'opérateurs de mutation plus restreints que JMutator. Dans le courant de l'année 2003, il devrait être complété pour intégrer tous les opérateurs de mutation usuels plus ceux récemment définis dans la littérature.

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, Damien Pollet, Romain Sagnimorte, Angeles Manjarres.

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

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 un ensemble de préconditions, le composant fournisseur s'engage à produire des résultats qui respectent des 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.) [54].

Les travaux réalisés en 2002 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 [61]. 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).

Le projet Triskell a conçu depuis 2001[77] une extension du méta-modèle UML qui permet d'ajouter des contrats (au sens de [54]) à un modèle de composant qui est lui-même une extension du modèle prédéfini dans le standard UML.

Au cours de l'année 2002, cette extension a servir de point de départ pour la définition d'une méthode de conception globale, intégrant

- les notions de contrats (y compris les contrats de qualité de service) [43][44] ;
- les notions de transformations progressives de modèles ;
- l'emploi de techniques de conception par aspects (*aspect oriented programming*).

6.1.1. Conception par transformation de modèles

Les techniques de conception développées par les membres de Triskell en 2002 s'appuient sur un procédé de conception par raffinements et transformations successifs de modèles. Pour développer une application, un concepteur définit un premier modèle de conception indépendant de la plate-forme finale de déploiement. Ce modèle indépendant est ensuite décliné en versions spécifiques de telle ou telle plate-forme, et plus généralement de telle ou telle configuration de logiciel. La production des modèle dépendants de la plate-forme à partir du modèle indépendant repose sur la transformation de modèles UML *via* des programmes de transformation [45].

L'OMG (pour *Object management group*) propose une approche assez analogue sous la forme d'un processus nommé MDE, pour *Model driven engineering*. Le projet Triskell a concrétisé et détaillé son processus de MDE, avec un support logiciel grâce au logiciel Umlaut. La réalisation de cette méthode et des outils *ad hoc* est au cœur du projet IST QCCS [38].

6.1.2. Conception par aspects

Les techniques de programmation par aspects commencent à être reconnues dans le monde de la recherche et du développement industriel [69]. L'extension naturelle de la notion aspects aux phases de conception de logiciels donne naissance à la *conception par aspects*. Au cours de l'année 2002, le projet Triskell a commencé la construction d'une technique originale de définition et d'utilisation des aspects dans des modèles UML.

Plus précisement, un fragment de modèle UML peut être transformé en définition d'aspect en utilisant une extension de la notation UML. L'originalité de l'approche repose sur l'emploi de *message sequence charts* (abrégé en MSC) pour spécifier comment les éléments présents dans la définition d'un aspect doivent interagir avec les éléments qui les entourent dans le modèle instanciant l'aspect. Le MSC peut alors servir de guide au processus de transformation de modèle qui remplace la référence à un aspect par une expansion de cet aspect dans le modèle résultat. Contrairement au concept de *template* déjà présent dans la notation UML, le mécanisme d'aspect défini dans le projet Triskell permet de spécifier un comportement dynamique. Cette spécification peut alors être employée pour modifier le comportement des éléments du modèle. Si ces éléments contiennent du code exécutable, ce code peut être transformé pour que le comportement à l'exécution satisfasse les spécifications données dans la définition de l'aspect [42].

6.2. Validation d'assemblages de composants

Participants : Jean-Marc Jézéquel, Yves Le Traon, Claude Jard, Simon Pickin, Hanh Vu Le, Benoit Baudry, Clémentine Nebut.

Mots clés : *Testabilité, conception testable, test d'intégration, tests parallèles, qualification des composants, diagnostic réparti, UML, MSC.*

La vérification et la validation d'assemblages de composants posent des défis nouveaux quant à l'estimation de la qualité et de la fiabilité de l'édifice obtenu [78] ; 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 fiabilité 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 [65], 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. Ceci nous a conduit en particulier à la conception d'un nouveau langage de test à base de scénarios pour le contexte de systèmes multi-composants, et à réflechir à la synthèse de testeurs répartis à partir de modèles de vrai-parallélisme ainsi qu'à l'utilisation de dépliages de réseaux de Petri pour le diagnostic modulaire et réparti.

Dans le cadre de techniques basées sur l'assemblages de composants, il semble également impératif d'intervenir 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.

6.2.1. Optimisation automatique de cas de test unitaires

Nous utilisons l'analyse de mutation pour la qualification de cas de test unitaires. Cette technique proposée dans [57] consiste à injecter des erreurs dans le programme sous test. La proportion de programmes erronés détectée par les cas de test correspond alors à une estimation de la qualité de ces cas de test. On appelle cette proportion le score de mutation des cas de test. L'hypothèse est la suivante : si les cas de test sont capables de détecter les erreurs introduites volontairement, alors ils seront capables de détecter les erreurs dans le programme original. Une étude a montré qu'il est facile d'écrire à la main des cas de test obtenant un score de mutation entre 50 et 70%, alors que l'obtention d'un score plus élevé réclame un effort important [59]. Nous avons donc étudié une technique pour améliorer automatiquement la qualité de cet ensemble initial de cas de test. Pour cela, nous avons appliqué un algorithme génétique [62]. Le but de cet algorithme est de parcourir l'ensemble des solutions d'un problème à la recherche de l'optimum. Une solution est modélisée comme un individu, et l'algorithme se charge de repérer les meilleurs individus, de les reproduire, les croiser et les muter jusqu'à obtenir une bonne solution. Il faut donc fournir à cet algorithme une fonction objective qui permet de déterminer si une solution est meilleure qu'une autre. Dans le cas de l'optimisation de cas de test, nous cherchons le meilleur ensemble de cas de test pour un programme particulier, les individus sont donc des ensembles de cas de test et la fonction objective est le score de mutation des individus [22][20]. Deux outils ont été développé pour expérimenter l'application d'un algorithme génétique sur les cas de test pour des classes Eiffel ou $C\sharp$. Les résultats de ces expériences étant décevants (évolution lente et instable, scores obtenus inférieurs à 80%), nous avons modifié le modèle génétique pour prendre en compte certaines caractéristiques de notre problème (croisement inefficace, petites populations...). Ces modifications ont conduit à un modèle dit bactériologique, qui correspond plus au phénomène d'adaptation bactériologique, qu'à un phénomène d'évolution. Cette nouvelle approche a permis d'obtenir des scores supérieurs à 90%.

6.2.2. 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. 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 [75].

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 [75]. 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 [76]. 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.

6.2.3. 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 dans [51]. 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 la 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.

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 dérivation d'un produit particulier.

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

Dans le cadre de l'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 possibilité de localiser 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 de qualité, d'autre part, des critères de modification d'une architecture pour la rendre plus testable, diagnosticable ou 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[50]. 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 [58]. Pour notre problème, l'analyse de mutation a d'abord permis de construire un ensemble de tests efficaces pour les systèmes étudiés, puis d'estimer la robustesse des composants d'un système selon la qualité des contrats embarqués.

6.2.5. *Un langage de scénario pour spécifier des tests*

En 2002, nous avons défini un nouveau langage de description de tests abstraits dans le contexte du test de composants logiciels. Ce langage a été appelé Tela [34][40]. L'originalité de ce travail a consisté à se placer dans le cadre composants et objets, à mettre l'accent sur les aspects comportementaux et d'ordre partiel. L'approche technique a été d'embarquer le plus possible le langage dans la notation UML, afin de s'intégrer à une pratique industrielle. Le résultat est une solution mieux intégrée que les propositions usuelles d'habillage d'un langage de test existant avec UML. Notre proposition a d'ailleurs alimenté largement les travaux d'un groupe de travail de l'OMG, chargé de la définition d'un profil de test pour UML. Nous avons principalement utilisé les diagrammes de séquence, et avons du discuter en profondeur des aspects sémantiques des scénarios, profitant de notre expertise sur les HMSC [15]. Tela est utilisé comme langage pivot dans la chaîne d'outils développée dans le projet COTE [46][41] (voir la section consacrée aux contrats industriels). Il constitue la notation cible pour Umlaut/TGV, alimenté par un objectif de test décrit dans une version simplifiée et abstraite de Tela (O-Tela). C'est aussi la notation source pour les générateurs de tests exécutables situés en aval (EJB, .Net et Corba).

6.2.6. *La synthèse de tests parallèles*

La synthèse automatique de tests de conformité a été jusqu'à présent principalement développée dans l'objectif d'obtention de tests séquentiels. Dans le contexte des systèmes répartis, il est tentant d'étendre les techniques de synthèse à la génération de testeurs multiples. Nous fondons notre proposition sur notre expérience dans l'utilisation des techniques de « model-checking » comme mis en oeuvre dans l'outil TGV [30]. Continuant les travaux de A. Ulrich et H. König, nous avons proposé en 2002 d'utiliser un modèle de vrai parallélisme (« true-concurrency » en anglais) basé sur le dépliage de graphes [16][29]. Nous avons conçu une chaîne complète de synthèse, partant de la définition d'objectifs de test, et allant jusqu'à la projection sur un ensemble de testeurs.

Le modèle théorique sous-jacent est celui des structures d'événements, capturant les informations pertinentes de causalité et conflits, et obtenus par abstraction de dépliages (similaires à ceux utilisés pour l'activité de diagnostic décrite plus loin). Les modèles sont décrits sous la forme de "tas de pièces" mettant en oeuvre une sémantique d'ordre partiel des systèmes communicants. Une connexion avec celle d'UML est en cours d'étude. Un premier prototype rudimentaire de démonstration a été fabriqué (futur outil TGP).

6.2.7. *Diagnostic et dépliages des réseaux*

Le cadre est celui des systèmes à événements discrets, répartis et asynchrones. Les réseaux de télécommunications, mais aussi les grandes architectures logicielles à base de composants sont des exemples de tels systèmes. Dans ceux-ci, on imagine que le système d'observation (les capteurs) est lui aussi réparti ; chaque capteur local n'ayant qu'une vue partielle de l'état global. Ils ne sont pas fortement synchronisés et les alarmes doivent être traitées de façon asynchrone. C'est typiquement le cas des applications de télécommunication étudiées dans le projet Magda2 (voir la section contrats).

Dans ce contexte où les alarmes se promènent sur le réseau, la vision d'une "séquence d'alarmes" n'est plus un ordre total, mais un ordre partiel. L'approche originale que nous développons travaille en fait sur des trajectoires que nous manipulons en tant que graphes. De plus, le futur de la supervision nous apparaît répartie, avec une notion de supervision locale dont les diagnostics doivent être maintenus cohérents (ils doivent être perçus comme des projections locales d'un diagnostic global que l'on ne cherche pas forcément à reconstruire). C'est ce que l'on a défini comme la problématique du diagnostic réparti.

Cette recherche est menée depuis plusieurs années en collaboration avec l'équipe Sigma2 (A. Benveniste, E. Fabre et S. Haar) et concerne les aspects modèles et algorithmes. En 2002, l'approche consistant à regarder le diagnostic comme des réseaux Bayésiens de systèmes dynamiques a été poursuivie, avec la mise en place d'un cadre abstrait algébrique à partir duquel des algorithmes répartis concrets de satisfaction de contraintes sont obtenus [26]. Une autre approche de modélisation, fondée sur l'utilisation de dépliages de réseaux de Petri a été explorée. Elle permet de donner un cadre concret de représentation compacte des modèles, observations et trajectoires à reconstruire [24]. L'année 2002 voit la convergence de ces travaux et l'obtention d'un cadre théorique et algorithmique solide pour le diagnostic modulaire et réparti.

6.3. Manipulations de modèles uml

Participants : Jean-Marc Jézéquel, Noël Plouzeau, Loïc Hélouët, Laurent Monestel, Tewfik Ziadi, Clémentine Nebut, Damien Pollet.

Mots clés : *Lignes de produits*, *composant*, *ordres partiels*, *UML*, *Scénarios*, *MSC*, *diagrammes de séquences*.

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, 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. En particulier, les scénarios permettent de définir à un haut niveau d'abstraction les interactions entre les éléments d'un système. Malgré leur apparente simplicité, les scénarios ont une puissance d'expression considérable, difficilement appréhendable sans outillage.

La maîtrise de la sémantique d'UML nous ouvre aussi des perspectives nouvelles quant au traitement des *lignes de produits*. L'approche *lignes de produits logiciels* est une transposition des chaînes de production au monde du logiciel. Le principe est de minimiser les coûts de construction de logiciels dans un domaine d'application particulier en ne développant plus chaque logiciel séparément, mais plutôt en le concevant à partir d'éléments réutilisables, appelés "assets". Un asset peut ainsi être une exigence, un modèle, un composant, un plan de test ou tout simplement un document. Les problèmes qui se posent autour des lignes de produits concernent autant la construction de la ligne (quelle architecture choisir, comment construire des assets génériques et facilement réutilisables, comment détecter les points communs et les variations dans les assets) que son utilisation (comment compose-t-on des assets pour dériver un produit, comment peut-on exploiter la généricité de la ligne pour effectuer des opérations sur un ensemble de produits).

6.3.1. Etude des langages de scénarios

Dans leur expression la plus simple, les scénarios décrivent des interactions entre des éléments d'un système. Leur représentation graphique est souvent donnée sous forme de chronogrammes. Un scénario peut aussi être représenté formellement par un ordre partiel sur un ensemble d'événements.

Cependant, ces chronogrammes ne sont pas suffisants pour décrire des comportements intéressant. Il donc indispensable de munir un langage de scénarios d'opérateurs de composition. Cette idée n'est pas récente : Pratt[73] proposait déja de modéliser la concurrence dans les systèmes distribués à l'aide d'ordre partiels.

La plupart des langages de scénarios comme les HMSC, par exemple, proposent ainsi des opérateurs de composition séquentielle, de choix, d'itération, et de composition parallèle. Choix et itération suffisent à donner aux scénarios une puissance d'expression similaire à celle des parties rationnelles [53]. Cette particularité fait qu'il est impossible de décider dans le cas général si deux HMSC décrivent les mêmes scénarios, si les traces de l'un sont incluses dans les traces de l'autre, ou dans un langage régulier donné, ...

Ces résultats d'indécidabilité ne signifient pas que les scénarios sont inutilisables, mais doivent plutôt inciter à la prudence lorsque l'on souhaite réaliser des manipulations formelles. En effet, il est souvent possible d'identifier une sous-classe du langage permettant l'automatisation d'une manipulation formelle. La classification d'un HMSC revêt donc un intérêt primordial.

Une partie des travaux réalisés cette année porte sur la classification de scénarios selon certains critères décidables, et l'identification des manipulations formelles possibles sur les classes connues. Parmi les critères identifiés, on peut citer :

- la confluence[72]
- la localité des choix[52]
- la reconstructibilité des ordres[64]
- la bornitude [48]
- la connexité [71]

L'apparente intuitivité des scénarios en fait aussi des langages bien adaptés à la capture d'exigences. Cependant, lorsque l'on cherche à définir des comportements typiques d'un système distribué, il est rare de pouvoir représenter l'ensemble des exigences comportementales par un seul et unique HMSC. Il est plus fréquent de devoir composer un ensemble de vues décrites par des scénarios spécifiant les comportements attendus dans une situation donnée. Cette composition n'est pas un HMSC. En effet les vues comportent des parties redondantes, et il ne suffit pas de composer des vues au moyen d'opérateurs de séquence ou de composition parallèle pour obtenir un comportement cohérent du système étudié.

Le comportement global que l'on souhaite obtenir peut être considéré comme un « tissage » de scénarios, tel qu'il est pratiqué dans la programmation par aspects. Pour tisser des comportements, il faut avant tout pouvoir identifier les parties identiques dans des vues différentes. Nous avons proposé une approche basée sur les projections de MSC permettant d'identifier les similitudes dans deux HMSC par projection ([28]). Ces premiers travaux devraient être utilisables dans des techniques de tissage de scénarios.

6.3.2. Ingéniérie des lignes de produits

L'ingénierie des lignes de produits est une approche récente du génie logiciel. Elle regroupe les activités de développement d'un ensemble de logiciels appartenant à un domaine particulier. Les "lignes de produits logiciels" sont une transposition des chaînes de production au monde du logiciel. Le principe est de minimiser les coûts de construction de logiciels dans un domaine d'application particulier en ne développant plus chaque logiciel séparément, mais plutôt en le concevant à partir d'éléments réutilisables, appelés "assets". Un asset peut ainsi être une exigence, un modèle, un composant, un plan de test ou tout simplement un document.

La première difficulté liée à cette approche réside dans la conception d'une architecture permettant de définir plusieurs produits. Les membres d'une ligne de produits sont caractérisés par leurs points communs, mais aussi par leurs différences (aussi appelées "points de variation"). La gestion de cette variabilité est un des concepts clé des lignes de produits. Dans une chaîne de production de véhicules, des voitures sont fabriquées à partir d'un ensemble d'éléments communs (roues, volant, vitres, etc) mais peuvent comporter certaines caractéristiques qui les différencient (nombre de chevaux du moteur, présence ou non de la climatisation,...). Dans le monde logiciel, les différences peuvent apparaître de manière analogue, en fonction de choix techniques (utilisation d'un type particulier de matériel), commerciaux (création d'une version limitée),...

Une autre difficulté réside dans l'utilisation d'une ligne de produits. La construction d'un produit logiciel (on parle aussi de "dérivation de produit") consiste en partie à figer certains choix (cristallisation) vis-à-vis des points de variation définis dans la ligne de produits. De toute évidence, certains choix sont incompatibles entre eux. Si l'on reprend l'analogie ci-dessus, une voiture comporte généralement un seul moteur, et il faut alors choisir entre une motorisation essence ou diesel. De la même manière, un choix particulier lors de la dérivation d'un logiciel peut exclure certaines variantes. Par exemple le choix d'un cabriolet à toit amovible exclura la possibilité de choisir un toit ouvrant. Une ligne de produits doit donc aussi intégrer des contraintes de cohérence permettant de faciliter les choix lors de la dérivation.

Les travaux effectués dans l'equipe Triskell cette année concernent trois aspects des lignes de produits : la définition d'une architecture logicielle permettant le représentation des aspects statiques et dynamiques des produits, la définition d'un ensemble de contraintes de cohérence associées aux lignes de produits, et enfin la conception de tests génériques.

6.3.2.1. Modèles

Une première partie du travail réalisé cette année a consisté à définir une architecture permettant de modéliser une ligne de produits logiciels. Notre choix s'est porté sur UML. Une des raisons est que UML est déjà largement répandu dans la communauté lignes de produits. Même si UML ne contient pas explicitement de mécanismes pour modéliser la variabilité, les possibilités d'extensions du langage par les stéréotypes offrent une solution quasi immédiate. Comme cela a été proposé dans d'autres approches[60], nous définissons des points de variation dans les diagrammes de classe par :

- des classes optionnelles, identifiées à l'aide d'un stéréotype « optional ».
- des classes paramétrées.

La variabilité ne saurait cependant se limiter aux seuls aspects statiques d'un logiciel, et de grandes différences comportementales peuvent appaître entre deux produits. Si la variabilité a été bien étudiée en ce qui concerne les aspects statiques des lignes de produits, il n'en est pas de même pour les aspects comportementaux. Des points de variation dans des statecharts[60] ont été proposés, mais sont de trop bas niveau pour être réellement utilisables.

Nous avons proposé une extension des diagrammes de séquence pour qu'ils contiennent des parties optionnelles, variables, ou virtuelles[36]. Ces points de variation permettent la définition de comportements génériques instanciables pour chaque produit de la ligne.

Les questions à résoudre pour ce modèle comportemental restent nombreuses. Entre autres on peut se demander comment construire ces comportements génériques à partir d'un ensemble de comportements concrets connus de produits. On peut aussi s'interroger sur les propriétes que doit préserver chaque variante de scénario : doit-on obligatoirement respecter un ordre précis sur les événements communs à deux variantes, tout raffinement est-il autorisé ? Connaissant certaines hypothèses sur les points de variation, peut on déduire des propriétés génériques de toutes les variantes ?

6.3.2.2. Contraintes

Comme nous l'avons déjà souligné, des incompatibilités peuvent apparaître entre différents points de variation. De plus, une ligne de produits doit respecter certaines contraintes syntaxiques (une partie non optionnelle, par exemple, ne doit pas utiliser une autre partie optionnelle, sous peine de permettre la dérivation d'un produit incomplet).

De telles contraintes peuvent être exprimées à l'aide d'expressions OCL portant sur le méta modèle UML. Ainsi, pour la contrainte syntaxique citée plus haut, on définira une contrainte imposant que toute classe qui n'est pas stéréotypée par « optionnal » ne doit utiliser que des classes également non optionnelles.

L'architecture d'une ligne de produits est donc constituée de modèles UML stéréotypés (diagrammes de classes, de cas d'utilisation, de séquences), accompagnés d'un ensemble de contraintes définies par le concepteur. Chaque modèle de ligne de produits doit vérifier un ensemble de contraintes syntaxiques, et chaque modèle de produit dérivé du modèle doit satisfaire l'ensemble des contraintes définies par l'utilisateur. Ce modèle d'architecture de lignes de produits a été proposé dans [36].

L'avantage d'exprimer un ensemble de contraintes OCL au niveau méta-modèle réside dans le fait que ces contraintes peuvent porter sur n'importe quel type de diagramme UML. L'intégration d'un interpréteur OCL à l'outil UMLAUT développé dans l'equipe devrait permettre d'implémenter cet environnement.

6.3.2.3. Tests

La problématique de test sous jacente aux lignes de produits logiciels concerne la réutilisabilité des tests. Il semble en effet naturel de vouloir tester de la même façon les parties communes à tous les produits, et de ne pas écrire autant de tests que de produits pour un même objectif de test. Cette réutilisation de tests ne peut bien sûr pas avoir lieu au niveau du code de test, puisque les variabilités entre les produits impliquent des différences assez grandes ne serait-ce qu'au niveau du type des objets manipulés.

Il est donc nécessaire d'écrire des tests génériques, et ce dans un format dédié. Ces tests sont ainsi définis au niveau de la ligne de produits et non pas au niveau des produits. Les tests génériques sont ensuite instanciés pour chaque produit au moment de la phase de dérivation.

Les tests génériques peuvent ainsi être considérés comme des assets de la ligne de produits. Construire de tels assets n'a d'intérêt que si d'une part l'écriture de tests génériques est une tâche de complexité similaire à l'écriture de tests concrets, et d'autre part si l'instanciation de tests concrets est automatisée.

C'est pourquoi nous proposons de définir les assets de tests comme un ensemble de diagrammes de séquences qui représentent plus un objectif de test qu'un test proprement dit. Un tel ensemble de diagrammes de séquence est appelé patron de test comportemental, ou *behavioral test pattern*. Les diagrammes composant un patron de test sont très simples, et sont choisis parmi les scénarios des cas d'utilisations. Leur généricité provient d'une abstraction des types et des noms des instances manipulées, ainsi que des paramètres des méthodes appelées. Chaque patron de test se compose :

- d'un diagramme dit « accept » qui représente l'objectif de test (ce que l'on veut voir apparaître dans le test),
- d'un diagramme dit « préfixe » qui est destiné à amener le système dans l'état dans lequel on souhaite voir apparaître l'objectif de test,
- d'un ensemble de diagrammes dits « reject » qui représentent ce que l'on ne veut pas voir apparaître dans le test, soit pour éviter des effets de bord, soit parce que ça impliquerait des échanges de messages non pertinents pour le test.

Une fois la généricité de ces patrons de test traitée et les produits de la ligne instanciés, la chaîne Umlaut/TGV génère pour chaque produit un cas de test correspondant au patron de test. Les cas de tests générés à partir du même patron de test sont potentiellement tous différents d'un produit à un autre. D'autres moyens de générer les cas de tests sont étudiés actuellement, ainsi que la possibilité de générer automatiquement un certain nombre de patrons de test.

7. Contrats industriels

7.1. ACCORD (rntl)

Participants : Jean-Marc Jézéquel, Noel Plouzeau, Jean-Philippe Thibault.

Mots clés : *UML*, *composants*, *contrats*, *QoS*.

Projet pré-compétitif RNTL ACCORD (Assemblage de Composants par Contrats en environnement Ouvert et Réparti). Contrat 202C022700 31330 011, 1/12/2001 - 1/12/2003).

La conception des systèmes d'information repose de plus en plus souvent sur une construction à partir de composants logiciels. Cette approche est attractive, car elle peut réduire coûts et délais, mais n'est pas exempte de risques notamment si l'on intègre des composants d'origines diverses. Un cadre d'analyse et de conception dédié à l'assemblage de composants métiers permettrait la construction fiable de systèmes complexes, évolutifs et répartis. L'objectif du projet ACCORD est l'étude de la sémantique des composants et de leur assemblage.

L'approche proposée est d'associer à chaque composant, une spécification décrivant précisément son contrat d'assemblage. Dans ce but, le projet ACCORD propose la définition d'un profil « UML pour composants métiers » permettant d'exprimer les contrats. Cette notion permettra d'adapter UML à la manipulation de composants et d'assemblages, tout en restant indépendant des mises en oeuvre concrètes sur des plates-formes telles que EJB, CCM ou .NET. La faisabilité de l'approche proposée sera démontrée sur deux études de cas.

Le projet ACCORD a pour objectif de proposer aux architectes et intégrateurs de systèmes d'information un cadre d'analyse et de conception fondé sur l'explicitation de la sémantique de contrats pour décrire et assembler des composants métier. Cette approche présente plusieurs avantages :

- Elle facilite la compréhension du système, dont la description fonctionnelle est complète à tous les niveaux : composants et assemblages de composants ;
- Elle rend le système plus flexible : ayant explicité la structure et la composition des assemblages, on peut plus facilement composer (ou recomposer) les éléments du système (un composant, une application, un serveur, etc.) ;
- Elle accroît les possibilités de réutilisation : d'une part un assemblage devient lui-même un élément réutilisable, et d'autre part la connaissance d'assemblages valides de composants favorise la compréhension de ces derniers, et donc leur réutilisation effective ;
- Elle n'impose rien quant à l'infrastructure logicielle qui sera utilisée pour l'intégration des composants (de type .NET, Enterprise Java Beans ou CORBA Component Model). La correspondance entre l'architecture fonctionnelle et l'infrastructure n'est plus implicite (camouflée dans la conception), il s'agit au contraire d'une projection explicite, susceptible d'être assistée par des outils.

De nombreuses infrastructures pour le développement par composants existent (COM+, .NET, CCM, EJB). Cependant, l'expérience des partenaires du projet montre qu'il est intéressant (et plus prudent) d'en rester indépendant, mais aussi que les modèles sous-jacents à ces infrastructures ne permettent pas de comprendre l'ensemble du système, vu comme un assemblage « reconfigurable », du point de vue du métier. Ce constat rejoint plusieurs recommandations de l'OFTA (Observatoire Français des Techniques Avancées) qui préconisent : de rester indépendant des infrastructures, de concevoir les architectures avec des composants, ou encore de rendre les systèmes réutilisables et flexibles par la métamodélisation (notamment avec UML). Le projet ACCORD n'a pas pour objectif de réaliser une plate-forme d'exécution indépendante des choix techniques, mais s'intéresse aux activités d'analyse et de conception situées plus en amont dans le cycle de vie d'une application répartie. Cependant, l'approche proposée améliore l'indépendance entre les choix fonctionnels et les choix techniques. Le projet ACCORD est décomposé en trois lots :

- lot 1 : définition d'un modèle abstrait de composants et d'assemblages. Dans ce modèle abstrait, l'accent est mis sur la description sémantique des propriétés fonctionnelles et non-fonctionnelles d'un composant, ainsi que sur la description de la coopération entre composants sous forme de contrats. En s'appuyant sur le langage UML, nous proposons des profils génériques pour la représentation des concepts nécessaires aux composants et à leurs assemblages.
- lot 2 : règles de projection des modèles abstraits vers les modèles technologiques des composants CCM et EJB.
- lot 3 : validation des deux premiers lots par la mise en oeuvre de deux applications à base de composants. La première concerne le système d'information de France Télécom. La seconde est une application pour les logiciels scientifiques d'EDF.

Le projet ACCORD a pour objectifs :

- de faire avancer l'état de l'art sur l'assemblage de composants. Dans le cadre bien accepté industriellement de l'utilisation d'UML, l'innovation principale du projet ACCORD est d'associer à chaque composant une spécification qui décrit précisément son contrat d'assemblage. Ceci

permettra d'assembler des composants en tenant compte de leur sémantique et de leurs propriétés non-fonctionnelles ; et non plus seulement en utilisant la signature de leurs méthodes.

- de favoriser la compréhension et l'utilisation d'une approche de développement par composants. A ce titre, le projet doit déboucher sur un modèle favorisant effectivement cette compréhension. La diffusion des résultats du projet se fera notamment par la soumission de publications et d'exposés de synthèse aux principales conférences du domaine (par exemple OOPSLA, TOOLS, ECOOP, OCM, UML, LMO) où plusieurs membres du projet jouent des rôles actifs. Les réalisations du troisième lot pourront faire l'objet de présentations lors de conférences abordant le thème des composants et devront alors y être perçues comme des expériences exemplaires.
- de diffuser les résultats au travers de travaux de normalisation et de standards tels que UML. En effet, UML évolue actuellement vers une démarche de développement par composants, qui devrait conduire à la version 2 du langage pour 2003. Le projet ACCORD s'appuiera sur l'expérience acquise pour lancer ou appuyer des propositions à l'OMG dans le cadre des RFP pour UML 2.0. Pour favoriser la visibilité sur les travaux engagés et la réutilisation des résultats, le modèle abstrait et son intégration à UML par le mécanisme des profils feront l'objet de publications et de diffusions, notamment en " open source ".

7.2. MAGDA2 (rnrt)

Participants : Claude Jard, Julien Thomas.

Mots clés : *Supervision de réseaux de télécommunications, dépliages de réseaux de Petri, diagnostic réparti, Viterbi.*

Projet exploratoire RNRT Magda2 (modélisation et apprentissage pour la gestion distribuée des alarmes de bout en bout). Novembre 2001 - Novembre 2003. Partenaires : Alcatel, France-Telecom R&D, Ilog, Irisa, LIPN. Dans le prolongement du précédent projet Magda, ce projet met en oeuvre une nouvelle technologie de diagnostic distribué, fondée sur une modélisation UML des pannes et alarmes dans des réseaux hétérogènes. Les projets Triskell et Sigma2 collaborent étroitement sur ce sujet.

Le projet Magda2 se situe dans le prolongement du succès du projet Magda. Le projet Magda a vu le développement et le prototypage d'une approche entièrement nouvelle dans le domaine de la corrélation d'alarmes et du diagnostic. Une nouvelle technologie de diagnostic distribué reposant sur une modélisation approchée du système de gestion a été développée. Les participants au projet Magda ont acquis une expérience significative en matière de modélisation des pannes et alarmes dans les réseaux SDH. Outre la modélisation par elle-même, la méthodologie de construction de cette modélisation constitue un acquis encore plus important, parce que transposable. Le point important est l'objectif fondamental vers lequel on doit tendre, à savoir que les tâches de modélisation qui ne sont pas automatisables doivent être réduites au minimum.

Le projet Madga2 se concentre sur plusieurs objectifs :

- Industrialisation des technologies ayant un certain de gré de maturité :
  - achever la ligne technologique "chroniques" par industrialisation de l'apprentissage et commercialisation éventuelle.
  - prototyper une intégration de la technologie de "pièces et puzzles" dans JRules.
- Exploration des technologies d'ouverture :
  - porter à maturité les technologies de diagnostic distribué en prenant en compte la reconfiguration dynamique, la robustesse des algorithmes en présence de pertes d'alarmes ou de communication, ainsi que l'incomplétude des modèles.
  - explorer de nouvelles technologies d'apprentissage.

- Exploration de nouvelles cibles applicatives :
  - corrélation d'événements et diagnostic de réseaux hétérogènes (WDM et GMPLS).
  - interaction réseaux-services : corrélations d'événements et application à la QoS et au SLA (Service Level Agreement).

La participation du projet Triskell s'est concentrée cette année sur la théorie du diagnostic fondée sur la construction de dépliage de réseaux de Petri, en collaboration avec des chercheurs du projet Sigma2, ainsi que la conception pratique d'une méthode permettant d'exprimer le modèle des pièces dans la notation UML et de produire par compilation les règles correspondantes de JRules (voir la partie 6.1).

7.3. ITR-Softeam

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

Mots clés : *tests d'intégration, UML.*

Cette collaboration avec la société Softeam se déroule sur une durée de trois ans. Elle a débuté en octobre 1999 et vise 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 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.

7.4. RNTL COTE

Participants : Claude Jard, Jean-Marc Jézéquel, Yves Le Traon, Clémentine Nebut, Simon Pickin, Didier Vojtisek.

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. Triskell, en liaison avec Vertecs amène ses compétences UMLAUT/TGV et participe 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.

Le travail de l'année 2002 a consisté principalement à concevoir un langage de test, appelé Tela, qui permet de décrire des tests (ainsi que des objectifs de test) à l'intérieur de la notation des diagrammes de séquences de UML (voir la partie 6.1). Un effort important a aussi été consacré à la consolidation du prototype UMLAUT (principalement de la partie simulateur) et à son utilisation dans le contexte de la création dynamique d'objets.

8. Actions régionales, nationales et internationales

8.1. Actions nationales

8.1.1. Action Spécifique du CNRS sur le test des systèmes complexes

Participants : Claude Jard, Yves Le Traon.

L'équipe Triskell, en collaboration avec l'équipe Vertecs, participe à cette action nationale du CNRS. Celle-ci s'est focalisée sur un travail de prospective sur le test de robustesse et son lien avec le test de conformité. Elle prépare un rapport de prospective pour fin 2002. Les laboratoires participants sont le LAAS, le LABRI, Vérimag, le LRI et l'Irisa.

8.1.2. Action de Recherche Coordonnée de l'INRIA : FISC

Participants : Claude Jard, Loïc Hélouët.

L'équipe Triskell, en collaboration avec l'équipe S4, participe à cette action nationale de l'INRIA. Celle-ci s'est focalisée sur la coordination de travaux de recherche sur la sémantique, la formalisation et l'instrumentation des langages de scénarios. Les laboratoires participants sont le LIAFA (Paris7), l'INRIA Rhône-Alpes, et l'Irisa. Cette action se termine fin 2002 et définit un ensemble de perspectives ouvertes et pertinentes sur le sujet.

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 [77].

8.2.2. ITEA CAFE (07/2001-07/2003)

Participants : Jean-Marc Jézéquel, Yves Le Traon, Benoît Baudry, Clémentine Nebut, Tewfik Ziadi, Loïc Hélouët, 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

8.3.1. ARTIST

Le projet Triskell participe au réseau d'excellence ARTIST (Advanced Real-Time Systems Information Society Technologies, IST-2001-34820), dont l'objectif est de définir des directions de recherche innovantes et pertinentes dans le domaine des systèmes embarqués temps-réels. Dans ce réseau, Triskell est plus particulièrement impliqué dans l'action 2 : *Component-based Design and Development*.

8.3.2. Normalisation à l'OMG

Le projet Triskell à participé à des actions de normalisation au sein de l'OMG, via deux soummissions à des RFP :

- RFP MOF2.0 QVT Query/view/Transformation : L'OMG défini un processus de développement centré sur le modèle pour suivre une approche appelée MDA (Model driven Architecture). Cette

approche décrit les étapes de conception comme des raffinements successifs de modèles. Au cours de leur cycle de vie, ces derniers passent d'une forme indépendante de la plateforme (PIM ou Plateform Independent Model), vers une forme dépendente de la plateforme (PSM ou Plateform Specific Model). Les moyens d'effectuer des transformations étant variés, l'OMG souhaite les normaliser afin de permettre une meilleure interopérabilité entre les outils supportant cette approche.

- RFP UML2.0 Test Profile : Jusqu'à lors, la technologie UML s'est focalisée principalement sur la définition de la structure et comportement du système. À présent en ce qui concerne le test, UML ne fournit que des moyens très limités pour décrire des procédures de test. Cependant, avec la promotion de l'approche MDA par l'OMG, et compte tenu l'accent mis dans cette approche sur la génération automatique de code, le besoin de traiter le test de conformité et la certification d'une manière solide à l'intérieur d'UML se fait sentir de plus en plus.
  Le profile de test UML est basé sur le métamodel d'UML 2, et vise à permettre :
  - la spécification de tests pour les aspects structurels (statique) ainsi que pour les aspects comportamentaux (dynamique) des modèles UML
  - l'inter-opération avec les tecnologies de test boîte noire existantes, notamment TTCN-3 et JUnit

8.3.3. Trusted Components

Jean-Marc Jézéquel participe au groupe de travail international *Trusted Components* (http://www.trusted-components.org/), qui a pour objectif l'étude et le développement de technologies permettant l'utilisation de composants logiciels dans des systèmes critiques.

8.3.4. Coopération Test et Objet

Le projet Triskell a développé des contacts avec l'université de Carleton (Ottawa - Canada) sur le thème test et objets. Cette coopération s'est traduite par deux séjours de doctorants pour travailler sur des thèmes communs (contrats et tests).

9. Diffusion des résultats

9.1. Animation de la communauté scientifique

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

- Zakia Marrakchi, mai 2002, université de Rennes (président) ;
- Philippe Boinot, juin 2002, université de Rennes (président) ;
- Dean Thompson, PhD Thesis, septembre 2002, université Monash (Australie)
- Erwan Breton, juin 2002, université de Nantes (rapporteur) ;
- Renaud Pawlak, décembre 2002, université de Paris VI (rapporteur) ;
- Carine Courbis, décembre 2002, université de Nice (rapporteur) ;
- Vu Le Hanh, décembre 2002, université de Rennes (directeur) ;

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

- UML'2002 : 5th International Conference on UML. Dresden, octobre 2001 (président su comité de programme)
- OCM'2002 (Objets, Composants et Modélisation). Nantes, mars 2002.
- LMO'2002 (Langages et Modèles à Objets), Montpellier, janvier 2002.
- AICCSA 2002 : Beyrouth, juin 2002.
- TOOLS Pacific 2002, Sydney, mars 2002.

Claude Jard a fait partie des jurys de thèse suivants :

- Patricia Bouyer, avril 2002, ENS Cachan (rapporteur) ;
- Yannick Pencolé, juin 2002, université de Rennes (président) ;
- Lucien Ghirvu, juillet 2002, université de Grenoble (rapporteur) ;
- Fabien Peureux, Décembre 2002, université de Franche-Comté (rapporteur) ;

Claude Jard a fait partie des comités de programmes des conférences :

- TESTCOM'2002 : International Conference on Testing Communicating Systems, Berlin, Allemagne, mars 2002.
- TACAS'2002 : International Conference on Tools and Analysis of Systems, Grenoble, avril 2002.
- CFIP'2002 : Conférence Francophone en Ingénierie des Protocoles. Montréal, Canada, juin 2002.
- FORTE'2002 : International Conference on Formal Description Techniques, Houston, Texas, novembre 2002.

Claude Jard est membre du bureau de la section 07 du comité national de la recherche scientifique et membre du conseil scientifique du département STIC du CNRS. Il est membre du Comité des Projets de l'IRISA. Il a été président de la commission personnel de l'Irisa jusqu'en avril 2002, commission qui instruit scientifiquement les dossiers de recrutement des personnels scientifiques temporaires (doctorants, post-doctorants, invités, ingénieurs-experts, ingénieurs associés, délégations, détachements). Soit environ 150 dossiers par an. C'est à ce titre qu'il a fait partie de l'équipe de direction du laboratoire. En 2002, Claude Jard a été membre du comité d'évaluation de Vérimag, de l'IMAG/LSR, de l'IMAG/ID, du LIRMM, du PRISM.

Claude Jard est membre du conseil stratégique de la société TNI-Valiosys. Il est membre du conseil scientifique du centre de recherche en informatique de Montréal, Canada.

Claude Jard est membre du directoire du Réseau Thématique Prioritaire du CNRS sur les systèmes embarqués complexes.

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

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

Yves Le Traon a aussi été Examinateur de la thèse de l'Institut National Polytechnique de Grenoble de Maisaa Khalil - " Stratégies de diagnostic de systèmes logiciel-matériels soutenue en mai 2002.

Jacques Malenfant a fait partie du comité de programme de la conférence :

- LMO'2002 (Langages et Modèles à Objets), Montpellier, janvier 2002.

et est coordonnateur du groupe « Objets, Composants et Modèles » du pôle « Spécification, Programmation et Logique » du GDR ALP.

Loïc Hélouët a participé au jury de thèse de Aziz Salah, mai 2002, Université de Montréal (Rapporteur externe).

Loïc Hélouët a fait partie des comités de programmes des conférences :

- VISS 2002 (Validation and Implementation of Scenario-based Specifications), Grenoble, France 7 Avril 2002.
- IWCSE 2002 (International Workshop on Communication Software Engineering),Marrakech, Maroc, 19 décembre 2002.

9.2. Enseignement universitaire

Jean-Marc Jézéquel anime un cours de méthodologie de conception par objets de logiciels en Iup3 et en Diic 3e année à l'Ifsic, ainsi qu'à Supélec (Rennes) et à l'ENSTB (Rennes).
Yves Le Traon a créé la filière Génie Logiciel du DESS-Isa dont il assure la responsabilité. Il est responsable du cours de Génie Logiciel de tronc commun du DESS Isa ainsi que du cours sur le test et la vaildation de logiciels dans 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 2002 un stagiaire de DEA de l'Ifsic, ainsi que plusieurs stagiaires d'été.

10. Bibliographie

Bibliographie de référence

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

[2] E. FABRE, A. AGHASARYAN, A. BENVENISTE, C. JARD. *Fault Detection and Diagnosis in Distributed Systems : an Approach by Partially Stochastic Petri Nets.* in « Journal of Discrete Events Dynamic Systems », volume 8, 1998, pages 203-231.

[3] L. HÉLOUËT, C. JARD, B. CAILLAUD. *An Event Structure Semantics for Message Sequence Charts.* in « Mathematical Structures in Computer Science (MSCS) journal », volume 12, 2002, pages 377-403.

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

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

[6] J.-M. JÉZÉQUEL. *Reifying Variants in Configuration Management.* in « ACM Transaction on Software Engineering and Methodology », numéro 3, volume 8, juillet, 1999, pages 284-295.

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

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

[9] Y. L. TRAON, T. JÉRON, J.-M. JÉZÉQUEL, P. MOREL. *Efficient OO Integration and Regression Testing.* in « IEEE Trans. on Reliability », numéro 1, volume 49, mars, 2000, pages 12-25.

Livres et monographies

[10] *Modeling and Verification of Parallel Processes, MOVEP'2002.* éditeurs F. CASSEZ, C. JARD, F. LAROUSSINIE, M. RYAN., série European Summer School, Univ. Nantes, Nantes, juin, 2002.

[11] *UML'2002 - The Unified Modeling Language : Model Engineering, Concepts and Tools, 5th Intl. Conference.* éditeurs J.-M. JÉZÉQUEL, H. HUSSMANN, S. COOK., série LNCS, Springer, septembre, 2002.

[12] J.-M. JÉZÉQUEL, N. PLOUZEAU, Y. LE TRAON. *Développement de logiciel à objets avec UML.* Polycopié IFSIC C119, Université de Rennes 1, septembre, 2002.

Thèses et habilitations à diriger des recherche

[13] V. L. HANH. *test et modèle UML : stratégie, plan et synthèse de test.* thèse de doctorat, Ecole doctorale MATISSE, Universite de Rennes 1, novembre, 2002.

Articles et chapitres de livre

[14] L. HÉLOUËT. *Distributed system requirement modeling with Message Sequence Charts : the case of the RMTP2 Protocol.* in « International Journal of Information and Software Technology », 2002, à paraître.

[15] L. HÉLOUËT, C. JARD, B. CAILLAUD. *An Event Structure Semantics for Message Sequence Charts.* in « Mathematical Structures in Computer Science (MSCS) journal », volume 12, 2002, pages 377-403.

[16] C. JARD. *Synthesis of Distributed Testers from True-concurrency Models of Reactive Systems.* in « International Journal of Information and Software Technology », 2002, à paraître.

[17] G. SUNYÉ, A. LEGUENNEC, J.-M. JÉZÉQUEL. *Using UML Action Semantics for Model Execution and Transformation.* in « Information Systems, Elsevier », numéro 6, volume 27, juillet, 2002, pages 445-457.

[18] T. WEIS, N. PLOUZEAU, K. GEIHS, A.-M. SASSEN, J.-M. JÉZÉQUEL. éditeurs F. BARBIER., *New Advances on CBSE.* Kluwer Academic Pusblishers, 2002, chapitre QCCS : Quality Controlled Component-based Software development.

[19] Y. LE TRAON, F. OUABDESSELAM, C. ROBACH, B. BAUDRY. *From diagnosis to diagnosability : axiomatization, measurement and application.* in « Journal of Systems and Software », 2002, à paraître.

Communications à des congrès, colloques, etc.

[20] B. BAUDRY, F. FLEUREY, J.-M. JÉZÉQUEL, Y. LE TRAON. *Automatic Test Cases Optimization Using a Bacteriological Adaptation Model : Application to .NET Components.* in « Proceedings of ASE02 (Automated Software Engineering) », Edimburgh, UK, September, 2002.

[21] B. BAUDRY, F. FLEUREY, J.-M. JÉZÉQUEL, Y. LE TRAON. *Computational Intelligence for Testing .NET Components.* in « Proceedings of Microsoft Summer Research Workshop », Cambridge, UK, September, 2002.

[22] B. BAUDRY, F. FLEUREY, J.-M. JÉZÉQUEL, Y. LE TRAON. *Genes and Bacteria for Automatic Test Cases Optimization in the .NET Environment.* in « Proceedings of ISSRE02 (International Symposium on Software Reliability Engineering) », Annapolis, USA, November, 2002.

[23] B. BAUDRY, Y. LE TRAON, G. SUNYÉ. *Testability Analysis of UML Class Diagram.* in « Proceedings of Metrics02 », pages 54-63, Ottawa, Canada, June, 2002.

[24] A. BENVENISTE, E. FABRE, C. JARD, S. HAAR. *Diagnosis of Asynchronous Discrete Event Systems, a Net Unfolding Approach.* in « WODES'02 : 6th Int. Work. on Discrete Event Systems », Zaragoza, Spain, octobre, 2002.

[25] E. CARIOU, A. BEUGNARD, J.-M. JÉZÉQUEL. *An Architecture and a Process for Implementing Distributed Collaborations.* in « Proceedings of EDOC 2002 », Lausanne, Switzerland, septembre, 2002.

[26] E. FABRE, A. BENVENISTE, C. JARD. *Distributed Diagnosis for Large Discrete Event Dynamic Systems.* in « 15th IFAC World Congress on Automatic Control », Barcelona, Spain, juillet, 2002.

[27] W. HO, J.-M. JÉZÉQUEL, F. PENNANEAC'H, N. PLOUZEAU. *A Toolkit for Weaving Aspect Oriented UML Designs.* in « Proceedings of 1st ACM International Conference on Aspect Oriented Software Development, AOSD 2002 », Enschede, The Netherlands, avril, 2002.

[28] L. HÉLOUËT. *Projection et comparaison de Message Sequence Charts.* in « Proc. of Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'2003) », 2003.

[29] C. JARD. *Principles of Distributed Test Synthesis based on True-concurrency Models.* in « TESTCOM'2002 : Testing Communicating Systems », Berlin, Germany, mars, 2002.

[30] C. JARD, T. JÉRON. *TGV : Theory, Principles and Algorithms.* in « The Sixth World Conference on Integrated Design and Process Technology », Pasadena, California, juin, 2002.

[31] A. MANJARRÉS, S. PICKIN, G. SUNYÉ, D. POLLET, J.-M. JÉZÉQUEL. *OO Analysis Patterns as UML Metalevel Collaborations.* in « proc. of ES2002. The 22nd SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence : Research and Development in Intelligent Systems XIX », série BCS Conference Series, Springer-Verlag, decembre, 2002.

[32] A. MANJARRÉS, G. SUNYÉ, D. POLLET, S. PICKIN, J.-M. JÉZÉQUEL. *AI analysis patterns as UML meta-model constructs.* in « proceedings of SEKE 2002. The 14th international conference on Software Engineering and Knowledge Engineering », ACM Press, pages 237-238, Jul, 2002.

[33] C. NEBUT, S. PICKIN, Y. LE TRAON, J.-M. JÉZÉQUEL. *Reusable Test Requirements for UML-Modeled Product Lines.* in « Proceedings of REPL 2002 International Workshop on Requirements Engineering for Product Lines », Essen,Germany, sep, 2002.

[34] S. PICKIN, C. JARD, Y. LE TRAON, T. JÉRON, J.-M. JÉZÉQUEL, A. LE GUENNEC. *System Test Synthesis from UML Models of Distributed Software.* in « FORTE'2002, IFIP Int. Conf. on Formal description techniques », Houston, Texas, novembre, 2002.

[35] H. W. SCHMIDT, R. H. REUSSNER. *Parameterised Contracts and Adaptor Synthesis.* in « Proc. 5th Intl Component-Based Software Engineering Workshop (CBSE5) of the ICSE Conference, IEEE », mai, 2002.

[36] T. ZIADI, L. HÉLOUËT, J.-M. JÉZÉQUEL. *Modeling behaviors in Product Lines.* in « Proceedings of REPL 2002 International Workshop on Requirements Engineering for Product Lines », Essen,Germany, sep, 2002.

Rapports de recherche et publications internes

[37] B. BAUDRY, J.-M. JÉZÉQUEL, C. NEBUT. *Driver de tests .NET.* COTE deliverable, numéro 4.4, Projet RNTL, 2002.

[38] K. GEIHS, T. WEIS, J.-M. JÉZÉQUEL, N. PLOUZEAU. *Design documentation for framework, tools and aspect weavers.* rapport technique, numéro D431, The QCCS consortium, september, 2002.

[39] C. JARD, T. JENSEN, T. JÉRON, J.-M. JÉZÉQUEL, S. PICKIN, L. VAN AERTRICK, L. DU BOUSQUET, Y. LEDRU. *Etat de l'art sur la synthèse de tests.* COTE deliverable, numéro 2.2, Projet RNTL COTE, 2002.

[40] S. PICKIN, T. HEUILLARD, N. BULTEAU, E. RIOU. *Spécification du format de description des tests en UML.* COTE deliverable, numéro 2.3, Projet RNTL COTE, 2002.

[41] S. PICKIN, T. JÉRON, A. LE GUENNEC, J.-M. JÉZÉQUEL, C. JARD, Y. LE TRAON. *Outil de synthèse de comportement.* COTE deliverable, numéro 4.4, Projet RNTL COTE, 2002.

[42] N. PLOUZEAU. *Final specification document of the weaving process.* rapport technique, numéro D361, The QCCS consortium, july, 2002.

[43] N. PLOUZEAU, T. WEIS. *Description of the contract metamodel.* rapport technique, numéro D341, The QCCS consortium, december, 2001.

[44] N. PLOUZEAU, T. WEIS. *Description of standard contracts.* rapport technique, numéro D351, The QCCS consortium, january, 2002.

[45] N. PLOUZEAU, T. WEIS. *Final specification document for the extended analysis and design method.* rapport technique, numéro D331, The QCCS consortium, may, 2002.

[46] E. RIOU, N. BULTEAU, J.-M. JEZEQUEL, S. PICKIN, L. VAN AERTRYCK, L. DU BOUSQUET. *Définition de l'architecture de l'atelier de modélisation des tests abstraits.* COTE deliverable, numéro 4.2, Projet RNTL COTE, 2002.

Divers

[47] D. POLLET, D. VOJTISEK, J.-M. JÉZÉQUEL. *OCL as a Core UML Transformation Language.* WITUML 2002 Position paper, Malaga, Spain, juin, 2002, http://ctp.di.fct.unl.pt/~ja/wituml02.htm.

Bibliographie générale

[48] R. ALUR, M. YANNAKAKIS. *Model Checking of Message Sequence Charts.* in « Proc. 10th Intl. Conf. on Concurrency Theory », Springer Verlag, pages 114-129, 1999.

[49] A. ARNOLD. *Systèmes de transitions finis et sémantiques de processus communicants.* série études et recherches en informatique, Masson, 1992, 196 p..

[50] 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.

[51] 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.

[52] H. BEN-ABDALLAH, S. LEUE. *Syntactic Detection of Process Divergence and non-Local Choice in Message Sequence Charts.* in « Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'97 », série Lecture Notes in Computer Science, volume 1217, Springer-Verlag, éditeurs E. BRINKSMA., pages 259 - 274, Enschede,The Netherlands, April, 1997.

[53] J. BERSTEL. *Transductions and Context-Free-Languages.* B.G. Teubner, Stuttgart, 1979.

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

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

[56] L. CASTELLANO, G. DE MICHELIS, L. POMELLO. *Concurrency versus Interleaving : An Instructive Example.* in « BEATCS : Bulletin of the European Association for Theoretical Computer Science », volume 31, 1987.

[57] R. DEMILLO, R. LIPTON, F. SAYWARD. *Hints on Test Data Selection : Help for the Practicing Programmer.* in « IEEE Computer », 1978.

[58] D. DEVEAUX, P. FRISON, J.-M. JÉZÉQUEL. *Increase Software Trustability with Self-Testable Classes in Java.* in « Proceedings of ASWEC 2001 », IEEE CS Press, pages 3-11, Canberra, Australia, August, 2001.

[59] D. DEVEAUX, J.-M. JÉZÉQUEL, Y. LETRAON. *Self-Testable Components : from Pragmatic Tests to Design-for-Testability Methodology.* in « TOOLS Europe 1999 », IEEE Computer Society Press, june, 1999.

[60] ESAPS. *Styles, structures and views for handling commonalities and variabilities.* rapport technique, ESAPS deliverable, 2001.

[61] S. FROLUND, J. KOISTINEN. *QML : A Language for Quality of Service Specification.* 1998.

[62] J. HOLLAND. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1974.

[63] L. HÉLOUËT. *Analyse des exigences des systèmes répartirs exprimées par des langages de scénarios.* thèse de doctorat, Université de Rennes 1, Octobre, 2000.

[64] L. HÉLOUËT, C. JARD. *Conditions for synthesis of communicating automata from HMSCs.* in « 5th International Workshop on Formal Methods for Industrial Cr itical Systems (FMICS) »,

GMD FOKUS, éditeurs S. GNESI, I. SCHIEFERDECKER, A. RENNOCH., Berlin, avril, 2000, http://www.gmd.de/publications/report/0091.

[65] J.-M. JÉZÉQUEL, D. DEVEAUX, Y. LETRAON. *Reliable Objects : a Lightweight Approach Applied to Java.* in « IEEE Software », numéro 4, volume 18, July/August, 2001, pages 76-83.

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

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

[68] J.-M. JÉZÉQUEL, B. MEYER. *Design by Contract : The Lessons of Ariane.* in « Computer », numéro 1, volume 30, janvier, 1997, pages 129-130.

[69] G. KICZALES, J. LAMPING, A. MENDHEKAR, C. MAEDA, C. VIDEIRA LOPES, JEAN-MARC. LOINGTIER, J. IRWIN. *Aspect-Oriented Programming.* in « ECOOP '97 - Object-Oriented Programming 11th European Confer ence », 1997.

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

[71] *Recognizable sets of Message Sequence Charts.* série LNCS, numéro 2285, Springer-Verlag, 2002, pages 523.

[72] A. MUSCHOLL. *Matching Specifications for Message Sequence Charts.* in « Lecture Notes in Computer Science », volume 1578, 1999, pages 273-287.

[73] V. PRATT. *Modeling Concurrency with Partial Orders.* in « International journal of Parallel Programming », numéro 1, volume 15, Mai, 1986, pages 33-71.

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

[75] Y. L. TRAON, T. JÉRON, J.-M. JÉZÉQUEL, P. MOREL. *Efficient OO Integration and Regression Testing.* in « IEEE Trans. on Reliability », numéro 1, volume 49, mars, 2000, pages 12-25.

[76] H. VU, 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 », série LNCS, volume 2072, Springer, éditeurs J. L. KNUDSEN., pages 381-400, Budapest, Hungary, June, 2001.

[77] T. WEIS, C. BECKER, K. GEIHS, N. PLOUZEAU. *An UML Meta Model for Contract Aware Components.* in « Proceedings of UML 2001 », série LNCS, volume 2185, Springer Verlag, pages 442-456, 2001.

[78] E. J. WEYUKER. *Testing Component-Based Software : A Cautionary Tale.* in « IEEE Software », numéro 5, volume 1, septembre, 1998, pages 54-59.

[79] G. WINSKEL. *Event Structures.* in « Petri Nets : Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course », volume 255,

Springer-Verlag, éditeurs W. BRAUER, W. REISIG, G. ROZENBERG., pages 325-392, Bad Honnef, september, 1986.