INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet PAMPA

Modèles et outils pour la programmation des architectures parallèles réparties

Rennes

THÈME 1C

Table des matières

1 Composition de l'équipe

Responsable scientifique

Claude Jard [DR CNRS]

Assistante de projet

Marie-Noëlle Georgeault [TR Inria]

Personnel Inria

Benoît Caillaud [CR]
Thierry Jéron [CR]
Vlad Rusu [CR]

Personnel CNRS

Jean-Marc Jézéquel [CR]

Personnel Université de Rennes 1

Yves Le Traon [maître de conférences]
César Viho [maître de conférences]

Post-doctorant

Lydie Du Bousquet [post-doc Inria, depuis le 1/10/99]
Laurie Ricker [post-doc Ercim, depuis le 1/9/99]
Naohito Sato [post-doc Inria, jusqu'au 28/2/99]
Gerson Sunye [post-doc Inria, depuis le 1/9/99]

Ingénieur-experts

Michel Bourdelles [ingénieur-expert Inria]
Solopho Ramangalahy [ingénieur-expert Dyade depuis le 1/8/99]
Séverine Simon [ingénieur-expert Inria depuis le 1/2/99]

Chercheurs doctorants

Hubert Canon [bourse BDI/Dret-CNRS]
Eric Cariou [bourse MENRT, depuis le 1/10/99]
Loïc Hélouët [bourse BDI/Inria-CNRS]
Wai Ming Ho [bourse Inria]
Alain Le Guennec [bourse MENRT]
Pierre Morel [bourse MENRT]
François Pennaneac'h [bourse BDI/Inria-CNRS]
Lenaïck Tanguy [bourse université Rennes I]

Collaborateurs extérieurs

Daniel Deveaux [maître de conférences, université Bretagne Sud]

2 Présentation et objectifs généraux

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.

2.1 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 Vérilog. 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.

2.2 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

Résumé : *le projet élabore de nouvelles technologies logicielles permettant d'aider le développement des logiciels répartis. Les problèmes centraux sont la modélisation des processus et comportements, et le développement d'algorithmes associés pour raffiner la conception, générer du code ou des tests. Ces questions sont examinées dans le cadre des architectures logicielles (à objets répartis). Les techniques de validation utilisées s'appuient sur des simulations complexes des modèles considérés.*

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.

3.2 Fondements mathématiques des systèmes réactifs et répartis

Mots clés : systèmes de transitions étiquetés, ensembles partiellement ordonnés.

Résumé : *La structure mathématique qui caractérise le mieux les fondements des travaux de recherche en vérification et génération de tests de programmes répartis sont les systèmes de transitions étiquetés (*labelled transition systems *en anglais, abréviation* LTS*)* [Arn92]*. Cette structure, développée il y a près de cinquante ans est l'un des fondements de l'informatique ; aussi il nous a paru utile de préciser*

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

de quelle façon nous utilisons cette structure, notamment sa construction au vol. L'autre aspect fondamental est la notion de causalité entre événements dans les exécutions réparties. C'est le concept central qui permet de parler de l'analyse des comportements des systèmes distribués [Jar94]*. Il est aussi à la base des plus beaux résultats en algorithmique répartie.*

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

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

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

Il est usuel de parler d'automate d'états finis pour désigner un système de transitions étiqueté dont l'ensemble des états et celui des événements sont finis. Il s'agit en fait du modèle de machine le plus simple que l'on puisse imaginer. Nous employons les LTS pour modéliser 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 SDL ou Lotos, voire UML. L'association d'un LTS à un programme se fait par l'intermédiaire d'une définition opérationnelle de la sémantique du langage et est en général formalisée sous la forme d'un système de déductions. Pour un langage aussi simple qu'une algèbre de processus (CCS par exemple), la définition de sa sémantique opérationnelle tient en moins de dix axiomes et règles d'inférences ; alors que pour un langage aussi complexe que LDS, cela est plutôt l'affaire d'un document de plus de cent pages.

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

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

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

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

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

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.

Méthodes déductives Une autre approche pour combattre l'explosion combinatoire est d'utiliser des méthodes déductives. Plutôt que de coder les valeurs des variables dans l'état du LTS (ce qui revient à s'intéresser uniquement aux effets des opérations sur les variables), on va conserver les opérations symboliques directement dans la représentation du LTS. L'analyse de tels LTS symboliques revient à manipuler des formules logiques qui caractérisent des ensembles d'états: c'est le champ d'activité des méthodes déductives. Celles-ci peuvent être utilisées conjointement avec les méthodes algorithmiques, par l'intermédiaire de relations d'abstraction. Des recherches récentes ont démontré la pertinence de cette intégration entre déduction et model-checking. Des outils existants (tels que PVS développé au Stanford Research Institute) permettent de combiner de manière efficace ces approches complémentaires [ORSvH95].

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

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

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

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

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

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

[ORSvH95] S. Owre, J. Rushby, N. Shankar, F. von Henke, « Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS », *IEEE Transactions on Software Engineering 21*, 2, février 1995, p. 107–125.

[Lam78] L. Lamport, « Time, clocks and the ordering of events in a distributed system », *Communications of the ACM 21*, 7, July 1978, p. 558–565.

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

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

3.3 Génération automatique de tests

Mots clés : Test de conformité, spécification, implantation sous test (IUT), cas de test, objectif de test, point de contrôle et d'observation (PCO), système de transitions à entrées sorties (IOLTS).

Résumé : *Le test de conformité est un test de type boîte noire. On se donne une spécification d'un système ouvert qui sert de modèle de référence et une implémentation réelle de ce système dont on ne connaît le comportement que par ses interactions avec l'environnement. Un test consiste à alterner le contrôle de l'*IUT *et son observation par un testeur et en déduire un verdict sur la conformité de l'*IUT *par rapport à sa spécification en fonction d'une relation de conformité. La génération automatique de test consiste à produire automatiquement des cas de tests en fonction de la spécification et de la relation de conformité choisie. La sélection d'un ensemble significatif de cas de tests peut se faire en utilisant des objectifs de test comme cela se fait lors de l'écriture manuelle des tests. Nous abordons la génération de tests par divers moyens dont les fondements sont l'algorithmique des graphes mais aussi les manipulations symboliques et la preuve.*

Modèles

Le modèle de base permettant la modélisation des objets relatifs au test de conformité est un modèle dérivé des systèmes de transition, que l'on appellera *système de transitions à entrée sortie* (IOLTS). Il distingue deux types d'actions, les actions internes inobservables, les actions observables elles-mêmes séparées en entrées et sorties, permettant ainsi de modéliser l'observation et le contrôle.

Un IOLTS est un système de transitions où l'alphabet est décomposé en $A = \{?\} \times A_I \cup \{!\} \times A_O \cup I$, A_I l'alphabet d'entrées, A_O l'alphabet de sorties, et I l'alphabet des actions internes.

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

Dans certains cas, nous enrichissons ce modèle d'états accepteurs pour leur faire jouer le rôle d'automate.

Nous utilisons aussi des modèles de plus haut niveau manipulant des variables. Les transitions sont alors étiquetées par des gardes et des actions composées d'événements (entrée, sorties, affectations).

Ces modèles sont utilisés pour modéliser les comportements des objets intervenant dans l'activité de test et sa génération. En particulier la spécification, l'implantation, les objectifs de test et les tests eux-mêmes utilisent des instances particulières de ces modèles. La génération partant de la spécification et d'un objectif de test implique des transformations de modèles tels que l'abstraction d'actions internes, la τ-réduction (encore appelée ϵ-clôture), la détermination, la minimisation qui créent des instances de sous-classes du modèle général d'IOLTS.

Une des relations qui lie les modèles d'IOLTS est la relation de conformité. La relation de conformité **ioco** choisie, définie par Jan Tretmans [Tre96] (Université de Twente) stipule qu'une IUT I est conforme à la spécification S si après toute trace (séquence d'actions observables y compris les blocages) de la spécification, les sorties possibles de l'IUT (y compris les blocages) sont incluses dans celles de la spécification.

Algorithmique à la volée

La génération de test implémentée dans l'outil TGV utilise une algorithmique de graphe, en particulier des algorithmes de parcours. Ces parcours font aussi de la construction: le graphe parcouru n'est pas connu a priori mais construit pendant le parcours de manière paresseuse. Ceci évite la construction de certaines parties du graphe: c'est ce qu'on appelle la génération à la volée. De plus ces algorithmes font aussi de la synthèse de sous-graphes.

TGV utilise plusieurs parcours en profondeur à plusieurs niveaux dans l'outil et en particulier des adaptations d'un algorithme de Tarjan [Tar72]. Cet algorithme permet de calculer les composantes fortement connexes maximales (CFCs) d'un graphe dirigé. L'algorithme original est récursif mais pour des raisons d'efficacité, nous utilisons une version itérative. Sa complexité est linéaire en temps et en mémoire.

Par définition, les CFCs sont des classes d'équivalence pour la relation d'accessibilité et de co-accessibilité: deux sommets u et v sont dans la même CFC si v est accessible depuis u et u accessible depuis v. On peut aussi remarquer que les CFCs peuvent se caractériser par une autre relation: depuis tous les sommets d'une même composante on peut atteindre exactement les mêmes ensembles de sommets. Cette caractérisation permet de se rendre compte de l'utilité du calcul de CFCs pour la vérification de propriétés d'accessibilité.

C'est cette idée qui est utilisée pour adapter l'algorithme de Tarjan à la génération de tests car justement plusieurs problèmes se réduisent à des problèmes d'accessibilité. La première adaptation est utilisée pour calculer une τ-réduction. Le problème se traduit en l'accessibilité aux transitions visibles. De plus les boucles d'actions internes τ sont caractéristiques des divergences pour les LTS finis et ces divergences sont considérées comme observables dans la

[Tre96] J. TRETMANS, « Test Generation with Inputs, Outputs and Repetitive Quiescence », *Software - Concepts and Tools 17*, 1996.

[Tar72] R. TARJAN, « Depth-first search and linear graph algorithms », *SIAM Journal of Computing 1*, 2, juin 1972, p. 146–160.

relation de conformité donc par un test. La seconde adaptation est utilisée dans l'algorithme principal de TGV. Ici, le problème est celui de l'accessibilité à un ensemble d'états puisque, pour schématiser, on veut calculer un sous-graphe du produit entre spécification et objectif de test contenant l'ensemble des états (ou seulement une partie) depuis lesquels les états accepteurs de l'objectif de test sont accessibles. Enfin, une troisième adaptation est utilisée pour extraire du sous-graphe précédant une partie dite *contrôlable* c'est à dire n'ayant jamais de choix entre une action contrôlable et d'autres actions (contrôlables ou non). De façon moins immédiate, ceci peut se voir aussi comme un problème d'accessibilité à l'état initial sur le graphe où on a inversé la relation de transition et modifié celle-ci au vol par coupure des conflits de contrôlabilité.

L'algorithme de Tarjan a aussi été adapté par F. Bourdoncle dans le contexte de l'analyse statique. L'idée de son algorithme est d'appliquer récursivement la décomposition en CFCs en enlevant à chaque récursion les racines de CFCs (la racine d'une CFC est le premier sommet atteint dans le parcours). L'algorithme se termine quand il n'existe plus de cycle. Ceci permet alors de décomposer complètement un graphe dirigé quelconque en un ordre topologique faible. La complexité est alors quadratique en temps et en mémoire. Nous avons adapté cet algorithme pour calculer un ordre pour le test d'intégration de composants objet. L'adaptation consiste à modifier le critère de choix du sommet à retirer à chaque itération sans modifier la complexité de l'algorithme: le critère de choix doit donc être calculé en temps constant.

Un des problèmes fondamentaux de la génération de tests est celui de l'explosion combinatoire du graphe d'états de la spécification. Le même problème apparaît dans le cadre de la vérification par modèle et une des techniques utilisées pour l'éviter est le calcul à la volée. Le principe est de vérifier les propriétés pendant la construction par un parcours en profondeur du graphe d'états de la spécification. Quand une violation de la propriété est détectée, la séquence courante donne un contre-exemple. En génération de test, le problème est à la fois plus compliqué car il faut abstraire les actions internes donc considérer plusieurs niveaux de graphes d'états simultanément, et plus simple car les propriétés (ici les objectifs de test) sont moins expressifs. Nous avons donc adapté l'idée de la vérification à la volée à la génération de test. Ceci nécessite un découpage de l'architecture logicielle par des couches utilisant et fournissant des Api composées de fonctions nécessaires au parcours de graphes. La génération de tests à la volée peut alors se voir comme une construction paresseuse du graphe d'état de la spécification et des graphes d'états intermédiaires guidée par l'objectif de test. Même si cela ne change pas la complexité théorique, l'efficacité est nettement meilleure en pratique.

3.4 Technologie objet dans un contexte de génie logiciel réparti

Mots clés : objets, composants logiciels, motifs de conception, "framework".

L'approche objet pour le génie logiciel

L'approche objet est aujourd'hui devenue incontournable pour l'analyse, la conception et la réalisation des grands systèmes d'information devant évoluer sur de longues périodes de temps, et pour lesquels l'effort principal en termes de logiciel (parfois jusqu'à 80% ou plus) est consa-

cré à la maintenance [Mey88]. Fondée sur la notion d'objets, c'est-à-dire d'instances de classes modélisant des types d'entités stables d'un système d'information en tant que modules autonomes organisés selon des relations d'héritage (classification) et d'utilisation, cette approche permet en effet de gérer la nature fondamentalement incrémentale, itérative et évolutive du développement de tels logiciels [Jac85,Boo94]. Les diverses phases d'analyse, de conception et de réalisation utilisent le même cadre conceptuel (fondé sur cette notion d'objet) et n'ont pas de frontières rigides entre elles, ce qui fait que le processus de développement objet est parfois qualifié de *continu* [Jéz96]. 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.

La notion d'objet fournit le substrat nécessaire au développement du concept de *composant logiciel*, vu ici comme unité de déploiement. 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 l'idée traditionnelle de la réalisation d'économies d'échelle dans le développement de logiciels en réutilisant des composants plutôt qu'en redéveloppant les applications à partir de zéro, il s'agit aussi aujourd'hui d'offrir la possibilité de modifier radicalement le comportement et les fonctionalités d'une application par substitution ou ajout de composants, même longtemps après son déploiement. La question n'est plus de développer une application répondant à un besoin précis à un moment donné, mais plutôt de construire un canevas d'application, ou "framework", qui va permettre de décliner une gamme d'applications répondant à une famille de besoins évoluant dans le temps.

Un *framework* fournit un ensemble intégré de fonctionnalités spécifiques à un domaine, implanté par une collection de classes liées entre elles par de multiples schémas (*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

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

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

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

[Jéz96] J.-M. Jézéquel, *Object Oriented Software Engineering with Eiffel*, Addison-Wesley, mars 1996, ISBN 1-201-63381-7.

l'utilisateur du framework.

L'utilisation de frameworks et de composants 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 ("frameworks")
- conception de composants utilisables comme unités de déploiement
- validation et assemblages de composants (d'origines diverses)
- gestion des composants (maintenance)

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

Parmi les défis à relever, le besoin de modèles d'assemblages de composants fondés formellement, ainsi que celui d'une qualité vérifiable se signalent tout particulièrement.

Si certains éléments de réponses à ces problèmes existent déjà dans des prototypes de laboratoire, l'adoption massive d'UML (Unified Modeling Language) 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 (OMTO, Booch, etc.) qui ne définissaient qu'une syntaxe graphique, UML est partiellement formalisé et dispose d'un sous-langage sophistiqué de description de contraintes appelé OCL (*Object Constraint Language*). Celui-ci offre la possibilité de définir des familles d'interprétations sémantiques permettant de manipuler formellement des modèles de composants, de frameworks et d'applications.

4 Domaines d'applications

Mots clés : Télécommunications, génie logiciel, test, UML, SDL.

Résumé : *le secteur des télécommunications est en grande expansion, avec la mise en place d'infrastructures mondiales, l'explosion des télécommunications mobiles et le développement de nouveaux services. Le contexte industriel français et européen est par ailleurs assez favorable. Du point de vue du logiciel, la pression est grande pour augmenter la généricité des solutions proposées (aspect méthode) tout en raccourcissant les délais de mise au point (aspect outils). Le projet Pampa, spécialiste de l'ingénierie des protocoles, trouve là un terrain privilégié d'applications. Ceci à condition de ne pas manquer des évolutions majeures du domaine qui sont : la nécessité de considérer globalement le cycle de développement du logiciel, notamment dans le cadre des méthodes de conception objet ; et l'intérêt croissant pour les protocoles des couches hautes exprimées sous la forme de services de communication ou de gestion (avec des données nécessitant des traitements symboliques).*

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

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 validation de conceptions UML*, la génération de tests de conformité pour les protocoles, la conception d'un langage d'interface pour objets communicants, et le suivi de pannes dans les réseaux.*

4.1 Logiciels pour les télécommunications

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

Conception fiable de logiciels communicants :

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

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

Test et diagnostic des systèmes communicants :

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. La surveillance et le diagnostic sont aussi des aspects du travail sur les implantations.

Le projet Pampa s'est focalisé sur la génération automatique de tests de conformité à partir de spécifications formelles, ainsi que sur le diagnostic en environnement réparti.

L'obtention d'une bonne suite de tests de conformité est d'un intérêt économique certain. C'est à l'heure actuelle un travail manuel coûteux et répétitif qui est ensuite utilisé à grande échelle. Nous participons au défi de l'automatisation en mettant particulièrement l'accent sur la qualité de la suite de tests (dans sa capacité à détecter les implantations non conformes et uniquement celles-ci). Sur plusieurs études de cas, nous avons montré la rentabilité de l'approche et le transfert industriel de notre outil TGV en est l'exemple. La principale difficulté non résolue actuellement est le test réparti (abordé souvent par le biais de l'interopérabilité); elle constitue une perspective importante de recherche pour le projet. Un premier travail sur

ce sujet nous a permis de proposer un schéma de distribution d'un testeur séquentiel en un ensemble de testeurs répartis ayant le même pouvoir de détection que le test original.

Cette question de test réparti apparaît d'ailleurs dans nos études sur la surveillance des comportements des réseaux. En effet, les questions de gestion des réseaux de plus en plus complexes sont d'une grande actualité. Beaucoup d'études sont menées sur l'architecture des protocoles et des services pour faire de la gestion. Un effort moins important est consacré à la conception d'algorithmes permettant d'offrir de tels services. Nos travaux sur la corrélation causale d'alarmes vont dans ce sens, avec la perspective principale de distribuer sur les capteurs la fonction de surveillance (notion de capteurs intelligents communicants que l'on souhaite développer dans le domaine des télécommunications).

5 Logiciels

5.1 TGV : un outil de génération automatique de tests de conformité

Participants : Thierry Jéron, Pierre Morel, Séverine Simon, César Viho, Claude Jard, Vlad Rusu.

Mots clés : Test de conformité, TGV, ObjectGéode, CADP, SDL, Lotos, TTCN.

Résumé :

TGV *(Test Generation with Verification technology) est un prototype de générateur automatique de tests de conformité à partir de spécifications formelles (*SDL *ou Lotos). Son originalité tient dans son algorithmique à la volée adaptée du domaine de la vérification par modèle (Model Checking).* TGV *est issu d'un projet commun avec le laboratoire Verimag Grenoble. Il utilise des bibliothèques de la boîte à outils* CADP *de Vérimag et de l'Inria Rhône-Alpes.* TGV *a été déposé par l'Inria à l'Agence de Protection des Programmes en 97.* TGV *va bientôt être distribué gratuitement dans la boîte à outils* CADP. *Il est déjà utilisé par les membres du projet sur plusieurs études de cas et aussi par quelques utilisateurs extérieurs. Un effort important a été fourni pour le transfert industriel des algorithmes de* TGV *dans l'outil ObjectGéode de Vérilog.*

Une première version de TGV a vu le jour en 95 lors d'un contrat financé par le STEI regroupant Vérilog, Cap Sesa Région Rennes, le Cnet Lannion et le Celar de Bruz, Vérimag et Pampa. Cette première version développée par Vérimag et Pampa était utilisable sur les graphes d'états de spécifications LDS produits par Géode.

Depuis cette première version, TGV a été constamment amélioré du point de vue algorithmique mais aussi par l'interfaçage avec de nouveaux environnements. TGV génère des tests à la volée (sans construire complètement le graphe d'état de la spécification), à la fois sur des spécifications LDS grâce à sa connexion à ObjectGéode, et sur des spécifications Lotos grâce à sa connexion à l'environnement CADP. TGV peut aussi fonctionner sur des graphes explicites décrits au format Aldébaran ou dans le format compressé BCG. En sortie, TGV construit un cas de test dans un format général (Aldébaran ou BCG) qui peut être traduit dans le langage TTCN, langage de description de tests standard de fait dans le monde des télécoms.

TGV est donc relativement indépendant des langages de spécification. Cette indépendance est obtenue par son architecture en couches par l'intermédiaire d'Api fournissant les fonctions de parcours de systèmes de transitions intermédiaires. D'autre part, ces différentes couches utilisent des bibliothèques de stockage de graphes de CADP. La première couche, la seule qui soit spécifique au langage d'entrée, permet de connecter TGV soit à l'Api du simulateur ObjectGéode pour LDS, soit à l'Api fournie par le compilateur Lotos Caesar de la boîte à outils CADP, soit encore aux Api de parcours de graphes explicites. Cette couche fournit une Api de parcours du graphe d'état de la spécification S. Une deuxième couche utilise cette Api et un objectif de test pour fournir l'Api du produit synchrone $PS = TP \times S$ par étiquetage des états de la spécification par les états correspondants de l'objectif (en particulier les états accepteurs de l'objectif). La troisième couche permet le renommage et le masquage d'actions internes fournissant une Api sur l'IOLTS PS_{abs} dans lequel toutes les actions internes sont indifférenciées par τ. La quatrième effectue une réduction des actions internes et une déterminisation et fournit les primitives de parcours du système de transition résultant PS_{vis} ainsi que ses états accepteurs. Enfin la dernière couche implante l'algorithme central qui effectue un parcours de PS_{vis} et synthétise un cas de test TC. Ce dernier algorithme peut être vu comme une extension d'un algorithme de model checking qui synthétise le sous-graphe de tous (ou seulement une partie contrôlable) les comportements menant aux états accepteurs de PS_{vis}, i.e. les témoins de la validité de la propriété exprimée par l'objectif.

En 99, TGV a bénéficié d'un lifting complet en vue de sa distribution. L'architecture de TGV a été repensée complètement pour correspondre à la découpe fonctionnelle et permettre une plus grande modularité. Les fonctionnalités ont été améliorées, en particulier pour permettre une plus grande expressivité des objectifs de tests, des fichiers de masquage et de renommage, en généralisant l'utilisation des expressions régulières. Le calcul des temporisateurs a été simplifié et incorporé au cœur de TGV. Le calcul des postambules a été affiné par la notion d'état stable. L'outil prend maintenant en compte le format de graphe BCG à la fois en entrée et en sortie. Un nouveau module appelé VTS et pouvant se substituer au module principal de TGV permet maintenant de vérifier et corriger à la volée des cas de tests. Ceci peut servir à la correction de tests manuels mais sert aussi à tester TGV par la vérification des cas de tests produits. Enfin en plus de la version Solaris, il existe maintenant une version Linux et une version NT sera prochainement disponible.

Une opération de transfert industriel de TGV avec la société Vérilog et le soutien de France Telecom a débuté en 98 et s'est terminée courant 99 [24]. L'objet en était l'implantation des algorithmes de TGV et son architecture dans l'outil ObjectGéode. Une bonne part des développements relatifs à TGV ont été effectués dans le projet. La version commerciale devrait être disponible avant fin 99.

Les différentes études de cas dans lesquelles TGV est utilisé nous permettent constamment d'enrichir et d'améliorer TGV sur différents aspects : l'expression des objectifs de test, de l'architecture de test, l'algorithmique de génération de test, de vérification de tests, la connexion avec d'autres outils.

5.2 UMLAUT : un outil de manipulation de modèles UML

Participants : Jean-Marc Jézéquel, Wai Ming Ho, Alain Le Guennec, François

Pennaneac'h, Gerson Sunye, Michel Bourdelles, Séverine Simon, Benoît Caillaud, Yves Le Traon.

Mots clés : UML, composants, patterns, validation.

Résumé :

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

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

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

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

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

L'outil UMLAUT vise à montrer la validité d'une telle approche. Cet outil intègre le méta-modèle UML et fournit un ensemble de facilités permettant de raisonner sur des modélisations UML. Il s'agit en fait d'un véritable framework de transformation de modèles permettant d'appliquer des manipulations complexes à un modèle UML [27]. Ces manipulations sont exprimées comme des compositions algébriques de transformations élémentaires réifiées. Elle sont donc extensibles au travers des mécanismes classiques d'héritage et d'aggrégation.

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

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

UMLAUT intégrant le méta-modèle UML, il est possible de réaliser simplement des transformations de modèles définies par un ensemble de règles fournies par l'utilisateur. Dans le cadre du projet RNRT Oural, nous avons commencé à appliquer cette idée pour transformer le modèle UML d'une application répartie en un système de transitions étiquetées ; dans l'objectif de le valider avec des outils avancés de validation de protocoles [13].

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

6 Résultats nouveaux

6.1 Test de conformité et d'interopérabilité

Participants : Thierry Jéron, Claude Jard, Pierre Morel, Vlad Rusu, Séverine Simon, Lénaïck Tanguy, César Viho.

Mots clés : Test de conformité, test d'interopérabilité, tests abstraits, tests exécutables, tests symboliques, abstraction, déterminisation, génération à la volée, vérification, objectifs de test, preuve.

Résumé : *L'activité relative au test de conformité revêt deux aspects: la génération automatique de tests de conformité qui produit des cas de tests abstraits et l'implantation de ces tests abstraits en tests exécutables. Concernant la génération de tests, nous continuons notre activité pour améliorer les tests produits, faciliter leur sélection, prendre en compte différentes architectures de test. En 99 nous avons continué notre travail sur l'algorithmique à la volée pour l'amélioration de l'efficacité et de la qualité des cas de test générés. Nous avons aussi conçu un algorithme permettant la vérification et la correction des tests. Un travail sur la génération et la vérification de tests symboliques par l'utilisation d'abstraction et de preuve a débuté. Un travail sur l'application des techniques d'observation répartie en vue d'améliorer la qualité des tests a aussi été démarré. Sur l'exécution des tests*

un outil prototype appelé PIXIT a été développé et nous sert de plate-forme de validation expérimentale des méthodes que nous développons.

6.1.1 Génération et vérification de tests

Algorithmique à la volée Nous avons conçu et implanté dans l'outil TGV un algorithme à la volée permettant de produire des tests en utilisant des techniques de model checking [30]. Le problème consiste à produire le sous-graphe de toutes les séquences menant dans un état accepteur d'un graphe obtenu par produit synchrone entre spécification et objectif de test, puis abstraction des actions internes et déterminisation. Le problème peut aussi être vu comme un problème de model-checking de propriété d'accessibilité. Le sous-graphe recherché est quasiment l'ensemble de toutes les séquences témoins de la validité de la propriété. L'algorithme est une adaptation de l'agorithme de Tarjan permettant le calcul des composantes fortement connexes maximales d'un graphe. Le sous-graphe obtenu n'est a priori pas contrôlable (il peut exister des choix entre actions contrôlables et observables) et n'est donc pas exactement un test. Mais il peut permettre de dériver tous les tests correspondant à un objectif. Si on veut produire un test, il faut couper le sous-graphe quand la condition de contrôlabilité est violée. Certaines coupures peuvent être effectuées pendant l'algorithme de Tarjan mais pas toutes. Les choix non-contrôlables résiduels doivent être éliminés par un deuxième parcours effectué en arrière sur le sous-graphe résultant obtenu. Là encore l'algorithme de Tarjan peut être utilisé. Comme effet de bord, l'algorithme permet de produire moins de cas inconcluants. On peut même montrer que pour un objectif de test donné, ils sont maximaux parmi l'ensemble des tests valides au sens où tout autre test valide pour cet objectif produit plus de cas non concluants. Cet algorithme a été intégré dans TGV et expérimenté sur plusieurs études de cas.

Un problème voisin de la génération de tests est celui de la vérification des tests. Il s'agit de vérifier qu'un test (écrit à la main par exemple) est non biaisé c'est à dire qu'il ne rejette aucune implantation conforme. Le non-biais peut se définir par une relation entre spécification et cas de test et se vérifie par un parcours de leur produit. On en profite pour vérifier aussi qu'il n'est pas trop laxiste c'est à dire qu'il n'accepte pas d'implantations qui pourraient être rejetées par des séquences de ce test. L'algorithme vérifie donc le non-biais et le non-laxisme et si besoin corrige le test à la volée. L'outil VTS (Verification of Test Suite) fait partie de TGV. Il a été utilisé pour le test de TGV en vérifiant la validité des tests générés. Nous avons aussi utilisé cet outil pour vérifier la suite de test du protocole ATM SSCOP fournie par l'ATM Forum, ce qui nous a permis d'identifier quelques erreurs.

Test et théorie des jeux Un travail en collaboration avec Pascale Le Gall (Université d'Evry) et Solofo Ramangalahy (auparavant au CEA Saclay et Université Evry) sur l'utilisation de la théorie des jeux pour la génération de tests a débuté en 98 et s'est concrétisé en 99 par deux publications [33, 16]. La modélisation de l'activité de test par un jeu entre le testeur et l'implantation, permet de voir la génération de test comme la recherche d'une stratégie gagnante, le but du jeu étant pour le testeur de contrôler le plus possible l'implantation pour l'amener à des erreurs ou à des situations particulières décrites par un objectif de test. Cette vision du test mène alors naturellement à regarder les rapports entre synthèse de contrôleur et génération de test, travail entrepris avec le projet EPATR.

Manipulations symboliques Les algorithmes de TGV sont basés sur des techniques énumératives. Lorsque, par exemple, la spécification d'un protocole contient des paramètres et/ou variables (paramètres de messages, compteurs, files d'attente...), TGV instancie toutes ces variables, qui seront traitées en "multipliant" les états de contrôle à explorer par le nombre de valeurs distinctes que les variables peuvent prendre. Ceci peut mener à l'explosion combinatoire ou même à la non-terminaison. Pour traiter ce problème, nous envisageons d'utiliser des techniques symboliques dans la génération de tests. Comme prélude à ce travail, nous étudions actuellement l'application des techniques symboliques à la *vérification* de cas de test écrits "à la main" (un problème plus simple que la génération). Ce problème est intéressant en soi: il existe, en pratique, des suites de tests "commerciales", (produites le plus souvent sans démarche formelle) dans lesquelles notre approche permettrait de détecter systématiquement les erreurs.

Nous avons défini une notion de correction d'un cas de test par rapport à une spécification et un objectif donnés. Cette définition fait intervenir des relations de simulation entre des systèmes de transitions symboliques. Nous caractérisons (de manière équivalente) ces relations en terme de propriétés de sûreté dans des produits de systèmes de transitions, et utilisons le prouveur PVS pour vérifier intéractivement ces propriétés. Cette approche permettra de prendre avantage, d'une part, de l'automatisation poussée de PVS, et d'autre part d'utiliser les méthodes basées sur l'intégration de model-checking, preuves et abstractions, qui donnent des bons résultats en vérification [34].

6.1.2 Exécution des tests

Il s'agit ici de compléter la chaîne de génération de tests pour produire des tests exécutables pour une architecture de test donnée. Au delà du test de conformité, notre objectif est la maîtrise du test d'interopérabilité entre IUT. Dans ce contexte, les questions de l'asynchronisme entre le testeur et l'IUT, et la répartition du tests sur plusieurs testeurs deviennent cruciales.

Génération de test réparti Nous travaillons sur une approche qui consiste à dériver automatiquement des testeurs répartis à partir de test centralisé. Nous avons proposé un schéma de distribution permettant de produire un ensemble de tests communiquant entre eux de manière asynchrone et s'appuyant sur un service global de consensus permettant de réaliser des choix non locaux. Notre travail actuel porte sur un raffinement de cette méthode par la synthèse automatique des procédures de coordination efficaces entre les testeurs.

Prise en compte de l'asynchronisme L'architecture réelle sur laquelle les tests répartis sont effectués a une importance évidente pour la génération de tests. Ainsi, une IUT dans son contexte est conforme à une spécification dans le même contexte si la composition de l'IUT avec son contexte est conforme (hors contexte) avec la spécification dans le même contexte. Pour une spécification donnée, on constate malheureusement que les ensembles d'IUT conformes dans et hors contexte ne correspondent pas : le contexte rend le test plus laxiste (des IUT non conformes deviennent conformes dans un contexte) mais ce qui est plus génant, des IUT conformes deviennent non conformes dans un contexte. Nous avons cependant montré que le deuxième problème pouvait être évité en rendant (ou considérant) les spécifications complètes en entrées et pour des contextes monotones (préservant l'inclusion de trace). Dans le cas d'un

contexte asynchrone réduit à un PCO (une file Fifo dans chaque sens), nous avons aussi montré qu'en instrumentant l'IUT d'un mécanisme simple de comptage d'événements observables, on pouvait également supprimer le problème du laxisme. En résumé, nous avons montré que le test distant asynchrone avec un PCO avait la même puissance d'observation (le contrôle est quand même réduit) que le test synchrone. Nous proposons deux techniques de génération/exécution de tests. La première consiste à générer les tests à partir de la spécification instrumentée et transformée par le contexte asynchrone. La deuxième consiste à générer les tests à partir de la spécification hors contexte et de les exécuter sur l'IUT instrumentée en utilisant un «driver» de test qui pilote le test et décode l'ordre d'occurrence des événements de l'IUT [28].

Exécution des tests Les cas de tests produits par nos algorithmes sont des tests abstraits. Pour être effectivement exécutés sur une IUT, il faut une phase de traduction vers les primitives d'accès à l'interface de l'IUT. Cette phase nécessite de renseigner certains paramètres du test comme la valeur effective des temporisateurs, les adresses effectives, etc. Pour maîtriser toute la chaîne de la génération à l'exécution des tests répartis, il convient d'étudier en quoi les problèmes rencontrés dans cette phase peuvent nous aider améliorer les tests que nous produisons. Nous commençons à regarder de plus près cet aspect d'un point de vue expérimental. En 1998, nous avions développé un prototype permettant de traduire les tests abstraits au format Aldebaran en tests exécutables pour une architecture de test distant. Cette année, nous avons rajouté la possibilité de simuler des IUTs par compilation de spécifications formelles (également au format Aldebaran).

Sur cette plateforme d'exécution des tests appelée $\mathcal{PIXIT}$, nous étudions l'instrumentation qui peut être faite du côté des testeurs que des IUTs pour affiner les observations et améliorer le contrôle.

6.2 Conception fiable à base de composants logiciels

Participants : Jean-Marc Jézéquel, Claude Jard, Benoît Caillaud, Yves Le Traon, Daniel Deveaux, Michel Bourdelles, François Pennaneach, Alain Le Guennec, Wai Ming Ho, Hubert Canon, Eric Cariou, Loïc Hélouët.

Mots clés : télécom, objets, composants, ordres partiels, concurrence, UML, MSC.

Résumé : *L'aspect central que nous étudions ici est la construction fiable et économiquement efficace d'applications par assemblage de composants logiciels, en particulier dans le domaine des systèmes répartis et réactifs ayant des temps de réponse statistiquement contraints ("temps réel mou"), ce qui est généralement le cas dans le domaine des télécoms. Nos travaux sur ce thème ont pour ambition de construire des modèles et des outils donnant à un concepteur de logiciel un certain niveau de confiance dans la fiabilité d'assemblages de composants pouvant provenir de sources tierces. Il s'agit notamment d'étudier des modèles permettant la spécification de propriétés à la fois fonctionnelles et non-fonctionnelles de composants devant être déployés sur des systèmes répartis, et de bâtir un continuum d'outils tirant partie de ces éléments de spécification, allant de vérificateurs hors-ligne à*

*des moniteurs supervisant en ligne le comportement des composants d'une application répartie. Au cœur du travail se situe la définition d'un modèle sémantique pour les vues comportementales d'*UML *et son utilisation pour intégrer des outils de validation. La recherche s'est concrétisée par l'utilisation du meta-modèle* UML *et la conception du modèle* BDL*, fondé sur des ordres partiels synchronisés.*

Manipulations de modèles UML

La principale particularité de la notation UML réside dans sa base formelle appelée *méta modèle* : la sémantique des concepts manipulés par la notation UML est décrite en utilisant la notation elle-même. Bien que cette approche soit source d'imprécisions dans la sémantique d'UML que nous nous efforçons par ailleurs de clarifier au travers de notre participation au groupe de travail *p*UML (precise UML), elle permet de développer des outils manipulant des modèles UML, qui ont la particularité de pouvoir aussi manipuler le méta-modèle (perspectives de «bootstrapping» complet d'un outil). Avec l'outil UMLAUT, qui fournit un ensemble de facilités permettant de raisonner sur des modélisations UML, nous avons démontré la possibilité d'utiliser de techniques formelles à des fins de vérification/validation [18], les éléments d'un modèle devant respecter les propriétés et contraintes définies dans le méta-modèle.

Nous l'avons conçu comme un véritable framework de transformation de modèles permettant d'appliquer des manipulations complexes à un modèle UML. Ces manipulations sont exprimées comme des compositions algébriques de transformations élémentaires réifiées. Elle sont donc extensibles au travers des mécanismes classiques d'héritage et d'agrégation.

Nous avons montré en particulier comment le modèle UML d'une application répartie peut être automatiquement transformé en un système de transitions étiquetées [27] ; dans l'objectif de le valider avec des outils avancés de validation de protocoles (e.g. TGV).

Patrons de conception et contrats

Il s'agit ici d'une recherche visant à découvrir, mettre en forme et proposer des pistes d'exploitation de motifs de conceptions (design patterns), en particulier dans le domaine des systèmes répartis et réactifs ayant des temps de réponse statistiquement contraints (e.g. télécom) [31, 14, 37].

Nous avons en particulier montré comment la notion de contrat logiciel entre composants peut être mise à profit pour contraindre l'application de motifs de conception [8], avec une étude poussée dans le domaine de la gestion de configuration de produit logiciels télécom [15].

Nous avons aussi montré comment la notion classique de contrat de conception portant sur des proriétés statiques et fonctionnelle pouvait être étendue à la description de propriétés non-fonctionnelles telles que contraintes de synchronisation ou qualité de service [10]. Nous recherchons maintenant comment intégrer au mieux ces notions dans des environnements de travail fondés sur UML.

Composants autotestables

Le succès de l'approche objet tient beaucoup de la notion de composant réutilisable. La réutilisation de composants logiciels devient un enjeu crucial tant pour éviter de réécrire inuti-

lement du logiciel, que pour réduire les coûts de développement. Un axe de recherche émergent vise donc à donner les moyens de développer des composants logiciels auxquels on puisse associer un degré de confiance afin de pouvoir les réutiliser (notion de «trusted components»).

Une contribution étudiée dans le cadre du projet Pampa consiste à considérer chaque objet comme un tout possédant en particulier la capacité de se tester et de tester les composants dont il dépend. Les tests sont donc considérés comme faisant partie du composant. Ceci permet de fournir une première contribution à la solution du problème de la réutilisation et de l'évolution du composant puisqu'il porte la capacité à se tester (autotest) qui peut être appelée depuis le système [23, 22].

L'approche développée apporte donc des éléments de solutions structurelles au problème du test d'intégration, du test de non-régression, de l'évolution des composants et de leur réutilisation. Elle a pour principale limitation le fait qu'elle ne permet pas le test fonctionnel du système (test système et validation). Pour ce dernier aspect, il faudrait prendre en compte de manière explicite les aspects dynamiques (aspects de communication entre objets) des modèles et proposer des méthodes de test spécifiques.

L'un des aspects de la conception de composants autotestables et réutilisables est celui de la qualification des tests. Il faut en effet indiquer avec quelle qualité les composants ont été testés, ce qui peut se faire à l'aide d'une analyse de mutation. 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.

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.

A 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 [29].

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 [18].

Langages de scénarios

L'usage de scénarios pour spécifier les comportements de systèmes est une pratique bien établie. Des notations formelles ont été proposées comme les "Message Sequence Charts" (MSC), principalement utilisés pour décrire l'activité de processus communicants dans le domaine des protocoles. On les retrouve sous une forme proche, les diagrammes de séquence, dans une des vues définies par la notation UML. L'inconvénient majeur de ce type de notation est la multiplication des différents scénarios que l'on est amené à décrire pour représenter le plus complètement possible l'activité d'un système. L'idée naturelle consiste à construire des familles de scénarios en utilisant une sorte d'automate de scénarios. La proposition la plus élaborée a été faite dans le contexte des MSC et est en cours de normalisation sous l'appellation "High level MSC" ou HMSC. Cette notation s'avère assez puissante. Nous essayons de contribuer à lui donner le statut de méthode formelle, permettant :

- la vérification de propriétés : par exemple, y a-t-il possibilité de divergence (accumulation indéfinie de messages dans le milieu de transmission) ? Dans l'objectif d'une implantation répartie, y a-t-il des choix non locaux qui nécessiteront des synchronisations non spécifiées entre processus ?
- la génération automatique de squelettes de programmes pour lesquels on pourra garantir que leur exécution respectera les dépendances spécifiées dans les scénarios.

Nous avons défini une sémantique d'ordre partiel pour des familles de scénarios représentés par des HMSC. Celle-ci permet d'associer à chaque HMSC une structure d'événements engendrable par une grammaire de graphes finie [26]. On peut alors décider effectivement l'équivalence (l'isomorphisme des structures d'événements) de deux HMSC. Cet algorithme est implanté et sera interfacé à UMLAUT pour permettre la manipulation formelle de diagrammes de séquences UML.

La représentation choisie pour les HMSC nous a permis de construire un environnement de simulation, utilisé dans le cadre de l'étude des interactions de services dans les réseaux intelligents. Une interaction de service se produit lorsque les requêtes d'un ou plusieurs clients ne peuvent être satisfaites sans contredire l'une des exigences d'un des clients. Le réseau intelligent et les services offerts aux clients sont modélisés par des HMSC, puis simulés.

La seule étude des causalités dans les HMSC n'est pas suffisante pour affirmer qu'une spécification répond bien aux exigences. En effet, assurer qu'une spécification peut effectuer une séquence d'actions ne permet pas d'assurer que cette séquence peut être effectuée dans un délai donné. Le respect des contraintes temporelles dépend de nombreux paramètres, tels que les durées des événements, les délais de transmission de messages, les valeurs des timers, etc. En associant une durée à tout événement d'un MSC, il est possible d'adopter une approche récente basée sur l'algèbre (Max,+) aux HMSC. Les MSC sont d'abord traduits en automates d'ordres, ce qui permet de répondre immédiatement à quelques questions habituelles sur les systèmes

temporisés (durée d'une exécution donnée, le temps moyen de transfert d'un message,...). Ce travail est effectué en collaboration avec l'équipe EPATR.

BDL : un formalisme comportemental pour les objets réactifs répartis

(ce travail est mené en étroite collaboration avec l'équipe EPATR).

La notation UML comporte un grand nombre de diagrammes.

Les ateliers de génie logiciel actuels n'offrent que des passerelles limitées entre ces formalismes ; ainsi il n'est actuellement pas possible de confronter des diagrammes de séquences avec la définition en StateCharts des comportements des objets qui composent un système. De même, tout changement d'architecture ou tout raffinement de diagrammes de séquences peut avoir un impact considérable sur les spécification en StateCharts de ces objets.

La production de code exécutable est largement manuelle : le code effectif à exécuter par un composant est généralement programmé "à la main", en C, C++, ou Java. Il est inséré manuellement dans les squelettes de programme engendrés. De même, les communications ou appels de procédure entre composants sont réalisés par appels à une bibliothèque de services de communication fournis par un ORB support. Il est de la responsabilité du programmeur de s'assurer de la correction de ces appels bibliothèque vis-à-vis du service attendu. Un formalisme "colonne vertébrale" pouvant servir de pivot entre les vues comportementales d'UML est hautement désirable ; c'est l'ambition de BDL.

BDL est un enrichissement des diagrammes de classes UML avec des concepts comportementaux. Chaque classe se présente comme un graphe orienté dont les sommets sont des événements valués. Ces graphes spécifient des contraintes d'ordonnancement entre leurs événements sommets. Une classe comporte une disjonction de tels graphes, éventuellement gardée. L'exécution d'une classe procède par réactions successives. Chaque réaction transforme l'état de la classe en réaction à l'environnement et émet lui-même des événements à destination de l'environnement. Deux classes se composent par unification de leurs réactions respectives. BDL récupère en fait la technologie développée pour les langages synchrones (Esterel, Lustre, et en l'occurrence plus particulièrement Signal), tout en offrant une notion d'abstraction appropriée. Le point nouveau, qui repose sur des résultats fondamentaux récents [19], est la possibilité d'interpréter une classe BDL indifféremment selon un mode synchrone ou asynchrone, avec la même faculté pour les communications entre classes. Ceci est à la base des techniques de génération de code système. Il est ainsi possible de donner en BDL la sémantique des MSC, des automates, des StateCharts, et de SDL.

Une première implémentation prototype de BDL est en cours de réalisation, grâce à une réutilisation d'une partie de la chaîne de compilation du langage Signal. Les outils réalisés (compilateur, simulateur, vérificateur, traducteurs MSC vers BDL et StateCharts vers BDL) seront interfacés à UMLAUT et manipuleront directement le méta-modèle (syntaxe abstraite) UML. Le simulateur et le vérificateur utiliseront l'environnement Caesar/Aldebaran (CADP).

Nous avons étudié dans [35, 19, 9] les liens entre modèles synchrones et asynchrones. Des modèles simples ont été proposés pour ces deux paradigmes, ainsi qu'un ensemble de théorèmes permettant de garantir la correction de la désynchronisation d'un programme BDL isolé ou bien d'un réseau de programmes BDL. Ces théorèmes sont assortis d'hypothèses qui constituent des obligations de preuves vérifiables sur la composition synchrone des programmes considérés. Si

ces conditions ne sont pas vérifiées, Il est possible d'adjoindre à chacun des programmes BDL, un nouveau programme synchrone permettant d'assurer la correction de la désynchronisation. Ceci peut être considéré comme étant une méthode de génération de protocoles de synchronisation en environnement asynchrone.

Application de la synthèse de réseaux de Petri à la génération de squelettes de programmes répartis

(ce travail est mené en collaboration avec l'équipe Paragraphe)

Les avancées récentes sur le problème de la synthèse de réseaux de Petri permettent maintenant le développement de techniques nouvelles en matière de synthèse de programmes répartis. Nous travaillons à la généralisation des techniques de synthèse de réseaux au modèle qui sous-tend les diagrammes de séquences : les expressions régulières d'ensembles ordonnés étiquetés.

Le problème de synthèse est de produire un réseau de Petri borné dont le graphe des marquages est isomorphe à un système de transition donné[1]. Les algorithmes de synthèse de réseaux développés dans le projet Paragraphe reposent sur le concept de régions dans les systèmes de transition et ramène la synthèse d'un réseau à la résolution d'un ensemble de programmes linéaires en nombres rationnels.

Nous avons amélioré les algorithmes de synthèse de réseaux de Petri bornés répartissables mis en œuvre dans l'outil SYNET (`http://www.irisa.fr/pampa/LOGICIELS/synet/synet.html`). L'optimisation consiste à ne considérer que les rayons extrémaux du polyèdre des régions. L'outil Synet permet la répartition d'automates réactifs en réalisant à la fois une extraction de la concurrence et le traitement de choix non localisés [20, 11]. Les domaines d'applications envisagés sont la synthèse de protocoles de communications, de squelettes de programmes répartis et de contrôleurs répartis asynchrones de systèmes à événements discrets.

Puzzle de Viterbi

(ce travail est mené en collaboration avec l'équipe Sigma2)

Poussés par la question du diagnostic en environnement réparti qui se pose régulièrement dans toutes les études que nous menons dans le projet Pampa, nous avons essayé d'utiliser les automates d'ordres comme un outil pour l'observation des comportements. Dans ce cadre, les dépendances causales attendues sont exprimées par un jeu de pièces (des motifs d'ordre de base) et doivent être confrontées aux dépendances observées. Nous avons résolu deux problèmes :

- l'évolution en ligne de l'automate conformément aux observations, sous l'hypothèse qu'à chaque observation on peut associer une transition du réseau.
- la prise en compte de "transitions cachées" qui ne sont associées à aucune observation. En particulier, ceci permet de traiter le cas d'observations incomplètes (avec des "trous").

Le travail est alimenté par l'application au suivi de pannes dans les réseaux de télécommunication (cf. module 7.1).

En 1999, nous avons spécialement étudié des extensions vers des modèles dynamiques et généralisé la méthode pour l'appliquer à d'autres formalismes que les réseaux de Petri dans l'objectif de l'application à UML dans le cadre du projet Magda.

1. Cette relation peut être affaiblie à l'égalité des langages de l'automate et du réseau.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 Diagnostic de pannes dans les réseaux de télécommunications (RNRT Magda)

Participants : Claude Jard, Laurie Ricker.

Mots clés : gestion de réseau, supervision, gestion d'alarmes, diagnostic, systèmes distribués, réseaux de Petri, HMM.

Glossaire : Gestion de réseau désigne la couche haute de gestion d'un réseau de télécommunications (surveillance, maintenance, etc).

Glossaire :HMM Hidden Markov Models (Modèles de Markov Cachés), technique consistant à inférer, à partir d'observations "bruitées", l'état interne caché d'un automate stochastique (ou chaîne de Markov). Technique très utilisée en reconnaissance de la parole, et plus récemment dans le diganostic de systèmes dynamiques discrets ou hybrides. Nous étendons cette technique aux réseaux de Petri.

Résumé : *Projet* RNRT*/Magda associant Cnet, Alcatel, Ilog, Irisa/Pampa, Sigma2 et Aïda,* LIPN *jusqu'en fin 2000.*

Cette activité est commune avec le projet Sigma2. Il s'agit de développer une approche systématique pour le diagnostic de réseaux de télécommunications, avec les objectifs suivants :

- *prendre en compte explicitement le caractère distribué des réseaux,*
- *suivre une approche "modèle", modèle sur lequel s'appuie l'algorithme de diagnostic,*
- *prendre en compte les aléas (perte d'alarme, confusions possibles,...),*
- *viser une mise en œuvre du logiciel de diagnostic qui soit répartie sur le réseau.*

Une technologie originale, fondée sur une notion nouvelle de puzzle de Viterbi sur des motifs d'ordres partiels. Cette technologie nous permet naturellement de prendre en compte les contraintes qui résultent du contexte "distribué" où nous nous situons. Pour en savoir plus, consulter le rapport de l'équipe Sigma2.

7.2 Méta-Framework pour Objets Répartis

Participants : Jean-Marc Jézéquel, Wai Ming Ho.

Mots clés : framework, design patterns, composant, systèmes distribués.

Glossaire : Framework Un *framework* fournit un ensemble intégré de fonctionnalités spécifiques à un domaine, implanté par une collection de classes liées entre elles par de multiples schémas (*patterns*) de collaboration statiques et dynamiques.

Résumé : *Contrat* CTI*-Cnet ref. 98 1B 070, mars 1998 - mars 2001*

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

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

Nous explorons depuis quelques années le domaine de la construction de frameworks d'objets répartis, avec des domaines d'application variés comme le calcul scientifique intensif, les télécommunications, ou le travail coopératif. L'objectif est d'étudier, concevoir et valider des modèles et des méthodes de construction de logiciels pour architectures distribuées par composition de composants logiciels dans un contexte de programmation par objets. Ceci nous a conduit à distinguer dans les frameworks applicatifs (ou métier) certaines classes de problèmes qui reviennent de manière répétitive lorsqu'on veut les utiliser dans un contexte distribué. Il s'agit en particulier de :

- Modèles de conception s'abstrayant des particularités de l'architecture répartie sous-jacente. Les travaux que nous avons menés autour de la notion d'*opérateurs parallèles* sont un exemple d'utilisation de ce type de modèle de conception.
- Gestion de versions et de configurations. Nous avons commencé à explorer une voie consistant à réifier la notion de variantes, et à doter les composants logiciels de moyens d'auto-configuration leur permettant de s'adapter automatiquement à des environnements matériels ou logiciels variés.
- Adaptations d'interfaces. Il s'agit de prendre en compte la sémantique du composant adapté en s'appuyant sur la notion de contrat, i.e. la spécification formelle du composant.

L'objectif général de ce projet est l'exploration de ces problèmes, et l'étude d'un modèle de conception d'architectures d'objets distribuées permettant de les régler. Il s'agit en quelque sorte d'un *Méta-Framework* encadrant le développement de frameworks d'objets métiers distribués. Les défis scientifiques à relever concernent à la fois la définition du modèle (ou plus probablement du méta-modèle) de conception d'architectures d'objets distribuées Metafor intégrant un certain nombre de patrons de conception —*Design Patterns*—, mais aussi la définition et la réalisation efficace des composants génériques nécessaires et des outils de validation associés.

7.3 Convergence SDL-UML (RNRT Convergence)

Participants : Jean-Marc Jézéquel, François Pennaneach.

Mots clés : UML, SDL, objets, protocoles.

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

LDS *(*SDL *en anglais) est un standard utilisé dans les télécommunications pour la conception de systèmes temps réel.* UML *est un nouveau standard pour l'analyse et la conception orientées objet d'applications de tous types. Des sociétés américaines*

étudient des évolutions de UML *pour le temps réel en réinventant de nombreux concepts sans tenir compte du fait que* LDS *les normalise déjà.*

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

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

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

Les objectifs de ce projet sont :

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

Le projet se déroule sur une durée de trois ans en cherchant à influencer les standards UML et LDS pendant cette période. Nos partenaires sont Verilog, Alcatel et Objectif Technologie.

7.4 FormalFame : Validation d'architectures multi-processeurs

Participants : César Viho, Solopho Ramangalahy, Claude Jard, Thierry Jéron.

Mots clés : test de hardware, protocole de cohérence de caches, génération de tests, exécution.

Résumé : *Dans le cadre de l'action Vasy/FormalFame (Validation de Systèmes) du* GIE *(Groupement d'Intérêt économique) Bull-Inria Dyade, il s'agit de fournir des méthodes et outils pour la vérification formelle et le test de protocoles de cohérence de caches pour des architectures multi-processeurs développées chez Bull. Ce travail est effectué en collaboration avec l'Inria Rhône-Alpes (avant-projet Vasy de H. Garavel).*

L'inria Rhône-Alpes s'occupe de la spécification formelle et la phase de vérification du protocole de cohérence de caches de l'architecture multi-processeurs de Bull. Pour ce qui concerne

la partie rennaise de cette action, il s'agit d'étudier la possibilité d'utiliser TGV pour la génération de tests pour le hardware. Sur la base de la spécification effectuée en Lotos (langage de spécification formelle normalisé par l'Iso) par l'Inria Rhône-Alpes, nous avons pris également en charge la génération automatique de tests abstraits et leur traduction en tests exécutables sur la plate-forme de Bull.

L'objectif initial de cette collaboration se situe très clairement dans notre volonté de confronter la méthodologie de tests de TGV à des domaines autres que les protocoles de communication ; l'idée étant d'enrichir notre méthode avec les "leçons" tirées de ces expérimentations. Une première phase nous a permis de montrer l'intérêt de la génération automatique de tests avec TGV dans le domaine des architectures hardware. Ceci nous a également permis d'enrichir TGV de fonctionnalités supplémentaires, notamment : la prise en compte de spécifications Lotos, la génération des boucles dans les tests, la génération des tests dits généralisés, etc. Notre approche de nature automatique et interactive (comparée à celle manuelle et batch utilisée habituellement dans le domaine du test hardware) a été très bien perçue par la communauté hardware de Bull [32]. Actuellement, cette approche est utilisée sur une autre architecture multi-processeur en cours de développement chez Bull.

7.5 Méthodologie de tests d'interopérabilité

Participants : César Viho, Lénaïck Tanguy, Claude Jard.

Mots clés : tests d'interopérabilité, test répartis, asynchronisme, Concurrent-TTCN, exécution de tests.

Résumé : *Contrat Université Rennes I/*KBKN *marché 98 42.561, novembre 1999 - octobre 2001*

L'objectif de cette collaboration Irisa/Pampa avec le Celar-Bruz (Centre Electronique et de l'armement) de la DGA *est de proposer un cadre méthodologique - analogue à ce qui existe pour le test de conformité - pour le test d'interopérabilité des protocoles de communication.*

Le test d'interopérabilité va au-delà du test de conformité : il vérifie l'interaction cohérente d'un ensemble de composants. La génération automatique de tests d'interopérabilité est plus complexe car cela revient à générer un ensemble de testeurs répartis et interagissant entre eux de manière asynchrone. Les problèmes d'asynchronisme et de nouveaux problèmes de contrôle et surtout d'observation répartie des entités en interaction se posent. En effet, les méthodes de génération de tests considèrent généralement une architecture de test centralisée : l'ensemble des stimuli de test envoyés à l'IUT ainsi que les observations en retour sont traités en un point unique représenté par un seul testeur. Cette hypothèse est contraignante pour les systèmes répartis sur de grandes distances dans lesquels l'injection des stimuli et la collecte des événements de test peuvent être déformées par l'environnement. Le test d'interopérabilité considére que :

– les différentes entités à tester sont réparties sur une architecture distribuée,

– le testeur est également implanté de façon répartie sous la forme d'un ensemble de testeurs parallèles connectés aux entités de protocole et interconnectés entre eux et échangeant des informations de synchronisation de manière asynchrone.

Ceci engendre de nombreux problèmes à résoudre simultanément :

– l'observation répartie des comportements par les testeurs répartis,
– la gestion de l'asynchronisme testeur/testé et entre testeurs,
– la synthèse de la procédure de coordination entre les testeurs répartis.

Le travail à effectuer peut donc se résumer à l'exploration du concept du test réparti de systèmes répartis. Il s'agit d'étudier des modèles formels (automates, ordre partiel, etc.) et des algorithmes efficaces dédiés à la synthèse automatique de testeurs répartis pour le test d'interopérabilité des protocoles de communication. Ceux-ci devront être connectés aux langages normalisés en vigueur dans le monde des télécoms (SDL, TTCN et Concurrent-TTCN). Le point de départ consiste à s'appuyer sur les résultats obtenus par l'équipe Pampa sur : (1) la génération automatique de tests de conformité pour les protocoles (prototype TGV), (2) l'observation et le traitement des dépendances causales en environnement réparti.

7.6 Oural

Participants : Jean-Marc Jézéquel, Alain Le Guennec, Michel Bourdelles, Claude Jard, Séverine Simon.

Mots clés : UML, TGV, Validation, objets.

Résumé : *Projet pré-compétitif* RNRT. *Contrat en cours de notification, durée deux ans.*

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

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

Le projet a pour ambition de fédérer autour du langage UML *les différents outils des partenaires, et de promouvoir un processus de développement industriel basé*

sur une modélisation unique de l'application. Pour ce qui concerne Pampa, Oural est la partie pré-compétitive de la recherche menée dans Reutel.

L'objectif de ce projet est de fournir un ensemble d'outils destinés à accélérer et améliorer le processus industriel de développement de produit par le biais de réutilisation de composants logiciels. Dans ce processus, nous nous focalisons essentiellement sur les phases "amont" que sont : la définition et analyse d'architecture, la validation de l'architecture, l'exploitation des informations en vue de la génération de scénarios exécutables et la génération de squelettes de code. Le projet se déroule sur une durée de deux ans, à partir de Février 1999. Nos partenaires sont Softeam, Alcatel et Thomson-LCR.

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

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

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

Le sous projet outils d'ingénierie non fonctionnelle a pour but de fournir un environnement de validation par la simulation d'une architecture de composants. Ces composants possèdent des propriétés aussi diverses que les performances temps-reel, l'utilisation de ressources, le degrés de fiabilité,.. qui seront extraites des descriptions UML/T. L'environnement de validation comprends un simulateur de comportement non fonctionnel et un générateur de tests. Afin de mener a bien les simulations, une modélisation non fonctionnelle de la plate-forme générique sera étudiée.

Le sous projet outils de déploiement a pour but de fournir les outils destinés a déployer une architecture de composants sur la plate-forme générique Alcatel/Thomson pour en faire une architecture exécutable. Cet outil exploite les informations de déploiement décrites dans UML pour générer le code destiner à piloter les services de la plate-forme (ORB, tolérance au défaillances, autres middleware). D'autre part, un environnement de test en ligne sera étudié à partir de l'exploitation des scénarios de tests résultants des précédents outils.

7.7 Modistarc : méthodes et outils pour la certification d'architectures réparties à base d'Osek/VDX

Participants : Benoît Caillaud, Claude Jard, Thierry Jéron

.

Mots clés : Osek, Protocoles automobiles, génération de tests, TGV.

Résumé : *Contrat Inria 1 97 A663 00 000MC 005, septembre 1997 - avril 1999*

Projet européen R&D conduit par Thomson-CSF Detexis (ex Dassault électronique), en collaboration avec plusieurs constructeurs automobiles, Modistarc a permis de mettre en place une méthodologie de tests pour les protocoles de communications et de gestion embarqués sur les automobiles. Le projet Pampa a participé en proposant une méthode automatique de génération de tests partant des spécifications formelles LDS existantes.

Statut du projet

- Projet Européen Esprit no 25332
- Organismes participants : Thomson-CSFF Detexis (ex Dassault Électronique), FZI (Karlruhe, Allemagne), Renault, BMW, Opel, Motorola, Siemens, Inria, Sagem, PSA, IIIT (université de Karlsruhe, Allemagne).
- Date de début : 1er septembre 1997
- Durée : 20 mois
- Participation de l'Inria : Projets Sosso et Pampa.
- Page Web du projet :
  `http://www-iiit.etec.uni-karlsruhe.de/osek/Modistarc.html`

Objectifs et résultats

Le projet Modistarc contribue à l'effort des industriels Européens du secteur automobile pour développer une architecture normalisée de systèmes répartis embarqués pour l'automobile. Dans le cadre des activités du consortium Osek/VDX (Open systems and the corresponding interfaces for automotive electronics), les industriels Européens de l'automobile ont défini trois nouveaux standards : un système d'exploitation pour calculateurs embarqués (Ecu) pour l'automobile (Osek/VDX OS), un protocole de communication inter et intra Ecu (Osek/VDX Com) et un protocole de gestion de réseaux idoine (Osek/VDX NM). Ces standards ont pour vocation, d'une part de permettre la portabilité des applications d'un système à un autre et d'autre part, de permettre l'interopérabilité des plate-formes et des applications à l'intérieur d'un même système.

Les travaux au sein de projet Modistarc ont porté sur :

- la définition d'une méthodologie de test de conformité.
- La définition d'une suite de test de conformité Osek/VDX, servant de base à un outil de validation et qui a vocation à être intégrée dans la norme Osek/VDX.
- Le développement d'un outil de validation permettant le test d'implantations des différents éléments de la norme Osek/VDX. Cet outil de validation est commercialisé par Thomson-CSF Detexis.

– La réalisation d'une plate-forme de démonstration regroupant des Ecu développées par différents constructeurs de systèmes embarqués membres du projets (Motorola, Sagem et Siemens) et mettant en œuvre une application automobile type développée par PSA.

Contribution du projet Pampa

La participation du projet Pampa a porté sur la définition de la méthodologie de test pour la communication et la gestion de réseaux. La solution préconisée a été la génération automatique des cas de tests à partir des spécifications formelles des protocoles Com et NM.

7.8 Alcatel/Reutel

Participants : Claude Jard, Jean-Marc Jézéquel, Benoît Caillaud, Hubert Canon, Alain Le Guennec, Thierry Jéron, Séverine Simon, Loïc Hélouet, François Pennaneach.

Mots clés : Méthode, UML, langage d'interface, IDL, BDL, synchrone, objets.

Résumé : *Contrat Inria 1 97 A 93600 000MC012, novembre 1995 - février 2001*

*La collaboration avec Alcatel a commencé en novembre 1995 dans le cadre de l'appel Artica du ministère de la recherche et la mise à disposition d'un post-doctorant Inria chez Alcatel en 1996. En septembre 1997, elle a pris un nouvel essor avec un contrat direct avec Alcatel permettant le démarrage de plusieurs thèses. Claude Jard en est le coordinateur à l'Inria. L'objectif de la collaboration est la maîtrise du développement logiciel d'applications de télécommunication par la conception d'outils de manipulations formelles à l'intérieur d'une chaîne de développement définie par Alcatel. En 1999, cinq équipes de l'Inria participent au projet (*ADP*, Compose,* EPATR*, Vasy et Pampa) autour de quatre actions scientifiques. Pampa est principalement engagé avec* EPATR *et Vasy dans la définition et l'outillage d'un langage d'interface appelé* BDL.

Le défi pour Alcatel est de réduire les coûts de développement tout en améliorant la qualité du logiciel. Il s'agit aussi d'assurer une flexibilité et réactivité élevées en regard des évolutions du marché et des technologies naissantes. Précisément, il s'agit d'utiliser et de contrôler au mieux la mise en œuvre d'une application sur une plate-forme donnée (par génération de code quand c'est possible).

Le cadre de développement d'Alcatel est défini dans l'esprit Corba (the Common Object Request Broker Architecture) et met l'accent sur l'écriture de schémas de programmes dans des langages d'interface comme IDL (Interface Definition Language). Il doit être compatible en amont avec une méthodologie de conception objet type UML. Il doit aussi pouvoir s'interfacer avec un choix libre de plate-formes d'exécution : il faut donc bien veiller au rôle tampon du langage d'interface qui doit abstraire correctement la plate-forme et permettre aussi l'utilisation de langages variés pour la programmation des objets logiciels.

Pour faire en sorte qu'il ne reste pas un simple guide de structuration du logiciel, et puisse vraiment répondre aux objectifs généraux de la maîtrise du développement, il est nécessaire de lui adjoindre une batterie d'outils pour assister le concepteur d'applications. C'est ici que

se situe l'offre de l'Inria proposant l'utilisation de méthodes formelles. Par méthode formelle, nous entendons un formalisme fondé sur une sémantique mathématique et un ensemble de techniques associées permettant la manipulation (en général automatique) des programmes, à savoir la génération de code, des transformations, des vérifications, des optimisations, de la génération de tests,... Le travail consiste à proposer des extensions aux langages d'interface pour faire apparaître des informations non fonctionnelles (comportementales) et à concevoir des outils de manipulation formelle traitant les schémas de programme définis par les interfaces. L'Inria participe aussi à l'extension de la chaîne pour la prise en compte de la résistance aux défaillances.

Les équipes Inria engagées dans Reutel en 1999 sont :

- EPATR : modélisation synchrone, vérification, génération de code optimisé.
- Pampa : modélisation asynchrone, vérification, génération de tests, méthodologie de conception objet.
- Compose : optimisation de code, spécialisation, évaluation partielle.
- ADP : services et algorithmes pour réaliser la fonction de tolérance aux défaillances.
- Vasy : technologie de vérification, ouverture de l'outil CADP.

et se sont regroupées sur quatre actions de recherche :

1. Corba "fault-tolerant" (M. Hurfin)
2. Spécialisation et analyse de codes (G. Muller)
3. Modélisation et validation UML. Extensions à Oural (JM. Jézéquel)
4. Outils de validation et de génération de code pour UML (JP. Talpin, B. Caillaud)

Le travail du projet Pampa se concentre principalement sur les actions 3 et 4 en coopération étroite avec le projet EPATR. Nous concevons un nouveau langage d'interface : BDL (pour Behavioural Description Language). Le rôle de BDL est d'accompagner le développeur depuis les tâches préliminaires de spécification jusqu'au travaux de codage.

BDL intervient dans cette chaîne comme support à cet accompagnement et comme interface à l'intégration d'outils. Il est tout autant, pour son utilisateur, un langage de spécification, riche, expressif et structuré, avec une forte orientation objet, qu'un moyen de représentation intermédiaire simple, flexible et efficace pour l'utilisation des logiciels d'analyse, de test, de vérification et d'optimisation envisagés.

BDL se présente techniquement comme un formalisme permettant d'abstraire un objet et ses méthodes sous la forme d'un ensemble de graphes orientés étiquetés par les états de l'objet et de son environnement. Ces graphes permettent de coder les communications et leur ordonnancement, ainsi que les dépendances de données et de contrôle. Du point de vue de l'utilisateur, BDL se situe entre les MSC et les StateCharts. Du point de vue sémantique, ils correspondent aux automates d'ordres partiels.

Lorsque ces ordres partiels sont synchronisés, on se retrouve dans le monde des langages synchrones à la Signal, ce qui permet de récupérer les techniques de preuve et de génération de code associés. Ces ordres partiels peuvent également être interprétés de manière asynchrone, donnant un fondement sémantique à la communication, et permettant aussi une connexion avec les outils de preuve et test CADP et TGV. En 1999 a été spécifié la réalisation d'une plate-forme de démonstration intégrant BDL à UMLAUT, CADP et TGV.

7.9 GAT : Génération automatique de tests

Participants : Thierry Jéron, Pierre Morel, Claude Jard, Séverine Simon.

Mots clés : Génération de tests, TGV, protocoles.

Résumé : *Contrat université Rennes 1 mars 1998 - avril 1999*

Nous sommes intervenus avec Verimag par l'intermédiaire de l'Université Joseph Fourier de Grenoble et l'université de Rennes comme sous-traitants pour le marché France Télécom 97.6M.842 entre Vérilog et France Télécom intitulé "Réalisation d'un outil industriel de génération automatique de tests à partir de TVEDA *et* TGV*". L'objet du marché principal est de réaliser un outil industriel, dans l'environnement ObjetGéode de Vérilog, en utilisant le meilleur des technologies de l'outil* TVEDA *du Cnet et de notre prototype* TGV*. La première phase du marché (spécification de l'outil) a été réalisée en 98. La deuxième phase (réalisation et expérimentation) s'est achevée en avril 99.*

Le Cnet développe depuis plusieurs années son propre outil TVEDA dont le principe est de générer automatiquement des objectifs de tests, un par branche de transition d'un automate étendu produit à partir d'une spécification Estelle ou LDS, puis de générer le test correspondant à chaque objectif par construction partielle du système de transition de la spécification. Verimag et le projet Pampa ont développé leur prototype TGV (cf. module 5.1) qui permet de produire un test pour un objectif de test donné par parcours à la volée du système de transition de la spécification. Vérilog a développé un prototype de génération de tests TTCgeN incorporé à l'environnement ObjectGéode basé sur la construction explicite du système de transition de la spécification. Le but du marché est de réaliser un outil commercial de génération de test, basé sur TTCgeN et incorporant le meilleur des technologies de TVEDA et TGV. Cet outil permet donc de calculer automatiquement des objectifs de tests à la façon de TVEDA en assurant une couverture des branches de transitions de la spécification LDS. A partir de ces objectifs calculés ou d'objectifs spécifiques décrits par des MSC ou dans le langage Goal, le moteur de génération calcule les tests correspondants. Deux moteurs sont disponibles. Le premier est basé sur TTCgeN et travaille sur des graphes d'états explicites et le second utilise la technologie de génération à la volée de TGV. Le projet Pampa a participé à la conception de l'outil, principalement au moteur de génération et au développement de ce moteur. Il est maintenant commercialisé sous le nom de "Test Composer" dans l'environnement ObjectGéode.

8 Actions régionales, nationales et internationales

8.1 Actions régionales

Une nouvelle action du programme ITR démarre en novembre 1999 sur le test dans les spécifications objets. Elle va durer trois ans et permet de financer une bourse de thèse.

8.2 Actions nationales

8.2.1 Action Forma

Contrat CNRS 96C0093, mai 1996 - mai 1999
Participants : Claude Jard, Thierry Jéron, Pierre Morel, César Viho.

Le projet national Forma a pour objectif l'évaluation et le transfert de techniques de validation de spécifications temporelles dans les systèmes critiques. Il est structuré en quatre opérations bien ciblées regroupant une ou plusieurs équipes de recherche avec des industriels et donneurs d'ordre autour d'études de cas et d'objectifs à court terme. Le projet fait participer une dizaine d'équipes académiques et plusieurs industriels. Il est partiellement financé par le MENRT, la DGA et le CNRS. L'Irisa, à travers le projet Pampa, anime une opération sur la vérification et test de protocoles ATM (SSCOP et ABR). Ses partenaires sont le Cnet, le CEA, le Labri, le LSV et Verimag. Pampa, en coopération étroite avec le projet Spectre de Verimag, y fait la démonstration de la maturité du prototype TGV.

Plus particulièrement Pampa travaille sur la génération automatique de tests de conformité pour le protocole SSCOP. Le protocole SSCOP est un protocole de l'ATM permettant de résoudre le problème de pertes d'unités de données sur des connexions ATM point à point. Une spécification LDS nous a été fournie par le Cnet, écrite à partir de la norme Itu définissant ce protocole. La taille et la complexité de cette spécification (10 états de contrôle, environ 50 planches LDS) en font un excellent exemple de taille industrielle pour expérimenter nos outils et les améliorer.

En 1999, nous avons fait une tentative pour prendre en compte les données dans la génération et la vérification de tests. Nous avons collaboré avec le CEA pour permettre d'utiliser les résultats de leur outil Agatha comme objectifs de tests en entrée de TGV. Nous avons également vérifié partiellement la suite de tests du SSCOP fournie par l'ATM Forum en utilisant le module VTS de TGV.

8.2.2 Action coopérative Inria Verdon : VERification et test de systèmes réactifs critiques comportant des DONnées

Participants : Benoît Caillaud, Thierry Jéron, Claude Jard, Vlad Rusu.

Verdon est l'une des "Actions de Recherche Coopérative" de la Direction Scientifique de l'Inria. D'une durée de deux ans (1998-1999), cette action regroupe quatre équipes de l'Inria Rennes, de l'Inria Rhône-Alpes, de l'Inria Sophia-Antipolis et du laboratoire LSR-Imag. Elle est consacrée à une recherche fondamentale dans le domaine de la vérification et du test de systèmes réactifs (synchrones ou asynchrones) comportant une partie contrôle et une partie données de complexité significative, notamment en combinant les méthodes de model-checking avec des techniques appropriées pour la représentation et l'analyse des données.

Le projet Pampa s'est intéressé dans ce cadre à la génération et à la vérification de tests manipulant des données. En effet les techniques énumératives ne permettent pas de produire des tests manipulant des données tels qu'ils sont écrits manuellement. Ceci requiert la mise en œuvre d'approches telles que la représentation symbolique, des techniques d'abstraction, des techniques de résolution de contraintes, de preuve.

Le problème de la vérification de tests étant plus simple que celui de la génération, nous avons d'abord commencé par celui-ci. Partant de modèles de spécifications, d'objectifs de tests et de cas de tests manipulant des données, nous avons défini les critères de correction des tests sur ces modèles : essentiellement non-biais et non-laxisme. Le codage de ces modèles et des critères de correction en PVS nous permet de prouver la correction ou non des cas de tests par rapport à la spécification et à l'objectif visé.

Une page Web décrivant les objectifs du projet est accessible à l'adresse:
http://www.inrialpes.fr/vasy/verdon.

8.2.3 Groupe de travail test et objets

Participants : Claude Jard, Jean-Marc Jézéquel, Thierry Jéron, Yves Le Traon, Alain Le Guennec.

L'équipe Pampa anime un groupe de travail national sur le thème du test et des objets. Ce groupe est affilié au PRC/GDR ALP. Claude Jard en est co-responsable (avec MC. Gaudel et P. Thevenod). Yves Le Traon en est le secrétaire. Une dizaine d'équipes en France participent. La page Web http://www.irisa.fr/testobjets peut être consultée.

8.3 Actions financées par la Commission Européenne

Voir projet Esprit/Modistarc dans les actions industrielles.

8.4 Réseaux et groupes de travail internationaux

L'équipe Pampa participe à deux actions de coopération concertée permettant l'échange de chercheurs : une avec l'université Carlos III de Madrid (C. Delgado Kloos) sur la conception objet et le test, l'autre avec l'université de Twente (E. Brinskma) sur la génération automatique de tests.

Nous avons aussi défini un accord de collaboration avec le SRI (Stanford Research Institute, J. Rushby), associant aussi le laboratoire Vérimag sur l'utilisation de techniques symboliques. Vlad Rusu a effectué deux séjours de un mois au SRI dans ce cadre.

L. Hélouet a été invité trois mois à l'université de Concordia à Montréal pour poursuivre sa recherche sur les MSC et les appliquer à la question de la détection d'interactions de services de télécom.

Jean-Marc Jézéquel est membre de Nice (*Non-Profit International Consortium for Eiffel*), qui a en charge la standardisation du langage Eiffel et de ses bibliothèques. Il participe au groupe de travail international *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. Il a été invité à plusieurs reprise dans ce cadre, notamment huit semaines à l'université de Monash à Melbourne en avril 1999.

Enfin, l'équipe Pampa participe à deux actions de coopérations, l'une avec l'institut Iti au Caire et l'autre avec l'institut Ifi à Hanoi, et a accueilli en 1999 des stagiaires de ces pays.

8.5 Accueils de chercheurs étrangers

Naohito Sato (université de Tokyo) effectue un postdoc de six mois dans Pampa (septembre 1998 – février 1999) sur le thème de la programmation parallèle et répartie par objets avec UML.

Brian Nielsen (université d'Aalborg, Danemark) est venu passer une semaine dans le projet en mai 1999 sur le thème de la génération de tests pour les systèmes temporisés.

L'équipe accueille L. Ricker (venant de l'université de Toronto) pour neuf mois dans le cadre du programme de post-doctorat Ercim.

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

Claude Jard est co-responsable de l'action nationale Forma.

Claude Jard a organisé un "Industrial tutorial" lors du congrès mondial sur les méthodes formelles (FM'99). Le thème était le test de logiciel en général (une cinquantaine de participants).

Claude Jard a participé en 1999 aux conférences suivantes en tant que membre du comité de programmes :

- Gres'99 : Gestion de réseaux, Montréal, juin 1999,
- CFIP'99 : Colloque francophone sur l'ingénierie des protocoles, Nancy, avril 1999,
- SDL-Forum Int. Conf., Montréal, juin 1999,
- MSR : Modélisation des systèmes réactifs, Paris, mars 1999,
- RenPar'11 : Rencontres du parallélisme, Rennes, juin 1999.

En tant que membre du comité de programme, Claude Jard est en train d'aider à la préparation des conférences suivantes :

- Tacas'2000 : Tools and Algorithms for the Construction of Systems, Etaps, Berlin, avril 2000,
- Testcom'2000 : Test of communicating systems, Ottawa, octobre 2000,
- Afald'2000 : Atelier sur l'assistance formalisée au développement logiciel, Grenoble, janvier 2000,
- CFIP'2000 : Ingénierie des protocoles, Toulouse, septembre 2000.

Claude Jard est membre du comité de pilotage de la série de colloques MSR. Il prépare (en collaboration avec le LRI, l'Ircyn et l'université de Birmingham) la tenue d'une école européenne sur la modélisation et la validation des systèmes réactifs, Movep'2000, Nantes, Juin

2000.

Claude Jard est membre du conseil scientifique du Labri (Bordeaux), membre du conseil de laboratoire et du comité des projets de l'Irisa, membre des commissions de spécialistes à l'université de Rennes 1 et l'Insa.

Claude Jard est membre du comité de pilotage de l'association Goétic qui rassemble plusieurs sociétés industrielles et publiques dans l'objectif de veille technologique dans le domaine des télécommunications.

Claude Jard est membre du comité d'expertise du programme canadien (NSERC) sur les réseaux d'excellence en informatique et télécommunication.

Claude Jard a fait partie des jurys des thèses suivantes :
- M. Sirigheanu (Inria Rhône-Alpes, Grenoble, janvier 1990) (rapporteur),
- A. Bouajjani (HDR, Vérimag, Grenoble, janvier 1999) (rapporteur),
- M. Bourdelles (Inria Sophia, mai 1999) (rapporteur),
- S. Ramangalahy (Lifo, Orléans, octobre 1999) (rapporteur),
- M. Bozga (LSR, Grenoble, décembre 1999).

Thierry Jéron est co-responsable avec Bruno Sericola du Séminaire Irisa, Marie Noëlle Georgeault se chargeant des aspects administratifs et diffusion.

Thierry Jéron a fait partie des jurys de thèse suivants:
- Michel Bourdellès (Inria Sophia, mai 99)
- Solofo Ramangalahy (Lifo Orléans, octobre 99)

Thierry Jéron est membre de la commission de spécialistes de l'ENS Cachan.

Thierry Jéron fait partie du comité de programme de Lfm'200 : Fifth Nasa Langley Formal Methods Workshop, juin 2000, Williamsburg, Virginie, USA.

Jean-Marc Jézéquel a fait partie des jurys de thèse suivants:
- Alain Plantec, février 1999, université de Brest.
- Gerson Sunye, juillet 1999, université Paris VI.
- Pascale Launay, septembre 1999, université Rennes 1.
- Benhur Stein, octobre 1999, université Grenoble 1.

Jean-Marc Jézéquel a organisé le workshop *Trusted Components* associé à Tools Europe 99, était responsable de l'organisation des workshops de la manifestation Objet'99 qui s'est tenue à Nantes en mai 1999 et est *Conference Chair* de Tools Europe 2000.

Jean-Marc Jézéquel a fait partie des comités de programmes des conférences :
- Iscope'99 : International Symposium on Computing in Object-Oriented Parallel Environments, San Francisco, décembre 1999

- ICSSEA 99 : Génie logiciel, Paris, 1999.
- Isorc 99 : International Symposium on Object-oriented Real-time distributed Computing, Saint-Malo, mai 1999
- Tools Europe 99 : Technologies Objets, Nancy, juin 1999.
- LMO2000 (Langages et Modèles à Objets), Montréal, janvier 2000.

César Viho participe à Goetic. Il est responsable du projet MTI (Méthodologie de tests d'interopérabilité).

César Viho est membre de la commission des spécialistes de l'Ifsic/université Rennes I, de l'Insa/Rennes et de l'université d'Angers.

Yves Le Traon a animé un groupe de travail sur test et objets lors de la conférence Objet1999.

9.2 Enseignement universitaire

Depuis septembre 1999, Benoît Caillaud est responsable de la filière "génie logiciel et méthodes formelles" du DEA d'informatique de l'université de Rennes 1. Il est également responsable du module "analyse comportementale des systèmes réactifs et répartis" de cette filière du DEA.

Claude Jard est responsable du module "Méthodes formelles pour le développement des logiciels réparti" du DEA informatique de l'Ifsic.

Claude Jard et Thierry Jéron enseignent la validation de protocoles (Vérification, model-checking, test, observation répartie) à l'ENSTB Rennes et en Diic à l'Ifsic.

César Viho enseigne les "mécanismes des systèmes et réseaux informatiques" dans les filières Maîtrise et Diic de l'Ifsic.

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

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

L'équipe Pampa a accueilli en 1999, un stagiaire de DEA de l'Ifsic, deux mini-projet Diic et deux stagiaires étrangers.

9.3 Participation à des colloques, séminaires, invitations

Claude Jard a été invité à donner les séminaires suivants :

- "Sémantique et validation des Message Sequence Charts" au Lifo (Orléans, février 1999)

et à l'Ircyn (Nantes, avril 1999).

- "Test generation for Reactive Systems", Int. School on Formal Validation, Cirm (Marseille).
- "La recherche en test de protocoles", Journée nationale SEE, Paris et Toulouse, mai 1999.
- "Méthodes formelles et tests", Trentenaire du génie logiciel, Paris, novembre 1999.

Thierry Jéron a été invité à donner les séminaires suivants :

- "Génération automatique de tests de conformité: l'approche de TGV" au Lami université d'Evry,
- "Efficient automatic generation of conformance test suites for protocols and reactive systems" au SAL lunch du Stanford Research Institute (SRI) en mars 99 et à Femsys, Workshop on Formal Design of Safety Critical Embedded Systems à Munich en mars 99
- "Génération automatique de tests de conformité pour les protocoles" dans le cadre de l'Ecole Nouvelles Directions En Modelisation, Test et Validation au Cirm, Marseille, en avril 1999,
- Test Generation for Reactive Systems à FM'99, World Congress on Formal Methods in the development of computing systems, dans le cadre de l'Industrial Tutorial : Testing & Formal Methods à Toulouse en septembre 99,
- Testing reactive systems using Model-based Verification Techniques à l'Université de Twente et au 5th Dutch Testing Day, Eindhoven, Pays-Bas en novembre 99.

Jean-Marc Jézéquel a été invité à donner les séminaires suivants :

- "Design Patterns with Contracts", université de Monash, Australie, mars 1999.
- "Testing OO Software", Distributed System Technology Center, Melbourne, Australie, avril 1999.
- "Trusted Components", TMM Software Development Technologies Seminar, Rennes, septembre 1999.

Yves Le Traon a été invité à donner un séminaire sur le thème "Trusted components" à la conférence Ericson, Rome, décembre 1999.

Benoît Caillaud a donné en mars 1999 un séminaire intitulé "From Synchrony to Asynchrony" au Stanford Research Institute (SRI). Il a également donné à la même date un séminaire intitulé "Distributing Finite Automata Through Petri-Net Synthesis" au Cadence Laboratory à Berkeley.

10 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] J. FERNANDEZ, C. JARD, T. JÉRON, L. MOUNIER, « On-the-fly Verification of Finite Transition Systems », *Formal Methods in System Design 1*, 1993, p. 251–273.

[2] J.-C. FERNANDEZ, C. JARD, T. JÉRON, C. VIHO, « An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology », *Science of Computer Programming*, 29, 1997, p. 123–146.

[3] E. Fromentin, C. Jard, G.-V. Jourdan, M. Raynal, « On-the-fly Analysis of Distributed Computations », *Information Processing Letters*, 54, 1995, p. 267–274.

[4] F. Guidec, J.-M. Jézéquel, J.-L. Pacherie, « An Object Oriented Framework for Supercomputing », *Journal of Systems and Software Special Issue on* Software Engineering for Distributed Computing, juin 1996.

[5] C. Jard, R. Groz, J. Monin, « Development of VEDA: a prototyping tool for distributed algorithms », *in : IEEE Trans. on Software Engin., 14,3*, p. 339–352, mars 1988.

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

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

Livres et monographies

[8] J.-M. Jézéquel, M. Train, C. Mingins, *Design Patterns with Contracts*, Addison-Wesley, octobre 1999.

Articles et chapitres de livre

[9] A. Benveniste, B. Caillaud, P. Le Guernic, « Compositionality in dataflow synchronous languages: specification and distributed code generation », *Information and Computation*, To appear.

[10] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, D. Watkins, « Making Components Contract Aware », *IEEE Computer 13*, 7, juillet 1999.

[11] B. Caillaud, « Bounded Petri-net synthesis techniques and their applications to the distribution of reactive automata », *JESA, European Journal on Automated Systems*, 1999.

[12] C. Jard, T. Jéron, « An educational case study in protocol verification and distributed observation », *Journal of Computer Science Education, ECASP Special Issue To be published*, 1999.

[13] C. Jard, J.-M. Jézéquel, A. L. Guennec, B. Caillaud, « Protocol Engineering using UML », *Annales des Télécoms To be published*, 4, novembre 1999, p. –.

[14] J.-M. Jézéquel, « An Object-Oriented Framework for Data Parallelism », *ACM Computing Surveys 6*, décembre 1999.

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

[16] S. Ramangalahy, P. L. Gall, T. Jéron, « Une application de la théorie des jeux au test de conformité », *Revue Electronique sur les Réseaux de l'Informatique Répartie (RERIR) To be published*, 1999, Version étendue de [33].

[17] O. Roux, V. Rusu, F. Cassez, « Hybrid Verifications of Reactive Programs », *Formal Aspects of Computing*, A paraître.

[18] Y. L. Traon, T. Jéron, J.-M. Jézéquel, P. Morel, « Efficient OO Integration and Regression Testing », *IEEE Trans. on Reliability To be published*, 4, décembre 1999, p. –.

Communications à des congrès, colloques, etc.

[19] A. Benveniste, B. Caillaud, P. Le Guernic, « From Synchrony to Asynchrony », *in : CONCUR'99, Concurrency Theory, 10th International Conference*, J. Baeten, S. Mauw (éditeurs), *Lecture Notes in Computer Science, 1664*, Springer, p. 162–177, August 1999.

[20] B. Caillaud, « Application des techniques de synthèse de réseaux de Petri bornés à la répartition d'automates réactifs », *in : Deuxième congrès sur la modélisation des systèmes réactifs (MSR'99)*, Cachan, France, mars 1999.

[21] H. Canon, C. Jard, « Un modèle sémantique pour la validation des logiciels objets en télécommunication », *in : Colloque Francophone sur l'Ingénierie des Protocoles, CFIP 99, Nancy, France*, A. Schaff (éditeur), Hermes, p. 83–98, avril 1999.

[22] 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, juin 1999.

[23] D. Deveaux, J.-M. Jézéquel, « Des classes autotestables », *in : LMO'99*, Villefranche sur Mer, janvier 1999.

[24] R. Groz, T. Jéron, A. Kerbrat, « Automated Test Generation from SDL specifications », *in : SDL'99 The Next Millenium, 9th SDL Forum, Montréal, Québec, Canada*, R. Dssouli, G. von Bochmann, Y. Lahav (éditeurs), Elsevier, p. 135–152, juin 1999.

[25] L. Hélouet, « Un environnement de simulation pour le Message Sequence Charts », *in : ICSSEA'99, 12th international conference on Software & System Engineering and their Applications*, CNAM,Paris, France, December 1999.

[26] L. HélouëT, « A Simulation Framework for Message Sequence Charts », *in : SDL'99, The Next Millenium*, Y. R. Dssouli, G.V. Bochmann (éditeur), Elsevier, p. 473–488, 1999.

[27] W.-M. Ho, J.-M. Jézéquel, A. LeGuennec, F. Pennaneac'h, « UMLAUT: an Extendible UML Transformation Framework », *in : Proc. Automated Software Engineering, ASE'99, Florida*, octobre 1999.

[28] C. Jard, T. Jéron, L. Tanguy, C. Viho, « Remote testing can be as powerful as local testing », *in : Formal methods for protocol engineering and distributed systems, FORTE XII/ PSTV XIX' 99, Beijing, China*, J. Wu, S. Chanson, Q. Gao (éditeurs), Kluwer Academic Publishers, p. 25–40, octobre 1999.

[29] T. Jéron, J.-M. Jézéquel, Y. L. Traon, P. Morel, « Efficient Strategies for Integration and Regression Testing of OO Systems », *in : 10th IEEE International Symposium on Software Reliability Engineering, ISSRE'99, Boca Raton, Florida*, novembre 1999.

[30] T. Jéron, P. Morel, « Test generation derived from model-checking », *in : CAV'99, Trento, Italy*, N. Halbwachs, D. Peled (éditeurs), Springer, LNCS 1633, p. 108–122, juillet 1999.

[31] J.-M. Jézéquel, N. Sato, « A Simple Dynamic Load-Balancing Scheme for Parallel Molecular Dynamics Simulation on Distributed Memory Machines », *in : High-Performance Computing and Network, Lecture Notes in Computer Science*, Springer-Verlag, avril 1999.

[32] H. Kahlouche, C. Viho, M. Zendri, « Hardware testing using a communication protocol conformance testing tool », *in : Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, W. R. Cleaveland (éditeur), *Lecture Notes in Computer Science, 1579*, Springer Verlag, p. 315–329, March 1999.

[33] S. Ramangalahy, P. L. Gall, T. Jéron, « Une application de la théorie des jeux au test de conformité », *in : Colloque Francophone sur l'Ingénierie des Protocoles, CFIP 99, Nancy, France*, A. Schaff (éditeur), Hermes, avril 1999.

[34] V. Rusu, E. Singerman, « On Proving Safety Properties by Integrating Static Analysis, Theorem Proving and Abstraction », *in : Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Lecture Notes in Computer Science, 1579*, Springer Verlag, p. 178–192, March 1999. an extended version was published as IRISA research report No 1256.

Rapports de recherche et publications internes

[35] A. Benveniste, B. Caillaud, P. Le Guernic, « From synchrony to asynchrony », *Research Report n° 1233*, IRISA, mar 1999.

[36] V. Rusu, E. Singerman, « Interactive Abstractions: Proving Safety Properties by Integrating Static Analysis, Theorem Proving and Abstraction », *Research Report n° 1256*, IRISA, July 1999, also published as INRIA research report No 3726.

[37] N. Sato, J.-M. Jézéquel, « A Simple Dynamic Load-Balancing Scheme for Parallel Molecular Dynamics Simulation on Distributed Memory Machines », *rapport de recherche n° 1237*, IRISA, mars 1999.