Projet Omega : Un profil UML et un outil pour la modélisation et la validation de systèmes temps réel

Iulian Ober[1], Ileana Ober[2], Susanne Graf[1] et David Lesens[3]

[1]VERIMAG, Grenoble
[2] Université Paul Sabatier, Toulouse (IRIT)
[3] EADS SPACE Transportation

1. Introduction

Le développement de systèmes temps réel embarqués de qualité à coûts maîtrisés représente un défi technologique et industriel très important. Dans certains secteurs industriels, tel que le transport, l'aéronautique, ou les télécommunications, la tenue de ces objectifs qualité/coût impliquent l'utilisation d'outils et de techniques de validation et de vérification formelles. Cela passe par l'utilisation d'une approche orientée modèle.

Une des caractéristiques de cette approche, est l'utilisation de modèles *exécutables* qui contiennent des informations sur les différents aspects du système : exigences fonctionnelles, architecture statique, exigences non fonctionnelles (en particulier temps réel), etc. Pour que l'utilisation de tels modèles soit pertinente, le formalisme utilisé doit être suffisamment expressif, et son utilisation doit être supportée, non seulement par des outils pour la génération de code, mais aussi par des outils de validation et de vérification formelle au niveau du modèle.

Dans ce papier nous présentons une approche basée sur des modèles dédiés au développement des systèmes temps réel embarqués critiques. La modélisation est basée sur le langage UML. Pour exprimer les exigences spécifiques au domaine du temps réel, nous avons, dans le cadre du projet européen OMEGA[1], défini un profil UML, raffinement formel du profil UML standard *Scheduling, Performance and Time (SPT)* [OMG05]. Nous avons développé la boite à outils IFx pour supporter l'utilisation de ce profil (dit profil OMEGA). IFx permet de simuler et de valider formellement des propriétés dynamiques de modèles UML utilisant ce profil OMEGA. L'outil implémente ainsi la sémantique du profil et ses extensions temps réel.

Nous avons expérimenté notre approche – modélisation avec un éditeur UML du commerce (type Rational Rose ou I-Logix Rhapsody), et validation des modèles avec la boite à outils IFx – à travers plusieurs études de cas industrielles. Dans ce papier nous présentons la plus représentative d'entre elles, qui a été effectuée en collaboration avec EADS SPACE Transportation. L'étude porte sur la validation par simulation et sur la vérification formelle de propriétés dynamiques d'un modèle obtenu par rétro-ingénierie d'une partie représentative du logiciel de vol du lanceur Ariane-5. A travers cet exemple, nous présentons de manière schématique le cycle de développement et les fonctionnalités supportées par IFx, ainsi que les résultats de la validation du modèle Ariane-5. Nous ferons également le point sur les questions techniques ouvertes et donnerons des pistes pour des recherches futures.

[1] Projet Européen OMEGA (IST-33522). Voir aussi http://www-omega.imag.fr

2. Développement de systèmes temps réel à base de modèles

Dans un contexte d'arrivée à maturité des langages, techniques et outils de modélisation, le développement de systèmes de plus en plus complexes et à coûts de plus en plus réduits conduit à repenser tout le processus de développement et à l'organiser autour de l'utilisation de modèles, ainsi qu'à utiliser des outils permettant l'automatisation d'une grande partie des activités post-conception, comme la génération automatique de code, la génération et l'exécution automatique de tests, le déploiement vers une architecture physique, etc.

Le modèle doit ainsi inclure plusieurs types d'informations : l'architecture du système, le comportement au niveau du système ou au niveau des composants individuels, informations spécifiques à la plateforme matérielle utilisée, etc. UML s'est imposé comme un standard pour la modélisation et commence à être utilisé dans le développement de systèmes temps réel embarqués. Il a pour ambition d'intégrer toutes les informations mentionnées ci-dessus.

Pour que le développement à base de modèles UML soit pertinent, des outils permettant la validation de modèles sont nécessaires. Par validation nous entendons toutes les activités qui permettent de gagner confiance dans le modèle développé : vérifications statiques du modèle (typage, règles de cohérence), simulation et test de son comportement, vérification formelle de propriétés dynamiques (en spécial dans le contexte des systèmes temps réel critiques), etc.

3. Profil UML pour la modélisation de systèmes temps réel

Pour répondre aux besoins spécifiques du domaine de la modélisation de systèmes temps réel, l'OMG a standardisé un profil UML dédié à ce domaine (SPT [OMG05]). Ce profil est un premier pas pour répondre à la demande de l'OMG d'« un paradigme basé sur UML pour la modélisation des aspects liés au temps, à l'ordonnancement et à la performance dans les systèmes temps réel ». SPT contient des constructions pour modéliser différents aspects des systèmes temps réel, tel que la temporisation, les ressources, la performance, l'ordonnancement, etc. Néanmoins, le profil se concentre plus sur les aspects syntaxiques des annotations, sans leur donner une sémantique formelle et sans faire le lien avec la sémantique de la partie fonctionnelle du modèle UML.

Le profil OMEGA, que nous utilisons dans notre approche, spécialise une partie du profil SPT dans le but de raffiner la description de contraintes temporelles sur le comportement du système et de formaliser la relation entre ces contraintes (annotations) et la sémantique du modèle fonctionnel. Il offre des moyens et une syntaxe concrète pour exprimer les contraintes temporelles à la fois de manière opérationnelle et de manière déclarative.

Concepts temporels opérationnels définis par le profil :

- On considère que le temps courant a la même valeur partout dans le système et que cette valeur peut être accédée explicitement dans le langage d'actions, à travers l'expression *now*.
- Les *temporisations* sont des instances (d'une classe prédéfinie *Timer*) qui permettent de compter la progression du temps. Une *temporisation* peut être *armée* pour compter le temps correspondant à une durée et elle génère un événement *timeout* après l'écoulement de la durée spécifiée, qui peut être utilisé pour déclencher une transition dans l'objet qui le détient.
- Les *horloges* (instances d'une classe prédéfinie *Clock*) sont un concept emprunté aux automates temporisés [AD94]. Ils permettent de mesurer et de consulter le temps écoulé depuis leur dernier démarrage.

Les classes *Timer* et *Clock* font partie d'une librairie prédéfinie, spécifique à notre profil.

Contraintes de durée entre événements temporisées. Une des originalités de notre approche est la possibilité de décrire des contraintes temporisées sur l'exécution du système en dessus

de son comportement fonctionnel, et sans avoir à le modifier. Cela se fait en définissant des événements, que l'on peut associer à tous les changements d'état du système, aussi qu'aux tous les changements de valeurs de données dans un état observable du système. Les propriétés temporelles sont ensuite définies comme des contraintes de durée entre deux événements.

Le langage de contraintes défini par le profil OMEGA permet de définir des types d'événements (par exemple, l'appel d'une certaine méthode), qui peuvent identifier plusieurs occurrences de l'événement durant une exécution du système (dans l'exemple, les appels effectifs de cette méthode). Pour permettre la description de contraintes complexes, une partie de l'état du système peut être déclarée comme observable à l'occurrence d'un certain type événement, et les contraintes portant sur ce type d'événement pourront dépendre d'une condition sur cette partie observable de l'état. Enfin, les contraintes de durée portent sur des paires d'occurrences d'événements, et plusieurs façons de construire des paires d'occurrences à partir de paires de types d'événements sont définies par le profil (par exemple, les deux dernières occurrences, les deux occurrences du même indice, etc.)

Les contraintes générales entre événements représentent le mécanisme primitif du profil. Des *opérateurs prédéfinis* permettent de contraindre des durées souvent utilisées dans la pratique, comme le *temps de réponse* d'une opération, le *temps d' exécution* d'une action, etc. Une partie de ces contraintes existent déjà dans SPT, dans ce cas l'apport de notre approche étant de définir leur sémantique en termes de contraintes sur paires d'événements.

Une contrainte peut représenter soit une *hypothèse* soit une *exigence* sur l'exécution du système. Nous permettons de faire cette distinction à l'aide de qualificateurs explicites.

Pour plus de précisions sur le profil temps réel OMEGA et sa sémantique, voir [GOO05].

Description fonctionnelle du système dans le profil OMEGA. Les contraintes temporisées présentées avant peuvent être combinées avec n'importe quelle sémantique opérationnelle non temporisée d'UML dans laquelle l'on peut identifier les événements utilisés dans les contraintes. Cette indépendance entre la sémantique de la partie fonctionnelle et la sémantique des contraintes temporisées est une des spécificités de notre approche.

Cependant, le profil OMEGA fixe un sous-ensemble d'UML pour décrire la partie fonctionnelle du système, ainsi qu'une sémantique particulière de ce sous-ensemble, définis dans [DJPV02,vdZH05,OGO05]. Les concepts couverts par le profil sont :

- La description de la structure, faite à travers des diagrammes de classes, incluant des relations d'héritage et d'association.
- Un modèle particulier de concurrence, défini à partir des notions de classe active et passive. Les primitives de communication entre objets sont les signaux asynchrones et les appels d'opérations. Ceci est aussi le modèle de concurrence utilisé par l'outil Rhapsody [DK04].
- La description du comportement, faite à l'aide de machines à états (associées aux classes) et d'opérations. On distingue deux types d'opérations, les unes (*primitives*) décrites par une méthode, les autres (*triggered*) décrites par la machine à états de la classe.
- Les actions des transitions et des méthodes sont décrites dans un langage d'actions impératif conforme à la sémantique d'actions d'UML (version 1.4 et suivantes).

Description des exigences. Pour formaliser les exigences temporelles et fonctionnelles du système, le profil OMEGA propose, en dehors des contraintes temporelles, l'utilisation d'objets observateurs. Ce sont des objets dont la machine à états réagit aux événements ayant lieu dans l'exécution du système. Des annotations sur les états permettent de définir les exécutions considérées *incorrectes*. Les observateurs peuvent ainsi exprimer des propriétés de

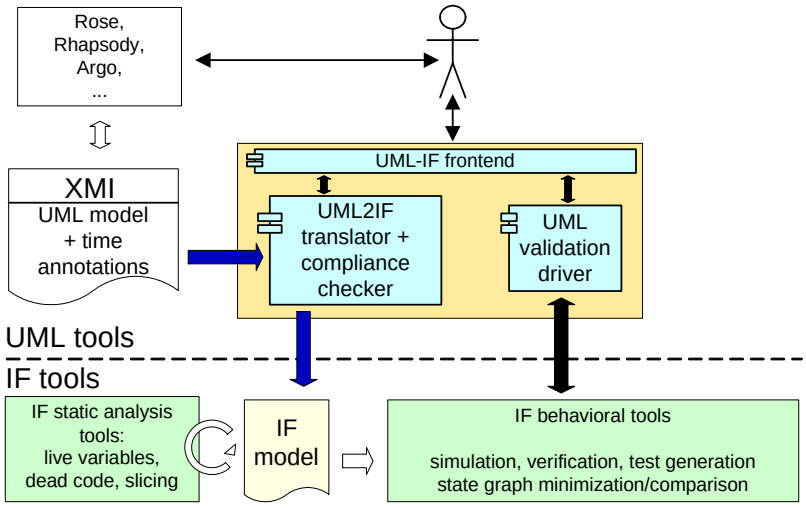

Figure 1. Architecture de la boîte à outils IFx

sûreté et sont utilisés dans la vérification par model-checking. Pour plus de détail, voir [OGO05].

4. Boite à outils IF/IFx

IF [BGM02,BGO+04] est une boite à outils qui permet de simuler et vérifier le fonctionnement de modèles décrits dans un langage spécifique (IF). IFx est un « frontend » pour la simulation et la validation de modèles UML OMEGA, destiné aux développeurs de systèmes temps réel critiques et embarquées.

IFx est utilisé en complément d'éditeurs UML compatibles avec le format d'échange XMI, telles que Rational Rose ou I-Logix Rhapsody. Le modèle UML est transformé en un modèle IF, ce qui offre accès aux fonctionnalités d'analyse et de vérification existantes dans l'outil IF. Pour que ce passage par IF soit transparent pour l'utilisateur, les fonctionnalités d'analyse et de vérifications ont été remontées au niveau de la spécification UML à travers une interface fournie par IFx. Ainsi, la connaissance de l'outil IF utilisé en dessous de l'interface n'est pas essentielle. L'architecture de la boîte à outils IFx est représentée dans la Figure 1. Les fonctionnalités qu'elle offre sont :

- La *simulation*, qui offre aux utilisateurs la possibilité d'exécuter et de déboguer un modèle de manière interactive. L'utilisateur peut faire des exécutions pas à pas, inspecter l'état du système, rajouter des points d'arrêt conditionnels, enregistrer et rejouer des exécutions, résoudre le non déterminisme (par exemple, crée par l'entrelacement d'activités concurrentes), contrôler les politiques d'ordonnancement, etc.
- La *vérification de conditions basiques* comme l'absence de blocages fonctionnels et temporels, la satisfaction d'invariants, etc.
- La *vérification de propriétés dynamiques* du comportement du système par des techniques de *model-checking*. Les propriétés à vérifier sont données par des contraintes temporelles ou par des objets observateurs [OGO05]. La vérification se fait par parcours exhaustive du graphe d'états du modèle. En cas d'erreur on obtient les traces défectueuses, qui peuvent être rejouées en mode simulation.

L'explosion de l'espace d'états est un des freins principaux à la généralisation d'outils de vérification automatique, Pour palier ce problème, l'outil IF/IFx utilise des techniques de réduction et d'abstraction dont l'efficacité est reconnue. Ainsi, des méthodes d'*analyse*

statique (comme la factorisation de variables mortes, ou la détection du code mort) sont utilisées pour optimiser le code IF dans le but de réduire l'espace d'états dynamiques du système, tout en préservant entièrement le comportement du système. A l'exploration, des techniques de *réduction d'ordres partielles* sont utilisées pour réduire l'explosion causée par l'entrelacement d'activités concurrentes indépendantes. En plus de ces techniques d'optimisation « exacte », l'outil fournit des fonctionnalités qui aident l'utilisateur à construire des abstractions. On mentionne ici entre autres le *tranchage* (*slicing*) qui permet de calculer la partie du système pertinente pour l'évaluation d'une propriété, ou *l'abstraction par minimisation* du graphe de comportement d'un composant du système.

5. Etude de cas : logiciel de vol du lanceur de Ariane

En collaboration avec EADS SPACE Transportation, nous avons utilisé IFx pour la validation d'un modèle UML obtenu par rétro-ingénierie d'une partie représentative du logiciel de vol du lanceur Ariane-5. Ce logiciel a pour objectif de contrôler le fonctionnement du lanceur depuis la phase sol jusqu'au placement en orbite de sa charge utile.

Les fonctions principales du logiciel sont :

- Le *contrôle* fin de la trajectoire du lanceur. Cette fonction est fournie par un ensemble de composants (GNC – pour Guidage, Navigation et Pilotage (Control en anglais)) qui implémentent essentiellement un cycle de contrôle s'exécutant avec une période de l'ordre de 100ms durant la totalité du vol du lanceur.
- La *configuration* et la *régulation* des étages du lanceur, correspondant au contrôle d'événements ponctuels comme l'allumage et l'extinction de moteurs, la séparation d'étages, la séparation de la coiffe ou de la charge utile, etc. Les composants qui implémentent ces fonctions forment essentiellement un sous-système réactif asynchrone.

Dans la modélisation et la validation, nous nous sommes concentrés sur le comportement temps réel critique du logiciel. Nous avons ainsi fait abstraction des aspects calculatoires, comme les algorithmes de pilotage. L'étude de cas reste néanmoins un system opérationnel représentatif.

Pour obtenir un modèle réaliste du logiciel de vol, en vue de la simulation et de la vérification de propriétés, nous avons dû modéliser également son environnement. Celui-ci est constitué les équipements embarqués et pilotés par le logiciel, qui ont été modélisés pour prendre en compte les contraintes temporelles qu'ils imposent (par exemple dans le cas du *Bus* utilisé pour transférer les données de mesure et les commandes) ou pour simuler les scénarios de panne matérielle (par exemple, panne d'ouverture d'une *Vanne*).

5.1. Validation par simulation et préparation pour la vérification

La fonction d'IFx qui compile le modèle UML vers une spécification IF permet une première validation du modèle. Les erreurs trouvées dans cette étape (erreurs syntaxiques ou de typage) sont simples, mais la plupart ne sont pas signalées par les éditeurs UML comme Rational Rose, qui a été utilisé dans cette étude.

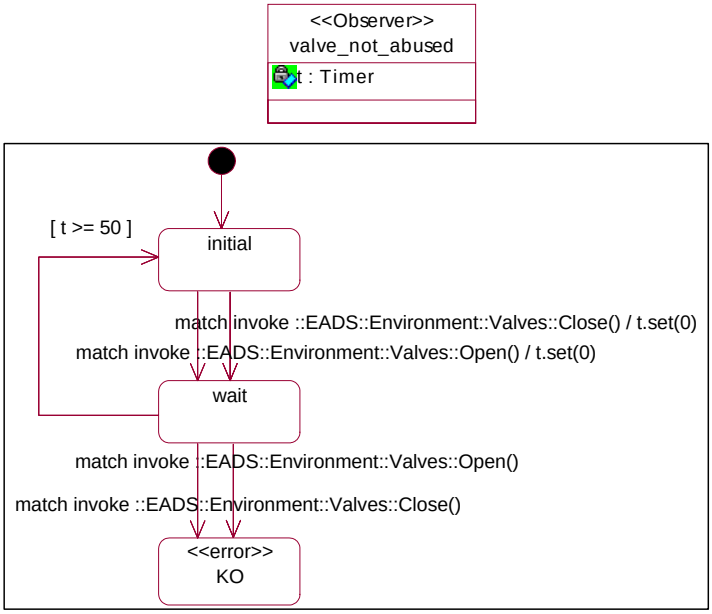

Figure 2. Une propriété de sûreté du modèle Ariane-5

Une seconde validation du modèle se fait par simulation interactive sur des scénarios nominaux du système. Des propriétés simples, comme l'absence de blocages peuvent être *testées* dans cette phase.

Enfin, une fois que l'utilisateur a gagné une certaine confiance dans son modèle, il peut essayer de le préparer en vue d'une vérification exhaustive, en appliquant par exemple des optimisations par analyse statique. Les optimisations « exactes » (qui préservent toutes les propriétés du comportement du modèle), comme la factorisation de variables mortes, peuvent être appliquées dans tous les cas et donnent en général des bonnes réductions de l'espace d'états.

5.2. Formalisation et vérification d'exigences de sûreté

Dans cette phase, il s'agit de formaliser un ensemble d'exigences de sûreté spécifiques au modèle étudié et de les vérifier par model-checking. Pour le cas d'Ariane-5, nous avons spécifié 9 propriétés de sûreté, la plupart faisant appel à des mesures quantitatives du temps. Le moyen le plus simple de spécifier ce type de propriétés en vue de la vérification avec IFx est d'utiliser les observateurs.

Un exemple simple de propriété est la suivante : « Le logiciel ne doit jamais envoyer deux commandes aux *Vannes* à moins de 50ms d'intervalle » (cette exigence provient de contraintes électriques des équipements utilisés). Cette propriété est formalisée par l'observateur de la Figure 2.

Pour rendre possible la vérification de ces propriétés, l'outil doit être capable de générer entièrement l'espace d'états du modèle. Pour arriver à cela pour des modèles réalistes comme celui de l'Ariane 5, l'utilisation des techniques de réduction dynamiques offertes par l'outil IF/IFx, comme la réduction par ordre partiel est absolument essentielle. Dans le cas d'Ariane-5, celle-ci a donné d'excellents résultats, en réduisant la taille de l'espace d'états par un facteur de plus de 1000.

Toutefois, dans certains cas, ceci n'est pas suffisant et l'on doit faire appel à des abstractions pour réduire encore plus l'espace de recherche. Par exemple, dans notre étude, la plupart des propriétés de sûreté portaient sur la partie acyclique du logiciel. Pour pouvoir les vérifier, nous avons construit une abstraction du modèle de la partie cyclique. L'abstraction que nous avons construite est une surapproximation du comportement de la partie cyclique.

Dans le cas où une propriété de sûreté est vérifiée avec cette abstraction de la partie cyclique, ceci nous assure que la propriété est tenue aussi par le système concret (sans abstraction).

Au final, l'utilisation d'abstractions nous a permis de construire des systèmes d'une taille très modérée, sur lesquelles toutes les propriétés ont pu être vérifiées.

5.3. Validation de la politique d'ordonnancement

Un type particulier de propriété qui peut être exprimé et vérifié avec des observateurs est la pertinence de la politique d'ordonnancement d'un système temps réel.

Dans le cas d'Ariane-5, les composants du système sont tous exécutés sur un processeur centralisé. Un modèle de tâches (*threads*) est défini, qui fait appel à une politique d'ordonnancement préemptive avec priorités fixes. Les activités de chaque composant du système sont affectées ensuite à une tâche (plusieurs composants peuvent partager la même tâche). La question posée est si la politique d'ordonnancement ainsi définie permet au système de tenir la réactivité requise, par exemple si tous les calculs relatifs à un cycle de pilotage se terminent avant la fin du cycle.

Les méthodes d'analyse classiques, qui pour répondre à cette question font totalement abstraction du comportement effectif du logiciel a l'intérieur de chaque période, ne sont pas applicables lorsque les durées d'exécution des actions sont très variables ou lorsqu'on a à faire avec des événements sporadiques. Ces deux difficultés apparaissent dans le modèle Ariane-5 : les durées de certaines actions sont très dépendantes de la phase courante de vol, et la partie *acyclique* traite essentiellement des événements sporadiques.

Pour cette raison, un des objectifs de notre étude a été de valider la pertinence de la politique d'ordonnancement en partant du modèle fonctionnel et en utilisant les techniques de vérification dans IF/IFx. Nous avons utilisé les moyens de description offerts par le profil Omega pour décrire les informations temporelles nécessaires (durées unitaires des actions, politique d'ordonnancement). Les objectifs d'ordonnancement (tenue des délais) sont des propriétés de sûreté qui ont été exprimés et vérifiés avec des observateurs.

6. Perspectives et conclusions

Dans le futur ces travaux peuvent se poursuivre sur plusieurs directions. A court terme, il s'agit d'un côté d'appliquer ces techniques sur plusieurs études de cas pour avoir un maximum de retours pouvant nous amener à améliorer nos résultats. D'un autre côté nous nous proposons d'adapter notre approche au standard UML 2.0. Comme au moment où nous avons démarré nos travaux, il n'y avait pas d'outil supportant UML 2.0 et le format d'échange XMI pour UML 2.0 n'était pas encore défini, notre travail porte sur la version 1.4.

L'objectif à long terme est d'améliorer l'efficacité de nos techniques de vérification sur des modèles de grande taille, notamment en les rendant plus adéquates à la modélisation orienté par objets et a l'utilisation de composants.

Les résultats obtenus lors de l'application de notre approche sur une étude de cas de taille moyenne nous laissent croire dans la pertinence d'une approche de développement qui utilise des modèles exécutables et des outils de vérification et validation appliqués sur ces modèles.

7. Références

[AD94] R. Alur et D.L. Dill. *A theory of timed automata*. Dans Theoretical Computer Science, vol. 126, no. 2, 1994.

[BGM02] Marius Bozga, Susanne Graf, et L. Mounier. *IF-2.0: A Validation Environment for Component-Based Real-Time Systems*. Dans Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen, LNCS 2404, Springer 2002.

[BGO+04] Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, et Joseph Sifakis. *The IF toolset*. Dans SFM-04:RT 4th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time, LNCS 3185, Springer, 2004.

[DJPV02] W. Damm, B. Josko, A. Pnueli, et A. Votintseva. Understanding UML: A Formal Semantics of Concurrency and Communication in Real-Time UML. Dans Proceedings of FMCO'02, LNCS 2852, Springer 2002.

[DK04] D. Harel and H. Kugler. The RHAPSODY Semantics of Statecharts (or, On the Executable Core of the UML). In Integration of Software Specification Techniques for Application in Engineering, LNSC 3147, Springer 2004.

[GOO05] Susanne Graf, Ileana Ober, et Iulian Ober. *Timed annotations in UML*. STTT, Int. Journal on Software Tools for Technology Transfer, 2005. Accepté pour publication.

[OGO05] Iulian Ober, Susanne Graf, Ileana Ober. *Validating timed UML models by simulation and Verification*. STTT, Int. Journal on Software Tools for Technology Transfer, 2005. Accepté pour publication.

[OMG05] OMG. *UML Profile for Schedulability, Performance, and Time Specification. OMG* document *formal/05-01-02,* January 2005.

[vdZH05] M. van der Zwaag et J. Hooman. *A Semantics of Communicating Reactive Objects with Timing*. STTT, Int. Journal on Software Tools for Technology Transfer, 2005. Accepté pour publication.