Transformations dirigées par des propriétés non fonctionnelles en conception logicielle

Julien Mallet, Siegfried Rouvrais

Transformations dirigées par des propriétés non fonctionnelles en conception logicielle

Julien Mallet et Siegfried Rouvrais

Institut Télécom ; Télécom Bretagne
Technopôle Brest Iroise, CS 83818, 29238 Brest Cedex 3
Université européenne de Bretagne
{mallet,rouvrais}@telecom-bretagne.eu

Résumé. Pour qu'un modèle de système logiciel critique soit digne de confiance, il est opportun de s'appuyer sur un processus rigoureux de transformations successives prenant en compte au plus tôt des propriétés non fonctionnelles. Pour cela, cet article se propose de décrire en logique quelques propriétés de sûreté de fonctionnement, de sécurité, ou encore de couplage, afin de diriger deux types de transformations : (i) une première transformation structurelle pour générer un assemblage de composants dits fonctionnels, (ii) une seconde pour incorporer successivement des mécanismes au modèle du système afin de satisfaire au mieux les exigences.

1 Un processus dirigé par le non fonctionnel

L'ingénierie dirigée par les modèles préconise de construire un modèle de son système par application de transformations successives. Le choix d'un modèle architectural impacte les propriétés finales du système. Encore trop souvent, les propriétés non fonctionnelles (p.ex. fiabilité, sécurité, couplage) sont peu prises en compte dans les phases amonts de la conception (Chung et al., 1999). Il convient pourtant de les associer au plus tôt au processus de construction pour satisfaire, au mieux, les exigences requises. Nous proposons pour cela de guider formellement les transformations à appliquer à l'aide de propriétés spécifiées en logique du premier ordre (cf. section 2). Deux types de transformations (cf. section 3) vont permettre, pas-à-pas, de raffiner le modèle initial afin de concevoir un modèle au plus près des exigences. En retenant également la logique pour spécifier les composants, cette proposition ne s'appuie pas sur un langage particulier de description du système. Un modèle de composants (p.ex. CCM, Fractal, SCA) pourrait toutefois être étendu pour y intégrer les expressions des propriétés retenues.

Premières alternatives de conception : choix d'un style. Un style d'architecture représente une famille d'architectures. C'est un modèle (c.-à-d. SDM, *Style Definition Model*) capitalisable qui amène à des premières alternatives de conception. Un style est défini par un vocabulaire spécifique (p.ex. composants et connecteurs) et un ensemble de contraintes structurelles portant sur les assemblages possibles entre composants. P.ex., pour un système distribué, les styles architecturaux client-serveur et pair-à-pair ne suivent pas les mêmes règles

de communication, et ce quel que soit le paradigme de conception choisi (p.ex. composants, services, orienté objet, processus). En suivant dans un premier temps une approche de type MDA, il est possible (Mallet et Rouvrais (2008)) de modéliser un système sous une forme purement fonctionnelle F (c.-à-d. SIM, *Style Independant Model*). Une première transformation T_s, dite structurelle, va permettre (p.ex. par composition selon le formalisme retenu) de générer un modèle F_s de l'architecture spécifique à un style (c.-à-d. SSM, *Style Specific Model*). Par construction, la transformation doit garantir $F_s \equiv F$. La partie gauche de la figure 1 présente une vue d'ensemble de cette première phase. Les exigences non fonctionnelles du système sont notées ENF. Un guidage (1) peut être proposé afin que l'architecte sélectionne un style dans

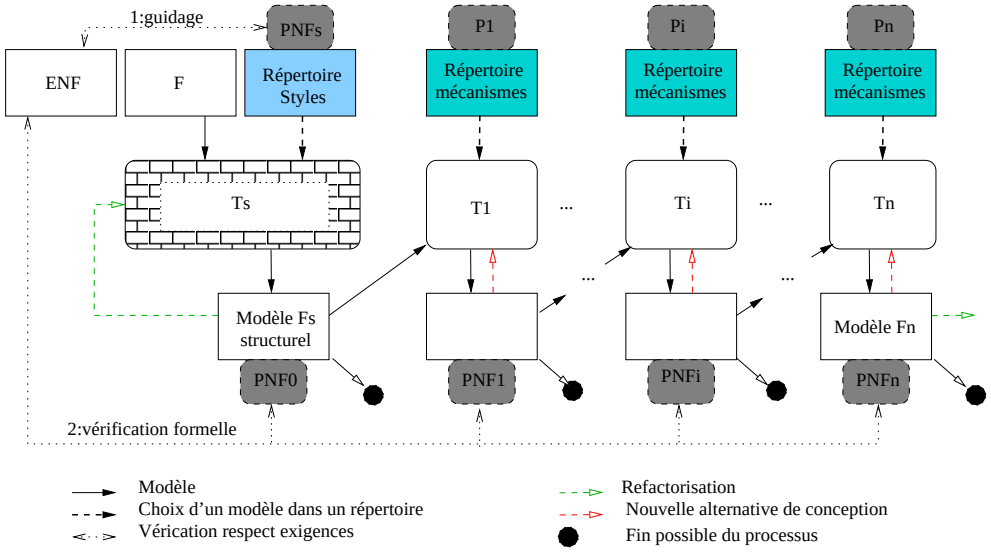

FIG. 1 – *Processus de transformations dirigé par des préoccupations non fonctionnelles.*

le cas où des exigences (inclus dans ENF) seraient déjà en conformité avec des propriétés intrinsèques PNF_s toujours satisfaites par le style. Après cette première transformation, une vérification (2) permet de déterminer si l'architecte dispose d'un modèle F_s satisfaisant non fonctionnellement. Autrement, il lui faudra soit tenter de trouver un autre style plus proche des exigences, soit chercher à introduire en deuxième phase des mécanismes dans le modèle généré afin de chercher à vérifier les propriétés non fonctionnelles.

Propriétés et transformations. Une fois un modèle structurel retenu et ne pouvant satisfaire les exigences non fonctionnelles, nous proposons de suivre ensuite un processus par étapes séparant les préoccupations (c.-à-d. expertises différentes par propriétés). Chaque étape s'attache à traiter une propriété non fonctionnelle P_i particulière. Des mécanismes également modélisés sont utilisés pour les transformations (p.ex. réplication active ou passive pour la fiabilité, hachage ou signature pour l'intégrité en tant que sous-propriété de sécurité incluant également la confidentialité et la disponibilité). Un mécanisme se doit d'offrir une conjonction de propriétés spécifiée dans le répertoire ad-hoc de P_i. Une transformation T_i va pouvoir intégrer un

mécanisme M_{ij} au modèle F_i afin de générer un modèle F_{i+1}. La transformation va prendre la forme d'un raffinement (c.-à-d. $F_i \sqsubseteq F_{i+1}$) dans le mesure où le mécanisme utilisé peut impacter d'autres propriétés ou introduire de nouvelles fonctionnalités.

À partir du moment où les propriétés du modèle généré respectent les exigences non fonctionnelles, la génération d'une implantation du modèle devient possible par la maîtrise d'œuvre. Mais la transformation T_i va parfois impacter plusieurs propriétés sur le modèle et risque de remettre en cause des propriétés satisfaites antérieurement. Dans ce second cas, il convient tout d'abord de chercher à appliquer un autre mécanisme de $\{M_i\}$. Si après l'application de toutes les transformations possibles, il s'avère que le modèle initial F_s étendu en F_n ne satisfait toujours pas les exigences, il conviendra de rechercher un autre style (c.-à-d. refactorisation à F constant) ou de relaxer des exigences. Plus globalement sur le cycle de vie, suite aux phases de maintenance et de maintient en condition opérationnelle, un système peut s'éroder peu à peu. Notamment, les évolutions ou contraintes technologiques non prévues initialement conduisent à remplacer des composants. De nouveaux composants peuvent être ajoutés à l'architecture afin de l'adapter à un nouveau contexte. Des composants peuvent être à intégrer pour fixer des erreurs non traitées ou repérées auparavant. Quand la « charpente initiale » n'est plus tout à fait adaptée et afin de tenir au mieux les exigences de qualité du système, il peut être raisonnable, par refactorisation, de proposer un nouveau modèle du système (c.-à-d. renouveler le SSM).

2 Description formelle de propriétés non fonctionnelles

Gage de correction, des expressions formelles peuvent être exploitées de manière systématique pour diriger les choix de conception par des caractéristiques non fonctionnelles.

Une logique étendue. Le formalisme retenu s'appuie sur les travaux sur la modélisation de la sûreté de fonctionnement de Saridakis et Issarny (1999). Les propriétés de sûreté de fonctionnement (Laprie et al., 1996) y sont décrites dans la logique des prédicats du premier ordre augmentée de la relation de précédence causale de Lamport. Un résumé des notations utilisées est donné figure 2. Un système F disposant d'une spécification Σ, est composé d'objets α, β, ... (la notation ε dénotant un objet quelconque). Il a un état σ à un instant donné. Les propriétés sont définies par des relations entre états ou entre changement d'états du système F. Le prédicat [ ] indique que le système est dans un état donné : $[\sigma]$ est vrai si et seulement si le système est dans l'état σ. Une action correspond à un changement d'états. Les actions considérées ici sont des actions d'entrées/sorties. L'action $\alpha.export(\beta, m)$ (resp. $\beta.import(\alpha, m)$) dénote que α a envoyé la donnée m à β (resp. β a reçu la donnée m envoyée par α).

Les relations sont basées sur un opérateur de précédence noté $<$. Si l'on considère deux actions a et b, $a < b$ indique que l'action a a eu lieu avant b.

Afin de décrire une large gamme de propriétés, des raffinements de l'état des objets ont été ajoutés. P.ex., pour indiquer qu'un objet est connecté au système, la variable $online$ est introduite. Ainsi, un objet α dans un état $\alpha.\sigma$ a la propriété $online \in \alpha.\sigma$ si α a envoyé dans le passé ou enverra dans le futur un message à un autre objet du système.

Exemples de propriétés pour systèmes critiques. Figure 2, nous présentons les propriétés de sécurité-innocuité (tirée de Saridakis et Issarny (1999)), de confidentialité et de découplage

F : système	$import()$, $export()$: action d'envoi/réception
Σ : spécifications du système	$[\sigma]$: prédicat indiquant que le système est dans l'état σ
α, β, γ : objets du système	$faulty(\sigma)$: prédicat indiquant que l'état σ est erroné
ε : objet quelconque du système	$online$: variable de l'état d'un objet indiquant que l'objet
σ : état du système	est connecté au système

$$Safety(F) = ([\sigma] \wedge faulty(\sigma)) \Rightarrow (\exists\, \sigma', \sigma'' \in \Sigma \mid ([\sigma] < [\sigma']) \wedge ([\sigma''] < [\sigma]) \wedge (\sigma' \subseteq \sigma''))$$

$$Confidentiality(F) = \forall\, \alpha,\ \beta \in F, \alpha.export(\beta, m) \wedge \varepsilon.import(\alpha, m) \wedge ((\alpha.export(\beta, m) < \varepsilon.import(\alpha, m)) \Rightarrow \varepsilon = \beta)$$

$$TimeCoupling_e(\alpha, \beta) = \alpha.export(\beta, m) \Rightarrow [\alpha.\sigma] \wedge [\beta.\sigma] \wedge online \in \beta.\sigma$$

$$TimeCoupling_i(\alpha, \beta) = \beta.import(\alpha, m) \Rightarrow [\alpha.\sigma] \wedge [\beta.\sigma] \wedge online \in \alpha.\sigma$$

FIG. 2 – *Notations et exemples de propriétés non fonctionnelles*

temporel (l'ensemble des propriétés, définies grâce au formalisme, est présenté dans (Belbis, 2008)).

La sécurité-innocuité, notée $Safety$, définit le fait que suite à une erreur, il existe un état σ' ultérieur à cette erreur qui n'est pas erroné. Cet état est un sous-ensemble d'un état σ'' précédant l'état erroné.

La confidentialité d'un système F, notée $Confidentiality$, indique que tout message ne peut être lu (et donc reçu par un $import$ sur son identité) par un utilisateur du système autre que celui auquel il était adressé.

Le couplage temporel pour l'échange de message est considéré des deux côtés de la communication, c.-à-d. d'une part l'envoi du message, et d'autre part sa réception. Deux composants α et β sont couplés temporellement du point de vue de l'émission ($TimeCoupling_e$) si et seulement si α ne peut envoyer de message vers β que si celui-ci est connecté au système. Deux composants α et β sont couplés temporellement du point de vue de la réception ($TimeCoupling_i$) si et seulement si β ne peut recevoir de message du composant α que si celui-ci est connecté au système. En général, la propriété requise d'un système est plutôt le découplage temporel. Cette propriété s'écrit, dans le formalisme, comme la conjonction des négations des propriétés précédentes.

3 Correction des transformations de modèles

La transformation initiale sur le style d'architecture, ou une future refactorisation, correspond finalement à proposer ou préconiser un changement de structure avec $F \equiv F_s$, elle n'ajoute pas de fonctionnalités (Fowler et al., 1999). Il convient toutefois de le vérifier pour garantir la correction du modèle généré, aux niveaux fonctionnel ainsi que non fonctionnel. Trois approches classiques existent : l'approche formelle où la sémantique de la transformation est prouvée, les tests où l'on vérifie après construction que les exigences sont atteintes et les analyses d'architecture pour caractériser des propriétés sur le modèle généré.

Précédemment, nous avons proposé une approche formelle, basée sur un calcul de processus (c.-à-d. FSP associé à son *model checker*), afin de composer (c.-à-d. transformation endogène) une description fonctionnelle dite « pure » d'un système avec un style d'architecture

(Mallet et Rouvrais, 2008). Les calculs orientés processus sont relativement bien adaptés aux analyses de propriétés de sécurité (p.ex. SafeAmbiant (Degano et al., 2000), Spi ou encore Kell-calcul). Au regard des propriétés non fonctionnelles visées (notamment sûreté de fonctionnement), un formalisme fondé sur la logique peut également être approprié (p.ex. OCL dans ATL (Bézivin et Jouault, 2006), TLA+ (Zschaler, 2009)) pour la modélisation. Dans cet article, les propriétés fonctionnelles et non fonctionnelles sont spécifiées en logique. La transformation d'introduction de style T_s, figure 1 page 2, revient à «porter» les composants et connecteurs fonctionnels sur ceux du style sélectionné. Du point de vue des propriétés, la transformation T_s est une fonction de substitution f des composants et connecteurs du modèle initial F vers ceux du style. En conséquence, les propriétés non fonctionnelles du style PNF_s ne changent pas et nous avons $PNF_0 = f(PNF_S) = PNF_S$. Pour les exigences fonctionnelles F, toute transformation T_s produisant un modèle F_s ne les conservant pas est considérée comme non valide et est éliminée des transformations possibles. La non-conservation des exigences est vérifiée par $PNF_0 \wedge f(F) \equiv false$, c.-à-d. que les propriétés non fonctionnelles du modèle généré et les exigences fonctionnelles portées sur le style sont non satisfiables.

Le choix du style revient à confronter les propriétés non fonctionnelles du style PNF_S aux exigences ENF après application de la substitution f. Si $PNF_s \Rightarrow f(ENF)$, le processus de sélection se termine : le style garantit les exigences. Par contre si $PNF_s \not\Rightarrow f(ENF)$, il convient d'envisager un autre style ou d'introduire des mécanismes spécifiques.

Le second type de transformations (c.-à-d. intégration successive de mécanismes) est un raffinement de modèle conservant les fonctionnalités initiales mais pouvant potentiellement en rajouter, c.-à-d. $F_i \sqsubseteq F_{i+1}$. Pour une catégorie de mécanismes donnée Pi (p.ex. confidentialité, sûreté de fonctionnement), la sélection se fait en balayant l'ensemble des mécanismes M_{i1} à M_{im}. Il convient de confronter les propriétés non fonctionnelles du modèle F_i défini par $PNF_i = P_{ij} \wedge PNF_{i-1}$ et les exigences non fonctionnelles initiales ENF. Finalement, soit un mécanisme garantit les exigences (c.-à-d. $PNF_i \Rightarrow ENF$) et le processus de conception est terminé, soit les exigences ne sont pas complètement satisfaites (c.-à-d. $PNF_i \not\Rightarrow ENF$) et il convient de sélectionner un autre mécanisme pour cette catégorie et de passer à une nouvelle étape.

4 Bilan et perspectives

L'ingénierie dirigée par les modèles gagne à s'appuyer sur des méthodes formelles afin de raisonner sur la correction du processus. Pour qu'un modèle d'architecture de système logiciel critique soit digne de confiance, il convient de prendre en compte les exigences non fonctionnelles dans les transformations successives. Cet article s'appuie sur la logique afin de vérifier des propriétés non fonctionnelles telles que la sûreté de fonctionnement, la sécurité, ou encore le couplage temporel, et ce au plus tôt dans les cycles de conception. Ces spécifications peuvent également permettre de diriger en partie le processus afin de faciliter les tâches des équipes confrontées aux alternatives de conception. À travers une transformation structurelle, les analyses de qualité permettent en premier lieu de guider le choix du style (c.-à-d. génération d'un SSM conforme à un métamodèle donné, p.ex. composants). Ensuite, les propriétés sont les principaux déclencheurs des transformations à appliquer (p.ex. intégrations successives de composants au SSM initial) pour répondre, au mieux, aux exigences de qualité.

L'ordre d'application des transformations a un impact non négligeable sur la tenue des exigences. En effet, les propriétés non fonctionnelles ont une regrettable tendance à interférer (Chung et al., 1999). P.ex., des propriétés de sécurité ou de couplage ne sont au final pas vérifiées à l'identique quand le concepteur sélectionne une réplication active ou passive pour garantir la fiabilité de son système. Autrement, si des alternatives pour le couplage ou la sécurité ont été abordées avant de s'intéresser aux propriétés de sûreté de fonctionnement, les transformations pourraient ne pas converger vers les mêmes modèles. Il serait donc judicieux de rechercher une méthode pour orienter l'ordre des transformations, en évitant, autant que faire se peut, les retours-arrières dans le processus.

Des transformations réalisées antérieurement dans le processus pourraient également être tracées et capitalisées (au niveau du modèle généré et des propriétés associées) dans le but de réutiliser celles-ci en cas de probables retours-arrières.

Finalement, l'approche exposée manipule abstraitement la description de l'architecture logicielle (propriétés logiques sur les composants et les connecteurs). La concrétisation de l'architecture dans un modèle à composants donné (p.ex. Fractal) permettrait d'étendre le processus de conception proposé jusqu'à la génération de code.

Références

Belbis, B. (2008). Spécifications de propriétés non fonctionnelles et analyses pour le choix du style d'architecture logicielle. Rapport de master recherche (M2RI), Télécom Bretagne.

Bézivin, J. et F. Jouault (2006). Using ATL for checking models. In *Proceedings of the International Workshop on Graph and Model Transformation (GraMoT)*, Volume 152 of *Electronic Notes in Theoretical Computer Science*, pp. 69–81. Elsevier.

Chung, L., B. A. Nixon, E. Yu, et J. Mylopoulos (1999). *Non-Functional Requirements in Software Engineering*. Kluwer.

Degano, P., F. Levi, et C. Bodei (2000). Safe ambiants: Control flow analysis and security. In *ASIAN Computing Science Conference, ASIAN'00*, Volume 1961 of *LNCS*. Springer.

Fowler, M., K. Beck, J. Brant, W. Opdyke, et D. Roberts (1999). *Refactoring: Improving the Design of Existing Code*. Addison Wesley.

Laprie, J., J. Arlat, J. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac, et P. Thévenod (1996). *Guide de la sûreté de fonctionnement, 2ème édition*. Cépaduès Editions.

Mallet, J. et S. Rouvrais (2008). Style-based model transformation for early extrafunctional analysis of distributed systems. In S. Becker, F. Plasil, et R. Reussner (Eds.), *4th. Intl. Conf. on the Quality of Software Architectures, QoSA 2008*, Volume 5281 of *LNCS*, pp. 55–70. Springer.

Saridakis, T. et V. Issarny (1999). Developing dependable systems using software architecture. In *Proceedings of the 1st Working IFIP Conference on Software Architecture*, pp. 83–104. Kluwer.

Zschaler, S. (2009). Formal specification of non-functional properties of component-based software systems: A semantic framework and some applications thereof. *Software and Systems Modelling (SoSyM)*. To appear.

Summary

For the development of critical software systems, it is worth adopting a rigorous process, including successive model transformations, which shall address, as soon as possible, non-functional properties. Accordingly, this paper offers to describe, from a logical standpoint, various properties relating to dependability, security and coupling, in order to direct two kinds of model transformation: (i) a structural transformation aiming at generating a first functional component architecture and (ii) a transformation aiming at incorporating, step by step, mechanisms into the whole system model with a view to maximizing requirement satisfaction.