Analyse de l'interopérabilité et de la conformité d'architectures logicielles

Anne-Lise Courbis, Thomas Lambolais, Hong-Viet Luong, Than-Liem Phan

Analyse de l'interopérabilité et de la conformité d'architectures logicielles

Anne-Lise Courbis, Thomas Lambolais, Hong-Viet Luong, Thanh-Liem Phan*

*LGI2P, école des mines d'Alès, France (e-mail : prénom.nom@mines-ales.fr).

Résumé. Ce travail concerne l'analyse de l'interopérabilité de systèmes dans les étapes amont de modélisation. Les systèmes à analyser sont modélisés en UML par des assemblages de composants dont les comportements sont définis par des machines d'états. Nous définissons l'interopérabilité des architectures selon deux niveaux : d'une part, l'absence de blocage, et d'autre part, le respect des services ainsi que des protocoles d'utilisation. Nous avons défini et implanté des relations de vérification de l'interopérabilité en faisant appel à une modélisation formelle des comportements des composants et des architectures par des systèmes de transitions étiquetées (LTS). Au problème de la construction d'architectures interopérables s'ajoute le problème de leur évolution. Pour cela, nous étudions sous quelles conditions il est possible de substituer un composant par un autre tout en garantissant l'interopérabilité de l'architecture.

1 Introduction

Le travail présenté porte sur la vérification de l'interopérabilité d'architectures logicielles UML en cours de conception ou d'évolution. Des travaux (Moisan et al. (2003); De Alfaro et Henzinger (2001)) abordent ce problème dans un contexte où il s'agit d'assembler des composants dont le comportement est défini soit par une spécification externe (cas de composants sur étagère considérés comme boîtes noires), soit par un modèle d'implantation (cas où l'architecte est également le concepteur du composant). Notre contexte d'étude est différent car nous nous intéressons à l'aide à la conception de systèmes où les architectures et les composants sont construits par raffinements et incréments successifs : ils sont définis à un niveau abstrait ; ils peuvent être incomplets et non déterministes. A l'inverse, la plupart des travaux portant sur l'analyse d'architectures font l'hypothèse que les composants sont déterministes. Nous mettons en évidence qu'il est possible de détecter des problèmes d'interopérabilité dans les étapes amont de spécification et de conception et qu'il n'est pas nécessaire d'attendre le modèle de la réalisation pour faire une étude d'interopérabilité. Notre approche est aussi valide pour les phases aval où les composants seraient définis en termes de modèles d'implantation.

Nous nous référons aux définitions de l'*interopérabilité* normalisées par l'ISO et l'ETSI dans le cadre du test d'interopérabilité (Monkewich (2006); Wiles (2003); Lynskey (2003)). Nous retenons qu'un système est interopérable si, sous certaines conditions, ses composants sont capables d'établir, maintenir et si nécessaire rompre un lien de communication, tout en garantissant un certain niveau de performance (Lynskey (2003)). L'interprétation minimale de

cette définition est qu'il est nécessaire que le système puisse s'exécuter sans blocage. C'est également cette notion que l'on retrouve dans la définition de Baldoni et al. (2009).

L'approche traditionnelle de construction d'architectures interopérables consiste à développer des composants conformes à des spécifications (des standards ou des préconisations) supposées interopérables. L'architecte s'attend à ce que le système soit de ce fait interopérable. Cette approche pose deux problèmes : vérifier la *conformité* des composants et s'assurer de l'interopérabilité de l'architecture. Bien que le terme *conformité* soit fréquemment utilisé, sa définition standard et ses implications ne sont pas toujours bien connues. Un équipement ou un composant est dit conforme à un standard s'il implante correctement tous les alinéas obligatoires de ce standard, certaines parties pouvant être optionnelles. Là aussi, nous nous référons à la norme ISO/IEC 9646-1 (1991) présentant le cadre du test de conformité. Cette notion a été par la suite formalisée par Brinksma et Scollo (1986) et Tretmans (1999) en traduisant le fait qu'une implantation est conforme à sa spécification si tout test que la spécification doit accepter, doit également être accepté par l'implantation. Il est important de comprendre que selon cette définition, une implantation conforme peut comporter davantage de fonctions ou services que sa spécification, mais peut également implanter moins de services lorsque certaines options sont omises. Ainsi, l'assemblage de composants conformes à des standards interopérables peut conduire à des systèmes non interopérables. La conformité est une condition nécessaire mais non suffisante pour garantir l'interopérabilité d'un système. En accord avec le modèle formel choisi par Brinksma et Scollo (1986), nous choisissons le formalisme des systèmes de transitions étiquetées (*Labelled Transition Systems* — LTS).

Le papier est structuré en quatre parties illustrées par des exemples se rapportant tous au même système. Le paragraphe 2 présente la relation de conformité que nous avons outillée afin de comparer des modèles comportementaux. Au paragraphe 3, nous définissons une méthode de vérification de l'interopérabilité d'architectures. Enfin, au paragraphe 4, nous présentons une relation de compatibilité qu'il est nécessaire d'associer à la relation de conformité et au prédicat d'interopérabilité pour garantir l'interopérabilité d'un système par substitution de composants.

2 Conformité de machines d'états UML

Nous nous intéressons à la mise œuvre de la définition formelle de la conformité proposée par Brinksma et Scollo (1986) afin de comparer les comportements de deux composants, exprimés sous forme de machines d'états UML. La relation de conformité comportementale ne pouvant être calculée directement sur les machines d'états, les comportements à analyser sont représentés par des LTS obtenus automatiquement par transformation des machines d'états. Nous ne donnons pas ici les détails de la transformation des machines d'états en LTS. Ils figurent dans Luong (2010). Dans cette partie, nous présentons les définitions de base du cadre des LTS et nous présentons la relation de conformité que nous avons implantée. Nous illustrons par un exemple le calcul de la conformité de deux LTS générés automatiquement à partir de deux machines d'états.

2.1 Caractéristiques principales des LTS

Les LTS ont été introduits comme modèle sémantique du parallélisme (Milne et Milner (1979)). Soit $\mathcal{P}$ un ensemble de noms d'états ou de noms de processus, et soit Act un ensemble de noms d'actions, avec $Act = \mathcal{L} \cup \{\tau\}$, où $\mathcal{L}$ est l'ensemble des actions visibles et τ une action silencieuse.

Définition 1 (LTS) *Un LTS $\langle P, A, \rightarrow, p\rangle$ est un quadruplet muni d'un ensemble non vide $P \subseteq \mathcal{P}$ d'états, d'un ensemble $A \subseteq Act$ d'actions, d'une relation de transitions $\rightarrow \subseteq P \times A \times P$, et d'un état initial $p \in P$. Les actions sont définies sur $A = L \cup \{\tau\}$, où $L \subseteq \mathcal{L}$.*

On désigne par p le LTS $\langle P, A, \rightarrow, p\rangle$. Remarquons que le terme *actions* utilisé ici se rapproche des concepts UML d'actions, d'activités mais aussi d'événements. Une action désigne en LTS un point de communication visible permettant une synchronisation entre un LTS et son environnement. On désigne par $Tr(p)$ l'ensemble des traces du LTS p, où une trace est une exécution (éventuellement partielle) d'actions observables. L'ensemble des états qui peuvent être atteints à partir de p après une trace σ est désigné par p after σ. $Out(p, \sigma)$ est l'ensemble des actions observables de p après une trace σ.

Définition 2 (Ensemble d'acceptance) *L'ensemble d'acceptance de p après la trace σ, noté $Acc(p, \sigma)$, est un ensemble d'ensembles d'actions représentant les actions que p doit accepter après la trace σ. $Acc(p, \sigma) = \{X \mid \exists p' \in p$ after $\sigma.\ X = Out(p', \varepsilon)\}$.*

Cette notion est au cœur de la relation de conformité car elle identifie, pour toute situation, ce qu'un composant *doit* ou *peut* accepter. Cela permet de distinguer les actions obligatoires des actions optionnelles. Par exemple, l'ensemble d'acceptance $\{\{a, b\}, \{b\}\}$ signifie que les actions a et b peuvent être acceptées et que b est une action obligatoirement acceptée (donc a peut être refusée).

2.2 Relation de conformité entre LTS

Telle que formalisée par Brinksma et Scollo (1986), la relation de conformité entre une implantation et sa spécification établit que toute action que la spécification doit accepter après une trace donnée, doit être acceptée par l'implantation après la même trace.

Définition 3 (Conformité) q conf p *si* $\forall \sigma \in Tr(p),\ Acc(q, \sigma) \subset\subset Acc(p, \sigma)$.

où $\subset\subset$ est une relation d'inclusion d'ensembles d'ensembles : $A \subset\subset B$ si, pour tout a de A, il existe b dans B tel que $b \subseteq a$.

Si q conf p, le processus q est plus déterministe que le processus p.

Bien que la conformité soit une relation assez ancienne, aucun algorithme de vérification formelle n'avait été proposé pour la vérifier sur des LTS. Dans Luong et al. (2008); Luong (2010),nous proposons un algorithme basé sur les travaux de Cleaveland et ceux de Khendek. Il est implanté dans un outil que nous avons appelé IDCM (*Incremental Development of Conformant Models*) réalisant également la transformation automatique des machines d'états en LTS. Ces travaux sont intégrés dans l'environnement TopCased (Farail et al. (2006)).

Plusieurs aspects de cette relation doivent être soulignés car leurs conséquences sont inattendues, bien que la définition 3 soit en accord avec celle de l'ISO. En premier lieu, cette relation n'est pas transitive. Elle ne peut donc pas être utilisée dans une démarche de construction incrémentale. Nous avons été ainsi amenés à utiliser d'autres relations (extension, réduction). Une autre propriété de la relation de conformité est que, bien qu'une implantation conforme à sa spécification soit plus déterministe que sa spécification, elle peut également offrir davantage de traces, à la condition que ces nouveaux comportements ne perturbent pas le comportement spécifié dans le modèle de référence. La conséquence est que dans un contexte d'interopérabilité, si un composant est remplacé par un composant qui lui est conforme, rien ne garantit l'interopérabilité du nouveau système. Nous étudierons et illustrerons cette propriété dans la partie relative aux architectures.

2.3 Illustration de la vérification de conformité de machines d'états

Considérons la spécification d'un composant appelé *TaskManagement* dont le but est de réaliser des traitements sur des tâches qu'il reçoit et de les renvoyer une fois traitées. Les tâches peuvent être réalisées en interne par *TaskManagement* ou envoyées à un composant externalisé. Les raisons de ce choix ne sont pas spécifiées à ce stade de la modélisation. Le comportement du composant *TaskManagement* est défini par une machine d'états donnée à la figure 1(a). Ses interfaces sont explicitées dans la figure 3(a).

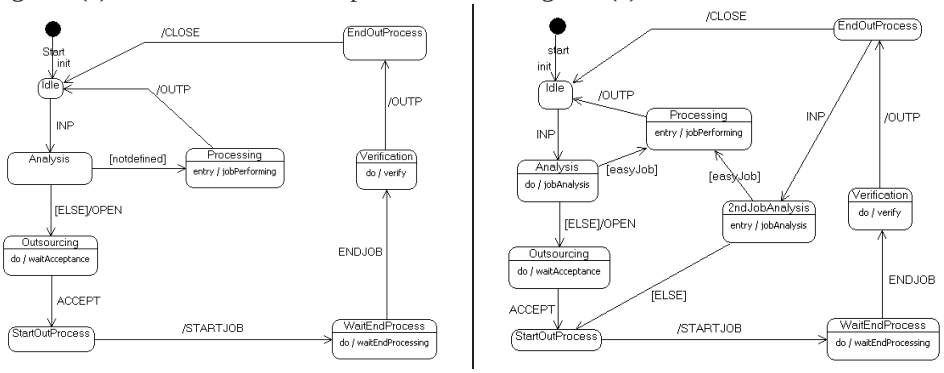

FIG. 1 – *(a) TMSpec*$_{SM}$ *(b) TMImp*$_{SM}$

La figure 1(b) présente un second modèle de machine d'états correspondant à une définiton plus détaillée du comportement du composant *TaskManagement*. La machine *TMImp*$_{SM}$ est ainsi vue comme un modèle de comportement d'implantation. Dans l'implantation, on fixe les conditions d'externalisation du traitement selon un prédicat *easyJob*. Ces machines d'états sont transformées en LTS par notre outil IDCM (cf. figure 2 (a) et (b)). Notons que que dans les représentations graphiques des LTS, l'action i correspond à τ. On vérifie sous IDCM que le LTS de l'implantation est conforme à celui de la spécification : « TMImp$_{LTS}$ conf TMSpec$_{LTS}$ ».

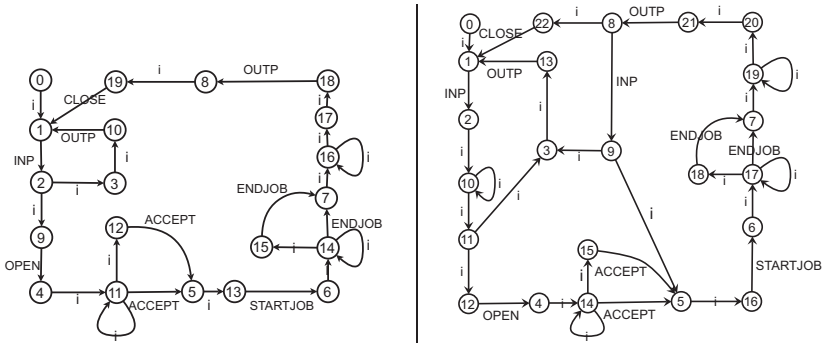

FIG. 2 – *(a) TMSpec$_{LTS}$* *(b) TMImp$_{LTS}$*

3 Analyse de l'interopérabilité d'une architecture

Grâce à la relation de conformité, nous pouvons étudier l'interopérabilité d'une architecture. En accord avec les normes ISO et ETSI, et selon Baldoni et al. (2009), un système est *interopérable* s'il ne présente pas de blocage (ou *dead-lock*). Ceci signifie que tous ses points d'arrêt (états dont l'ensemble d'acceptance est vide) sont des états terminaux.

Définition 4 (intéropérabilité) *Un système s de composants p_1, p_2, …, p_n est* interopérable *si tous ses points d'arrêts coïncident avec des états terminaux p_1, p_2, …, et p_n : pour toute trace σ de s correspondant aux sous-traces σ_1 de p_1, …, σ_n de p_n, $Acc(s, \sigma) = \{\varnothing\} \Rightarrow Acc(p_1, \sigma_1) = \cdots = Acc(p_n, \sigma_n) = \{\varnothing\}$.*

L'analyse d'interopérabilité demande une sémantique formelle de l'architecture. Celle-ci est définie en langage LOTOS NT (Sighiereanu (1999)) qui est une extension du langage LOTOS (ISO/IEC 8807 (1989)), où les composants UML correspondent à des processus LOTOS NT, le langage LOTOS NT étant lui même traduit en LOTOS (ISO/IEC 8807 (1989)). La sémantique opérationnelle de LOTOS est donnée en LTS. Nous montrons sur un exemple comment se fait la transformation d'une architecture en LOTOS NT puis LOTOS, ainsi que son analyse d'interopérabilité. Nous mettons ensuite en évidence, sur le même exemple, les problèmes soulevés par la substitution d'un composant par un composant auquel il est conforme.

3.1 Spécification d'architectures : modélisation et analyse

Considérons le composant *TaskManagement* dont une machine d'état est représentée à la figure 1 (a). Dans le cas d'une externalisation du traitement de la tâche, *TaskManagement* a été conçu pour travailler avec le composant *TaskTreatment* (figure 3 (b)). Considérons l'architecture A_0 (figure 3 (a)) formée par ces deux composants. L'interface fournie de l'architecture possède un service (INP), représentant la demande de traitement d'une tâche. Son interface requise possède un service (OUTP), représentant la restitution de la tâche venant d'être traitée.

A_0 est alors traduite en LOTOS NT. Nous montrons ici la spécification LOTOS en résulte (figure 4). Même si les concepts des langages de modélisation UML et LOTOS NT diffèrent, on retrouve des notions communes : les composants UML sont associés à des processus LOTOS ; les différents types d'événements UML sont à comparer à la notion d'action LOTOS ; la

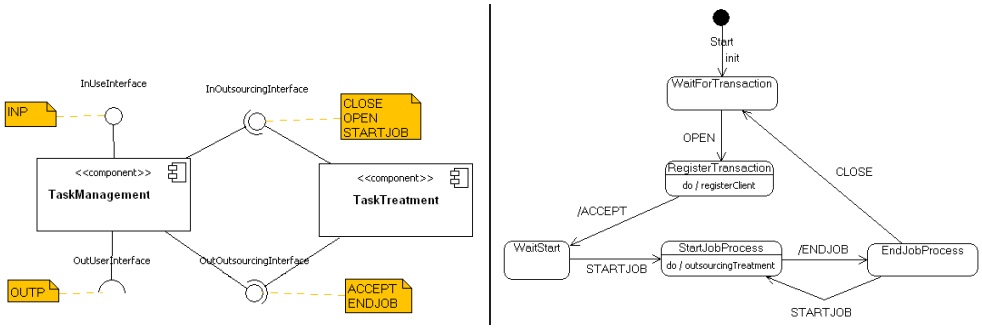

FIG. 3 – *(a) Architecture A0* *(b) TTSpec$_{SM}$*

connexion entre composants est à comparer à la notion de composition parallèle. Par exemple, en LOTOS, l'expression $P|[\mathcal{A}]|Q$, où P et Q sont deux processus et $\mathcal{A}$ un ensemble d'actions observables, signifie que P et Q sont *synchronisés* sur toutes les actions de $\mathcal{A}$. Dans le cas de communications asynchrones, la transformation en LOTOS nécessiterait d'utiliser explicitement des processus supplémentaires représentant des *buffers*.

```
specification A0[INP, OUTP]:noexit
behaviour
hide [OPEN,START,CLOSE,ACCEPT,ENDJOB] in
    (TMSpec_LTS[INP, OUTP, OPEN,START,CLOSE,ACCEPT,ENDJOB]
      |[OPEN,START,CLOSE,ACCEPT,ENDJOB]|
     TTSpec_LTS[OPEN,START,CLOSE,ACCEPT,ENDJOB] )
endspec
```

FIG. 4 – *Description* LOTOS *de l'architecture* A_0

Nous obtenons ainsi l'architecture en LOTOS. L'automatisation de l'obtention de spécifications LOTOS NT à partir d'UML est en cours de développement. Le LTS obtenu pour A_0 compte environ 70 états et 150 transitions. On vérifie sous CADP que l'architecture A_0 n'a pas de blocage.

3.2 Raffinement d'architectures : modélisation et analyse

Considérons l'architecture A_1 obtenue à partir de A_0 en raffinant le composant *TaskManagement* en un modèle d'implantation par exemple. La machine d'états de *TaskManagement* est celle de la figure 1(b). Le LTS obtenu pour A_1 compte environ 125 états et 270 transitions. En revanche, l'analyse de A_1 sous CADP détecte des blocages. Bien que les composants soient conformes à leur spécification, A_1 n'est donc pas interopérable. Une trace provoquant un blocage est : INP ; OPEN ; ACCEPT ; STARTJOB ; ENDJOB ; OUTP ; INP ; OUTP ; INP. Son analyse montre que la première tâche est externalisée, la seconde non, et la troisième est à nouveau externalisée mais *TaskTreatment* ne peut l'accepter car il attend une fermeture de transaction. Le comportement de *TaskManagement* est parconséquent rectifié : on appelle TMImpC$_{SM}$ la machine d'états corrigée en ajoutant une demande de fermeture de transaction CLOSE sur la transition sortant de l'état 2ndJobAnalysis vers l'état Processing.

On constate que la substitution d'un composant par un composant conforme dans une architecture interopérable peut provoquer des problèmes d'interopérabilité. En effet, la relation

de conformité assure que toute séquence de la spécification est respectée par l'implantation, mais cette dernière peut faire plus que sa spécification et introduire de nouvelles traces non analysées. Des blocages peuvent survenir dans ces nouvelles traces. Dans l'exemple choisi, l'architecture A_1 peut traiter plusieurs tâches dans la même transaction d'externalisation, ce qui correspond à de nouvelles traces sources de blocages.

Cette étude soulève deux problèmes. Soit $A(C, D)$ une architecture interopérable formée de deux composants C et D. (1) Existe-t-il une relation entre C et l'un de ses raffinements R garantissant l'interopérabilité de $A(R, D)$? (2) Existe-t-il une relation entre les composants assemblés R et D garantissant l'interopérabilité de $A(R, D)$?

4 Construction d'architectures interopérables par substitution

Nous avons étudié comment analyser l'interopérabilité d'un système. Dans cette partie, nous souhaitons étudier quelle relation doit exister entre des spécifications de composants et leur implantation de façon à garantir l'interopérabilité du système réalisé.

4.1 Substituabilité

Si on remplace un composant C par un autre composant R, il est nécessaire de vérifier non seulement que R fait ce que doit faire C, mais aussi de vérifier qu'il ne présente pas de nouveaux comportements observables car ils correspondent aux nouvelles traces non analysées par la relation de conformité. C'est en ce sens que la relation de conformité n'est pas une relation suffisante. D'un point de vue algébrique, des relations satisfaisant des propriétés de substituabilité sont des relations de congruence. Milne et Milner (1979) a défini des relations de congruence observationnelles, notées "$=$". Il faut noter que la relation d'équivalence observationnelle, implantée dans de nombreux outils de *model checking*, n'est pas une congruence. Nous avons choisi comme relations cext et cred définies dans Brinksma et Scollo (1986) et Leduc (1992). Ces relations présentent un intérêt pour nous car elles sont basées sur la relation de conformité et nous avons pu les implanter dans IDCM.

Définition 5 (cred**,** cext**)** *Soient p et q deux LTS,*
q cred $p \Leftrightarrow$ *pour tout contexte* $C[.], C[q]$ red $C[p]$,
q cext $p \Leftrightarrow$ *pour tout contexte* $C[.], C[q]$ ext $C[p]$.

Dans cette définition, le contexte fait référence à l'ensemble des composants qui interagissent avec le composant sous analyse. Les relations red et ext sont des relations de conformité avec en plus, respectivement, des réductions de trace et des extensions de traces entre le nouveau composant et le composant substitué. Pour illustrer la relation cred, considérons la machine d'états TMImp2$_{SM}$ de la figure 5. On vérifie que TMImp2$_{LTS}$ cred TMSpec$_{LTS}$. En effet, TMSpec$_{LTS}$ est conforme à TMSpec$_{LTS}$ et n'a pas plus de traces. La relation cred vérifie en plus que tous les états *stables* sont préservés. Un état est qualifié de *stable* si la machine ne peut pas le quitter par elle-même (après un *complete event* par exemple, ou encore un *time event* ou un *change event*).

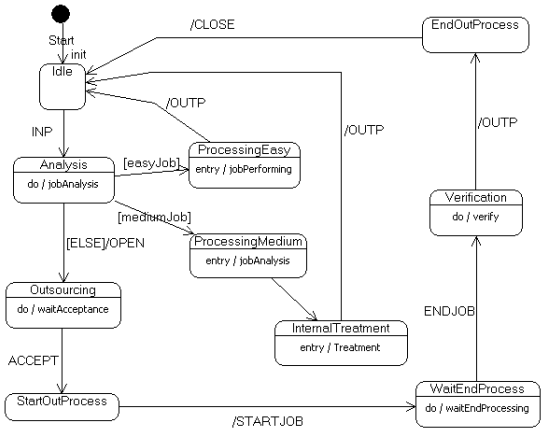

FIG. 5 – *TMImp2$_{SM}$: seconde implantation de la machine d'états de TaskManagement*

On appelle A2 l'architecture de même structure que A0 (cf. figure 3(a)), dans laquelle TaskManagement a pour comportement la machine d'états TMImp2$_{SM}$. On vérifie que l'architecture A2 est interopérable.

Les relations de congruence peuvent être utilisées pour vérifier l'interopérabilité. Ce sont néanmoins des relations fortes et on souhaiterait définir une relation plus faible. Pour cette raison, nous introduisons la notion de compatibilité.

4.2 Compatibilité

On se place dans le contexte où il n'y a pas de relation de congruence entre le nouveau composant et le composant substitué, comme c'était le cas entre TMImpC$_{SM}$ et TMSpec$_{SM}$. On vérifie alors si le composant est compatible avec l'environnement dans lequel il va se trouver, c'est-à-dire l'ensemble des composants de l'architecture auxquels il va être connecté. L'environnement peut être considéré comme un sous-système connecté au composant à analyser, sous-système dont le comportement peut être déduit de celui de ses composants (par une traduction LOTOS comme précédemment). Le comportement du sous-système peut donc être traduit en LTS. On ramène ainsi le problème à l'analyse de compatibilité entre deux LTS : celui de l'environnement et celui du composant à étudier.

Définition 6 (Compatibilité) *Deux LTS p et q sont compatibles sur un ensemble d'actions S, ce qu'on écrit p comp_S q, si pour toute trace $\sigma \in S^*$, $Acc(p/S, \sigma)$ comp $Acc(q/S, \sigma)$.*

où p/S est le LTS obtenu à partir de p en masquant toute action qui n'est pas dans S. Le masquage consiste à remplacer l'étiquette d'une transition par l'action silencieuse τ. Deux ensembles A and B sont compatibles, ce qu'on écrit A comp B, si $\forall x \in A, \forall y \in B, x \cap y \neq \varnothing$.

Théorème 1 *Si p comp_S q, alors $p|[S]|q$ est interopérable.*

En effet, après toute trace $\sigma \in S^*$, $Acc(p/S, \sigma) \neq \{\varnothing\}$ et $Acc(p/S, \sigma)$ comp $Acc(q/S, \sigma) \Rightarrow Acc(q/S, \sigma) \neq \{\varnothing\}$. Donc pour toutes traces σ_p et σ_q telles que $p' \in p$ after $\sigma_p \Leftrightarrow p' \in p/S$ after σ et $q' \in q$ after $\sigma_q \Leftrightarrow q' \in q/S$ after σ, $Acc(p, \sigma_p) \neq \{\varnothing\} \Rightarrow Acc(q, \sigma_q) \neq \{\varnothing\}$.

On peut ainsi démontrer par exemple que TMImpC_{LTS} est compatible avec le composant TTImp_{LTS}. Les actions qui ne sont pas masquées sont celles des interfaces InOutSourcing et OutOutsourcing : S = {OPEN, CLOSE, STARTJOB, ACCEPT, ENDJOB}. Pour toutes les traces, les ensembles d'acceptance des deux LTS sont les mêmes (donc compatibles), à l'exception de ceux obtenus après la trace t = OPEN ; ACCEPT ; END. $Acc(\mathrm{TMImpC}_{LTS}/S, t)$ = {{CLOSE},{CLOSE, STARTJOB}} et $Acc(\mathrm{TTImp}_{LTS}/S, t)$ = {{CLOSE, STARTJOB}}. Ces deux ensembles sont compatibles, donc les deux composants sont déclarés compatibles.

La compatibilité assure un premier niveau d'interopérabilité. Le second niveau consiste à s'assurer que le nouveau système offre toujours les mêmes services sous les mêmes conditions d'utilisation (c'est à dire pour les mêmes traces). c'est la relation de conformité entre architectures qui permet de vérifier ce second niveau.

5 Conclusion

Nous avons défini la notion d'interopérabilité d'architectures modélisées sous forme de diagrammes de composants UML, chaque composant étant défini par une machine d'états. Nous avons implanté sous TopCased la transformation de modèles UML en LTS ainsi que les relations de compatibilité, interopérabilité et conformité. Deux voies sont possibles pour vérifier l'interopérabilité d'une architecture où un composant aurait été substitué : analyser la compatibilité du composant avec son environnement ou recalculer sous forme de LTS le comportement du système global et détecter les blocages. Une application de ce travail est la recherche automatique de composants de substitution dans une bibliothèque, que ce soit pour réparer un composant défectueux ou augmenter la performance du système. Des applications de ce travail seraient la construction automatique d'architectures à partir de spécifications comportementales, la construction se faisant de façon incrémentale par ajouts de composants compatibles. Une fois les architectures élaborées, on peut les comparer et sélectionner celles répondant au mieux à des critères prédéfinis comme par exemple la minimisation du nombre de composants, de connexions ou d'interfaces, ou encore l'utilisation de composants fiables.

Références

Baldoni, M., C. Baroglio, A. K. Chopra, N. Desai, V. Patti, et M. P. Singh (2009). Choice, interoperability, and conformance in interaction protocols and service choreographies. In S. Decker, Sichman et Castelfranchi (Eds.), *8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*. Budapest, Hungary.

Brinksma, E. et G. Scollo (1986). Formal Notions of Implementation and Conformance in LOTOS. Technical Report INF-86-13, Dept. of Informatics, Twente University of Technology.

De Alfaro, L. et T. Henzinger (2001). Interface automata. *ACM SIGSOFT Software Engineering Notes 26*(5), 120.

Farail, P., P. Gaufillet, A. Canals, C. L. Camus, D. Sciamma, P. Michel, X. Crégut, et M. Pantel (2006). The TOPCASED project : a toolkit in open source for critical aeronautic systems design. *Embedded Real Time Software (ERTS)* (781), 54–59.

ISO/IEC 8807 (1989). LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève.

ISO/IEC 9646-1 (1991). Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1 : General concepts.

Leduc, G. (1992). A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems 25*(1), 23–41.

Luong, H., T. Lambolais, et A. Courbis (2008). Implementation of the conformance relation for incremental development of behavioural models. In K. Czarnecki (Ed.), *MoDELS 2008*, Volume 5301/2009 of *LNCS*, pp. 356–370. Springer-Verlag.

Luong, H.-V. (2010). *Construction Incrémentale de Spécifications de Systèmes Critiques intégrant des Procédures de Vérification*. Ph. D. thesis, Université de Toulouse III.

Lynskey, E. (2003). Importance of last mile interoperability. OptiCom 2003, 1st Internation Workshop on Community Networks and FTTH/P/x.

Milne, G. et R. Milner (1979). Concurrent processes and their syntax. *Journal of the ACM (JACM) 26*(2), 302–321.

Moisan, S., A. Ressouche, et J. Rigault (2003). Behavioral substitutability in component frameworks : A formal approach. *SAVCBS 2003 Specification and Verification of Component-Based Systems*, 22.

Monkewich, O. (2006). Conformance and interoperability testing tutorial. UIT-T SG17, Telecommunication Languages and Softwares. Second Informal Workshop. Switzerland, Genève.

Sighiereanu, M. (1999). *Contribution à la définition et à l'implémentation du langage "Extended LOTOS"*. Ph. D. thesis, université Joseph Fourrier, Grenoble.

Tretmans, J. (1999). Testing concurrent systems : A formal approach. In S. M. Jos C.M. Baeten (Ed.), *CONCUR'99 Concurrency Theory*, Volume 1664 of *LNCS*, pp. 46–65. Springer-Verlag, Berlin Heidelberg.

Wiles, A. (2003). Relevance of conformance testing for interoperability testing. ACATS ATS-CONF. Stuttgart.

Summary

Our work deals with the analysis of the interoperability of systems during high level modelling steps. Systems to be analysed are modelled as UML architectures in terms of assembly of components, the behaviour of which is defined by State Machines. We define two interoperability levels: the system is deadlock free, and services and usage protocols are preserved. We have defined and implemented relations to verify interoperability on Labelled Transition Systems (LTS). We also address the problem of architecture evolution. For that purpose, we study under which conditions a component can be substituted by another one, while maintaining architecture interoperability.