Protocoles d'utilisation de composants : spécification et analyse en Kmelia

Pascal André, Gilles Ardourel, Christian Attiogbé

Protocoles d'utilisation de composants

Spécification et analyse en Kmelia

Pascal André — Gilles Ardourel — Christian Attiogbé

LINA - FRE CNRS 2729
2, rue de la Houssinière, BP 92208, F-44322 Nantes Cedex 3
(Pascal.Andre,Gilles.Ardourel,Christian.Attiogbe)@univ-nantes.fr

RÉSUMÉ. *L'approche des composants logiciels constitue une solution pour développer des logiciels de façon modulaire et en (ré) utilisant l'existant. Cependant, ce n'est pas facile de trouver des composants appropriés à un besoin spécifique et pouvoir se servir convenablement des services offerts. Nous proposons de répondre à la fois à la problématique de la description de "mode d'emploi" pour les composants et à celle de l'expression de contraintes d'utilisation par la spécification de multiples modes d'emploi qui peuvent être utilisés comme des services quelconques. Pour s'assurer que les protocoles sont exempts d'anomalies, nous avons élaboré une technique d'analyse de leur cohérence. Notre étude et les expérimentations sont faites sur la base du modèle à composants* Kmelia.

ABSTRACT. *The Component-Based Engineering is an approach for developing software in a modular way by reusing existing software pieces. However finding software components that fit a specific requirement and using correctly the provided services are quite difficult tasks. In this paper, we propose to answer both the issues of expressing a component's user guide and that of expressing usage constraints by specifying multiple user guides, which can be used as services. To ensure the correctness of the protocols, a technique of consistency analysis is provided. This study is experimented on the* Kmelia *component model.*

MOTS-CLÉS : *protocoles, modèles de composants, propriétés de correction et cohérence.*

KEYWORDS: *protocols, component models, correctness and consistency properties.*

1. Introduction

L'approche des composants logiciels constitue une solution pour développer des logiciels de façon modulaire et en (ré)utilisant l'existant. Cependant, ce n'est pas facile de trouver des composants appropriés à un besoin spécifique et pouvoir s'en servir convenablement. Plusieurs modèles à composants proposent un langage de description d'interface (IDL) qui fournit la documentation utilisateur des composants sous forme d'interface. Le langage IDL permet de décrire les fonctionnalités offertes dans l'interface du composant, ses services offerts, et pour chacun de ces services le profil des opérations. Cependant, selon les modèles à composants, l'interface proposée à l'utilisateur est plus ou moins expressive et inclut ou non plusieurs aspects parmi lesquels :

– la présence de services requis dans l'interface (ceux nécessaires à la réalisation des services offerts),

– le support multi-interface (plusieurs interfaces disponibles, en général liées à des vues différentes),

– les contrats d'utilisation des services (indiquant les droits et devoirs du client et du fournisseur),

– les règles ou protocoles d'utilisation des services (précisant les enchaînements licites de services).

Dans cet article, nous nous intéressons plus particulièrement aux règles d'utilisation des services offerts par les composants et à leur vérification statique. Nous avons étudié les moyens de définir simplement des modes d'emploi des services d'un composant ainsi que différentes analyses à réaliser pour garantir la cohérence des protocoles et donc des composants. Pour ce travail, nous nous appuyons sur le modèle Kmelia dont le noyau a été décrit dans (Attiogbé *et al.*, 2006). Ce modèle permet de définir des composants d'une manière abstraite (vis-à-vis des modèles opérationnels) en se focalisant sur les services et les interfaces, l'objectif sous-jacent étant la vérification statique d'assemblage hétérogènes de composants.

Les principales contributions de ce travail sont : *i)* la proposition d'une méthode de description des modes d'emploi des composants qui reprend la description des services offerts et *ii)* une technique d'analyse de la cohérence des protocoles.

L'article est organisé de la façon suivante : dans la section 2 nous exposons nos motivations pour l'usage des protocoles et les solutions existantes. Nous détaillons l'approche des protocoles Kmelia dans la section 3. La section 4 est consacrée à la conception et à la vérification des protocoles : l'analyse de cohérence des protocoles et l'interaction entre les composants dont les services sont contrôlés par des protocoles. Nous concluons dans la section 5 et nous indiquons les perspectives de ce travail.

2. Le protocole, un mode d'emploi des composants

Cette section discute de l'usage de protocoles pour documenter l'utilisation des composants logiciels.

2.1. *Motivations*

Pour se servir d'un composant, l'utilisateur recherche dans la documentation de l'interface du composant la description individuelle des services offerts et celle des services qui l'intéressent. Les dépendances et les contraintes d'enchaînement pouvant exister entre les services sont rarement décrites explicitement ou se trouvent réparties dans la documentation. Sans assistance, l'utilisateur pourrait invoquer de manière inappropriée ces services (utiliser n'importe quel service dans n'importe quel ordre) en se basant sur la description individuelle des services. Si des interdépendances sont décrites dans la spécification des services, par exemple sous forme de contrats, alors les préconditions risquent d'être complexes en prenant en compte les enchaînements. Laisser à l'utilisateur le soin de l'analyse des cas possibles n'est guère envisageable, en raison de l'explosion combinatoire. Enfin, il y a des risques de confusion liés au fait que les contraintes soient implicites (cachées dans les descriptions des services) et non explicites. Pour toutes ces raisons, il est nécessaire de définir des règles d'usage et d'enchaînement de services du composant de manière explicite dans l'interface même du composant. Nous proposons de le faire au travers de *protocoles*.

2.2. *Divers usages de la notion de protocole*

La notion de protocole dans les modèles à composants n'est pas nouvelle mais sa définition ou son utilisation varie d'un modèle (langage) à l'autre. Pour clarifier le concept, nous proposons une brève comparaison[1] entre les approches en retenant quatre critères :

– Le contenu des protocoles : invocations de services, calculs, envois de messages, structures de contrôle (conditionnnelles, répétitives).

– L'unité de rattachement des protocoles : le composant, l'interface, le service, le connecteur ou l'architecture.

– Le formalisme de description de la dynamique des protocoles : automates, machines à états, expressions régulières, etc.

– Les techniques de description et de preuve des propriétés des protocoles : logique temporelle, langage de marques, algorithmes, etc.

Cette synthèse indique aussi la présence d'outils de description, de vérification des protocoles (*model checking*, *theorem proving*), de raffinement ou de génération de code.

Le contenu est relatif aux actions élémentaires, un message correspondant à un échange atomique : service, opération, méthode ou simple communication selon le modèle. Le formalisme est souvent une variante des algèbres de processus (CSP), des machines à états (FSM, LTS, STS, statecharts) ou des expressions régulières. Les pro-

1. une version plus détaillée est accessible à
`lina.atlanstic.net/fr/equipes/team10/Kmelia/protocolesCBSE.html`

priétés sont définies par des expressions (variantes de logique temporelle) ou par des algorithmes. Elles sont établies par des méthodes de test, de démonstration automatique ou de vérification de modèles.

Approche	Critères				Outils
	Contenu	Rattachement	Formalisme	Propriété	
Allen (Allen *et al.*, 1997)	msg, calculs	connecteur	CSP	traces, TL	X
Attie (Attie *et al.*, 2003)	msg, calculs	interface	automates	CTL	-
Becker (Becker *et al.*, 2004)	msg, calculs	composant	CC-FSM	algo.	-
Beugnard (Beugnard *et al.*, 1999)	java	interface	-	-	-
Canal (Canal *et al.*, 2003)	msg	interface	exp. regu.	π-calcul	-
Inverardi (Inverardi *et al.*, 2000)	opérations	composant	CHAM	algo.	
Henziger (de Alfaro *et al.*, 2001)	msg	interface	automates	equiv.	-
Kramer[a]	msg, calculs	composant	LTSA	LTSA, LTL	X
Moizan (Moisan *et al.*, 2003)	evts	classe, methode	statecharts	LTL	-
Plasil (Plasil *et al.*, 2002)	msg	comp., interface	exp. regu.	algo.	X
Royer (Pavel *et al.*, 2005)	méthode	composant	STS	-	
Sudölt (Südholt, 2005)	méthode	composant	exp. non reg	algo.	-
Schmidt (Schmidt, 2003)	msg	interface	FSM	-	X
UML 2.0 (OMG, 2005)	msg, actions	composant	statecharts	-	
Yellin (Yellin *et al.*, 1997)	msg	interface	FSM	algo.	-
Zimmermann[b]	opération	composant	gram. alg.	algo.	-
Kmelia (Attiogbé *et al.*, 2006)	msg, actions	comp., services	FSM	export	X

[a] (Giannakopoulou *et al.*, 1999) [b] (Zimmermann *et al.*, 2006)

Tableau 1. *Synthèse comparative des protocoles dans les modèles à composants*

On peut classer sommairement les approches en trois catégories en accordant plus de poids au critère "unité de rattachement" qui est un critère structurant :

1) Celles dont le protocole définit le « cycle de vie » du composant (Becker, Inveradi, Kramer, Moizan, Royer, Sudölt, UML, Zimmermann). Un unique protocole est associé au composant (ou à son unique interface, ce qui est équivalent). Dans ces approches, le composant est un processus et les services sont soit atomiques (messages) soit définis par un comportement spécifique (Moizan, UML).

2) Celles dont le protocole définit le « cycle de vie » d'une vue du composant (Allen, Attie, Beugnard, Canal, Henziger, Plasil, Schmidt, Yellin). Un protocole est associé à une interface et plusieurs interfaces coexistent. Les interfaces peuvent être limitées à un seul composant cible (assimilable à un connecteur) ou pas.

3) Celles dont le protocole définit un usage du composant (Kmelia). Plusieurs protocoles coexistent au sein du composant dans une ou plusieurs interfaces.

Sur ce critère, les approches retenues ne sont pas radicalement différentes en terme d'expressivité mais elle le sont en termes de facilité d'expression et d'utilisation car c'est au concepteur d'architectures de gérer la complexité induite, au développeur de la résoudre et à l'utilisateur de la comprendre. Prenons l'exemple du multi-protocole généralisé (plusieurs protocoles par composants et un protocole par service) pour l'illustrer la nuance. Avec un modèle à composant simple (qui autorise une seule interface par composant, un seul protocole et des services réduits à des profils d'opération) on peut représenter (a) des protocoles de services (en modélisant un seul service par composant) (b) plusieurs protocoles ou bien plusieurs interfaces de composants (par composition de composants) et (c) différents connecteurs (par spécialisation des com-

posants). Dans ce cas, le concepteur d'architectures devra encapsuler les protocoles dans des composants composites, gérer la cohérence des interfaces (proches des problèmes d'héritage en objet) et définir des bibliothèques de composants spécialisés. Il en résulte donc une certaine lourdeur des modélisations. Autrement dit, certains usages favorisent plus la documentation du composant que d'autres. Par exemple, les approches de la catégorie 1 (et une partie de celles de la catégorie 2) interprètent le protocole plutôt comme une contrainte pour l'implantation du composant qu'un mode d'emploi. Le rattachement du protocole à un connecteur permet à certains modèles de se rapprocher du point de vue de l'orchestration de services, dans laquelle l'enchaînement de messages a un sens indépendamment des composants concernés et peut par conséquent être décrite en dehors d'un composant. L'utilisation de protocole en tant que mode d'emploi embarqué dans un composant en facilite la documentation et l'intégration dans un cadre non déterminé à l'avance.

De façon orthogonale, les protocoles peuvent s'inscrire dans une démarche contractuelle en s'associant avec des assertions (Becker, Beugnard, Kmelia, Schmidt, UML-OCL, Schmidt) et des contraintes non-fonctionnelles comme la qualité de service (Becker, Beugnard). Il est évident que ces aspects sont importants pour la documentation et la vérification d'assemblage, de même que le traitement de l'adaptation (Allen, Becker, Canal, Kmelia, Schmidt, Yellin).

Notons que certaines approches traitent de la récursion (Sudölt, Zimmerman), du sous-typage de protocoles (Becker, Plasil, Yellin) ou du raffinement (Henziger, Moizan). D'autres relient les protocoles à des implantations de composant (Allen, Kramer, Royer) ou proche de modèles comme CORBA (Canal), ce qui leur donne à la fois une certaine crédibilité mais aussi des structures moins abstraites. Mais ces aspects ne concernent pas directement la documentation et l'intégration de composants.

2.3. *Protocole d'utilisation*

Un *protocole* définit un mode d'emploi de son composant. Plus précisément il définit des règles d'enchaînement de services du composant : c'est un protocole d'utilisation. En explicitant la dynamique de l'ensemble des services, on réduit la combinatoire d'utilisation des services. Du point de vue du fournisseur, les protocoles permettent d'alléger la description individuelle des services et dégagent des services les contraintes d'enchaînement et des conditions générales du composant. Le protocole est défini à un niveau de granularité plus large que celui des services, ce qui favorise l'abstraction, la réutilisabilité et la modularité des composants.

Prenons quelques exemples pour illustrer l'emploi des protocoles d'utilisation. Pour utiliser correctement la lecture et l'écriture dans un serveur de fichiers, il faut d'abord ouvrir les fichiers et ne pas oublier de les refermer, il faut aussi s'assurer de l'exclusivité lors de l'écriture. Dans un guichet automatique bancaire (GAB) pour réaliser des opérations (services) de retrait, consultation, dépôt, transfert, l'usager doit au préalable s'identifier (se connecter), on doit aussi établir les liaisons distantes né-

cessaires au bon fonctionnement des services. Par ailleurs tous les usagers n'ont pas les mêmes droits et possibilités d'utilisation : un usager quelconque peut retirer de l'argent, un usager ayant un compte dans la banque du guichet peut en plus réaliser des opérations sur compte, un personnel de la banque peut réaliser des services de maintenance. Dans un protocole réseau tel que TCP, les échanges sont assimilables à des services et l'ordre des échanges (envois, accusés, connexions, déconnexions, renvois...) est régi par des règles bien précises. Dans une architecture pair-à-pair, un nœud est serveur et client de services comme le partage de fichier, l'échange de données ou la téléphonie.

En résumé, les protocoles peuvent servir à contrôler des sessions, des processus ou des classes de services en plus de l'aspect « protocoles de communication » qui détaillent les principes de bonne communication entre participants (Allen *et al.*, 1997). Dans la suite nous proposons une méthode de prise en compte de ces protocoles dans le modèle Kmelia.

3. Protocoles dans Kmelia

Après une brève description du modèle Kmelia, nous montrons comment décrire en Kmelia des protocoles de composants répondant aux exigences de lisibilité, de flexibilité mais aussi de simplicité de mise en œuvre pour le concepteur de composants.

3.1. *Kmelia*

Kmelia est un modèle de spécification de composants basés sur des descriptions de services complexes. Les composants sont *abstraits*, indépendants de leur environnement et par conséquent non exécutables. Kmelia peut servir à modéliser des architectures logicielles et leur propriétés, ces modèles étant ensuite raffinés vers des plateformes d'exécution (André *et al.*, 2006b). Il peut aussi servir de modèle commun pour l'étude de propriétés de modèles à composants et services (abstraction, interopérabilité, composabilité). Les caractéristiques principales du modèle Kmelia sont : les composants, les services et les assemblages. Leur description formelle est donnée dans (Attiogbé *et al.*, 2006).

Un *composant* est défini par un espace d'états, des services et une interface *1*. L'espace d'état est un ensemble de constantes et de variables typées, contraintes par un invariant. Dans l'interface d'un composant on distingue les *services offerts* I_p qui réalisent une fonctionnalité et les *services requis* I_r qui déclarent les besoins des fonctionnalités (Allen *et al.*, 1997; Medvidovic *et al.*, 2000). Les *services* sont eux-mêmes constitués d'une interface (qui peut inclure des sous-services), d'une description d'état et d'assertions (pre-post conditions). Formellement, un service s est défini par un couple $(I_s,\ \mathcal{B}_s)$ où I_s est l'interface du service et $\mathcal{B}_s$ est un éventuel comportement dynamique (les services requis n'en n'ont pas). L'interface du service est une abstraction des relations de composition de services : soit une composition horizon-

tale (ou dépendance), soit une composition verticale (ou inclusion). Le comportement (dynamique) d'un service offert est caractérisé par un automate, qui précise les enchaînements d'actions autorisés. Ces actions sont des calculs, des communications (émissions, réceptions de messages), des invocations ou retours de services.

Un des objectifs de Kmelia est de permettre l'expression de services complexes et leur composition. Les services appelables au sein d'un autre service sont nommés sous-services et sont déclarés dans l'interface du service I_s. La composition verticale de services repose sur l'ajout de *points d'expansion* (aux états ou parmi les actions sur $\mathcal{B}_s$) qui permettent d'inclure un service dans un autre et qui sont expansés lors des vérifications de compatibilité de services. Un appel de sous-service au sein d'un service est évalué dans son contexte. Différents types de points d'expansion peuvent être utilisés selon que l'on désire un appel *obligatoire* ou *optionnel*, de *service* ou de *comportement*. La syntaxe est donnée dans le tableau 2.

Point d'expansion		**Appel**	
Type	**Rattachement**	**service**	**comportement**
obligatoire	transition	`[[nom_service]]`	`[|nom_service|]`
optionnel	état	`<<nom_service>>`	`<|nom_service|>`

Tableau 2. *Syntaxe Kmelia des points d'expansion*

Les points d'expansion de comportement se distinguent des points d'expansion de service par la possibilité d'atteindre le comportement d'un service sans passer par l'appel du service (une pseudo-transition qui court-circuite le début de service est notamment ajoutée). Les points d'expansion de comportements ne peuvent s'appliquer qu'aux services sans paramètres d'entrée ni de sortie (qui ne transmettent donc d'information que par leurs messages), à moins de proposer une adaptation de service (André *et al.*, 2006a).

La figure 1 illustre les concepts précédents dans le cas d'un guichet automatique bancaire, GAB. Il s'agit ici d'une représentation condensée qui met à plat plusieurs niveaux distincts de description tels que composant et interface, service et interface, dynamique et contrats. Une seule interface est représentée. Les services offerts (resp. requis) sont représentés par des rectangles grisés (resp. blancs). Un lien d'assemblage est représenté par un trait reliant un service requis à un service offert.

Dans le comportement (dynamique) du service `requete` offert par le composant `IHM_CLIENTR`, deux points d'expansion de type service optionnel permettent l'appel éventuel des services `code` et `montant` à l'état `e1`. Ces services sont considérés ici sous une forme réduite pour faciliter la lecture, en réalité ils sont suffisamment complexes pour ne pas se réduire à un simple envoi de message (contrôle du demandeur et cryptage des informations). Dans cet exemple, `code` et `montant` sont des sous-services offerts par `requete` via le service requis `retirer`. Les pseudo-transitions en pointillé entre l'état `e1` de `requete` et les états initiaux et finaux de `code` illustrent l'expansion qui est générée systématiquement lors de la vérification comportementale

et selon chaque contexte d'invocation de service d'un composant. Elles ne figurent pas dans une spécification Kmelia. Les points d'expansions de type obligatoire se placent sur des transitions et sont expansées en utilisant le même principe (cf figure 2).

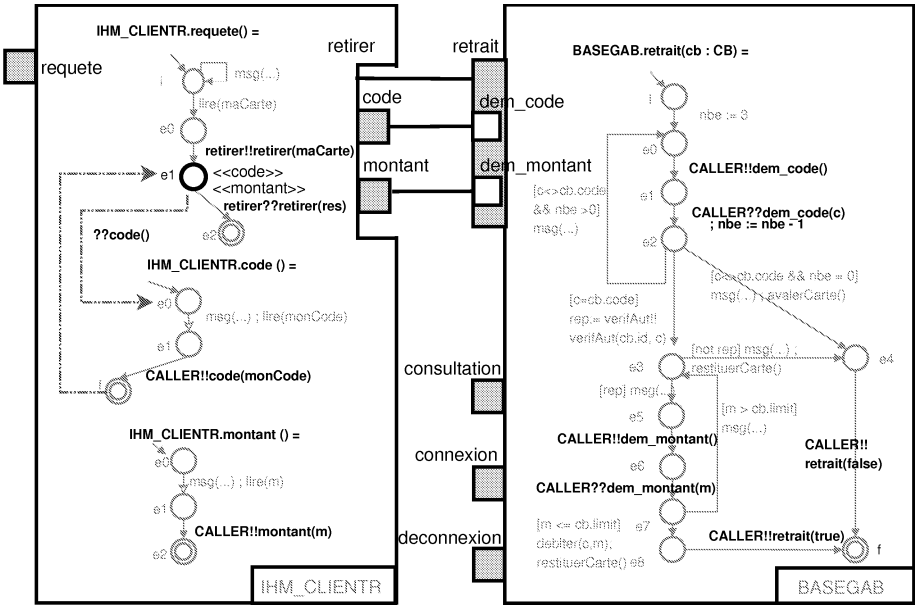

Figure 1. *Extrait du Guichet Automatique Bancaire modélisé dans Kmelia*

Formellement, I_s, l'**interface d'un service** s d'un composant C est spécifiée par un 5-uplet $\langle \sigma,\ P,\ Q,\ V_s, S_s \rangle$ où σ est la signature, P la précondition d'appel, Q la postcondition de déroulement, V_s un ensemble de déclarations de variables locales au service et $S_s = \langle sub_s,\ cal_s,\ req_s,\ int_s \rangle$ un quadruplet d'ensembles finis et disjoints de noms de services tels que $req_s \subseteq I_r$ et sub_s (resp. cal_s, req_s, int_s) est l'ensemble des services offerts (resp. les requis de l'appelant, les requis d'un composant quelconque, les offerts en interne) dans le cadre du service s.
$\mathcal{B}_s$, le **comportement d'un service** s est un système de transitions étiquetées étendu (ou *eLTS*) spécifié par un sexuplet $\langle S, L, \delta, \Phi, S_0, S_F \rangle$ où S est l'ensemble des états de $\mathcal{B}_s$, $S_0 \in S$ est l'état initial, $S_F \subset S$ est l'ensemble non vide des états finaux (le service se termine toujours), L est l'ensemble des étiquettes des transitions entre les éléments de S (les actions autorisées, y compris les points d'expansion obligatoire). $\delta : S * L \rightarrow S$ est la relation de transition entre les états de S selon les étiquettes. $\Phi : S \leftrightarrow sub_s$ est la relation d'étiquetage des états par des points d'expansion de type optionnel.

Les composants Kmelia peuvent être assemblés ou composés via des liens entre services. Dans un *assemblage*, les services requis par certains composants sont liés (connectés) aux services offerts d'autres composants. Ces liaisons, appelées liens d'assemblage, établissent des canaux implicites pour les communications entre services.

Les canaux sont point-à-point (dans la version actuelle du modèle) mais bidirectionnels. La figure 1 illustre une vue partielle d'un assemblage du GAB dans laquelle on se focalise sur le lien `retirer-retrait` pour lequel deux sous-liens sont définis (il n'y a pas de restrictions à la profondeur des sous-liens, elle est liée à la composition verticale des services). Une *composition* est un assemblage encapsulé dans un composant. La hiérarchisation des services ainsi que des composants est une des caractéristiques de Kmelia qui permet une bonne lisibilité, flexibilité et une bonne traçabilité dans la conception des architectures. Cet aspect a été abordé dans (André *et al.*, 2006b). La continuité des services est mise en œuvre par des liens de compositions qui servent à la promotion des services d'un composant vers ceux d'un composite.

3.2. *Spécification de protocoles en Kmelia*

Pour contrôler les services offerts par les composants Kmelia, nous souhaitons introduire des protocoles d'utilisation pour décrire des enchaînements licites d'appels de services. Parmi les services offerts par un composant, ceux qui apparaissent dans la description de protocoles sont dits *contrôlés*, ceux qui n'y figurent pas sont dits *libres*. Ces qualificatifs viennent compléter ceux qui gouvernent l'accessibilité directe des services, exposée dans l'interface du composant. Le modèle permet ainsi l'existence de services contrôlés par un ou plusieurs protocoles mais néanmoins visibles et appelables à tout moment. Un composant peut offrir un ou plusieurs protocoles. Certains protocoles d'un composant peuvent être qualifiés d'*ininterruptibles* par le concepteur du composant : les appels aux services que contrôle le protocole doivent se dérouler donc sans interruption, c'est-à-dire que lorsque le protocole est démarré (par l'effet d'un appel d'un service client), il doit se poursuivre jusqu'à achèvement ; le client s'engage à enchaîner convenablement les appels de services.

Une manière simple et efficace d'introduire des protocoles dans Kmelia est de les faire apparaître comme un service offert qui donne accès à d'autres services du composant. De nombreux éléments du modèle Kmelia permettent de spécifier les caractéristiques des protocoles exprimées ci-dessus : (1) les systèmes de transitions étiquetés étendus expriment des enchaînements d'activités dans le composant ; (2) les interfaces de services et de composants permettent de contrôler l'accès à certains services et de subordonner l'utilisation d'un service à un autre ; (3) les points d'expansion permettent de hiérarchiser des déroulements de service au sein d'un système de transition. Du point de vue du modèle, les protocoles constituent donc une classe particulière de services, caractérisée notamment par une restriction sur les actions présentes dans le comportement : chaque transition, éventuellement gardée, est étiquetée soit par un point d'expansion correspondant à un service qui doit être appelé (en l'absence de choix) par le service qui utilise le protocole, soit par une manipulation de variable locale limitée aux types simples. pour exprimer par exemple des conditions d'enchaînement (compteur de boucles, prédicats de chemin). Le modèle Kmelia permet d'associer des propriétés nommées aux entités du langage (services et composants notamment). Il s'agit d'un mécanisme d'extension dans le sens où les propriétés peuvent être exploi-

tées par la suite, à l'instar des restrictions exprimées dans des stéréotypes UML. La restriction (prédicat) qui caractérise la propriété *protocole* est associée au qualificatif `protocol` dans l'entête du service. La représentation de protocoles comme classe de services présente plusieurs avantages : simplicité du support multiprotocole (pas de niveaux structurants complexes qui rendent plus délicate l'utilisation par d'autres composants), interopérabilité avec les modèles qui ne supportent pas les protocoles (pour l'assemblage de composants hétérogènes), évolutivité des modèles (un protocole peut devenir un service dans un autre composant ou un raffinement de composant), extension cohérente de Kmelia (une nouvelle classe de services).

Prenons un exemple simplifié dans lequel les éléments de visibilité et de structuration sont masqués et qui fait apparaître le comportement dynamique du protocole. La figure 2 présente le protocole `protocoleRetrait` du composant `BASEGAB` qui spécifie un ordre sur l'usage des services `connexion`, `retrait` et `deconnexion`. Les points d'expansion de service de type obligatoire impliquent les appels respectifs de ces trois services aux états `i`, `e0` et `e1`.

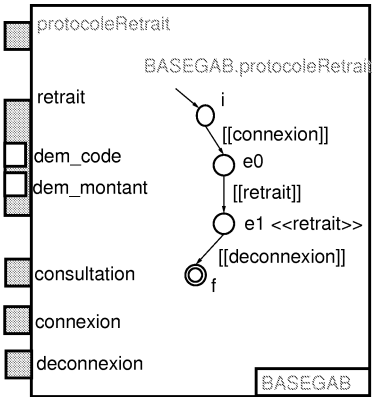

Figure 2. *Un protocole pour le composant BASEGAB*

Ce protocole est très simple car il n'intègre ni boucles (à l'exception de celle, implicite, que constitue `<<retrait>>`), ni gardes, ni actions élémentaires sur des variables pour établir des compteurs de boucles ou des restrictions de chemins. Il est présent dans l'interface du composant au même titre qu'un service et peut être appelé comme n'importe quel autre service présent dans l'interface. Les services présents dans le protocole sont appelés dans le cadre du protocole tout comme les sous-services d'un service sont appelés dans son contexte (cf section 3.1).

Dans la spécification des services en Kmelia, un protocole est distingué d'un service par l'emploi du qualificatif `protocol` dans l'entête du protocole ci-dessous.

```
provided protocoleRetrait()
   Properties = {protocol, nonInterruptible, ...}
   Pre  true
   Post true
```

Formellement, le service $p = (I_p, \mathcal{B}_p)$ est un protocole si

$$sub_p \neq \emptyset \wedge cal_p = \emptyset \wedge req_p = \emptyset \wedge L_p \subseteq (sub_p \cup int_p \cup L_p^-)$$

où L_p^- est l'ensemble des actions élémentaires portant uniquement sur les variables locales au protocole et les variables de l'espace d'état du composant. A cette contrainte s'ajoutent des règles méthodologiques de bonne définition (cf section 4).

Une partie de la sémantique d'un protocole est celle d'un service Kmelia : elle est obtenue par le développement des points d'expansion. La définition actuelle des points expansions et le partage implicite du canal de communication entre le service et le protocole qui le propose font qu'il n'y a pas parallélisme possible entre les deux. Le développement consiste (1) pour la partie dynamique à inclure —à renommage près des états et des actions— l'automate dynamique du service offert dans l'automate du protocole et (2) pour la partie fonctionnelle à établir la chaîne des contrats (pre/post) —à renommage près des variables locales au service et des actions. Les interfaces des services et leurs comportements doivent être compatibles. Dans cette interprétation, les mêmes règles de contrôle s'appliquent aux services et aux protocoles. Cependant, un concepteur ou un programme pourra utiliser le fait qu'il possède la propriété *protocole* pour le considérer autrement ; par exemple s'il existe des règles de conception concernant des protocoles. On peut alors exprimer des règles ou des vérifications qui ne sont valables que pour les protocoles, comme la section suivante va le montrer.

4. Conception et vérification des protocoles

Un composant mis à disposition des développeurs doit être exempt d'anomalies. Dans un article précédent (Attiogbé *et al.*, 2006), nous avons énoncé des propriétés attendues pour les composants et les services relatives notamment à la sûreté et la vivacité. Afin d'expérimenter nos propositions et offrir, à terme, des outils d'aide aux développeurs de composants, nous avons entrepris le développement d'une plate-forme pour l'étude des composants (COSTO[2]). La prise en compte de protocoles ne doit pas perturber le contrôle du bon déroulement des interactions entre composants. Le bon déroulement d'une interaction part de l'hypothèse du respect des règles contractuelles entre service appelant et protocoles ou services appelés. Un composant qui offre des services s'engage à fournir des protocoles cohérents servant de modes d'emploi des services offerts. Nous nous concentrons ici sur les vérifications et les règles de conception liées aux protocoles.

4.1. *Vérifications de protocoles en tant que services*

Dans la mesure où un protocole est un service offert, nous pouvons le soumettre aux mêmes vérifications que n'importe quel autre service offert. Nous exploitons pour

2. `lina.atlanstic.net/fr/equipes/team10/Kmelia/`

cela des résultats précédents (Attiogbé *et al.*, 2006) : du point de vue de chaque composant sont vérifiées la syntaxe et la cohérence des interfaces de services avec leur comportement ; du point de vue des assemblages sont vérifiées la compatibilité des interfaces de services (offerts-requis) et leur compatibilité comportementale. Dans ce dernier cas, il est à noter que les problèmes d'explosion combinatoire sont limités puisque la correction globale des interactions est l'union des corrections locales d'interaction (Attie *et al.*, 2003). Dans notre cas, une interaction est la donnée de trois services (un **contexte**) : un service offert (à vérifier) lié à un autre service offert (l'appelant) via un service requis dans l'assemblage. Ces vérifications sont faites à l'aide des outils existants de la plate-forme COSTO permettant l'analyse syntaxique et structurelle des composants *via* des algorithmes ad hoc et la vérification des compatibilités comportementales (par exemple la vérification de la bonne interaction avec des services offerts) en s'appuyant sur les environnements LOTOS/CADP ou MEC.

4.2. *Cohérence des protocoles*

La détection d'incohérence dans les protocoles fait partie des vérifications nécessaires pour assurer la correction d'un composant. Un protocole r d'un composant C est *incohérent* si un des enchaînements de services qu'il décrit peut être impossible du seul fait de l'enchaînement. Les deux causes d'incohérence suivantes peuvent être détectées :

– L'existence de chemins gardés sans alternatives menant à l'état final du protocole, dans le cas où l'expression de ces gardes ne peut être évaluée à vrai.

– L'existence dans un protocole d'une séquence d'appels $s_1; ...; s_n; s_{n+1}$ telle que la prise en compte des postconditions de s_1 à s_n implique la négation de la précondition de s_{n+1}, c'est-à-dire qu'un des services appelés avant s_{n+1} et dont les effets n'ont pas été remis en cause empêche le déroulement correct de s_{n+1} : il s'agit d'une séquence infaisable. Par exemple, si le service `connexion` possède une précondition $not(connected)$ et une postcondition $connected$, la séquence `connexion ;connexion` rend le protocole incohérent, ainsi que toute autre séquence ne contenant aucune modification de $connected$ entre deux appels de `connexion`.

Pour l'analyse des suites de séquences infaisables, nous expérimentons une passerelle avec des prouveurs de théorèmes basés sur la logique du premier ordre tel que Atelier B[3].

Le principe est le suivant : on considère un protocole non expansé et les sous-séquences d'appels de services allant d'un état initial à un état final du protocole pour éviter les boucles. Pour chacune de telles séquences, on vérifie que pour toutes les sous séquences $s_i; s_j$ (avec $j = i + 1$) qu'elle contient :

$$\neg(post(s_i) \Rightarrow \neg pre(s_j))$$

3. `www.atelierb.societe.com`

Le prédicat R postcondition d'un service s_i est exprimé avec des variables locales vl_i du service s_i qui sont les paramètres du service et des variables globales vg_k du composant, avec des informations de typage issues des paramètres de l'interface de s_i et du composant :

$$vl_i : tl_i;\ vg_k : tg_k \bullet R(vl_i, vg_k)$$

De la même façon, le prédicat Q précondition du service s_j est exprimé avec les variables locales vl_j du service s_j et des variables globales vg_k du composant, avec des informations de typage issues des paramètres de s_j et du composant :

$$vl_j : tl_j;\ vg_k : tg_k \bullet Q(vl_j, vg_k)$$

Puisque nous raisonnons en dehors du contexte de déroulement effectif des services, les valeurs des variables locales sont inconnues (on les suppose dans le meilleur des cas, convenables pour la vérité des prédicats) au moment des appels de services, les seules hypothèses de travail sont celles sur les variables globales ; on restreint alors les prédicats $R(vl_i, vg_k)$ et $Q(vl_j, vg_k)$ à $R'(vg_k)$ et $Q'(vg_k)$.

La propriété de départ
$\neg(vl_i : tl_i; vg_k : tg_k \bullet R(vl_i, vg_k) \Rightarrow \neg(vl_j : tl_j; vg_k : tg_k \bullet Q(vl_j, vg_k)))$
devient alors

$$\neg(vg_k : tg_k \bullet R'(vg_k) \Rightarrow \neg(vg_k : tg_k \bullet Q'(vg_k)))$$

Lever partiellement cette restriction sur la portée des prédicats est faisable dans certains cas (un paramètre peut être contraint dans une précondition et une postcondition peut établir qu'une variable globale a la valeur de ce paramètre comme nouvelle valeur) mais très coûteux en terme de preuve. A partir de là, nous devons intégrer les prédicats R' et Q' avec leurs contextes (variables, types) dans le prouveur cible en transformant au préalable nos obligations de preuve dans le formalisme pour pouvoir relever les points d'incohérence dans le protocole.

En reprenant l'exemple de la séquence `connexion ; connexion`, l'application du principe de notre analyse aboutit à faire la preuve de

$$\neg(post(\texttt{connexion}) \Rightarrow \neg pre(\texttt{connexion}))$$

or cette formule est fausse (donc non prouvable) ; on détecte ainsi l'incohérence.

4.3. *Méthodologie et règles de conception de protocoles*

Le définition adoptée pour les protocoles est relativement souple et flexible, ce qui permet de modéliser des cas variés. A l'inverse, il est intéressant de définir des règles de "bon" usage des protocoles pour éviter des incohérences. Nous en discutons ici quelques pistes.

La création de protocoles peut intervenir dans une démarche de conception descendante ou ascendante. Dans le premier cas il est probable que les services mentionnés seront des coquilles vides destinées à être enrichies ou raffinées. Dans le second cas plusieurs services existent avant que l'on crée des protocoles pour organiser ou figer une ou plusieurs utilisations ce qui implique potentiellement des changements dans

l'interface du composant. Par exemple si certains services deviennent contrôlés par des protocoles alors les composants qui les utilisaient doivent modifier leur interfaces de services requis.

Il est possible de hiérarchiser l'usage des services et protocoles, dans une présentation à deux niveaux, en contrôlant tous les services par des protocoles. Cette clarification d'usage des concepts devient lourde pour les services communs indépendants d'un composant, tels que les consultations. Rappelons que rien n'oblige a priori les concepteurs d'un composant de ne proposer que des protocoles comme services offerts dans l'interface de ce composant. Plusieurs cas de figure peuvent alors éventuellement poser problème :

– Si un des protocoles est interruptible, tout service présent dans l'interface peut être appelé à chaque état du protocole. Cela peut provoquer une explosion combinatoire de la recherche de séquences incohérentes dans le protocole.

– Lorsqu'un service s_i est le seul qui permettre d'établir la précondition d'un autre service s_j alors s_i doit apparaître dans le protocole d'utilisation (et avant s_j).

– Afin d'éviter des incohérences dues aux interférences avec l'état d'un composant, il est déconseillé d'avoir dans l'interface du composant les services opérant des modifications d'une partie de l'état prise en compte dans les préconditions de services contrôlés par des protocoles interruptibles.

– Un service possédant dans sa précondition un prédicat portant seulement sur l'état du composant (et pas un paramètre éventuel) n'a pas sa place dans l'interface du composant, mais devrait plutôt être contrôlé par un protocole.

Le modèle Kmelia n'inclut pas de contraintes dans la définition de composants et de protocoles restreignant l'interface d'un composant à certains services si un protocole est présent : une telle contrainte contredirait le modèle dans la mesure ou celui-ci est basé sur des composants et des services, et non des protocoles qui ne sont que des services affublés d'une propriété particulière non spécifique au modèle. Cependant, il est possible d'écrire les propriétés correspondantes qui peuvent être vérifiées et utilisées pour établir des contraintes ou émettre des messages d'avertissements. Une partie de ces propriétés a été écrite et implantée (de manière *ad hoc* pour l'instant).

4.4. *Outillage*

La plate-forme COSTO (*Component Study Toolbox*) est une boîte à outils pour spécifier, analyser et développer des composants Kmelia. COSTO contient un analyseur/compilateur de spécifications qui permet de traiter des spécifications Kmelia et de les représenter dans une structure abstraite sous forme d'objets. COSTO comprend également des traducteurs de spécifications Kmelia en Lotos et en MEC afin d'obtenir des spécifications en entrée des outils d'analyse formelle tels que CADP et MEC.

L'introduction de protocoles dans COSTO est réalisée pour la compilation et la vérification de compatibilité (interface et comportement) dans les assemblages de

composants. La vérification des contrats de protocoles est en cours d'expérimentation. Nous procédons d'une manière similaire à celle de l'analyse comportementale, à savoir par traduction dans des langages adaptés et outillés. Pour la vérification de contrat, nous expérimentons avec le langage B (Atelier B) et le langage Z (prouveur Z/EVES). Les aspects liés à la concurrence entre protocoles et services ne sont pas encore traités.

5. Conclusion et perspectives

Après avoir positionné notre approche des protocoles avec les autres usages courants qui en sont faits dans le domaine, nous avons présenté une solution pour décrire des modes d'emploi de services de composants sous la forme de protocoles d'utilisation. Cette solution reprend la description des services offerts Kmelia en la spécialisant et l'annote avec la propriété *protocole* afin d'en tirer parti en termes de représentation ou d'analyse. De cette façon notre modèle de composants n'est pas modifié ce qui permet de réutiliser les outils existants dans notre plate-forme COSTO pour analyser la compatibilité des protocoles. Nous avons donné la sémantique des protocoles dans le modèle de composant Kmelia et montré comment leur description est expansée pour obtenir des services ordinaires. Nous avons proposé une technique d'analyse formelle de la cohérence interne de protocole et des règles de conception de protocoles. Pour pousser plus loin nos analyses (de cohérence en l'occurrence) nous avons envisagé une passerelle avec un prouveur de théorèmes (Atelier B).

Les travaux en perspectives, en plus de l'extension à d'autres exemples d'application, concernent trois points principaux : i) la prise en compte d'autres propriétés en plus de la cohérence (correction des fonctionnalités fournies sous contrôle de protocoles), ii) l'exploitation des cas d'anomalies pendant l'analyse de cohérence pour suggérer des modifications des composants/protocoles ou bien une introduction de services ou de composants d'adaptation, iii) le raffinement vers des plate-formes existantes (Fractal, SOFA). Chacun de ces points fera l'objet d'une mise en œuvre sous forme de modules et de passerelles dans notre plate-forme expérimentale COSTO.

6. Bibliographie

Allen R., Garlan D., « A Formal Basis for Architectural Connection », *ACM Transactions on Software Engineering and Methodology*, vol. 6, n° 3, p. 213-249, July, 1997.

André P., Ardourel G., Attiogbé C., « Coordination and Adaptation for Hierarchical Components and Services », *Third International ECOOP Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'06)*, p. 15-23, 2006a.

André P., Ardourel G., Attiogbé C., « Spécification d'architectures logicielles en Kmelia : hiérarchie de connexion et composition », *1ère Conférence Francophone sur les Architectures Logicielles*, Hermès, Lavoisier, p. 101-118, 2006b.

Attie P. C., Lorenz D. H., Establishing Behavioral Compatibility of Software Components without State Explosion, Technical Report n° NU-CCIS-03-02, College of Computer and Information Science, Northeastern University, 2003.

Attiogbé C., André P., Ardourel G., « Checking Component Composability », *5th International Symposium on Software Composition, SC'06*, vol. 4089 of *LNCS*, Springer, p. 18-33, 2006.

Becker S., Overhage S., Reussner R., « Classifying Software Component Interoperability Errors to Support Component Adaption », *7th International Symposium CBSE 2004*, LNCS, Springer, p. 68-83, 2004.

Beugnard A., Jézéquel J.-M., Plouzeau N., Watkins D., « Making Components Contract Aware », *Computer*, vol. 32, n° 7, p. 38-45, 1999.

Canal C., Fuentes L., Pimentel E., Troya J. M., Vallecillo A., « Adding Roles to CORBA Objects », *IEEE Trans. Softw. Eng.*, vol. 29, n° 3, p. 242-260, 2003.

de Alfaro L., Henzinger T. A., « Interface automata », *Proceedings of the 8th European software engineering conference ESEC/FSE-9*, ACM Press, p. 109-120, 2001.

Giannakopoulou D., Kramer J., Cheung S.-C., « Behaviour Analysis of Distributed Systems Using the Tracta Approach. », *ASE*, vol. 6, n° 1, p. 7-35, 1999.

Inverardi P., Wolf A. L., Yankelevich D., « Static checking of system behaviors using derived component assumptions », *ACM Trans. Softw. Eng. Methodol.*, vol. 9, n° 3, p. 239-272, 2000.

Medvidovic N., Taylor R. N., « A Classification and Comparison Framework for Software Architecture Description Languages », *IEEE Transactions on Software Engineering*, vol. 26, n° 1, p. 70-93, january, 2000.

Moisan S., Ressouche A., Rigault J., A Behavioral Model of Component Frameworks, Technical Report n° RR-5065, INRIA, December, 2003.

OMG, *The OMG Unified Modeling Language Specification, version 2.0 Rfp*, Superstructure Specification available at www.omg.org/docs/ptc/05-07-04.pdf, Infrastructure Specification available at www.omg.org/docs/ptc/03-09-15.pdf. 2005.

Pavel S., Noye J., Poizat P., Royer J.-C., « Java Implementation of a Component Model with Explicit Symbolic Protocols », *4th International Symposium on Software Composition, SC'05*, vol. 3628 of *LNCS*, Springer, p. 115-124, 2005.

Plasil F., Visnovsky S., « Behavior Protocols for Software Components », *IEEE Trans. Softw. Eng.*, vol. 28, n° 11, p. 1056-1076, 2002.

Schmidt H., « Trustworthy components-compositionality and prediction », *J. Syst. Softw.*, vol. 65, n° 3, p. 215-225, 2003.

Südholt M., « Model of Components with Non-regular Protocols », *4th International Symposium on Software Composition, SC'05*, vol. 3628 of *LNCS*, Springer, p. 99-113, 2005.

Yellin D., Strom R., « Protocol Specifications and Component Adaptors », *ACM Transactions on Programming Languages and Systems*, vol. 19, n° 2, p. 292-333, 1997.

Zimmermann W., Schaarschmidt M., « Checking of Component Protocols in Component-Based Systems », *5th International Symposium on Software Composition, SC'06*, vol. 4089 of *LNCS*, Springer, p. 1-17, 2006.