Vérification d'une architecture UML 2.0 avec l'ADL Wright

Mohamed Graiet, Mohamed Tahar Bhiri, Jean-Pierre Giraudin, Abdelmaj Hamadou

Vérification d'une architecture UML2.0 avec l'ADL Wright

Mohamed Graiet [I, III] Mohamed Tahar Bhiri [II], Jean-Pierre Giraudin [I], Abdelmajid Ben Hamadou [III]

[I] LSR-IMAG, équipe SIGMA, BP 72, 38402 SAINT MARTIN D'HERES CEDEX
Mohamed.graiet@imag.fr
Jean-Pierre.giraudin@imag.fr

[II] Faculté des Sciences de Sfax, rue soukra, BP 802, 3018 SFAX
Tahar_bhiri@yahoo.fr

[III] LARIM-Institut Supérieur d'Informatique et du Multimédia de Sfax, rue mharza, 3018 SFAX
Abdelmajid.benhamadou@isimsf.rnu.tn

Résumé : UML2.0 offre des nouveaux concepts tels que composant, port, interfaces offertes, interfaces requises, connecteur, structure composite permettant de décrire une architecture logicielle. Mais UML2.0 ne permet pas l'étude formelle de deux critères fondamentaux sur une architecture logicielle qui sont **la cohérence** et **la complétude**. Par contre certains ADL comme Wright autorisent une telle étude. Dans cet article, nous préconisons une approche de traduction permettant de transformer une architecture UML2.0 en une architecture Wright afin de réaliser des vérifications formelle d'architectures UML2.0.
Mots Clés : Architecture logicielle, UML2.0, Wright, Traduction, vérification formelle.

1 INTRODUCTION

L'architecture logicielle [Garlan, 1994] est devenue une activité de conception explicite dans le processus de développement. Elle modularise un système sous forme d'une configuration de composants et de connecteurs. L'architecture logicielle permet d'identifier des entités de connexion appelées connecteurs et de décrire de manière précise les protocoles de communication entre composants [Allen, 1996] [Allen, 1997a]. Les travaux sur les formalismes de description d'architectures logicielles ont donné naissance à plusieurs ADL (Architecture Description Langage) [Medvidovic, 2000]. Chaque ADL s'est spécialisé dans un thème particulier fournissant des fonctionnalités complémentaires pour la description et l'analyse architecturale [ACCORD, 2002] : Aesop pour les styles architecturaux, le style C2 pour les systèmes distribués et évolutifs, Darwin pour les systèmes d'envoi de messages distribués, Rapide concernant la simulation et Wright pour la description comportementale et la vérification formelle des propriétés sur une architecture logicielle. Plusieurs travaux ont essayé de définir un méta-langage pour profiter des avantages de ces différents ADL. Par exemple, ACME [Garlan, 2000] fournit une représentation intermédiaire exploitable par les outils architecturaux associés aux différents ADL.
Pour mettre à la disposition des utilisateurs UML et par conséquent du monde industriel, des fonctionnalités offertes par les ADL notamment la vérification formelle des architectures logicielles, nous préconisons une approche de transformation permettant la traduction d'une architecture logicielle décrite en UML2.0 [OMG03] en une architecture décrite en Wright. Le choix de l'ADL Wright est justifié par ses aptitudes liées à la validation formelle des architectures logicielles.
Cet article est organisé comme suit. Les deux sections 2 et 3 présentent brièvement le cadre de notre travail à savoir UML 2.0 et l'ADL Wright. Ensuite la section 4 propose et justifie des règles de correspondance supportant le processus de traduction d'une architecture UML 2.0 vers une architecture en Wright. Enfin, la section 5 conclut et présente les perspectives de cette recherche.

2 ARCHITECTURE LOGICIELLE UML2.0

UML2.0 offre des nouveaux concepts (interfaces offerte et requise, port, structure composite et connecteur) permettant de définir l'architecture de l'application que l'on désire développer. Le méta-modèle UML2.0 considère les deux concepts class et component comme des classifieurs. De plus, la méta-classe component est une sous-classe de la méta-classe class.
UML2.0 spécifie un composant comme étant une unité modulaire, réutilisable qui interagit avec son environnement par l'intermédiaire de points d'interactions appelés ports. Les ports sont typés par les

interfaces. Celles-ci contiennent un ensemble d'opérations et de contraintes OCL2.0 (langage d'expression de contraintes pour UML2.0). Le comportement d'une interface est décrit par un protocole de type « machine à états » (protocol statemachine). Les interfaces peuvent être fournies ou requises. De même, les ports installés sur des composants ou des classes peuvent être fournis ou requis. Le comportement d'un port est issu de la composition des comportements des ces interfaces. Un tel comportement des ports est décrit à l'aide d'un protocole «machine à états». Le comportement interne du composant ne doit être ni visible, ni accessible autrement que par ses ports. Enfin, le composant est voué à être déployé un certain nombre de fois, dans un environnement non connu a priori lors de la conception.
Il existe deux types de modélisation de composants dans UML2.0 : le composant atomique (ou basique) et le composant composite (structure composite). La première catégorie définit le composant comme un élément exécutable du système. La deuxième catégorie étend la première en définissant le composant comme un ensemble cohérent des parties appelées parts. Chaque part représente une instance d'un autre composant.
La connexion entre les ports requis et les ports fournis se fait au moyen de connecteurs. Deux types de connecteurs existent : le connecteur de délégation et le connecteur d'assemblage. Le connecteur de délégation est utilisé pour lier un port du composant composite vers un port d'un composant situé à l'intérieur du composant composite : relier par exemple un port requis à un autre port requis. Le connecteur d'assemblage est utilisé pour les liens de construction : relier un port requis à un port fourni.

3 L'ADL WRIGHT

Wright est un langage de spécification d'architecture issu de la thèse de Robert J.Allen [Allen, 1997b]. Les principaux concepts offerts par Wright sont : composant, connecteur, configuration et style.
Les descriptions des composants contiennent deux parties :

- L'interface consiste en un ensemble de ports, chacun représentant une interaction avec l'extérieur à laquelle le composant peut participer. La spécification d'un port décrit deux aspects de l'interface du composant : le comportement partiel attendu du composant et ce que le composant attend du système dans lequel il va interagir.
- Le calcul (ou computation) décrit ce que le composant fait de point de vue comportemental.

Ces deux parties sont décrites en CSP [Hoare, 1985], et fournissent l'enchaînement des évènements émis ou reçus. Un composant proxy peut se définir successivement par la description de deux ports et par l'expression de calculs :

Component Proxy

Port s = requete?u → $\overline{\text{reponse!r}}$ → s □ §
Port c = $\overline{\text{requete!u}}$ → reponse?r → ıı §
Computation = s.requete?u → $\overline{\text{c.requete!u}}$ → c.reponse?r → $\overline{\text{s.reponse!r}}$ → Computation □ §

Ce composant possède deux ports : s et c. Le port s reçoit les requêtes et envoie les réponses à un client. Le port c est utilisé pour déléguer la requête à un autre composant. La partie calcul rend la délégation explicite (requête reçue sur s, envoi d'une requête sur c, attente de la réponse, puis envoi de cette réponse sur s).
La structure d'un connecteur est similaire à celle d'un composant. Elle consiste en un ensemble de Rôles et une Glue :

- Chaque **Rôle** décrit les attentes du connecteur dans ses interactions avec les composants.
- La **Glue** décrit comment les Rôles travaillent ensemble pour créer cette interaction

Les Rôles et la Glue sont également décrites en CSP.
La configuration décrit comment est agencé le système. Elle comporte pour cela quatres aspects : les définitions des types composants et connecteurs, les instances, les liens et les hérarchies (connecteurs composites et composants composites).
Wright permet de décrire des styles d'architectures qui regroupent une famille d'architectures ayant un ensemble de propriétés communes caractérisant cette famille. Pour cela Wright fournit trois aspects pour définir un style : les types (composant, connecteur, et interface), les paramètres et les contraintes. Pour exprimer les contraintes, Wright propose une notation basée sur la logique de premier ordre.
Les connecteurs sont toujours utilisés pour connecter deux ports de deux composants. Plusieurs tests sont effectués pour vérifier la validité de la configuration. Wright utilise la spécification CSP des ports et des rôles, pour spécifier entre autres leur compatibilité.

4 TRADUCTION DE SPECIFICATIONS UML2.0 EN WRIGHT

Une architecture logicielle de qualité devrait distinguer d'une façon explicite les composants fonctionnels des composants de communication [Cariou, 2003]. D'ailleurs les ADL comme Wright proposent deux constructions component et connector permettant de modéliser respectivement les composants fonctionnels et de communication. Mais le concept de connecteur d'UML2.0 n'est pas considéré comme classifier et ne peut pas modéliser un composant de communication. Pour résoudre ce problème, nous avons opté pour les choix suivants :

- Un composant fonctionnel est modélisé par le concept de composant d'UML2.0 doté des ports.
- Un composant de communication est modélisé par le concept de classe d'UML2.0 doté des

ports. Une telle classe sera stéréotypée « communication » avec une contrainte de ports obligatoires.

Sachant que les deux concepts d'UML2.0 composant et classe ont le même pouvoir expressif. Mais ils possèdent deux icônes différentes. Ceci permet de distinguer explicitement les composants fonctionnels et les composants de communication dans une architecture logicielle décrite en UML2.0.

La topologie ou la configuration d'une architecture logicielle décrite en UML2.0 est modélisée par un diagramme d'instances comportant des instances de type composant fonctionnel et des instances de type composant de communication. Les liens entre les instances de type composant fonctionnel et les instances de type composant de communication sont modélisés par des connecteurs d'assemblage d'UML2.0.

Dans la suite, nous allons proposer des règles de correspondances entre une architecture logicielle UML2.0 -comportant les éléments décrits précédement- et une spécification Wright.

4.1 Les composants fonctionnels

Un composant fonctionnel est modélisé par un composant UML2.0. Les points d'interaction d'un composant fonctionnel avec son environnement sont modélisés par des ports UML2.0. Chaque port est typé par des interfaces offertes ou requises. Le comportement de chaque port est modélisé par un protocole «machine à états» d'UML2.0. De même, le comportement du composant est modélisé par un protocole «machine à états». La figure 1 donne la règle de traduction d'un composant UML2.0 en son équivalent en Wright.

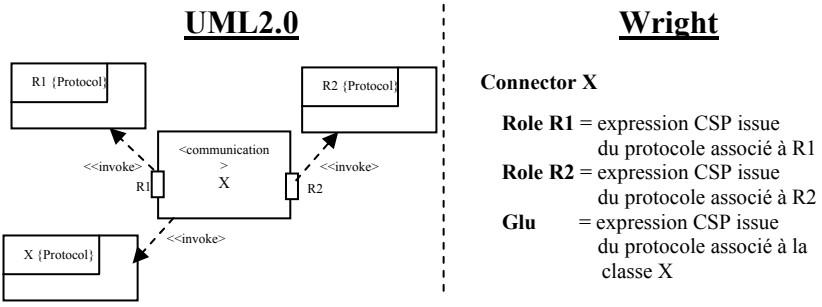

Figure 1: Règle de traduction d'un composant UML2.0 en Wright

Les aspects structuraux d'un composant UML2.0 sont traduits en Wright en utilisant également les aspects structuraux attachés à un composant Wright : chaque port UML2.0 est traduit par un port Wright ayant le même identificateur. Les aspects comportementaux d'un composant UML2.0 sont traduits en Wright sous forme des expressions CSP (voir section 4.4).

4.2 Les composants de communication

Un composant de communication est modélisé par une classe UML2.0. Les rôles joués par les composants fonctionnels connectés au composant de communication sont modélisés par des ports UML2.0. Les comportements des rôles et du composant sont modélisés par des protocoles machines d'états. La figure 2 donne la règle de traduction d'une classe UML2.0 vers Wright.

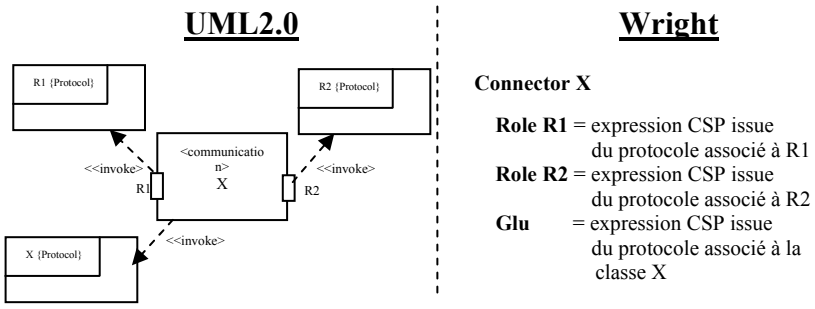

Figure 2: Règle de traduction d'une classe communication UML2.0 en Wright

Les aspects structuraux d'une classe UML2.0 modélisant un composant de communication sont traduits en utilisant les aspects structuraux attachés à un connecteur Wright : chaque port UML2.0 est traduit par un rôle Wright ayant le même identificateur. Les aspects comportementaux d'une classe UML2.0 sont traduits en Wright sous forme des expressions CSP (voir section 4.4)

4.3 Architecture logicielle

Une architecture logicielle en UML2.0 est modélisée par un diagramme d'instances. Ces instances sont issues de deux types UML2.0 : component et class. La figure 3 donne les principes de traduction d'une architecture logicielle UML2.0 de type client-serveur en configuration Wright.

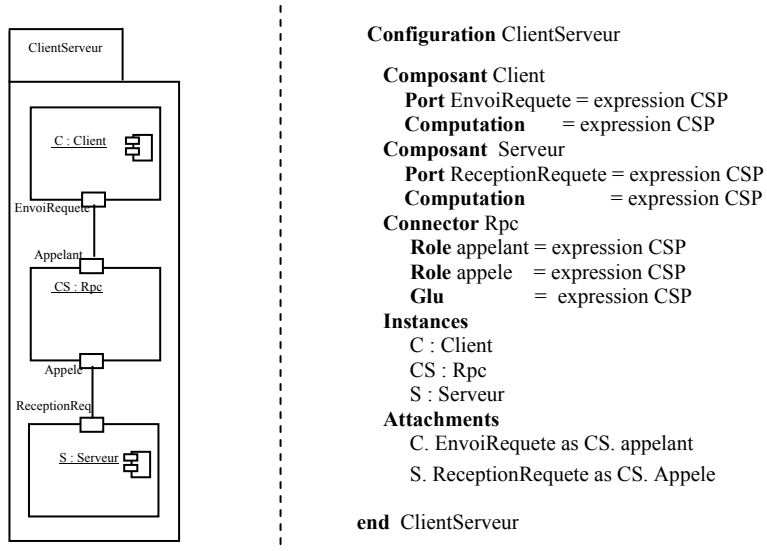

Figure 3: Principes de traduction d'une architecture UML 2.0 en Wright

4.4 Expression en CSP-Wright des protocoles «machine à états» UML2.0

L'ADL Wright utilise un sous-ensemble de CSP compatible avec le CSP standard [Hoare, 1985] permettant de décrire les comportement partiels (port et role) et globaux (computation et glu) de deux constructions offertes par Wright à savoir component et

connector. De plus l'ADL Wright augmente la notation CSP pour faire la différence entre un événement initialisé recevant des données (événement avec une barre, suivi d'un point d'exclamation et d'une variable qui représente les données) et un événement observé fournissant des données (événement sans barre, suivi d'un point d'interrogation et d'une variable qui représente les données).

Des travaux liés à la consistance des modèles UML existent [Engels2001] [Küster, 2004]. Ces travaux prennent comme domaine sémantique le langage CSP et l'outil de validation FDR [FDR93] [FDR98]. Le travail décrit dans [Engels2001] propose des règles permettant de traduire une machine à états UML en CSP. Nous avons adapté ces règles à notre contexte : protocole «machine à états» UML2.0 et CSP augmenté par des événements initialisés et observés. A titre illustratif, la figure 4 donne la description en UML2.0 d'un pipe utilisé notamment dans les architectures de type Filtre/Pipe [Allent, 1997]. Le port source exige une interface requise appelée Ecriture. Tandis que le port Destination offre une interface appelée Données et exige une interface appelée Lecture. La figure 5 donne les deux expressions CSP de Wright Source et Destination issues des protocoles machines d'états UML2.0 associées à ces deux ports. Toute transition étiquettée par des appels d'opérations appartenant à des interfaces requises est traduite par un événement initialisé en CSP de Wright. De plus, le lien entre opération UML2.0 et événement Wright est traité selon des correspondances adaptées de [Sanlaville, 1997].

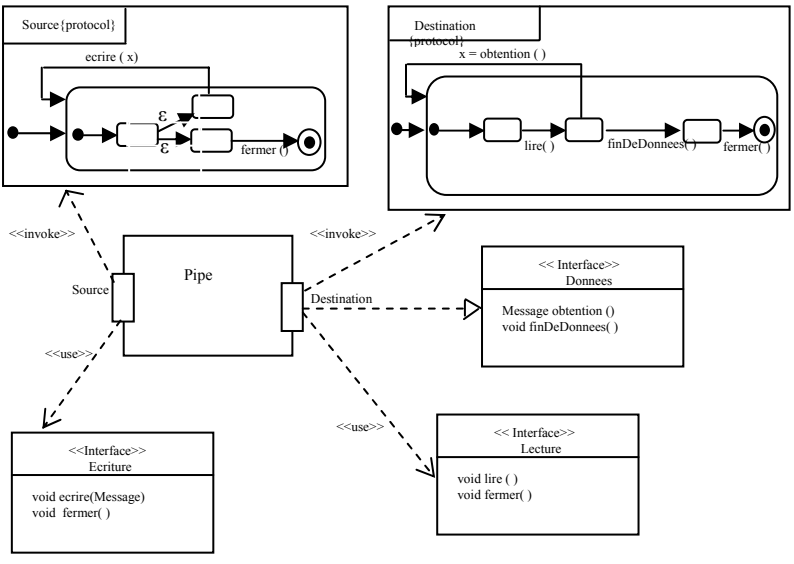

Figure 4: Description en UML2.0 du connecteur Pipe

Source = $\overline{\textbf{ecrire!x}}$ → **Source** ∥ $\overline{\textbf{fermer}}$ → §
Destination = $\overline{\textbf{lire}}$ → **obtention?x** → **Destination** □
finDeDonnees → $\overline{\textbf{fermer}}$ → §)

Figure 5: Expressions CSP associées aux ports Source et Destination du connecteur Pipe

5 CONCLUSION ET PERSPECTIVES

Pour mettre à la disposition des utilisateurs UML les concepts et les mécanismes sous-jacents issus des ADL, nous avons préconisé une approche de traduction d'UML2.0 vers Wright. Ainsi nous avons proposé des règles permettant de traduire une architecture UML2.0 en une architecture Wright. Ceci ouvre des perspectives liées à la vérification formelle des architectures UML2.0. En effet l'architecture Wright issue de l'architecture UML2.0 est traduite en une spécification CSP contenant des processus CSP décrivant les aspects comportementaux de l'architecture et des assertions exprimant les propriétés standard définies par Wright. Une telle spécification CSP peut être analysée par l'outil de vérification formelle FDR [FDR98].

A l'issue de ce travail exploratoire, des questions se posent que nous évaluons actuellement :

- Wright propose une dizaine de propriétés appelées tests permettant de vérifier la cohérence et la complétude d'une architecture Wright. Il est important d'évaluer la pertinence de chacun de ces tests vis-à-vis d'une architecture en UML2.0.
- Les contraintes OCL2.0 associées à des spécifications UML2.0 sont à combiner avec les expressions Wright issues des traductions que nous proposons. Nous utiliserons en particulier le langage des contraintes propres à Wright.

Ces travaux de traduction et de vérification formelle s'intègrent dans notre projet plus général de combinaison d'architectures selon différents points de vue système d'information et architecture logicielle compatibles avec une approche MDA [OMG00] [OMG01a] [OMG01b].

BIBLIOGRAPHIE

[ACCORD, 2002] « Projet ACCORD: Etat de l'art sur lesLangages de Desciption d'Architecture » (ADL) 2002. http:// www.infres.enst.fr.projet/accord/

[Allen, 1996] Allen R et Garlan D. "The wright architectural specification langage". Rapport technique CMU-CS-96-TBD, Carnegie Mellon University, School of Computer Science, Pillsburgh, PA, septembre 1996.

[Allen1997a] Allen R et Garlan D. « A Formal Basis for Architectural Connection ». ACM Transactions on Software Engineering and Methodology, July 1997.

[Allen, 1997b] Allen R. « A Formal Approch to Software Architecture ». Phd Thesis, Carnegie Mellen University, School of Computer Science, May 1997.

[Cariou, 2003] Cariou E, « Contribution à un processus de réification d'abstractions de communication ». Thèse de doctorat, Université de Rennes1, Juin 2003.

[Engels2001] Engels C, Heckel R and Küster J M « Rule-based Specification of Behavior Consistency based on the UML Meta-Model ». In M.Gogolla an C.Kobry, editors, proceedings of the 4th International Conference on the Unified Modeling Language (UML 2001), pages 272-287, LNCS 2185, Toronto, Canada, October 2001.

[FDR93] Failures Divergence Refinement: User Manual and Tutorial. Formal Systems (Europe) Ltd, Oxford, England, 1.3 edition, August 1993.

[FDR98] http://www.formal.demon.co.uk/FDR2.html

[Garlan, 2000] Garlan D, Monroe R M and Wile D. « Acme: Architectural Description of Component-Based Systems ». Eds: Cambridge University Press,2000.

[Garlan,1994] Garlan D, and Shaw M. « An Introduction to Software Architecture ». CMU-CS-94-166, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890.

[Hoare,1985] Hoare C A R. « Communicating Sequential Processes ». Prentice Hall, Englewood Califfs NJ, 1985.

[Medvidovic, 2000] Medvidovic N, and Taylor R N. « A classification and comparison framework for software architecture description languages ». IEEE Transactions on Software Engineering, 26 (1): 70-93, January 2000.

[Küster, 2004] Küster J M. « Consistency Management of Object-Oriented Behavioral Models ». Faculty of Computer Science, Electrical Engineering, and Mathematics of the University of Paderborn, March 2004.

[OMG00] R. Soley, OMG Staff Strategy Group: Model Driven Architecture, www.omg.org, novembre 2000.

[OMG01a] OMG, Model Driven Architecture: The Technical Perspectives, www.omg.org, juillet 2001.

[OMG01b] Object Management Group. OMG Unified Modeling Language Specification V. 1.4. http://www.omg.org/docs/formal /01-09-67.pdf (September 2001).

[OMG03] Object Management Group. UML2.0 Superstructure Specification: Final Adopted Specification. http://www.omg.org/docs/ptc /03-08-02.pdf (August 2003).

[Sanlaville, 1997] Sanlaville R. « Description d'architectures logicielles: utilisation du formalisme Wright pour l'interconnexion de machines abstraites B ». DEA, LSR, IMAG, Grenoble, 1997.