LE PARADIGME ACTEUR DANS LA MODELISATION DES SYSTEMES EMBARQUES

Mokhoo Mbobi
Ecole Supérieure d'Electricité
Département Informatique
91192 Gif-sur-Yvette cedex, France
Mokhoo.Mbobi@supelec.fr

Frédéric Boulanger
Ecole Supérieure d'Electricité
Département Informatique
91192 Gif-sur-Yvette cedex, France
Frederic.Boulanger@supelec.fr

Abstract

La conception d'un système embarqué implique une phase préalable de modélisation grâce à des outils (langages ou plate-formes) de modélisation. Chacun de ces outils utilise une approche de modélisation appropriée ; ce qui induit une dépendance forte entre la méthodologie et l'outil de modélisation. Il s'ensuit que le choix de la méthodologie par le concepteur est dicté par celui de l'outil à utiliser. Ce choix dépend à la fois des contraintes spécifiques du système et du niveau d'abstraction auquel il sera modélisé. Il est déterminant au niveau de la mise en œuvre du système car, il oriente la modélisation et contraint donc la conception. Ainsi, un mauvais choix peut compromettre la qualité et le coût de la conception.

Cet article présente la méthodologie de modélisation basée sur des composants appelés "acteurs". Sa valeur distinctive est présentée sur le plan de la structuration du système, de la séparation des préoccupations, et de la décomposabilité et recomposabilité modulaire.

Keywords *— Conception ; Modélisation ; Composants ; Acteurs ; Systèmes embarqués ; Génie Logiciel ; Hétérogénéité.*

1 Introduction

La croissance du marché des systèmes embarqués est en pleine explosion et ceci pour des raisons durables. Cependant, ces types de systèmes étant fabriqués à grande échelle et le risque d'une erreur préjudiciable étant à écarter, la réalisation *"a priori"* d'un prototype s'avère indispensable. De plus, ces prototypages étant long à développer et onéreux, les modèles logiciels sont souvent définis pour être utilisés comme référence comportementale pour une meilleure exploration et évaluation des propriétés du système. C'est pourquoi une phase préliminaire de modélisation est nécessaire. De cette phase, logiciel et matériel, communication et calcul, contrôle et données sont manipulés séparément et ne sont rassemblés que dans la dernière phase de prototypage. Du fait de l'hétérogénéité des système embarqués [18], cette phase préliminaire de modélisation requiert des connaissances scientifiques pluridisciplinaires ; d'où le besoin de plusieurs spécialistes dans différents domaines scientifiques et techniques. Ainsi, chaque équipe apporte ses compétences à travers un formalisme spécifique. Quant à cette modélisation, elle est faite grâce à des outils qui peuvent être des langages ou des environnements (plate-formes) de modélisation. Cependant, chacun de ces outils utilise une approche de modélisation appropriée, ce qui induit une dépendance forte entre la méthodologie et l'outil utilisé pour la modélisation. Il s'ensuit que le choix de la méthodologie par le concepteur est souvent dicté par celui de l'outil qu'il aura à utiliser. Quant au choix de cet outil, il dépend à la fois du niveau d'abstraction auquel le système sera modélisé et de ses contraintes spécifiques. Ce choix reste déterminant au niveau de la mise en œuvre du système, car, il oriente la modélisation et par conséquent contraint la conception. Ainsi, un mauvais choix peut soit compromettre la qualité de la conception soit mener le concepteur dans une implémentation plus onéreuse ou moins fiable [17].

Cet article présente l'approche de modélisation basée sur les composants appelés acteurs. La valeur distinctive de cette approche est présentée sous forme d'avantages qui militent pour son utilisation dans le cadre de la modélisation des systèmes embarqués.

2 Modélisation des systèmes embarqués

2.1 Modèles

Dans [6], les auteurs définissent la modélisation comme la représentation formelle d'un concept, d'un système, d'un sous-ensemble etc. Ainsi, un modèle peut être mathématique, auquel cas il peut être vu comme un ensemble d'assertions concernant les propriétés d'un système. Un modèle peut également être constructif, et dans ce cas, il définit un procédé informatique qui calcule un ensemble de propriétés d'un système. Les modèles constructifs appelés également *"modèles exécutables"* sont souvent utilisés pour décrire le comportement d'un système répondant à une stimulation qui lui est extérieure. La conception est par contre la définition d'une implémentation, i.e., la définition d'un modèle exécutable sur une plate-forme cible qui impose des contraintes sur le fonctionnement du système. Souvent, elle implique la réalisation de plusieurs modèles, chacun étant un raffinement du précédent. Le premier modèle peut être considéré comme la spécification formelle du système, et le dernier comme son implémentation. En somme, la conception et la modélisation sont étroitement liées ; le but de la modélisation est l'exploration des modèles pour une conception finale alors que le but de la conception demeure l'implémentation.

2.2 Tendances actuelles en modélisation

Dans l'analyse des aptitudes des langages existants à spécifier des systèmes embarqués, deux constats se dégagent ; d'une part, ces langages ne peuvent nullement couvrir tout le flot de conception d'un système et d'autre part, ils

ne tiennent pas toujours compte des concepts spécifiques tels que le temps, le parallélisme, la communication interprocessus, c'est-à-dire, les signaux et protocoles, la réactivité et les types de données spécifiques. C'est pourquoi, en modélisation des systèmes hétérogènes, trois principales approches s'opposent : Certaines initiatives tentent d'adapter les langages comme C ou C++ à la description de matériel en tenant compte de ses concepts spécifiques, ce qui implique le développement de bibliothèques ou de classes nécessaires pour représenter les composants matériels et leurs interactions. Une deuxième approche cherche à étendre les langages de description matérielle à la description de systèmes. Une troisième voie est la spécification de nouveaux langages qui peuvent suivre une nouvelle syntaxe, le langage Rosetta en est un exemple [4]. Ces langages peuvent être classés dans les différentes familles suivantes :

a *les langages de description de matériel* qui supportent les concepts spécifiques matériels. Parmi ces langages on peut citer VHDL [23], SystemC [21].

b *les langages de description des architectures* qui permettent une définition formelle de l'architecture de haut niveau d'un système complexe. Parmi ces langages on peut citer SpecC [10].

c *les langages orientés objet* qui permettent une description des systèmes complexes à un très haut niveau d'abstraction par sa décomposition en sous-systèmes plus faciles à maîtriser. UML [22] est un langage de ce type.

d *les langages pour la modélisation des systèmes temps réel*, systèmes dont le temps de réponse est du même degré d'importance que la précision du résultat. Parmi ces langages, il y a ceux qui sont synchrones comme Lustre [11], Esterel [5] [8] et ceux qui sont asynchrones tel que SDL [19].

e *les langages dérivant de l'extension des langages de programmation utilisés en informatique* par l'ajout des concepts spécifiques au matériel. Le choix de ces langages est justifié par le fait qu'ils fournissent le contrôle et les types de données nécessaires. Les langages développés consistent en la définition des fonctions ou classes pour modéliser les concepts spécifiques au matériel. Parmi ces langages, il y a SystemC [21] [20], SpecC [10] et JavaTime [24].

2.3 Environnements de modélisation

Les environnements de modélisation utilisent des langages et des méthodologies appropriées. En effet, dans la modélisation d'un système hétérogène, chacune des spécificités des modèles de calcul utilisés doit être prise en compte. Pour cela, l'utilisation d'un seul langage est généralement insuffisante, car il est très difficile d'exprimer toutes les contraintes des différents modèles de calcul dans un seul langage du fait que certains sont orientés matériels et d'autres sont orientés systèmes. C'est pourquoi, les environnements traditionnels de modélisation utilisent généralement une approche multilangage. Néanmoins, il existe aussi des environnements de modélisation utilisant une approche unifiée. Ces environnements permettent la modélisation et la conception d'un système par utilisation d'un seul langage. Quelques environnements unifiés de modélisation sont : SystemC [21] [20], Ptolemy II [6] [13].

2.4 Méthodologies de modélisation

Puisque la méthodologie utilisée pour la modélisation d'un système dépend du langage ou de la plate-forme de modélisation utilisée, le concepteur établit son choix en prévision du langage qu'il aura à utiliser. Si ce dernier a prévu utiliser le langage UML par exemple, sa réflexion sur la décomposition de son système sera faite selon les règle de l'approche objet.

Puisque cette étude adresse l'approche acteur, la section suivante sera consacrée à l'approche composant de manière générale et particulièrement à l'approche acteur.

3 Modélisation orientée acteurs

3.1 Méthodologies orientées Composants

Les méthodologies de modélisation et de conception orientées composants préconisent des approches qui permettent la décomposition d'un système en sous-ensembles à la fois facilement maniables, spécifiques à un domaine et ayant une interface bien définie. Chacun de ces composants encapsule certaines fonctionnalités, telles que le calcul et la communication. La modélisation et la conception orientée objet contrôle la complexité dans un système par des mécanismes d'abstraction comme l'encapsulation de l'état de l'objet, les hiérarchies de classe, et les interfaces d'appel de méthode. Elle s'intéresse au comportement du système qu'elle modélise par l'interaction des comportements des objets qui le composent. Un objet ne peut interagir qu'avec un autre objet dont il a une référence, i.e., un identificateur unique. L'interaction entre objets se fait par invocation de méthode, ce qui entraîne le transfert du contrôle de l'objet demandeur à l'objet qui fournit le service demandé. Cette approche a été adaptée à la conception de systèmes embarqués et au logiciel temps-réel, tant au niveau de la modélisation, grâce à l' évolution de UML et à la création de profils spécifiques, qu'au niveau de l'intégration des méthodes propres à l'embarqué dans l'approche objet, en l'occurrence les objets synchrones [7] et les extensions temps-réel de CORBA.

3.2 Concept d'orientation acteurs

Historiquement, la notion de modélisation et de conception orientées acteur tire sa source des travaux de Gul Agha et autres [1] [2] [3]. Quant au terme acteur, il a été présenté pour la première fois dans les années 1970 par Carl Hewitt du MIT pour décrire le concept des agents intelligents autonomes [12]. Actuellement, la modélisation par acteurs reprend les idées développées par Agha et Hewitt en ce sens qu'elle décrit le comportement d'un système comme celui de composants autonomes dont cha-

cun communique avec un ensemble connu d'autres composants qui lui sont connectés. Par contre, certains aspects comme le fait que chaque acteur dispose de son propre flot de contrôle, ou que les communications entre acteurs sont asynchrones, ne sont pas nécessairement conservés dans la modélisation par acteurs. Cette méthodologie basée sur les composants appelés *"acteurs"* est particulièrement efficace pour la modélisation et la conception au niveau système, car, elle décompose un système du point de vue de ses *"actions"*. En effet, la méthodologie orientée acteurs préconise la décomposition des systèmes en composants interagissant et la recomposition des composants avec des modèles de calcul bien définis. Dans ce type de modélisation, les acteurs s'exécutent et communiquent avec d'autres acteurs dans un modèle. Essentiellement, contrairement à la méthodologie objet, un acteur définit des activités locales sans se référencer implicitement à d'autres acteurs. De part sa composition, un acteur a une composante interface bien définie. Cette interface différencie son état interne de son comportement, et définit de plus l'interaction entre cet acteur et son environnement. L'interface d'un acteur inclut les ports qui représentent ses points de communication et les paramètres utilisés pour configurer ses opérations par une entité extérieure telle qu'une interface utilisateur par exemple. Un acteur peut aussi être observé comme une encapsulation des actions paramétrables effectuées sur des données d'entrée et produisent des données de sortie après son exécution. Des données d'entrée et de sortie sont communiquées par des ports bien définis. Concernant son contrôle, il est fait par le modèle qui l'englobe à travers un ensemble de méthodes généralement identiques à tous les composants. Typiquement, ces opérations exécutent des fonctions comme *"initialisation"*, *"activation"* et *"arrêt"* du composant. En ce qui concerne sa communication, elle est définie par un ensemble d'actions, dont chacune traite des données d'entrée pour produire des données de sortie. A la différence des objets, un port d'un acteur n'a pas la sémantique de retour d'appel et son interface doit déclarer les propriétés dynamiques telles que des protocoles de communication et des propriétés temporelles. En ce qui concerne le modèle, sa configuration contient des canaux de communication explicites qui véhiculent des données d'un port d'un acteur à celui d'un autre conformément à un schéma de messages.

La question fondamentale pour les plates-formes orientées acteurs est la définition des modèles d'interaction entre ces acteurs. Pour son exécution, un acteur s'exécute dans le modèle dans lequel il réside. Ainsi, les sous-systèmes et les acteurs inclus dans un modèle définissent un *"système"*. Lorsque les différents sous-systèmes implémentent des modèles de calcul différents, le système est dit *"hétérogène"*. Comme pour les acteurs, un modèle hiérarchique peut aussi définir une interface externe appelée *"abstraction hiérarchique"* [16] [15]. Elle est composée des ports et des paramètres externes, qui sont distincts des ports et des paramètres des différents acteurs dans le modèle. Les ports externes d'un modèle peuvent être reliés par des canaux à d'autres ports

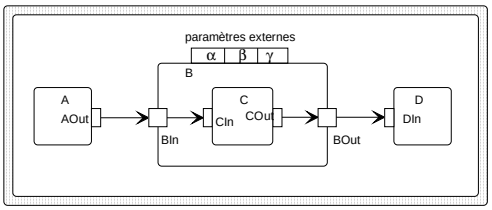

Figure 1: Exemple d'un modèle

externes d'un autre modèle ou aux ports des acteurs inclus dans le modèle. Des paramètres externes d'un modèle peuvent être utilisés pour déterminer les valeurs des paramètres des acteurs à l'intérieur du modèle. Sur la figure 1, l'abstraction hiérarchique de ce modèle est composé de port BIn et des paramètres externes α, β et γ.

3.3 Structures d'un acteur et d'un modèle

En considérant un acteur A, celui-ci dispose d'un ensemble des variables noté A.X qui peut être subdivisé en deux sous-ensembles : le sous-ensemble des variables d'interface appelées *ports* et celui des variables internes (paramètres et ses états) noté S. Ses variables d'interface sont de deux types : les ports d'entrée rassemblés dans le sous-ensemble P et ceux de sortie rassemblés dans le sous-ensemble Q. De la même manière, un modèle M dispose d'un ensemble de variables que l'on note M.X contenant le sous-ensemble noté M.S des variables internes (acteurs et canaux de communication) et les deux sous-ensembles M.P et M.Q des variables d'interfaces (ports d'entrée et de sortie) du modèle M. Ainsi, $\mathrm{M.X} = \{\{\mathrm{M.P}, \mathrm{M.Q}\} \bigcup \{\mathrm{M.S}\}\}$ où $\mathrm{M.S} = \{\{\bigcup \mathrm{A}_i\} \bigcup \{\bigcup \mathrm{c}_i\}\}$ avec $\bigcup \mathrm{A}_i$ et $\bigcup \mathrm{c}_i$ respectivement union des acteurs et union des canaux du modèle. Quant à ses opérations il en existe deux types :

- *Les opérations qui capturent le flot de données* : elles traitent de la manière de calculer les nouvelles données à partir des anciennes et de les envoyer. Ces opérations ont la caractéristique de changer les évaluations des variables. Elles vont répondre à la question *"comment recevoir, comment calculer et comment envoyer les données"*
- *Les opérations qui capturent le flot de contrôle* : celles-ci ne changent pas directement les valeurs des variables, mais par contre s'occupent du contrôle des opérations entre elles. Elles vont plutôt déterminer à quel moment le calcul et la communication devront se produire, c'est-à-dire, elles vont répondre à la question *"quand le calcul et la communication devront se produire"*.

3.4 Avantages de la méthodologie acteurs

Plusieurs plates-formes utilisent cette méthodologie entre autre Moses [9], et Ptolemy II [6]. La méthodologie orientée acteurs répond aux attentes des concepteurs de systèmes hétérogènes pour des raisons de structuration, de séparation des préoccupations et de décomposabilité et composabilité

modulaire :

- Sur le plan de la structuration du système, c'est une manière efficace de structurer et de conceptualiser un système. Elle considère un système comme une structure de sous-systèmes. Ceci met en évidence la structure causale et les dépendances de communication entre les sous-systèmes. Ce qui permet aux concepteurs d'établir efficacement la séparation entre le flot de contrôle et le flot des données.
- Sur le plan de la séparation des préoccupations, elle complète les techniques objet en y rajoutant une vision de découplage entre la transmission de données et le transfert de contrôle. Elle utilise la séparation des préoccupations [14], condition nécessaire dans la réutilisabilité des composants et la flexibilité du système.
- Sur le plan de la décomposabilité et de la recomposabilité modulaire, elle préconise la décomposition des systèmes en composants interagissant et la recomposition de ces composants avec des modèles de calcul bien définis.

4 Conclusion

Dans cet article nous avons présenté les différents outils ainsi que les approches de modélisation des systèmes hétérogènes. Nous nous sommes focalisés sur l'approche de modélisation basée sur les composants et avons détaillé la méthodologie orientée acteurs en présentant le concept d'acteur, sa structure, sa communication ainsi que son comportement, et en montrant les avantages que procure cette approche.

References

[1] G. Agha, *"Abstracting Interaction Patterns : A Programming Paradigm for Open Distributed Systems"*, IFIP Transactions, E. Najm and J.-B. Stefani, Eds., Chapman and Hall, 1997.

[2] G. Agha et al, *"Abstraction and modularity mechanisms for concurrent computing"*, IEEE Parallel and Distributed Technology : Systems and Applications, 1(2) :3-14, May 1993.

[3] G. Agha, I. A. Mason, S. F.Smith, and C. L. Talcott, *"A foundation for actor computation"*, Journal of Functional Programming, 7(1) :1-72, 1997.

[4] P. Alexander, C. Kong, D. Barton, P. Ashenden et C. Menon, *"Rosetta, Strawman V. 0.1"*, September 2002.

[5] G. Berry, *"The fundation of Esterel"*, MIT Press, Robin Milner edition, 1998.

[6] S.S. Bhattacharyya et al, *"heterogeneous concurrent Modeling and design in java, Volume I to III"*, Memorandum UCB/ERL M05/21 eecs, University of California at Berkeley, July, 2005

[7] F. Boulanger, *"Intégration de modules synchrones dans un langage à Objets"*, Ph.D. Thesis Université Paris XI, Decembre, 1993.

[8] F. Boussinot et R. De Simone, it "The Esterel language", Proceedings of the IEEE, 79(9), 1991.

[9] R. Esser, J.W. Janneck, *"Moses - A tool suite for visual modeling of discrete-event systems"*, Proc. of Symposium on Visual/Multimedia Approaches to Programming and Software Engineering, Italy, Sept, 2001.

[10] D.D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhu., *"SpecC : Specification Language and Methodology"*, Kluwer Academic Publishers, Dordrecht, Hardbound, ISBN 0-7923-7822-9, March 2000.

[11] N. Halbwachs and F. Lagnier and C. Ratel*"Programming and Verifying Real-Time Systems by Means of the Synchronous Data-Flow Language LUSTRE "*, IEEE Transactions on Software Engineering, 18(9), September, 1992.

[12] C. Hewitt, *"Viewing control structures as patterns of passing messages"*, Journal of Artificial Intelligence, 8(3) :323-363, June 1977.

[13] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, H. Zheng,*"Overview of the Ptolemy project"*, July 2, 2003, Technical Memorandum UCB/ERL M03/25.

[14] K. Keutzer, S. Malik, A.R. Newton, J.M. Rabaey and A. Sangiovanni-Vincentelli., *"System-Level Design : Orthogonalization of Concerns and Platform-Based Design"*, IEEE transactions on computer aided design of integrated circuits and systems, VOL.19,NO.12,December 2000, Pages 1507-1522.

[15] S. Neuendorffer and E.A. Lee, *"Hierarchical Reconfiguration of Dataflow Models"*, Invited paper, Conference on formal methods and models for codesign, MEMOCODE, San Diego, California, June 22-25, 2004.

[16] E.A Lee and S. Neuendorffer, *"Classes and Subclasses in Actor-Oriented Design"*, Invited paper, Conference on formal methods and models for codesign, MEMOCODE, San Diego, California, June 22-25, 2004.

[17] M. Mbobi, *"Modélisation Hétérogène Non-Hiérarchique"*, Ph.D. Thesis, Supélec, Université Paris XI (Orsay), Décembre, 2004.

[18] M. Mbobi, F. Boulanger, M. Feredj, *"Issues of Hierarchical Heterogeneous Modeling in Component Reusability"*, Proc. of the 2005 IEEE (IRI 2005), 15-17 August, 2005, Las Vegas, USA, pages 84 to 89

[19] SDL, *"Computer Networks and ISDN Systems"*, CCITT SDL, 1987.

[20] S. Swan, *"An Introduction to System Level Modeling in SystemC 2.0"*, White paper of OSCI, Mai 2001.

[21] SystemC Transaction Level Modeling Working Group Charter., June 2003, http ://www.systemc.org/

[22] UML, http ://www-306.ibm.com/software/rational/uml/

[23] VHDL, IEEE, Standard VHDL Language, Reference manual, STD 1076-1993. IEEE, 1993.

[24] J.S. Young and al, *"Design and specification of embedded systems in Java using successive, formal rafinement"*, Proceedings of Of 35th DCA, pg. 70-74, 1998.