Document de travail : vérification des compoants Kmelia en utlisant la méthode B

Mohamed MESSABIHI

Laboratoire d'Informatique de Nantes-Atlantique
2, rue de la Houssinière
B.P. 92208
44322 NANTES CEDEX 3
Prenom.Nom@univ-nantes.fr

— Modèle à base de composant, méthodes formelles, vérification de propriétés —

RAPPORT DE RECHERCHE

N^{o} xx.yy

Mars 2009

LINA, Université de Nantes – 2, rue de la Houssinière – BP 92208 – 44322 NANTES CEDEX 3
Tél. : 02 51 12 58 00 – Fax. : 02 51 12 58 12 – http://www.sciences.univ-nantes.fr/lina/

Mohamed MESSABIHI

Document de travail : vérification des compoants Kmelia en utlisant la méthode B

14 p.

Document de travail : vérification des compoants Kmelia en utlisant la méthode B

Mohamed MESSABIHI

Résumé

Catégories et descripteurs de sujets : D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Modules and interfaces*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal Methods*; D.2.11 [**Software Engineering**]: Software Architectures—*Languages*

Mots-clés additionnels et phrases : Composants, Services, Spécification formelle

Table des matières

1 Introduction

2 Principes généraux

La traduction des composants Kmelia vers B se passe en deux étapes

1. La traduction des composants primitifs :
   À chaque composant C décrit en Kmelia correspond une machine B qui porte le même identificateur dont l'état est obtenu en traduisant les différentes clauses Kmelia dans B (par ex. TYPES, CONSTANTES, VARIABLES, ...) et dont à chaque service $Serv$ dans C on associe une opération $Serv$ dans la machine C. La section 3 décrit brièvement la correspondance entre clauses Kmelia et celles de B . Le détail des algorithmes de traduction est donné dans la section 4.

2. La traduction des assemblages et compositions :
   Les assemblages et les composites vont se traduir pour l'essentiel par des compositions et rafinnements entre les machines construites précédemment en se servant des fameuses clauses REFFINES, INCLUDS, SEES, USES, PRMOMOTES, ..., avec parfois une réecriture de certaines machines ou renommage de certains éléments. Le détail de cette traduction est donné dans la section 5

3 Correspondance globale des concepts

Cette section montre une vue globale sur la traduction des différentes clauses Kmelia vers B (voir le tableau 1 et les listings).

Kmelia	**B**	**Commentaire**
COMPONENT Comp	MACHINE Comp	même identificateur
INTERFACE I	-	non encore définie
TYPES T	SETS T	l'invariant peut être utilisé pour compléter la définition des types
CONSTANTES C	CONSTANTES C	seuls les noms des constantes sont listés
	PROPERTIES Props	définition et/ou initialisation des constantes
VARIABLES VxT	VARIABLES V	Le typage se fait dans l'invariant
INVARIANT Inv	INVARIANT Inv & VxT	l'invariant de la machine est l'invariant du composant en rajoutant le typage de toute les variables
INITIALISATION Init	INITIALISATION Init	substitusions
SERVICES	OPERATIONS	chauqe service est décrit comme une opération B
serv (list_param : list_types) : type	res <– serv (list_param) =	cas général de signature
Pre P	PRE P & list_param x list_types	typage de la liste des paramètres en plus de la précondtion P
Post Q	ANY temp WHERE Q THEN list_var := temp	On choisi des valeurs quelconques pour que la post soit vérifiée
END_SERVICES ou end	END	fin de service ou fin du composant

TABLE 1 – Table de mapping entre clause Kmelia et celles de B

```
COMPONENT      Comp

INTERFACE      I

TYPES          T

CONSTANTS      C


VARIABLES      VxT

INVARIANT      Inv

INITIALISATION Init

SERVICES

   provided serv(id : Type) : r_Type

      Pre      P

      Post     Q

   end

   ...

END_SERVICES

# end of component
```

```
MACHINE        Comp


SETS           T

CONSTANTS      C

PROPERTIES     Props

VARIABLES      V

INVARIANT      Inv & VxT

INITIALISATION Init

OPERATIONS

   res <-- serv (id) =

      PRE P & id : Type
      THEN
      ANY List_var WHERE QQ
      THEN LIST_VAR := List_var
   END

   ...

END

# end of machine
```

Traduction d'un compsant Kmelia vers une machine B

4 Traduction des composants primitifs

4.1 Component

Pour chaque composant Kmelia primitif C, on crée une machine B qui porte le même identificateur C.

```
COMPONENT ATM
   ...
```

```
MACHINE ATM
   ...
```

4.2 Types

4.2.1 Integer

Le type de base Integer dans Kmelia peut être traduit systématiquement par le type $\mathbb{Z}$ (l'ensemble des entiers relatifs INT) en B .

Remarque 4.1 *Toutes les exprssions Kmelia sur les entiers sont également supportées par B sauf l'oérateur "/", car B ne sait pas encore manipuler les réels au niveau langage. Nous traduisons l'expression a/b par a div b (division entière).*

4.2.2 Boolean

Le type Boolean est modélisé en B par le type BOOL.

Remarque 4.2 *La valeur de vérité d'une expression booléenne b_exp en B est donnée par la fonction prédéfinie bool(b_exp).*

4.2.3 String

Le type String en Kmelia peut être traduit par STRING en B .

Remarque 4.3 *Cette solution ne permet d'utiliser que les deux opérations "$=$" et "$\neq$" sur les chaînes de caractères. Une alternative consiste à modéliser String par le type seq($0..255$) en B .*

```
VARIABLES
   available_cash : Integer;
   name : String;
   success = Boolean
```

```
VARIABLES
   available_cash, name, success
INVARIANT
   available_cash : INT &
   name : STRING &
   success : BOOL
```

4.2.4 range et enum

Les deux types Kmelia range et enum sont utilisés également dans B .

```
TYPES
   CASH : range 0..3000;
   STATE : enum {on, off};
```

```
DEFINITIONS
   CASH = 0..3000;
SETS
   STATE = {on, off}
```

4.2.5 array

Un tableau peut être modélisé par une fonction (ou application) dont l'ensemble de départ est constitué des index et l'ensemble d'arrivée est constituée des données. La fontion utilisée peut être *injective*, *surjective* ou *bijective* suivant les propriétés du tableau à modéliser.

```
TYPES
   Last_transactions : array [0..100] of CASH
```

```
VARIABLES
   Last-tansactions
INVARIANT
   NOTES : 0..100 --> CASH
```

4.2.6 struct

Le langage B ne propose pas de type pour définir des *structures*, donc le type struc de Kmelia ne peut pas être traduit directement en un type B .

Proposition 4.4 *On propose une solution basée sur les fonctions pour représenter les structures, où le nom du type est définicomme un type de base (générique) en B , et chaque champ de la structure est représenté par une fonction du type de base vers le type du champ en question.*

```
TYPES
   CashCard : struct{ code : Integer;
                      ident : String }
```

```
SETS
   CashCard
INVARIANT
   code : CashCard --> Integer &
   ident : CashCard --> String
```

4.2.7 setOf

La construction setOf Type dans Kmelia est traduite dans B par $\mathbb{P}$(Type).

```
TYPES
   swallowed_cards : setOf CashCard
```

```
VARIABLES
   swallowed_cards
INVARIANT
   swallowed_cards : pow(CashCard)
```

4.3 CONSTANTS

Les consatantes Kmelia sont modélisées par des constantes B où les noms des ces dernières sont listés dans la clause CONSTANTS, et leurs définitions sont exprimées par la clause PROPERTIES dans B .

```
CONSTANTS
   swallowed_size : Integer := 100
```

```
CONSTANTS
   swallowed_size
PROPERTIES
   swallowed_size : INT &
   swallowed_size = 100
```

4.4 VARIABLES

Les varaiables Kmelia sont modélisées par des varaibles B où leur noms sont listés dans la clause VARIABLES et leur typage est défini dans l'INVARIANT (voir l'exemple dans la section 4.2.3).

Remarque 4.5 *Tout identificateur dans B doit contenir au moins deux caractères. Les identificateur Kmelia contenant qu'un seul caractère seront renomés.*

4.5 INVARIANT

4.6 INITIALISATION

4.7 SERVICES

5 Traduction des assemblages

6 Exemple

7 Vérifications

7.1 Liste de propriétés à vérifier

7.2 Expérimentation

8 Conclusions et perspectives

Références

[1] Pascal ANDRÉ, Gilles ARDOUREL, et Gerson SUNYE. The Bosco Project, A JMI-Compliant Template-based Code Generator. In W. DOSCH et N. DEBNATH, réds., *Proceedings of the 13th International Conference on Intelligent and Adaptive Systems and Software Engineering*, pages 157–162, July 2004. ISBN 1-880843-52-X.

[2] Pascal ANDRÉ et Jean-Claude ROYER. *Ingéniérie Objet, Concepts et techniques, Mourad Oussalah (Ed.)*, Chapitre de livre 8 : Objets et spécifications formelles, pages 271–314. ISBN : 2-7296-60642-4. Interéditions, Mai 1997.

[3] Pascal ANDRÉ et Alain VAILLY. *Exercices corrigés en UML ; Passeport pour une maîtrise de la notation.*, volume 5 of *Collection Technosup*. Editions Ellipses, 2003. ISBN 2-7298-1725-5.

[4] Desmond D'SOUZA et Alan WILLS. *Objects, Components and Frameworks With UML : The Catalysis Approach*. Addison-Wesley, 1998.

[5] Andy S. EVANS, Jean-Michel BRUEL, Robert B. FRANCE, Kevin C. LANO, et Bernhard RUMPE. Making uml precise. In *OOPSLA'98 Workshop on "Formalizing UML. Why and How ?"*, October 1998. Vancouver, Canada.

[6] Martin GOGOLLA. UML for the impatient. Research Report 3/98, Universität Bremen, 1998.

[7] Object Management GROUP. The OMG Unified Modeling Language Specification, version 1.5. Rapport technique, Object Management Group, available at http ://www.omg.org/cgi-bin/doc ?formal/03-03-01, Juin 2003.

[8] Ludwik KUZNIARZ, Zbigniew HUZAR, Gianna REGGIO, Jean-Louis SOURROUILLE, et Miroslaw STARON, réds.. *Proceedings of «UML» 2003 Workshop on Consistency Problems in UML-based Software Development II*, San Francisco, California, USA, October 20 - 24 2003. IEEE and Blekinge Institute Of Technology. ISSN 1103-1581 also Research Report 2003 :06, http ://www.ipd.bth.se/consistencyUML/Consistency_Problems_in_UML_II.pdf.

[9] Ludwik KUZNIARZ, Gianna REGGIO, Jean-Louis SOURROUILLE, et Zbigniew HUZAR, réds.. *Proceedings of «UML» 2002 Workshop on Consistency Problems in UML-based Software Development*, Dresden, Germany, September 30 - October 4 2002. Blekinge Institute Of Technology. ISSN 1103-1581also Research Report 2002 :06, http ://www.ipd.bth.se/uml2002/RR-2002-06.pdf.

[10] Stephen J. MELLOR et Marc J. BALCER. *Executable UML - A Foundation For Model-Driven Architecture*. Addison Wesley, 2002. ISBN 0-201-74804-5.

[11] Chris RAISTRICK, Paul FRANCIS, Ian WILKIE, John WRIGHT, et Colin B. CARTER. *Model Driven Architecture with Executable UML*. Cambridge University Press, 2004. ISBN 0-521-53771-1.

[12] Gianna REGGIO et Roel WIERINGA. Thirty one problems in the semantics of uml 1.3 dynamics. In *Proceedings of the OOPSLA'99 Workshop on "Rigorous Modelling and Analysis of the UML : Challenges and Limitations*, 1999.

[13] James RUMBAUGH, Ivar JACOBSON, et Grady BOOCH. *The Unified Modeling Language*. Object-Oriented Series. Addison-Wesley, 1999. User Guide (ISBN 0-201-57168-4), Reference Manual (ISBN 0-201-30998-X).

[14] James RUMBAUGH, Ivar JACOBSON, et Grady BOOCH. *The Unified Software Development Process*. Object-Oriented Series. Addison-Wesley, 1999. ISBN 0-201-57169-2.

[15] Anthony SIMONS et Ian GRAHAM. 30 things that go wrong in object modelling with uml 1.3. In I Simmonds H KILOV, B RUMPE, réd., *Behavioral Specifications of Businesses and Systems*, chapitre 17, pages 237–257. Kluwer Academic Publishers, 2001.

[16] Ambrosio TOVAL, Jose SÀEZ, et Francisco MAESTRE. Automated Property Verification in UML Models. In Stefan Gruner MICHAEL LEUSCHEL et Stéphane Lo PRESTI, réds., *Proceedings of the 3rd Automated Verification of Critical Systems (AVoCS'03)*. DSSE Technical Report DSSE-TR-2003-2, Avril 2003.

Table des matières

Document de travail : vérification des compoants **Kmelia** en utlisant la méthode **B**

Mohamed MESSABIHI

Résumé

Catégories et descripteurs de sujets : D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Modules and interfaces*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal Methods*; D.2.11 [**Software Engineering**]: Software Architectures—*Languages*

Mots-clés additionnels et phrases : Composants, Services, Spécification formelle

LINA, Université de Nantes
2, rue de la Houssinière
B.P. 92208 — F-44322 NANTES CEDEX 3