Construction et réutilisation de spécifications LOTOS

Samira Sadaoui

▶ **To cite this version:**

Samira Sadaoui. Construction et réutilisation de spécifications LOTOS. AFADL : Approches Formelles dans l'Assistance au développement de Logiciels, 1998, Toulouse, France, 13 p. inria-00098706

HAL Id: inria-00098706
https://inria.hal.science/inria-00098706

Submitted on 26 Sep 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Construction et réutilisation de spécifications LOTOS

Samira SADAOUI
Loria, BP 239, F-54506, Vandœuvre-les-Nancy Cedex
e-mail : sadaoui@loria.fr

Résumé. Notre objectif est d'assister le spécifieur dans sa démarche de construction, de réutilisation et d'adaptation de spécifications LOTOS. Nous utilisons pour cela le modèle Proplane qui permet de décrire différentes étapes de développement de spécification et de mémoriser les décisions prises ainsi que leurs justifications. Dans ce papier, nous présentons une approche de construction de spécifications LOTOS : le style orienté processus et étudions deux exemples de réutilisation : adjonction d'un composant dans une architecture et réutilisation d'un style d'architecture.

1 Introduction

La réutilisation est considérée comme un moyen de surmonter la crise du logiciel, permettant d'obtenir des gains significatifs sur la qualité du logiciel, les coûts de maintenance et de développement [Par89, Kru92]. En général, quatre étapes sont présentes dans un processus de réutilisation [BP89] : la sélection du composant à réutiliser, sa compréhension, sa modification et sa composition. Nous nous intéressons aux spécifications formelles qui présentent un cadre intéressant pour la réutilisation, associant à la fois un bon niveau d'abstraction, le concept de modularité et des possibilités de preuve de correction entre une spécification et son implantation. Plusieurs approches ont été proposées pour unifier et adapter des composants existants : unification de fonctions et de modules [ZW95], application de techniques de raisonnement par analogie [Mai91, MS92], réutilisation par modification en unifiant les structures de types abstraits [DS97]. Par ailleurs, nous pouvons citer le projet REPLAY [CCJ+91] qui permet de réutiliser des développements formalisés dans le langage DEVA [CJ+89] en rejouant leurs étapes de développements.

Dans ce papier, nous nous intéressons à la construction et à la réutilisation de spécifications LOTOS. Ces constructions sont assistées en utilisant le modèle d'aide au développement Proplane [SD95]. Ce modèle a été expérimenté pour la construction de structures de données en GLIDER [Sou93], d'approches orientées modèle en Z [DS93] et de structures de contrôle en LOTOS [Lam97, LLS97]. La réutilisation dans Proplane [SS97] consiste à étudier la construction de spécifications non seulement à partir de spécifications existantes mais également à partir de leurs processus de développement. Le développement peut être réutilisé grâce à la mémorisation des décisions prises lors des différentes étapes. La réutilisation est réalisée à l'aide d'opérateurs permettant d'utiliser et d'adapter des textes de spécifications et de rejouer des étapes de développements.

Le papier est structuré comme suit. Dans le paragraphe 2, nous présentons brièvement le modèle Proplane et son application au langage de spécification LOTOS [ISO88]. À partir d'un exemple, le paragraphe 3 présente notre approche de construction d'une spécification LOTOS. Cette approche consiste à concevoir l'architecture d'un système puis à décrire son comportement. Nous étudions également deux cas de réutilisation d'une spécification existante. Les phases de sélection et de modification de composants s'appuient sur les architectures des spécifications LOTOS.

2 Construction d'une spécification LOTOS dans Proplane

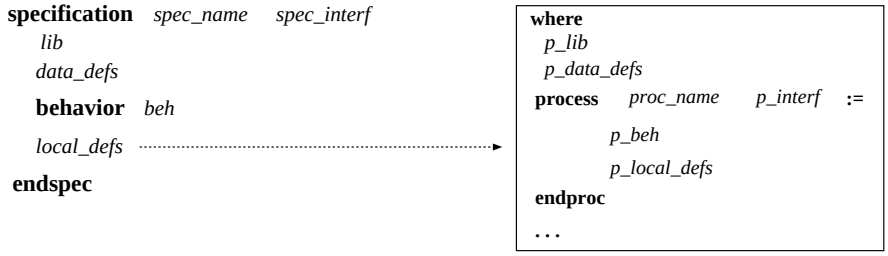
Le développement d'une spécification est vu comme une suite d'étapes. Une étape se définit comme une transition entre deux états et résulte de l'application d'un *opérateur*. Un état du développement est constitué de deux partie, un *plan de travail* et une spécification en cours

du développement. Le plan de travail est une collection de *tâches* reliées par l'intermédiaire de *réductions*. Une réduction consiste à décomposer une tâche en sous-tâches correspondant à de nouveaux sous-problèmes. Chaque étape a pour objectif de définir une ou plusieurs parties de la spécification désignées par des *variables* associées aux différentes tâches. La définition d'une variable est composée d'un type, d'un état (**I**nitial, **G**iven, **P**artial et **T**erminated) et d'une expression fonctionnelle. L'ensemble des variables et leurs définitions constituent le *méta-programme* [Lév95] décrivant la spécification en cours du développement.

Avec Proplane, une spécification LOTOS de nom *spec_name* se construit à l'aide de cinq parties nommées *spec_interf, lib, data_defs, beh* et *local_defs :*

- *spec_interf* dénote l'interface globale de la spécification
- *lib* désigne les types importés de la bibliothèque LOTOS
- *data_defs* contient les types définis par le spécifieur
- *beh* désigne l'expression de comportement global
- *local_defs* regroupe des définitions locales nécessaires à la définition du comportement *beh*, c'est-à-dire les importations de types, les définitions de types et les définitions de processus.

Dans le méta-programme, ces cinq parties sont représentées par des variables typées, *spec_interf*: INTERFACE, *lib* : NAME_LIST, *data_defs*: TYPE_DEF_LIST, *beh*: BEHAVIOR et *local_defs*: LOCAL_DEF_LIST. Dans ce qui suit, la spécification LOTOS est donnée avec ces cinq variables. La définition d'un processus de nom *proc_name* est constituée de l'interface *p_interf*, du comportement *p_beh* et des définitions locales *p_local_defs*. *p_lib* et *p_data_defs* contiennent les types utilisés dans l'interface du processus.



Des opérateurs de développement peuvent être appliqués pour aider à définir ces variables. Ils travaillent en parallèle sur le plan de travail et sur le méta-programme, réduisant une tâche et faisant évoluer le produit associé. Ils permettent de construire directement ou par réutilisation soit des types abstraits de données soit des expressions de comportement en termes d'algèbres de processus.

La figure 1 illustre la première étape de construction d'un exemple de spécification nommée CALCUL étudié dans la suite du papier. La partie gauche de la figure représente un état du plan de travail et la partie droite le méta-programme avec les variables associées aux tâches du plan.

La figure 2 correspond à la vue utilisateur. Elle est formée de la partie visible du plan de travail et de l'évaluation du méta-programme. Il est à noter que toutes les variables ont une valeur par défaut, i.e. qu'elles n'ont pas encore été définies par des réductions. Ces valeurs permettent d'évaluer à chaque étape du développement le méta-programme pour obtenir une spécification LOTOS.

3 Un exemple de réutilisation

Un composant CALCUL est présent dans la bibliothèque des composants réutilisables. À partir de ce composant, nous souhaitons proposer des réutilisations basées sur *(i)* sa composition avec un nouveau comportement à définir et *(ii)* adaptation des composants figurant dans son

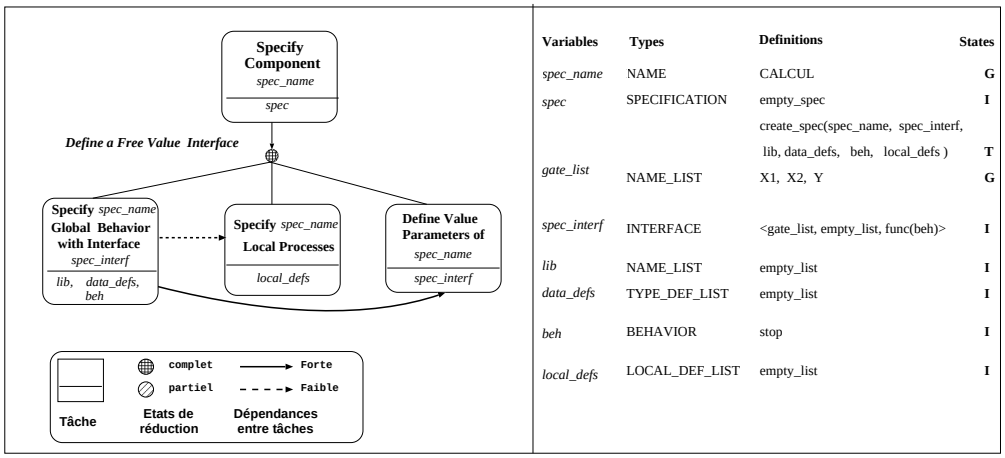


FIG. 1 – *Plan de travail et méta-programme du composant CALCUL*

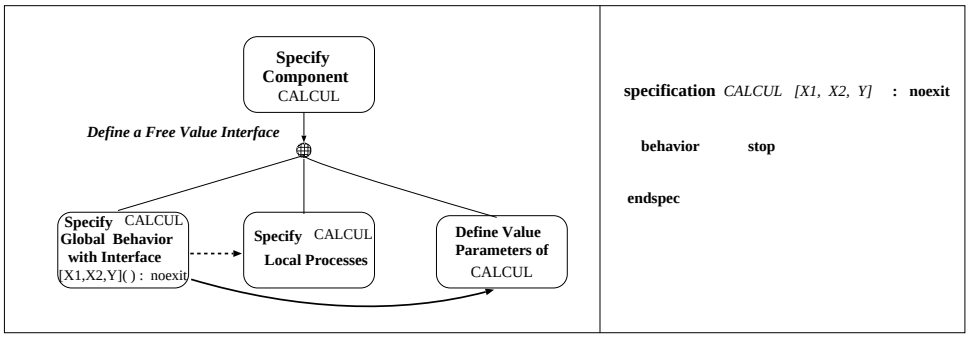


FIG. 2 – *Vue utilisateur correspondant à la figure 1*

architecture. On s'appuie sur la notion d'architecture pour guider la construction et la réutilisation de spécifications LOTOS. Cette architecture décrit une structure de processus parallèles communicants ou non.

3.1 Le composant CALCUL

3.1.1 Description informelle

Le but du composant CALCUL est de calculer la valeur y à partir des valeurs x_1, x_2, w_1 et w_2 et telle que $y = x_1 * w_1 + x_2 * w_2$. Son architecture ou sa structure est représentée figure 3. Elle permet de dégager :

- les sous-composants ou processus ADDER et CELL ;
- les interfaces des processus : ADDER$[Y_1, Y_2, Y]$, CELL$[X_1, Y_1](w_1)$ et CELL$[X_2, Y_2](w_2)$;
- les compositions parallèles entre ces processus : composition indépendante et composition communicante sur des portes privées Y_1 et Y_2.

Le calcul de la valeur y se fait de la façon suivante :

- x_1 et x_2 sont les valeurs envoyées respectivement sur les portes X_1 et X_2 par l'environnement externe ;
- $y_1 = x_1 * w_1$ est la valeur calculée et envoyée par le premier processus CELL sur la porte Y_1, w_1 étant un paramètre fixe ;
- $y_2 = x_2 * w_2$ est la valeur calculée et envoyée par le second processus CELL sur la porte Y_2, w_2 étant un paramètre fixe ;
- $y = y_1 + y_2$ est la valeur calculée et envoyée par le processus ADDER sur la porte Y.

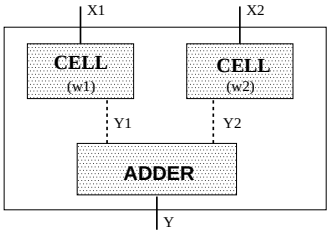


FIG. 3 – *Architecture du composant CALCUL*

Les valeurs manipulées sont des expressions symboliques décrites à l'aide du type abstrait EXP. Ce type est muni de deux opérations d'addition + et de multiplication ∗.

3.1.2 Description formelle du développement de CALCUL

Le plan de travail du développement complet du composant CALCUL est décrit figure 4 où les numéros sur les réductions correspondent aux étapes de développement.

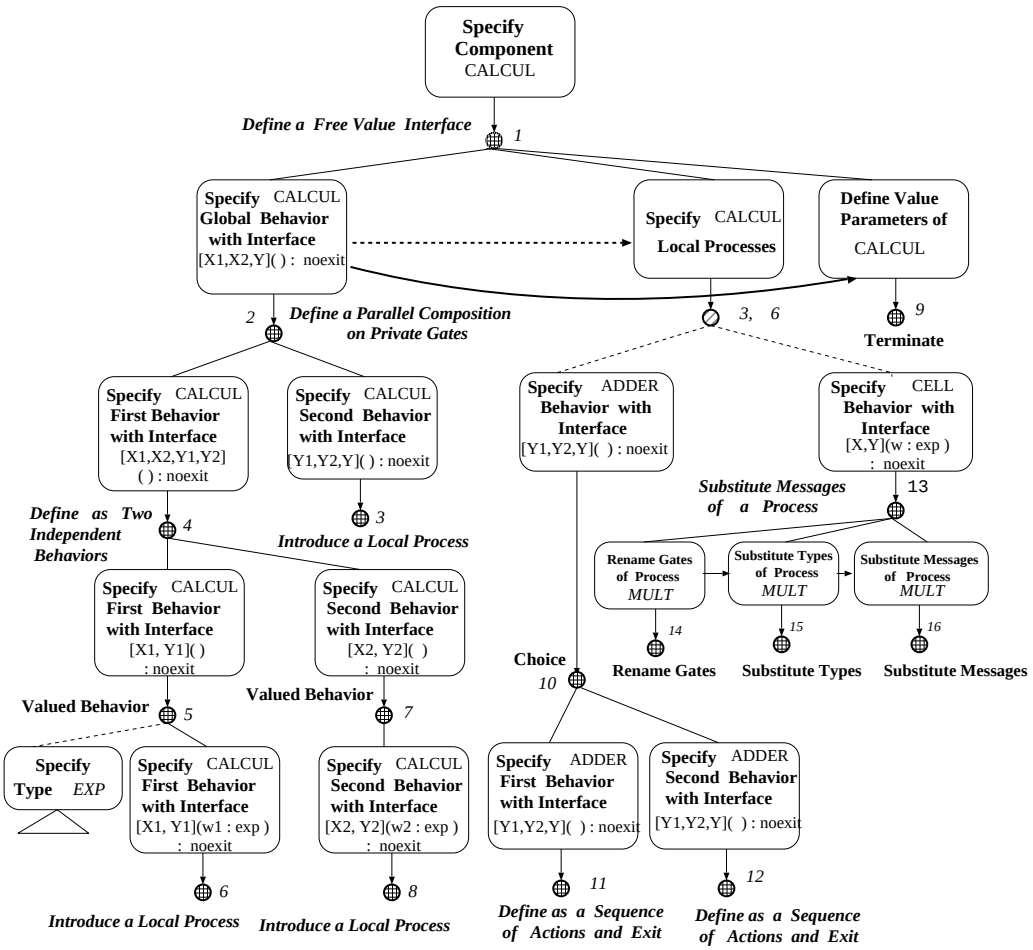


FIG. 4 – *Plan de travail du composant CALCUL*

La spécification LOTOS construite à l'aide de ce plan de travail est présentée figure 5.

3.1.3 Style orienté processus

Le langage LOTOS autorise plusieurs styles de spécification (monolithique, orienté contraintes, orienté ressources, orienté état, ...) [VSvSB90]. Ces styles ont pour but de concevoir des spécifica-

```
specification CALCUL [X1, X2, Y] : exit
    type     EXP    is
    sorts    exp
    opns     x1, x2, w1, w2 : -> exp
             _+_, _*_ : exp, exp -> exp
    endtype

    behavior
       hide   Y1, Y2   in
       ( CELL[X1, Y1](w1) ||| CELL[X2, Y2](w2)
                          |[Y1, Y2]|
                   ADDER[Y1, Y2, Y]    )
    where
      process  ADDER[Y1, Y2, Y] : exit    :=
      Y1 ? y1 : exp ;   Y2 ? y2 : exp;   Y ! (y1+y2) ;   exit
      [ ]
      Y2 ? y2 : exp ;   Y1 ? y1 : exp;   Y ! (y1 + y2) ;  exit
      endproc

      process  CELL[X, Y](w : exp) : exit    :=
        X ? x : exp  ;  Y !  (x * w)   ;  exit
      endproc
endspec
```

FIG. 5 – *La spécification CALCUL*

tions lisibles et bien structurées. Nous présentons une approche de construction de spécifications LOTOS inspiré des travaux de [Lam97] qualifié d'orienté processus : un problème est spécifié en utilisant en priorité des processus, les données émergeant lors de la définition des comportements.

En utilisant cette approche, nous avons quatre phases de construction du composant CALCUL (cf. figure 4) :

- définition de l'interface globale (*étapes 1 et 9*) : chaque système offre une interface lui permettant d'être en communication avec son environnement. Cet interface est composée des portes externes, de la liste des paramètres valeurs typées et de la fonctionnalité (capacité d'un système à terminer, **exit**, ou non, **noexit**) ;
- conception d'architecture (*étapes 2 à 8*) : une description plus ou moins abstraite de la structure du système est élaborée. Elle consiste à composer des processus parallèles en connaissant leurs interfaces ;
- description du comportement (*étapes 10 à 16*) : les mécanismes de réalisation du système sont étudiés. Cela consiste à définir le comportement des processus figurant dans l'architecture ;
- définition des données : l'interface et le comportement d'un système peuvent utiliser des types de données qui peuvent être construites simultanément ou bien différées après l'expression des comportements. Dans notre exemple, le type de données utilisé est EXP. Nous ne détaillons pas son développement.

L'architecture est conçue progressivement. On peut développer partiellement l'architecture, détailler les comportements des processus déjà introduits puis revenir sur l'architecture. L'interface globale est valuée une fois que l'architecture et les types utilisés dans les interfaces des processus ont été complètement définis. Il est à noter qu'il faut restreindre l'ensemble des constructions LOTOS pour concevoir une architecture.

Dans le plan de travail de la figure 4, nous avons deux types d'opérateurs :

- opérateurs définis par combinaison des opérateurs de contrôle LOTOS (composition parallèle, masquage, préfixage, choix, appel de processus, ...) ;
- opérateurs basés sur des mécanismes définis à l'aide du méta-programme (réutilisation

partielle de composants, renommage sur le texte de spécification et propagation, ...).
À ces opérateurs, on associe des guides, des raisonnements et des propriétés sur les comportements spécifiés.

Dans ce qui suit, nous décrivons, par exemple, les deux étapes de développement 1 et 12. Chaque étape est caractérisée par une décision formalisée par :

Tâche sélectionnée : la tâche à réduire dans l'agenda des tâches

Opérateur appliqué : choisir un opérateur parmi les opérateurs applicables i.e. dont les pré-conditions sont vérifiées,

Paramètres interactifs : les paramètres saisis par le spécifieur

Justification : description informelle justifiant le choix effectué

Description de l'étape 1 : Définition d'une interface non valuée

Tâche sélectionnée : **Specify Component** *CALCUL*

Opérateur appliqué : *Define a Free Value Interface*

Paramètres interactifs :

- les portes externes : X_1, X_2, Y

Justification : le problème est spécifié avec une algèbre de processus . On introduit l'interface globale consistant à donner les entrées (x_1 et x_2) et les sorties (y) du problème. Il s'agit ici d'une interface ne contenant pas de paramètres valeurs car ces derniers sont spécifiques aux processus qui seront introduits dans l'architecture. Il sera toujours possible de compléter l'interface de CALCUL ultérieurement par réduction de la sous-tâche **Define Value Parameters of** CALCUL.

Description de l'étape 13 : Définition du processus CELL par réutilisation

Tâche sélectionnée : **Specify** *CELL* **Behavior with Interface** *[X, Y](w : exp) : noexit*

Opérateur appliqué : *Substitute Messages of a Process*

Paramètres interactifs :

- nom du processus à réutiliser : MULT
- nom du composant contenant le processus : SPEC

Justification : dans la bibliothèque des composants, une spécification SPEC contient la définition d'un processus MULT dont le comportement est identique à celui de CELL modulo le renommage des messages envoyés sur les portes. La suppression des messages (valeurs algébriques) dans les deux processus CELL et MULT engendre le même processus non valué. Pour construire CELL, il suffit de substituer les messages envoyés sur les portes de MULT ainsi que leurs types.

3.2 Composition avec un nouveau comportement

Nous souhaitons raffiner la spécification CALCUL pour filtrer les valeurs qui arrivent sur les deux portes X_1 et X_2. Une solution consiste par exemple à composer CALCUL avec un nouveau comportement FILTRE jouant le rôle du filtre. La nouvelle spécification FILTRE-CALCUL est alors obtenue en composant parallèlement FILTRE et CALCUL avec les deux portes X_1 et X_2. Une architecture peut être spécifiée à différents niveaux d'abstraction. Ainsi, l'architecture du composant FILTRE-CALCUL est donnée figure 6.

Le comportement de la spécification FILTRE-CALCUL est comme suit :

- le sous-composant FILTRE reçoit deux valeurs sur la porte X et les distribue sur les deux portes X_1 et X_2 en n'autorisant qu'un certain ensemble de valeurs ;

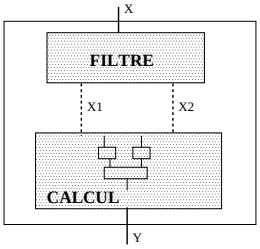


FIG. 6 – *Architecture du composant FILTRE-CALCUL*

- le sous-composant CALCUL reçoit les deux valeurs sur les portes X_1 et X_2 et renvoie le résultat $x_1 * w_1 + x_2 * w_2$ sur la porte Y.

En analysant la nouvelle architecture , nous remarquons que :

- les deux portes X_1 et X_2 sont internes ;
- CALCUL est maintenant un sous-composant de FILTRE-CALCUL. Dans la spécification FILTRE-CALCUL, il va falloir effectuer une transformation syntaxique de la spécification CALCUL en un processus.

Pour construire la spécification FILTRE-CALCUL, nous activons l'opérateur de réutilisation *Parallel Composition of a Specification on Private Gates*. Cet opérateur est basé sur les deux opérateurs de masquage (**hide ... in**) et de composition parallèle (**|[...]|**) de LOTOS et sur le mécanisme d'importation.

Application de l'opérateur *Parallel Composition of a Specification on Private Gates*

Justification

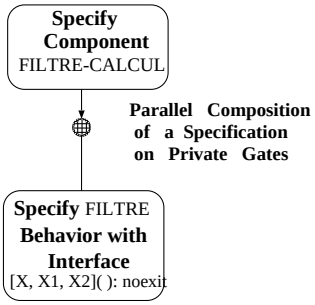
Nous souhaitons composer une spécification existante, CALCUL, avec un nouveau comportement, FILTRE. Il s'agit ici d'une composition parallèle communicante sur une liste de portes privées.

Rôle

L'opérateur permet :

(i) d'importer la spécification CALCUL en la transformant en un processus dans la nouvelle spécification, d'effectuer la composition parallèle de CALCUL et de FILTRE et de calculer automatiquement l'interface globale de la nouvelle spécification ;

(ii) de planifier la définition du nouveau processus FILTRE et éventuellement la définition des types utilisés par l'interface de ce processus.



Paramètres interactifs

- "La spécification à composer : " reused_spec = CALCUL
such that *type(reused_spec) = SPECIFICATION and gates(reused_spec)* $\neq \emptyset$
Pour notre type de composition, le composant existant doit possèder des portes externes.

- "Le nouveau processus : " new_process = FILTRE

- "Les portes externes du processus FILTRE :" gates_proc = X, X_1, X_2
such that *gates_proc* $\neq \emptyset$ *and gates_proc* $\cap$ *gates(reused_spec)* $\neq \emptyset$
Pour notre type de composition, les deux composants à assembler doivent possèder des portes communes.

- "Les paramètres valeurs du processus FILTRE :" values_proc = empty_values

- "Les portes de synchronisation internes :" hidden_gates = X_1, X_2
such that *hidden_gates* $\neq \emptyset$ *and hidden_gates* $\subseteq$ *gates(reused_spec)* $\cap$ *gates_proc*
Les portes de communication entre les deux composants doivent être incluses dans l'ensemble des portes communes.

La spécification obtenue après application de cet opérateur est illustrée figure 7 où les boîtes représentent les parties importées du composant CALCUL.

```
specification  FILTRE-CALCUL [X, Y] :  noexit                (* nouvelle spécification *)
  behavior  (* son comportement *)
    hide  X1, X2  in
      ( CALCUL[X1, X2, Y]   |[X1, X2]|   FILTRE[X, X1, X2] )
    where
      type  EXP  is
      sorts  exp
      opns  x1, x2, w1, w2 : -> exp
            _+_, _*_ : exp, exp -> exp
      endtype (* type de données utilisé dans CALCUL *)

      process CALCUL[X1, X2, Y] : exit :=
        hide Y1, Y2 in
        ( CELL[X1, Y1](w1) ||| CELL[X2, Y2](w2)
                  |[Y1, Y2]|
          ADDER[Y1, Y2, Y] )
        where
          process ADDER[Y1, Y2, Y] : exit :=
          Y1 ? y1 : exp;  Y2 ? y2 : exp;  Y ! (y1+y2) ;  exit
          [ ]
          Y2 ? y2 : exp;  Y1 ? y1 : exp;  Y ! (y1+y2) ;  exit
          endproc

          process CELL[X, Y](w : exp) : exit :=
            X ? x : exp;  Y ! (x * w) ;  exit
          endproc
      endproc (* transformation de la spécification CALCUL en un processus *)
    process  FILTRE[X, X1, X2] : noexit :=
      stop
    endproc (* nouveau processus *)
endspec
```

FIG. 7 – *La spécification FILTRE-CALCUL*

Remarques

- L'interface globale *spec_interf* est déduite automatiquement à partir du comportement global *beh* de la spécification FILTRE-CALCUL et une fois que tous les types utilisés dans *beh* ont été complètement définis : *spec_interf* = < *gates(beh), value_params(beh), func(beh)* > où
  - *gates(beh)* est l'ensemble des portes de *beh*, la porte interne i et les portes masquées

étant supprimées,

- *value_params(beh)* est l'ensemble des paramètres valeurs de *beh*,
- et *func(beh)* est la fonctionnalité de *beh*.

Dans notre cas, *spec_interf* = $< X, Y,$ *empty_list, noexit* $>$

– Avec la même démarche que précédemment, nous pouvons définir d'autres types de composition de spécifications : composition séquentielle, composition parallèle indépendante, générale, etc.

– La réutilisation dans Proplane ne fait pas appel à des copies mais à des liens. Ainsi, les composants réutilisés ne sont pas dupliqués dans le nouveau composant mais référencés par leur noms. Ce n'est qu'à l'évaluation du méta-programme du nouveau composant que les définitions des composants réutilisés sont effectivement importées. Ces liens permettent de bénéficier des corrections d'anomalies et des évolutions ultérieures du composant réutilisé. Ces modifications sont faites sans perturber les applications qui utilisent le composant.

3.3 Extension du calcul de la valeur y

L'objectif est d'étendre le calcul de y de la manière suivante $y = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4$ en réutilisant au maximum la spécification CALCUL. Pour ce faire, nous décomposons le calcul de y de la manière suivante :

$y_5 = x_1 * w_1 + x_2 * w_2$ calculée avec la spécification CALCUL ;

$y_6 = x_3 * w_3 + x_4 * w_4$ calculée avec la spécification CALCUL ;

$y = y_5 + y_6$ calculée avec le processus ADDER.

La structure de la nouvelle spécification EXT-CALCUL permettant de calculer la valeur y est visualisée figure 8.

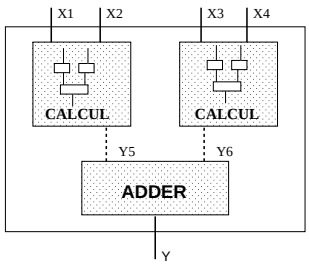


FIG. 8 – *Architecture du composant EXT-CALCUL*

Nous remarquons que cette architecture est similaire à celle du composant CALCUL :

- même nombre (3) de processus
- et même types de composition parallèles entre ces processus : une composition communicante sur des portes privées et une composition indépendante.

Nous pouvons donc réutiliser l'architecture de CALCUL pour construire celle de EXT-CALCUL en tenant compte des différences entre les deux architectures :

- l'interface globale $[X_1, X_2, Y]$ est substituée à $[X_1, X_2, X_3, X_4, Y]$;
- les deux portes de communication Y_1 et Y_2 sont substituées à Y_5 et Y_6 ;
- les deux instances du processus CELL sont substituées à deux instances de la spécification CALCUL :
  - CELL$[X_1, Y_1](w_1 : exp)$ est substituée à CALCUL$[X_1, X_2, Y_5]$
  - CELL$[X_2, Y_2](w_2 : exp)$ est substituée à CALCUL$[X_3, X_4, Y_6]$

Ces données correspondent aux paramètres introduits par le spécifieur dans les étapes de définition de l'interface et de l'architecture de la spécification CALCUL. Il existe deux solutions pour réutiliser l'architecture existante en tenant compte des substitutions :

- réutiliser la manière dont l'architecture a été construite en rejouant les étapes d'interface et d'architecture du développement CALCUL. Ceci consiste à réappliquer les opérateurs associés à ces étapes dans le nouveau développement EXT-CALCUL en introduisant les nouvelles valeurs des paramètres interactifs. Il faut donc rejouer dans l'ordre les étapes numérotées de 1 à 8 tout en remettant en cause les deux étapes 5 et 7 car le processus CALCUL ne possède pas de paramètres valeurs ;
- réutiliser directement la partie architecture de la spécification CALCUL en y appliquant et propageant les substitutions. Ceci est réalisé avec l'opérateur de réutilisation *Adapt a Specification from the Architecture*.

Nous choisissons de prendre en compte la seconde solution. Dans ce qui suit, nous donnons les étapes de développement permettant de construire la spécification EXT-CALCUL par adaptation de la spécification CALCUL.

3.3.1 Étape de réutilisation de l'architecture

Dans cette étape, nous appliquons l'opérateur *Adapt a Specification from the Architecture* permettant d'importer l'architecture d'une spécification puis de substituer ses objets (processus, interfaces et portes de synchronisation).

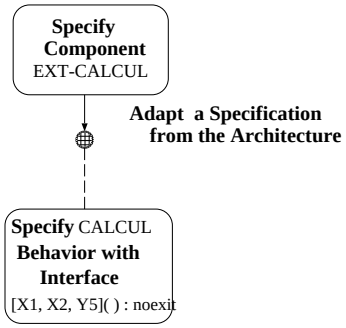
Application de l'opérateur *Adapt a Specification from the Architecture*

Justification
La nouvelle spécification, EXT-CALCUL, est obtenue en adaptant la spécification existante, CALCUL, à partir de son architecture jugée adéquate.

Rôle
L'opérateur permet :

(i) d'importer l'architecture de la spécification CALCUL en effectuant les substitutions introduites par le spécifieur et d'importer la définition des processus non substitués et les types utilisés. L'interface globale est calculée automatiquement ;

(ii) de planifier la définition des nouveaux processus introduits par les substitutions et éventuellement la définition des types utilisés par les interfaces de ces processus.



Paramètres interactifs

- "La spécification à réutiliser :" reused_spec = CALCUL
such that *type(reused_spec) = SPECIFICATION and behavior(reused_spec) = architecture*

La spécification réutilisée posséde bien une structure de processus parallèles.

- "Substitution des processus : " (old_procs, new_procs) = ((CELL), (CALCUL))

- "Interface de la première instance de CALCUL : " first_interf = $[X_1, X_2, Y_5]$

- "Interface de la seconde instance de CALCUL : " second_interf = $[X_3, X_4, Y_6]$

- "Substitutions des portes de communication :" (old_gates, new_gates) = $((Y_1, Y_2), (Y_5, Y_6))$

L'application de cet opérateur génère la spécification suivante où les boîtes représentent les substitutions effectuées sur la spécification réutilisée CALCUL et le reste les parties réutilisées et non adaptées. Comme le processus ADDER est non substitué, i.e. présent dans la nouvelle architecture, l'opérateur importe donc sa définition dans la spécification courante.

```
specification  EXT-CALCUL [X1, X2, X3, X4, Y]   : noexit   (* nouvelle spécification *)
  behavior (* architecture préservée *)
    hide  Y5, Y6  in
    ( CALCUL[X1, X2, Y5]  |||  CALCUL[X3, X4, Y6] )
            |[ Y5, Y6 ]|
          ADDER[ Y5, Y6 , Y] )
  where
    process ADDER[Y1, Y2, Y] : exit :=
      Y1 ? y1 : exp;  Y2 ? y2 : exp;  Y ! (y1+y2) ;  exit
      [ ]
      Y2 ? y2 : exp ;   Y1 ? y1 : exp;  Y ! (y1 + y2) ;   exit
    endproc  (* processus non substitué *)
    process  CALCUL[X1, X2, Y5] : noexit  :=
        stop
    endproc  (* nouveau processus *)
endspec
```

Remarques

Nous pourrions proposer une bibliothèque de schémas de composition parallèles ou de modèles d'architectures tels que : producteur-consommateur, client-serveur, réseau en étoile, en anneau, pipe-line, mémoires partagées, etc. Il suffit ensuite d'instancier ces schémas à l'aide d'opérateurs pour obtenir le système voulu.

3.3.2 Étape de définition des nouveaux processus

Dans cette étape, nous devons spécifier les nouveaux processus introduits dans la phase de réutilisation de l'architecture existante. Il s'agit ici de décrire le processus CALCUL. Comme ce dernier correspond au fait au composant CALCUL, il suffit de le réutiliser en effectuant la transformation syntaxique de la spécification CALCUL en un processus. Nous appliquons l'opérateur *Reuse a Specification* sur la sous-tâche **Specify** CALCUL **Behavior with Interface** *[X_1, X_2, Y_5]( ): noexit* avec comme paramètre la spécification CALCUL. Cet opérateur permet d'importer à partir de la spécification CALCUL les parties non définies dans la spécification courante EXT-CALCUL. Le processus ADDER a déjà été importé par l'opérateur *Adapt a Specification from the Architecture*. ADDER est maintenant un processus partagé par le processus CALCUL et la spécification EXT-CALCUL. Après application de l'opérateur *Reuse a Specification*, l'état de la spécification du processus CALCUL est donné figure 9.

```
type      EXP     is
sorts     exp
opns      x1, x2, w1, w2  :  ->  exp
          _+_,  _*_ : exp,  exp  ->  exp
endtype

process     CALCUL[X1, X2, Y5]( ) :  exit      :=
   hide     Y1,  Y2  in
   ( CELL[X1, Y1](w1)  |||  CELL[X2, Y2](w2)
                      |[Y1, Y2]|
             ADDER[Y1, Y2, Y5]     )
where
     process  CELL[X, Y](w : exp)  :  exit      :=
       X  ? x : exp;   Y  !  (x *  w) ;   exit
     endproc
endproc
```

FIG. 9 – *Parties importées de la spécification CALCUL*

4 Conclusion

Notre but est de mettre en œuvre des approches de construction et de réutilisation de spécifications LOTOS. De telles constructions sont guidées et facilitées par des opérateurs de développement. Ces derniers se définissent à partir des opérateurs de contrôle LOTOS et/ou des mécanismes définis à l'aide du méta-programme. Nous avons défini deux opérateurs de réutilisation, l'un permettant d'étendre une architecture existante par ajout d'un nouveau composant et l'autre de réutiliser un modèle d'architecture en ne modifiant que les comportements des processus impliqués. Ces opérateurs permettent de raffiner des spécifications, de modifier certaines parties et d'automatiser le calcul de leurs interfaces globales ainsi que certaines transformations. La réutilisation est réalisée en deux étapes : *(i)* comparaison d'architectures et *(ii)* adaptation de la spécification réutilisée en tenant compte des différences architecturales. L'aide est apportée dans l'étape d'adaptation.

Références

[BP89] Biggerstaff (T. J.) et Perlis (A. J.) (édité par). – *Software Reusability.* – ACM Press, 1989, Addison Wesley éditionvolume 1, 2.

[CCJ+91] Cazin (J.), Cros (P.), Jacquart (R.), Lemoine (M.) et Michel (P.). – Construction and reuse of formal program developments. *In : Proceedings of International Conference on the Theory and Practice of Software Develpment, TAPSOFT'91.* – LNCS 494, 1991.

[CJ+89] Cazin (J.), , Jacquart (R.), Lemoine (M.) et Michel (P.). – *Manipulation of formal developments expressed in DEVA.* – In K.H. Bennet Editor, Sofware Engineering Environments, Ellis Horword, 1989.

[DS93] Darimont (R.) et Souquières (J.). – A Development Model: Application to Z Specifications. *In : Proc. of the WG 8.1 Working Conference on Information System Development Process.* – Como (It), 1993.

[DS97] Darimont (R.) et Souquières (J.). – Reusing Operational Requirements : a Process-Oriented Approach. *In : Proceeding of the third International Symposium on Requirements Engineering.* – I.E.E.E. Press, January 1997.

[ISO88] ISO, IS 8807. – *Information Processing Systems, Open Systems Interconnection, LOTOS —A Formal Description Technique Based On the Temporal Ordering of Observational Behaviour*, July 1988.

[Kru92] Krueger (C. W.). – Software Reuse. *ACM Computing Surveys*, vol. 24, n 2, June 1992, pp. 131–183.

[Lam97] Lambolais (T.). – *Modélisation du développement de spécifications LOTOS.* – CRIN, Thèse, INPL, octobre 1997.

[LLS97] Lambolais (Th.), Lévy (N.) et Souquières (J.). – Assistance au développement de spécifications de protocoles de communication. *In : Conférence AFADL, Approches formelles dans l'assistance au développement de logiciels.* – Toulouse, Mai 1997.

[Lév95] Lévy (N.). – Improving a framework for modelling specification development. *In : Proceedings of the 5th International Symposium ISAS'95 : Information Systems Analysis and Synthesis, Baden-Baden, Germany*, pp. 20–24. – August 1995.

[Mai91] Maiden (N. A.). – Analogy as a paradigm for specification reuse. *Software Engineering journal*, January 1991.

[MS92] Maiden (N. A.) et Sutcliffe (A. C.). – Exploiting Reusable Specifications Through Analogy. *Communication of the ACM*, vol. 35, n4, April 1992, pp. 55–64.

[Par89] Partsch (H.). – *Generalize and reuse : An Exercise in reusing Transformational developments.* – Rapport technique nMIP-8915, Passau, University of Nijmegen, 1989.

[SD95] Souquières (J.) et Darimont (R.). – La description du développement de spécifications. *Technique et Science Informatiques*, vol. 14, n9, novembre 1995.

[Sou93] Souquières (J.). – Aides au développement de spécifications. – Thèse d'état, CRIN/Université de Nancy 1, Janvier 1993.

[SS97] Sadaoui (S.) et Souquières (J.). – Quelques approches de la réutilisation dans le modèle Proplane. *In : Conférence AFADL, Approches formelles dans l'assistance au développement de logiciels.* – Toulouse, Mai 1997.

[VSvSB90] Vissers (C.A.), Scollo (G.), van Sinderen (M.) et Brinksma (E.). – On the use of specification styles in the design of distributed systems. *Theoretical Computer Science 89 (1) 179-206*, 1990.

[ZW95] Zaremski (A. M.) et Wing (J. M.). – Specification Matching of Software Components. *Proc. of the 3th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, vol. 20, n4, October 1995, pp. 6–17.