Quelques éléments d'un langage de construction de spécifications

Jeanine Souquières

INRIA

UNITE DE RECHERCHE
INRIA-LORRAINE

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1053

Programme 1

QUELQUES ELEMENTS D'UN LANGAGE DE CONSTRUCTION DE SPECIFICATIONS

Jeanine SOUQUIERES

Juin 1989

RR-1053

Quelques éléments d'un langage de Construction de Spécifications

Towards a Framework for the Process Specification

Jeanine Souquières

CRIN-LORIA, BP. 239,
54506 Vandoeuvre-les-Nancy Cédex (France)

Résumé:

Le souhait d'obtenir des logiciels fiables, évolutifs et de moins en moins contraints par le contexte d'implantation a conduit à faire jouer un rôle de premier plan à la spécification. Celle-ci doit posséder de bonnes propriétés par exemple de modularité, correction et réutilisabilité. De telles propriétés peuvent être garanties par l'utilisation de mécanismes de construction bien connus tels l'enrichissement, l'instanciation ou des mécanismes moins classiques basés sur la réutilisation. Nous présentons, à partir d'un exemple, les mécanismes d'agrégation et de propagation permettant de réutiliser une spécification.

Absract:

The design of software products which are reliable, evolutive and not implementation dependent enforces the specification to play an important rôle. In order to support this paradigm, the specification must provide the user with some good properties such as modularity, correctness and reusability. Such properties will be warranted using well-known construction mechanisms like enrichment, renaming and instanciation or mechanisms based on reuse technics. We will study aggregation and propagation mechanisms allowing to reuse a source specification.

DIXIEME SEMINAIRE TUNISO-FRANCAIS D'INFORMATIQUE, MAI 1989, TUNIS

Quelques éléments d'un langage de Construction de Spécifications

Jeanine Souquières

CRIN-LORIA, BP. 239,
54506 Vandoeuvre-les-Nancy Cédex (France)

1 Introduction

Un intérêt croissant se manifeste depuis quelques années pour l'étude du développement de logiciels. Il s'est traduit par l'émergence de projets européens tels TOOL'USE [Dev86] et PROSPECTRA, des travaux de groupe [Ann86] et des wokshops internationaux [Wss87] sur ce thème. Dans les travaux sur le processus, les règles sont souvent décrites de manière informelle [Fea87], [Fic87]. Cependant des tentatives de formalisation d'opérateurs relevant du processus, notamment dans le cadre de la réutilisation sont en cours [Wir88], [GM88].

Dans [DL87], les opérateurs du processus proposés sont très syntaxiques. Ils sont fortement liés au langage de spécification et à ses structures de contrôle. Le langage de spécification utilisé dispose de trois types de constructions, aussi bien pour la description des opérations (AND, OR, FOR-ALL) que pour la description des types de données (CARTESIAN-PRODUCT, UNION, SEQUENCE ou TABLE ou SET) et à chacune de ces constructions, sont associés des mécanismes de construction tel la décomposition ou l'agrégation. Une formalisation est proposée dans [Dub88].

Feather propose un modèle de construction de spécifications basé sur des développements en parallèles. Le point de départ est une description simplifiée du problème. Il développe ensuite plusieurs élaborations en parallèle, chacune correspondant à un opérateur particulier (raffiner, ajouter un cas particulier) et à une motivation (prise en compte d'une contrainte particulière vis à vis du problème). La construction est incrémentale et son histoire est conservée. Cette approche favorise la maintenance et l'évolutivité du logiciel car il est possible de modifier aisément les étapes de construction et d'en dériver ensuite une nouvelle spécification. Une construction assistée faciliterait cette approche.

Une idée intéressante consiste à vouloir réutiliser le processus de construction plutôt que la spécification elle-même. Ceci nécessite de décrire précisément les opérateurs du processus et représente un travail de longue haleine. Une première étape consiste à comprendre les mécanismes de base utilisés et les stratégies d'utilisation de ces mécanismes. Cette approche favorise:

- l'évolutivité du logiciel,
- la réutilisation lors de la première construction. Actuellement la réutilisation a été étudiée sous l'aspect approche paramétrée dans laquelle on a prévu a priori les évolutions. Il existe deux orientations:

- définition de modules généraux et génériques qui seront utilisés pour des applications particulières. Mais pour quelles applications? En fait, les modules obtenus sont trés généraux donc peu efficaces.

- généralisation d'applications particulières afin de les réutiliser dans un problème similaire. Si les applications sont très spécifiques, elles sont candidates à la réutilisation mais satisfont rarement aux problèmes spécifiques posés.

- le développement d'outils supports de la construction.

Nous avons suivi une approche expérimentale en déroulant des exemples de construction avec un intérêt, non plus pour l'objet **spécification**, mais pour l'objet **processus** ou développement de la construction de cette spécification. Le cadre formel est celui des spécifications algébriques des types abstraits de données avec la possibilité de définir les opérations de manière constructive.

Dans le paragraphe 2, nous tenterons de mettre en évidence ce qui rélève du langage de spécifications et ce qui relève de la construction de la spécification. Dans le paragraphe 3, nous présenterons à partir d'un exemple une famille d'opérateurs particulière: **agréger** et **propager**. Nous en donnerons une description plus précise au paragraphe suivant.

2 Mécanismes de construction de spécifications

De nombreux travaux portent sur l'étude de spécifications plutôt que sur l'expression du développement ayant conduit à son élaboration. Certains de ces travaux ont abouti à la définition précise de langages de spécifications et à l'implantation de systèmes. Dans le cadre des types abstraits algébriques, on peut citer les langages ASL [SW83], PLUSS [Gau86], LPG [BDE87], OBJ [FGJM84], LARCH [GHW85] et le système SACSO [LPS87], [DLS86].

Dans la plupart des langages algébriques, il n'y a pas une distinction très claire entre le noyau du langage: sortes, opérations, axiomes et les mécanismes de construction tels la paramétrisation et l'enrichissement. De notre point de vue, ces mécanismes ne font pas partie du langage de spécifications mais sont des opérateurs de plus haut niveau portant sur une spécification. Cette distinction est faite dans ASL [SW83] ou une spécification est définie par une signature et un modèle. ASL peut être vu comme un langage présentant des opérateurs sur les spécifications.

Un certain nombre de mécanismes de construction sont maintenant bien connus, d'autres sont en cours d'étude.

- Parmi les mécanismes classiques de construction, citons:
  - l'enrichissement correspondant à l'adjonction d'une nouvelle sorte d'intérêt et de ses opérations. Il correspond aux opérateurs **use** de PLUSS [Gau86], **sum** d'ASL [SW83], **protecting** d'OBJ [FGJM84].
  - la paramétrisation et l'instanciation avec l'utilisation de bibliothèques de constructeurs de types,
  - les mécanismes de sous-sortes (héritage),
  - le renommage,
  - les mécanismes de masquage tels l'exportation, l'importation (**forget** et **restrict** d'ASL),
  - les mécanismes de restructuration [DLS87].
- Parmi les mécanismes moins classique, nous décrirons plus précisément au paragraphe suivant les opérateurs d'**agrégation** et de **propagation** permettant d'introduire un niveau de

structuration supplémentaire. Ces opérateurs sont basés sur la réutilisation de spécifications appartenant à une bibliothèque ou à la spécification courante. La formalisation des opérateurs proposés est en cours.

3 Agréger ou comment passer de un à plusieurs

Agréger et **propager** correspondent à une famille d'opérateurs dont le but est de réutiliser un type source par encapsulation. Ils consistent à:

- ajouter une nouvelle sorte d'intérêt paramétrée par le type source,
- définir ses opérations associées par abstraction des opérations associées au type source: c'est le rôle de l'opérateur **propager**.

Nous allons détailler, à partir d'un exemple, l'utilisation de ces opérateurs.

3.1 Un exemple

Un continuum [BT85] est un moyen de communication entre plusieurs utilisateurs. Chaque utilisateur peut envoyer des messages dans le continuum et lire des messages de la manière suivante: lors de la première utilisation, tout utilisateur doit lire le premier message du continuum. Il peut, indépendamment des autres utilisateurs lire le message suivant car il dispose d'une position individuelle dans les messages disponibles indiquant le dernier message lu. Il a la possibilité de savoir s'il y a des messages qu'il n'a pas encore lus.

Soit une spécification répondant à l'énoncé précédent. Nous nous plaçons ici dans le cadre de spécifications algébriques de types abstraits de données avec utilisation de constructeurs de types prédéfinis. Cette opération est constituée de la description du type de données du système nommé **CONTINUUM** et des fonctionnalités suivantes:

. *VIDE*: opération d'initialisation,
. *LIRE*: lecture d'un message par un utilisateur particulier et mise à jour de sa position de lecture dans le système,
. *ENVOYER*: envoi d'un message par un utilisateur dans le continuum,
. *ABONNER*: abonnement au continuum d'un nouvel utilisateur,
. *DESABON*: désabonnement d'un utilisateur particulier,
. *NOUVEAU*: y-a t'il des messages non encore lus par un utilisateur particulier dans le continuum?

Le profil de ces opérations est le suivant:

$$
\begin{array}{lll}
VIDE & : & \longrightarrow CONTINUUM \\
LIRE & : CONTINUUM, UTIL & \longrightarrow PC\,[CONTINUUM, MESSAGE\,] \\
ENVOYER & : CONTINUUM, UTIL, MESSAGE & \longrightarrow CONTINUUM \\
ABONNER & : CONTINUUM, UTIL & \longrightarrow CONTINUUM \\
DESABON & : CONTINUUM, UTIL & \longrightarrow CONTINUUM \\
NOUVEAU & : CONTINUUM, UTIL & \longrightarrow BOOLEEN
\end{array}
$$

Structure du système:

$$CONTINUUM = PC\,[\,FORUM : S\,[\,MESSAGE],\ POSITIONS : T\,[\,UTIL,\ NBLUS]\,]$$

. $CONTINUUM$ est composé d'une suite de messages et d'une table qui à un utilisateur associe le nombre de messages lus,
. PC, S et T désignent les constructeurs des types Produit Cartésien, Suite et Table,
. $MESSAGE$, $UTIL$ et $NBLUS$ sont définis dans l'environnement de la spécification,
. $BOOLEEN$ est un type de base prédéfini.
A titre d'exemple, on trouvera en annexe la définition d'une opération particulière, l'opération de lecture.

3.2 Objectif

Spécifier un nouveau système composé de plusieurs continua et muni des mêmes fonctionnalités que le système $CONTINUUM$. Notre point de départ est donc la spécification $CONTINUUM$ décrite précédemment.
Le système nommé **CA** est maintenant composé de plusieurs continua. Chacun d'eux doit pouvoir être caractérisé, par exemple par son **nom**. La notion de **plusieurs** peut être traduite:

- soit par un **ensemble** qui a un paramètre, un objet d'un ensemble étant caractérisé par son appartenance à l'ensemble,
- soit par une **suite** qui a un paramètre, un objet d'une suite étant caractérisé par son rang dans la suite,
- soit par une **table** qui a deux paramètres (le type indice et le type valeur), un objet d'une table étant caractérisé par son indice.

3.3 Quelle définition donner au type de données CA?

Choisissons la troisième solution: CA peut être défini à l'aide d'une table T, qui à un nom de continuum $NOMC$ associe un continuum $CONTINUUM$, ce continuum correspondant au type de données décrit dans la spécification précédente.

$$CA = T\,[\,NOMC,\ CONTINUUM\,]$$

3.4 Fonctions associées à CA = propager la définition de CA dans le profil des fonctions

Nous souhaitons munir CA des mêmes fonctionnalités que $CONTINUUM$, ces fonctionnalités étant généralisées à CA. Examinons comment obtenir de manière systématique les diverses possibilités en partant du profil des fonctions associées à $CONTINUUM$. Pour chaque possibilité, nous donnerons le profil des fonctions et une description intuitive. Nous détaillerons la définition de la fonction de lecture.

1. **Première possibilité:** *les fonctionnalités sont disponibles pour un continuum particulier caractérisé par son $NOMC$.*
   Nous définissons un mécanisme permettant d'accéder aux opérations de $CONTINUUM$ à travers CA.

Mise en oeuvre: pour chaque fonction associée à *CONTINUUM*, définir une fonction associée à CA de la manière suivante :
. Nommer la fonction en concaténant le nom du nouveau type (CA) et de la fonction,
. Dans le profil de la fonction, *CA* est substitué à *CONTINUUM*,
. Le paramètre *NOMC* est systématiquement introduit dans le domaine de la fonction. *NOMC* sert de clé d'accès à un des éléments de l'agrégation. Si on avait introduit une *SUITE*, on aurait introduit le *RANG* dans la suite au lieu de *NOMC*.

$$
\begin{array}{lll}
CA-VIDE1 & : & \longrightarrow CA \\
CA-LIRE1 & : CA,\ UTIL,\ NOMC & \longrightarrow PC\ [\,CA,\ MESSAGE\,] \\
CA-ENVOYER1 & : CA,\ UTIL,\ MESSAGE,\ NOMC & \longrightarrow CA \\
CA-ABONNER1 & : CA,\ UTIL,\ NOMC & \longrightarrow CA \\
CA-DESABON1 & : CA,\ UTIL,\ NOMC & \longrightarrow CA \\
CA-NOUVEAU1 & : CA,\ UTIL,\ NOMC & \longrightarrow BOOLEEN
\end{array}
$$

Intuitivement:,
. $CA-LIRE1$: lecture du dernier message non encore lu par util dans le continuum nomc et mise à jour de sa position de lecture dans ca,
. CA-ENVOYER1: envoi d'un message dans le continuum nomc par util,
. $CA-ABONNER1$: abonnement de util au continuum nomc,
. $CA-DESABON1$: désabonnement de util du continuum nomc,
. $CA-NOUVEAU1$: tous les messages du continuum nomc ont-ils été lus par util?

Remarque: Pour que le type CA soit complètement défini, il faudrait rajouter un constructeur permettant d'ajouter un nouveau continuum à un continuua, soit

$$CA-AJOUT\ :\ CA,\ NOMC \longrightarrow\ CA$$

Définition de la fonction de lecture: CA-LIRE1

lexique	**définitions**
res : $PC\ [CA,\ MESSAGE\,]$ continua mis a jour et message lu $ca,\ ca'$: CA continua avant et après mise à jour u : $UTIL$ nom de l'utilisateur m : $MESSAGE$ lu par util $nomc$: $NOMC$ du continuum concerné $conti,\ conti'$: $CONTINUUM$ concerné avant et après mise à jour	$res = CA-LIRE1(ca,\ u,\ nomc)$ $= < ca',\ m >$ **avec** $ca' = Modif_{table}(\ ca,\ u,\ conti')$ $< conti', m > = LIRE(conti,\ u)$ $conti = Acces_{table}(ca, nomc)$

La nouvelle fonction de lecture se définit par réutilisation directe de la fonction de lecture d'un *CONTINUUM* et des fonctions d'accès et de modification associées au constructeur de type *TABLE*.

Cette définition pourrait en partie être proposée par un système, en partie car il semble difficile de générer automatiquement la modification du système. Ceci est dû au fait que la fonction de lecture est une fonction assez complexe. Cette fonction a deux rôles: fonction d'accès à un message du continuum nomc et fonction de modification avec la mise à jour de la position de lecture de l'utilisateur dans le continuum nomc.

2. **Deuxième possibilité:** *généralisation de tous les types du domaine des fonctions*

Mise en oeuvre: pour chaque fonction associée à *CONTINUUM*,
. Nommer la fonction en concaténant le nom du nouveau type (*CA*) et de la fonction,
. Dans le profil de la fonction, *CA* est substitué à *CONTINUUM*,
. Tout **type** du domaine de la fonction est remplacé par *T [NOMC, type]* de même que *CONTINUUM* est remplacé par *T [NOMC, CONTINUUM]*

$$
\begin{array}{lll}
CA-VIDE2 & : CA & \longrightarrow CA \\
CA-LIRE2 & : CA, TUTIL & \longrightarrow PC\ [CA,\ MESSAGE\ ] \\
CA-ENVOYER2 & : CA, TUTIL, TMES & \longrightarrow CA \\
CA-ABONNER2 & : CA, TUTIL & \longrightarrow CA \\
CA-DESABON2 & : CA, TUTIL & \longrightarrow CA \\
CA-NOUVEAU2 & : CA, TUTIL & \longrightarrow BOOLEEN
\end{array}
$$

où *TUTIL* désigne *T [NOMC, UTIL]* et *TMES* désigne *T [NOMC, MESSAGE]*.

Intuitivement,
. Quel est le sens de *CA − LIRE2*?
Cette fonction pourrait être interprétée comme une fonction de lecture d'un message destiné à un seul utilisateur (application plus proche du courrier électronique).
. Que désigne *TUTIL*?
. *CA − ENVOYER2*: envoi d'un message particulier par util dans tous les continua de son choix,
. *CA − ABONNER2*: abonnement aux continua de son choix par util,
. *CA − DESABON2*: désabonnement aux continua de son choix par util,
. *CA − NOUVEAU2*?

Remarque: Cette possibilité n'a pas de signification immédiate ici.

3. **Troisième possibilité:** *généralisation de tous les types du codomaine des fonctions*

Mise en oeuvre: pour chaque fonction associée à *CONTINUUM*,
. Nommer la fonction en concaténant le nom du nouveau type (*CA*) et de la fonction,
. Dans le profil de la fonction, *CA* est substitué à *CONTINUUM*,
. Tout **type** du codomaine de la fonction est remplacé par *T [NOMC, type]* de même que *CONTINUUM* est remplacé par *T [NOMC, CONTINUUM]*

$CA-VIDE3$: $\longrightarrow CA$
$CA-LIRE3$: $CA,\ UTIL \longrightarrow PC\ [CA,\ T\ [NOMC,\ MESSAGE]\ ]$
$CA-ENVOYER3$: $CA,\ UTIL,\ MESSAGE \longrightarrow CA$
$CA-ABONNER3$: $CA,\ UTIL \longrightarrow CA$
$CA-DESABON3$: $CA,\ UTIL \longrightarrow CA$
$CA-NOUVEAU3$: $CA,\ UTIL \longrightarrow T\ [NOMC,\ BOOLEEN]$

Intuitivement,
. $CA-LIRE3$: lecture du premier message non encore lu dans chaque continuum par util et mise à jour de ses positions de lecture dans les divers continua,
. $CA-ENVOYER3$: envoi d'un message à tous les continua par util,
. $CA-ABONNER3$: abonnement à tous les continua par util,
. $CA-DESABON3$: désabonnement complet des continua par util,
. $CA-NOUVEAU3$: pour chaque continuum, y-a-t'il un message non encore lu par util?

Définition de la fonction de lecture : CA-LIRE3

lexique	**définitions**
$res : PC\ [CA,\ T[NOMC,\ MESSAGE]]$ continua mis à jour et messages lus $u : UTIL$ nom de l'utilisateur $ca, ca' : CA$ avant et après mise à jour $tmessages : T[NOMC,\ MESSAGE]$ messages lus par util $conti, conti' : CONTINUUM$ $nomc : NOMC$ $m : MESSAGE$	$res = CA-LIRE3(\ ca,\ u)$ $= < ca',\ tmessages >$ **pour chaque** $nomc$ **dans** ca : $ca' = Modif_{table}(ca,\ nomc,\ conti')$ $tmessages = Insert_{table}(tmessages,\ nomc,\ m)$ $< conti', m > = LIRE(Acces_{table}(ca,\ nomc),\ u)$ $tmessages_0 = tvide()$

La fonction de lecture se définit à l'aide d'une quantification sur les éléments du nouveau type de données et de l'opération de lecture dans un continuum.

4. **Quatrième possibilité:** *généralisation de tous les types du profil des fonctions*

Mise en oeuvre: pour chaque fonction associée à $CONTINUUM$,
. Nommer la fonction en concaténant le nom du nouveau type (CA) et de la fonction,
. Dans le profil de la fonction, CA est substitué à $CONTINUUM$,
. Tout **type** du profil de la fonction est remplacé par $T\ [NOMC,$ **type** $]$ de même que $CONTINUUM$ est remplacé par $T\ [\ NOMC,\ CONTINUUM\ ]$

$CA-VIDE4$: $\longrightarrow CA$
$CA-LIRE4$: $CA,\ TUTIL \longrightarrow PC\ [\ CA,\ TMES]$
$CA-ENVOYER4$: $CA,\ TUTIL,\ TMES \longrightarrow CA$

$CA - ABONNER4 : CA, TUTIL \longrightarrow CA$
$CA - DESABON4 : CA, TUTIL \longrightarrow CA$
$CA - NOUVEAU4 : CA, TUTIL \longrightarrow T\,[NOMC,\ BOOLEEN]$

où $TUTIL$ désigne T $[NOMC,\ UTIL]$ et $TMES$ désigne T $[NOMC,\ MESSAGE]$.

Intuitivement,
. Que désigne $TUTIL$?
Il s'agit d'une table qui à un nom de continuum associe un nom d'utilisateur. Elle pourrait être interprétée comme répertoire des noms de continua auxquels util est abonné,
. $CA - LIRE4$: lecture du premier message non encore lu dans chaque continuum auquel util est abonné et mise à jour des positions de lecture dans les divers continua,
. $CA - ENVOYER4$: envoi d'un message particulier par util dans tous les continua de son choix,
. $CA - ABONNER4$: abonnement aux continua de son choix par util,
. $CA - DESABON4$: désabonnement des continua de son choix par util,
. $CA - NOUVEAU4$: tous les messages des continua auxquels il est abonné ont-ils été lu par util?

Définition de la fonction de lecture: $CA - LIRE4$

lexique	**définitions**
res : PC $[CA,\ T[NOMC,\ MESSAGE]]$ continua mis à jour et messages lus u : UTIL nom de l'utilisateur ca, ca' : CA avant et après mise à jour $tmessages$: $T[NOMC,\ MESSAGE]$ messages lus par util $abonnement$: $T[NOMC,\ UTIL]$ $conti, conti'$: $CONTINUUM$ $nomc$: $NOMC$ m : $MESSAGE$	$res = CA - LIRE4(\ ca,\ abonnement)$ $= < ca',\ tmessages >$ **pour chaque** $nomc$ **dans** $abonnement$: $ca' = Modif_{table}(ca,\ nomc,\ conti')$ $tmessages = Insert_{table}(\ tmessages,\ nomc,\ m)$ $< conti',\ m > = LIRE(Acces_{table}(\ ca,\ nomc),\ u)$ $u = Acces_{table}(abonnement,\ nomc)$ $tmessages_0 = tvide()$

La fonction de lecture se définit à l'aide d'une quantification sur les éléments du nouveau type de données, d'un filtrage sur les continua auxquels util est abonné et de l'opération de lecture dans un continuum.

3.5 Conclusion sur l'exemple

Un des objectifs de ce travail est d'aider l'utilisateur à définir de manière systématique les fonctions associées au nouveau type de données. Sur l'exemple du continua, nous avons étudié les propagations possibles de la transformation de types dans les profils des opérations, en appliquant la même

transformation sur toutes les opérations. Une autre possibilité serait d'utiliser des propagations différentes suivant les fonctions.

Il est à noter que toutes les propagations étudiées n'ont pas une interprétation immédiate. L'une échoue et une autre, celle correspondant à la généralisation de tous les types du profil d'une fonction nécessite certaines hypothèses. Un système d'aide à la construction peut proposer les différentes possibilités: le spécifieur choisira en fonction des besoins requis par le problème à traiter.

Le choix d'une propagation étant fait par le spécifieur, la propagation peut s'effectuer de manière très assistée. Par exemple, dans le cas de la propagation standard permettant de généraliser les opérations de type existant à nouveau type, un système proposera la fonction d'accès associée au constructeur de type utilisé et la fonction correspondante associée au type existant.

4 Vers une formalisation de l'opérateur agréger

4.1 Sens intuitif

L'idée initiale est de réutiliser un type de données existant en lui rajoutant "une capsule": le type source est englobé. Son utilisation est liée à une approche ascendante et correspond à un enrichissement particulier: adjonction d'une nouvelle sorte d'intérêt et de ses opérations associées.

4.2 Description

Elle est donnée en terme d'une composition d'opérateurs et d'un nouvel opérateur consistant à propager l'introduction d'un niveau de structuration supplémentaire dans les profils des fonctions associées au type.

1. NOMMER(type-cible). Le profil de NOMMER est:

   SPEC, NOM-TYPE $\longrightarrow$ SPEC

2. REPRESENTER(type-cible).
   REPRESENTER est un opérateur de modification de spécifications de profil:

   SPEC, NOM-TYPE-SOURCE, CONSTRUCTEUR, INFO $\longrightarrow$ SPEC

   Son application s'effectue ici par l'adjonction à la spécification d'une équation de représentation du nouveau type:

   nom-type-cible = *CONSTRUCTEUR [info, nom-type-source]* où

   . *CONSTRUCTEUR* correspond au niveau de structuration introduit. Il peut être choisi parmi les constructeurs de types prédéfinis *TABLE, SUITE, ENSEMBLE, PILE, FILE.*

   . *info* correspond aux informations supplémentaires requises à l'introduction du niveau de structuration, par exemple NOMC dans l'exemple précédent.

   . *nom-type-source* est le nom du type de données de la spécification à partir duquel est défini le type cible.

3. PROPAGER(fonctions, type-cible)
Cet opérateur porte sur les fonctions. Il consiste à définir les opérations associées au type cible par abstraction des opérations associées au type source. Son profil est le suivant:

SPEC, NOM-FONCTIONS, NOM-TYPE-CIBLE $\longrightarrow$ SPEC

où NOM-FONCTIONS désigne la liste des noms des fonctions associées au type source.

Plusieurs propagations sont possibles. La propagation est définie à l'aide de la transformation permettant de définir le **type-cible** en fonction du **type-source**, transformation appliquée aux éléments du profil des fonctions. Une question se pose: doit-on propager de la même façon pour toutes les fonctions? Une piste de travail consiste à faire une différenciation entre les constructeurs du type de données et les autres opérateurs.

(a) **Généralisation du type d'intérêt et adjonction de info au domaine de la fonction**

Les fonctionnalités obtenues se rapportent à un élément agrégé, élément de **type-source**. Ceci correspond à une propagation minimum de base consistant à disposer sur le type cible de la même fonction que dans type source. Les nouvelles fonctions se définissent par réutilisation directe des fonctions associées au type-source et des fonctions associées à CONSTRUCTEUR utilisé de la manière suivante.

Soient les types X, YS et $SOURCE$ (YS fait partie de l'expression de $SOURCE$). $SOURCE$ est muni des opérations suivantes:

- **Constructeurs:**
  $Cree_{source}$: $\longrightarrow SOURCE$
  $Ajoute_{source}$: $SOURCE, X, YS \longrightarrow SOURCE$

- **Modificateurs:**
  $Modif_{source}$: $SOURCE, YS \longrightarrow SOURCE$

- **Fonction d'accès:**
  $Acces_{source}$: $SOURCE, X \longrightarrow YS$

Notre objectif est de définir un type $CIBLE$ dont l'expression est:

$$CIBLE = CONSTRUCTEUR\ [\ INFO,\ SOURCE\ ]$$

et ses opérations associées sont dérivées à partir des opérations associées au type $SOURCE$ de la manière suivante:

- **Adjonction d'un élément de type YS**
  $Ajoute_{cible}$: $CIBLE, YS, INFO \longrightarrow CIBLE$

  $Ajoute_{cible}(cible, ys, i)$
  $= Modif_{constructeur}(cible, i, Ajoute_{source}(Acces_{constructeur}(cible, i), ys))$

- **Modification d'un élément de type** *YS*
  $Modif_{cible}$: $CIBLE$, $INFO$, X, YS $\longrightarrow$ $CIBLE$

  $Modif_{cible}(cible,\ i,\ ys)$
  $= Modif_{constructeur}(cible,\ i,\ Modif_{source}(Acces_{constructeur}(cible,\ i),\ x.\ ys))$

- **Accès à un élément de type** *YS*
  $Acces_{cible}$: $CIBLE$, $INFO$, X $\longrightarrow$ YS

  $Acces_{cible}(cible,\ i,\ x) = Acces_{source}(Acces_{constructeur}(source,\ i),\ x)$

NB: Ces opérations sont des opérations partielles relativement au type *SOURCE*. Il peut y avoir autant d'opérations partielles que de composants dans l'expression du type *SOURCE*.
Cette propagation peut-être partiellement automatisée.

(b) **Généralisation de tous les types du codomaine de la fonction**

Les fonctions portent sur le type cible. Elles se définissent par une quantification sur tous les éléments de la nouvelle structure et utilisation directe des fonctions associées à type-source. Cette propagation peut-être partiellement automatisée.

(c) **Généralisation de tous les types du profil des fonctions**

Les fonctions portent sur le type cible. Les nouvelles fonctions se définissent par une quantification sur tous les éléments de la nouvelle structure et filtrage sur la clé d'accès.

4.3 Propagation systématique

L'application des propagations s'effectue de manière systématique. Si un type appartenant au domaine et au codomaine de la fonction est généralisé par l'application d'une des propagations précédentes, il est généralisé dans chacune de ses apparitions.

5 Conclusion

Nous avons détaillé sur un exemple l'utilisation des opérateurs **agréger** et **propager** qui définissent un enrichissement particulier. Un début de généralisation de cet opérateur a été présenté au paragraphe précédent. D'autres opérateurs sont en cours d'étude, par exemple l'opérateur **étendre** permettant de modifier un type source par adjonction de nouvelles composantes et propagation de cette modification sur les opérations associées au type de données modifié à l'aide de l'opérateur **propager** ci-dessus.

L'objectif poursuivi est de formaliser ces opérateurs de construction de spécifications en vue de proposer des aides à la construction basées sur la réutilisation de spécifications mais aussi sur la réutilisation de processus. Nous pensons arriver à proposer des heuristiques d'application des diverses propagations possibles en faisant une distinction entre les constructeurs du types de données et les autres opérations de ce type. L'idée à long terme, c'est de disposer d'un langage d'expression

du développement, les opérateurs introduits ici constituant les primitives de base du langage.

Remerciements: Ce travail s'effectue dans le cadre du projet SACSO visant à développer un environnement d'aide à la construction de spécifications. Je tiens à remercier ici J. P. Finance, N. Lévy, J. L. Rémy et M. de Silvestri pour les nombreuses discussions concernant ce travail.

6 Bibliographie

- [*Ann*86] Anna Gram, *Raisonner pour Programmer*, DUNOD (éd), 1986.
- [*BDE*88] Bert D., Drabik P., Echahed R. *Manuel de référence de LPG*, Rapport LIFIA, 1988.
- [*BT*85] Broy, M. and Tarlecki, A. *Algebraic Specification of the abstract data type "Continuum"*, Bulletin 26 of EATCS, pp. 32-35, 1985.
- [*Dev*87] *DEVA: Current Specifications and experiments*, April 1987, Research Report 87-13, TOOL'USE, Esprit Prject 510.
- [*DLS*86] Dubois, E., Levy, N. and Souquières, J. *SACSO : Methods and Tools for Constructing Requirements Specifications*, Proc. of Third Conference on Software Engineering, Versailles, 1986, pp 177-188.
- [*DL*87] Dubois E. and Van Lamsweerde A. *Making specification processes explicit*, Proc. of 4th International Workshop on Software Specification and Design, IEEE, Monterey (CA), 1987, pp. 169-177.
- [*DLS*87] Dubois, E., Levy, N. and Souquières. J. *Formalising Restructuring Operators in a Specification Process*, Proc. of First European Conference on Software Engineering, Strasbourg (FR), 1987, pp. 173-184.
- [Dub88] Dubois E. *Logical Support for Reasonning about the Specification and the Elaboration of Requirements* in The Role of Intelligence in Databases and Information Systems, WG2.6/WG8.1 Conference, Guangzhou, 43, nos. 3/4, July 1988.
- [*Fea*87*a*] Feather M. S. *Constructing Specifications by Combining Parallel Elaborations*, 1987, to appear in IEEE TSE.
- [*Fea*87*b*] Feather M.S. *The Evolution of Composite System Specifications*, Proc. of 4th International Workshop on Software Specification and Design, IEEE, Monterey (CA), april 1987, pp. 52-57.
- [*Fea*87*c*] Feather M. S. *Language support for the specification and development of composite systems*, ACM Transactions on Programming Languages and Systems, Vol. 9, Num. 2, April 1987.
- [*Fic*87] Fickas S. *Automating the Specification Process*, CIS-TR-87-05, University of Oregon, 1987.
- [*FGJM*84] Futatsugi, J., Goguen, A., Jouannaud J-P and Meseguer, J. *Principles of OBJ2*, Proc. 11th POPLE, 1984.
- [*GHW*85] Guttag J.V., Horning J.J. and Wing J.M. *The Larch familly of Specification Languages*, IEEE Software 2(5), September, 1985.

- [*Gau*86] Gaudel, M-Cl. *Towards Structured Algebraic Specifications*, ESPRIT'85: Status Report of Continuing Work, North-Holland, 1986, pp. 493-510.
- [*GM*88] Gaudel M.C., Moineau Th. *A theory of Software Reusability*. ESOP'88, Nancy (FR), pp.115-130, 1988.
- [*Gre*86] Green C., Luckmam D., Balzer R., Cheatham T. and Rich C., *Report on a knowledge-based software assistant*, In Readings In Artificial Intelligence and Software Engineering, C. Rich and R. Waters (Eds.), Morgan Kaufmann, Los Altos (CA), 1986.
- [*LPS*87] Levy N., Piganiol A. and Souquières J. *Specifying with SACSO*. Proc. of 4th International Workshop on Software Specification and Design, IEEE, Monterey (CA), 1987, pp. 236-241.
- [*SW*83] Sannella, D. and Wirsing, M. *A Kernel Language for Algebraic Specification and Implementation*, Proc. Int. Conf. on Foundations of Computer Theory. Bergholm Sweden, LNCS, 1983, pp. 308-322.
- [*Sin*86] Sintzoff, M. *Software Engineering Formalization*. Proc. 3rd Conference on software Engineering, Versailles, 1986, pp. 33-39.
- [*Wir*88] Wirsing, M. *Algebraic Description of Reusable Software Components*. Internal report, Passau, MIP-8816, 1988.
- [*Wss*87] Worshop on Software Specification and design, 4th, IEEE, Monterey (CA), 1987.

A Annexe

Définition de l'opération de lecture d'un CONTINUUM

lexique	**définitions**
but lecture d'un message par un utilisateur et mise à jour de sa position de lecture **objets** *res* : *PC* [*CONTINUUM*, *MESSAGE*] continuum mis à jour et message lu *message* : *MESSAGE* lu *conti*, *conti'* : *CONTINUUM* avant et après lecture du message *nomu* : *UTIL* nom de l'utilisateur désirant lire un message *forum* : *S* [*MESSAGE*] du continuum *positions*, *positions'* : *T*[*UTIL*, *NBLUS*] positions de lecture des utilisateurs avant et après lecture *nblus*, *nblus'* : *ENTIER* nombre de messages lus par *nomu* avant et après lecture du continuum	$res = LIRE\ (conti, nomu)$ $= < conti', message >$ **avec** $conti' = < forum, nblus' >$ $message = Acces_{suite}(forum, nblus')$ $forum = Acces - forum(conti)$ $positions' =$ **si** $Appt_{table}(positions, nomu)$ **alors** $Modif_{table}(positions, nomu, nblus')$ **sinon** $Insert_{table}(positions, nomu, 1)$ $nblus' =$ **si** $nblus < Taille_{suite}(forum)$ **alors** $nblus + 1$ **sinon** $nblus$ $nblus =$ **si** $Appt_{table}(positions, nomu)$ **alors** $Acces_{table}(positions, nomu)$ **sinon** 0 $positions = Acces - positions(conti)$

. $Insert_{table}$, $Modif_{table}$, $Acces_{table}$, $Appt_{table}$ sont les fonctions associées au constructeur de types *TABLE*,

. $Acces_{suite}$, $Taille_{suite}$ sont des fonctions associées au constructeur de types *SUITE*.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique