

PROJET RNTL ARCAD*

D3.1 - Spécification d'un modèle d'interaction

T0+18 : interne - T0+24 : public

Laurent Berger, Mireille Blay, Anne-Marie Pinna-Dery et Michel Rveill
Université de Nice / ESSI
930 route des Colles
06903 Sophia Antipolis Cedex

28 mai 2003

Résumé

Ce document est composé de Dans la partie précédente, nous avons étudié les solutions proposées par d'autres travaux pour la prise en compte des interactions et déterminé un ensemble de critères définissant les caractéristiques qui nous semblent essentielles pour un support "idéal" des interactions.

L'objectif de document est la définition d'une part, d'un modèle à interactions distribuées offrant une sémantique claire aux interactions et ce dans un contexte proche des réalités industrielles indépendante d'un langage de programmation ou d'une plate-forme donnée. Nos objectifs sont de pouvoir par exemple utiliser C++ avec CORBA, mais aussi Java et Java RMI. Et, d'autre part, la spécification d'un langage dédié à la description des interactions : Interaction Specification Language (ISL).

Nous nous basons sur le modèle à objets défini dans [Jau00] (lui même issu de [San95]) pour modéliser les interactions distribuées.

Avant d'entrer dans le vif du sujet avec la description formelle du modèle, le chapitre 1 donne une idée concrète de l'utilisation de notre modèle dans le monde compilé et fortement typé qu'est C++ et CORBA. Il explique, en effet, comment définir et enregistrer un schéma

*Le projet ARCAD (Architecture Extensible pour Composants Adaptables) a été labellisé en décembre 2000. Les différents partenaires sont France Télécom R&D (équipe ASR - Thierry.Coupaye@francetelecom.com), INRIA (projet Oasis - Denis.Caromel@inria.fr, projet Sardes - Daniel.Hagimont@inria.fr), Ecole des Mines de Nantes (équipe OCM - Thomas.Ledoux@emn.fr), le laboratoire I3S commun à l'Université de Nice - Sophia Antipolis et au CNRS (projet Rainbow - Anne-Marie.Pinna@unice.fr). Le coordonateur du projet est Michel.Riveill@unice.fr

d'interactions auprès d'un dépôt de schémas d'interactions (mécanisme permettant une représentation des comportements réactifs à l'exécution) et comment l'instancier sur un ensemble d'objets, puis présente ces différentes étapes de la vie d'un schéma d'interactions à travers un exemple de gestion des feux (tricolores et pour piétons) d'un croisement.

Avec le chapitre 2, nous entrons dans le vif du sujet. Ce chapitre fournit une description formelle de notre modèle, que nous appelons modèle à interactions distribuées, en se basant sur un modèle à objets distribués. Il montre l'intégration et la réutilisation qui est faite des principaux concepts du modèle sous-jacent. Il décrit également la sémantique des comportements réactifs.

Ensuite, le chapitre 3 décrit la fusion comportementale des règles d'interaction. Il présente les règles de réécriture (en sémantique naturelle) qui décrivent la sémantique de la fusion comportementale des règles d'interaction (cette sémantique est utilisée par le mécanisme d'héritage des interactions et lors de l'exécution des comportements réactifs) et démontre les deux propriétés fondamentales de cette fusion, à savoir sa commutativité et son associativité vis-à-vis de la sémantique des opérateurs réactifs.

Un modèle à interactions distribuées

Ce chapitre fournit une description formelle de notre modèle, que nous appelons modèle à interactions distribuées, en se basant sur un modèle à objets offrant un support du distribué. Il présente tout d'abord la syntaxe abstraite du langage ISL. Il montre l'intégration et la réutilisation qui est faite des principaux concepts du modèle sous-jacent : héritage, réification et instanciation notamment. Il décrit également la sémantique des comportements réactifs.

NOTE. — Dans ce chapitre, et le suivant, nous décrivons notre modèle indépendamment de tout modèle à objets. En effet, notre modèle à interactions distribuées est basé sur le langage ISL qui est indépendant de tout langage de programmation. Cependant, afin d'employer un vocabulaire connu et de simplifier la définition des propriétés du modèle, nous supposons que le modèle à objets sous-jacent offre la notion de métaclasses et définit les classes comme des objets.

Nous basons nos notations sur celles définies dans [Jau00] (elles mêmes issues de [San95]). Certes ce choix de représentation du modèle à objets sous-jacent est assez éloigné de celui utilisé par les langages que nous visons (C++ et Java en particulier), mais nous verrons dans la partie III que ce choix est malgré tout bien adapté (grâce à l'utilisation de systèmes réflexifs et de la programmation par aspects [KLM⁺97]).

1 Syntaxe abstraite du langage ISL et définitions préliminaires

La syntaxe abstraite décrite dans ce paragraphe utilise le formalisme METAL [KLM83] défini dans CENTAUR [INR89]. Par souci de simplicité la syntaxe abstraite du langage ISL est décomposée en deux sous ensembles, chacun faisant référence à l'autre : un élément en italique dans une syntaxe abstraite indique une référence à un élément de la syntaxe abstraite de l'autre sous-ensemble.

La syntaxe abstraite du tableau 5.1 décrit les éléments de portée globale dans le code ISL, c'est-à-dire les éléments ayant une visibilité de leur point de définition jusqu'à la fin du code ISL. Ces éléments sont des définitions des schémas d'interactions (déclaration *interactingScheme*). Ceux-ci sont constitués d'un ensemble de règles réactives (déclaration *interactingRule*) décrivant des comportements réactifs. La syntaxe abstraite des comportements réactifs est, quant à elle, décrite dans le tableau 5.2. Ceux-ci sont décrits à l'aide d'opérateurs (appelés opérateurs réactifs).

DÉFINITIONS PRÉLIMINAIRES. — Nous notons \mathcal{A} l'ensemble des attributs d'un objet, \mathcal{O} l'ensemble des objets, \mathcal{N} l'ensemble des noms de variables (identificateurs), \mathcal{C} l'ensemble des classes et métaclasses (nous avons $\mathcal{C} \subset \mathcal{O}$) et \mathcal{B} l'ensemble des méthodes. Nous notons $P^{fin}(X)$ l'ensemble des parties finies de l'ensemble X et $L^{fin}(X)$ l'ensemble des listes ordonnées et finies de l'ensemble X .

definition of ISL is**abstract syntax**

schemes	→	SchemeDef + ...
intScheme	→	InteractingScheme
scheme	→	SchemeName Objects Inherited InteractingRules
schemeClass	→	SchemeName Objects Inherited Class InteractingRules
interactingObjects	→	InteractingObject * ...
typeObject	→	Path Class Obj
inheritedSchemes	→	InheritedScheme * ...
inheritDeclaration	→	SchemeName ObjVars
vars	→	Obj + ...
importPath	→	SchemeName * ...
rules	→	InteractingRule + ...
rule	→	InteractingMessage Variables <i>InteractingBehavior</i>
message	→	TypeModifier MessageTarget MessageSelector MessageParameters
parameters	→	MessageParameter * ...
parameter	→	MessageParam
modifierParameter	→	MessageModifier MessageParam
modeParam	→	ModeModifier
modeTypeParam	→	ModeModifier TypeModifier
typeParam	→	TypeModifier
interactingVariables	→	InteractingVariable * ...
typeVar	→	TypeModifier Variable
name	→	implemented as Identifier
object	→	implemented as Identifier
target	→	implemented as Identifier
selector	→	implemented as Identifier
param	→	implemented as Identifier
mode	→	atomic in inout out
type	→	atomic void integer char string float mailbox
var	→	implemented as Identifier
ISL	::=	schemes
SchemeDef	::=	intScheme
InteractingScheme	::=	scheme schemeClass
SchemeName	::=	name
Objects	::=	interactingObjects
InteractingObject	::=	typeObject
ObjVars	::=	vars
Path	::=	importPath
Class	::=	name
Obj	::=	object
Inherited	::=	inheritedSchemes
InheritedScheme	::=	inheritDeclaration
InteractingRules	::=	rules
InteractingRule	::=	rule
InteractingMessage	::=	message
MessageTarget	::=	target
MessageSelector	::=	selector
MessageParameters	::=	parameters
MessageParameter	::=	parameter modifierParameter
MessageParam	::=	param
MessageModifier	::=	modeParam modeTypeParam typeParam
ModeModifier	::=	mode
TypeModifier	::=	type name
Variables	::=	interactingVariables
InteractingVariable	::=	var typeVar
Variable	::=	var

end definitionTAB. 5.1 – *Syntaxe abstraite des schémas d'interactions et des règles réactives*

definition of InteractingBehavior **is**
abstract syntax

delegate	→	InteractingBehavior
sequential	→	InteractingBehavior InteractingBehavior
concurrency	→	InteractingBehavior InteractingBehavior
ifThen	→	StrictMessageCallBehavior InteractingBehavior
ifThenElse	→	StrictMessageCallBehavior InteractingBehavior InteractingBehavior
call	→	MessageBehavior
exception	→	ExceptionId + ...
try	→	InteractingBehavior CatchBehaviors
cascadedCatch	→	CatchBehavior + ...
catch	→	ExceptionId InteractingBehavior
msg	→	MessageTarget MessageSelector MessageArgs
waiting	→	MessageBehavior Variable
assignment	→	Variable MessageCallBehavior
args	→	MessageArg * ...
modifierArg	→	ModeModifier MessageParam
except	→	implemented as String
InteractingBehavior	::=	delegate sequential concurrency ifThen ifThenElse call exception try
MessageArgs	::=	args
MessageArg	::=	parameter modifierArg
StrictMessageCallBehavior	::=	msg waiting
MessageCallBehavior	::=	msg assignment
MessageBehavior	::=	msg waiting assignment
CatchBehaviors	::=	cascadedCatch
CatchBehavior	::=	catch
ExceptionId	::=	except

end definition

TAB. 5.2 – *Syntaxe abstraite des comportements réactifs*

2 Règles réactives et comportements réactifs

Dans le chapitre précédent, nous avons introduit les concepts de règles réactives et de comportements réactifs comme moyen de décrire les sémantiques de communications au sein des schémas d'interactions. Dans ce paragraphe et avant de définir formellement les schémas d'interactions, nous définissons la structure des règles réactives ainsi que celle des comportements réactifs. Nous notons \mathcal{R} l'ensemble des règles réactives et \mathcal{G} l'ensemble des comportements réactifs.

2.1 Une métaclasse pour les règles réactives

Les règles réactives disposent d'un ensemble d'informations qui leur sont propres, dont notamment le message déclencheur et un comportement réactif. Pour ce faire, le modèle à interactions distribuées définit d'une part les règles réactives sous la forme d'une classe et, d'autre part, une classe spécifique pour toutes les règles réactives (métaclasse). Nous appelons $c_{metarule}$ cette métaclasse.

La classe $c_{metarule}$ des règles réactives définit les trois attributs suivants (il s'agit d'attributs de classe dont la valeur est présente dans l'état des règles réactives) :

- Un attribut $D \in \mathcal{A}$, dont le domaine des valeurs est $\mathcal{C} \times \mathcal{B}$ et dont la valeur est un doublet (P, M) , décrivant le message déclencheur de la règle d'interaction instance de la règle réactive. Il spécifie à quel message formel¹ est associé le comportement réactif décrit par la règle réactive.
- Un attribut $CR \in \mathcal{A}$, dont le domaine de valeurs est \mathcal{G} , qui sera exécuté en lieu et place du message déclencheur. Ce comportement réactif peut contenir au plus une invocation à la méthode spécifiée par le message déclencheur (afin d'exécuter le comportement initial).
- Un attribut $N \in \mathcal{A}$, dont le domaine de valeurs est \mathcal{N} , décrivant, dans le schéma d'interactions où est définie la règle réactive, le participant formel (son nom) déclenchant le comportement réactif CR .

1. Un message formel est un doublet *méthode + classe de l'objet sur lequel s'applique la méthode*. Il se distingue du concept de message du modèle à objets par le fait qu'il n'est pas défini sur un objet particulier.

NOTE. — Il est à noter que les comportements réactifs peuvent être définis sur n'importe quelle méthode publique (méthode d'interface) de n'importe quel objet du système (critères C2.2 et C3.2). Par conséquent, la méthode déclenchante (valeur de M) est une méthode publique (d'interface) de la classe (désignée par P) de l'objet déclencheur.

EXEMPLE. — Soit la règle réactive (définie dans le schéma d'interactions VisualiserNiveauSonore présenté au chapitre 4)

```
b.enDeplacement (int delta) -> b.enDeplacement (delta) // c.modifierVolume (delta)
```

Le message déclencheur (valeur de l'attribut D) est le doublet (Bouton, enDeplacement (int delta)), le nom du participant (valeur de l'attribut N) est b et le comportement réactif (valeur de l'attribut CR) est

```
b.enDeplacement (delta) // c.modifierVolume (delta).
```

2.2 Structure d'une règle réactive

La structure de classe des règles réactives définit, entre autres, les trois attributs suivants (attributs d'instance):

- Un attribut $p \in \mathcal{A}$, dont le domaine de valeurs est \mathcal{O} , décrivant l'objet pour lequel l'invocation de la méthode définie par l'attribut de classe M va déclencher le comportement réactif.
- Un attribut $cr \in \mathcal{A}$, dont le domaine de valeurs est \mathcal{G} , correspondant à l'instanciation de l'attribut de classe CR (ce mécanisme d'instanciation est décrit plus en avant dans ce chapitre).
- Un attribut $v \in \mathcal{A}$, dont le domaine de valeurs est $L^{fin}(\mathcal{O})$, correspondant aux valeurs des variables définies dans la règle d'interaction.

De plus, la structure des règles réactives définit le comportement suivant :

- Une méthode d'exécution du comportement réactif cr . Cette méthode est appelée lorsque le message déclencheur a été invoqué. Sa sémantique consiste à exécuter le comportement réactif cr .

Pour ce faire, le modèle à interactions distribuées définit une classe abstraite² c_{rule} , instance de la classe $c_{metarule}$ qui sera héritée par toutes les règles réactives et qui définit ces deux attributs d'instance. Ainsi toute règle réactive hérite de la classe c_{rule} et est une instance de la classe $c_{metarule}$ (ou de l'une de ses sous-classes). La figure 5.1 présente ceci, à l'aide de la notation UML [BJR97], pour le schéma d'interactions LierPistes (exemple de l'égaliseur graphique du chapitre précédent).

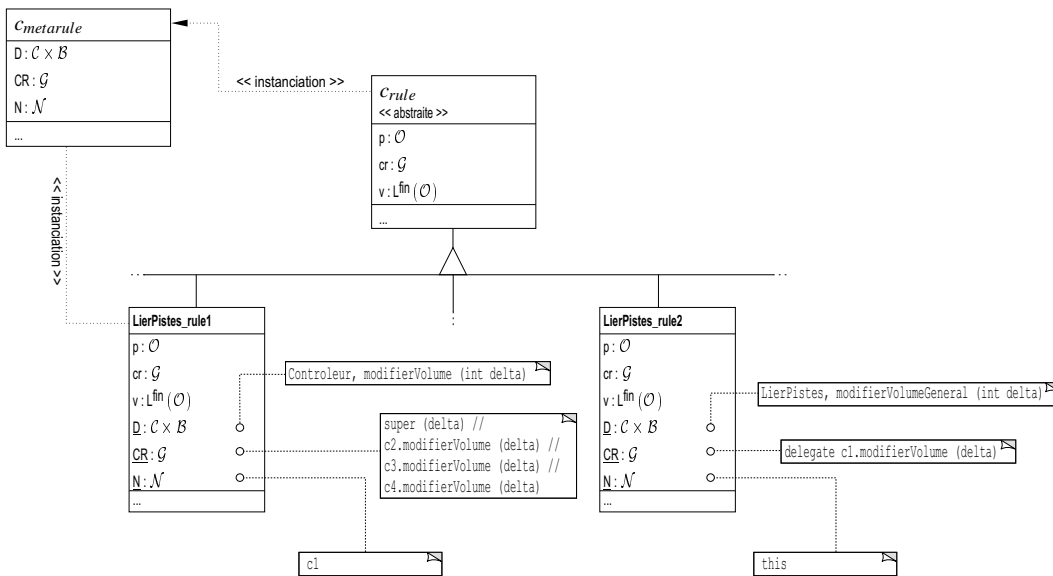


FIG. 5.1 – Description UML des règles réactives

2.3 Définition des comportements réactifs

Nous notons \mathcal{G} l'ensemble dénombrable des comportements réactifs pouvant être obtenus à partir de la syntaxe abstraite décrite dans le tableau 5.2. Le modèle à interactions distribuées définit, de manière récursive, les comportements réactifs à l'aide de l'ensemble des opérateurs réactifs décrits ci-dessous (chaque opérateur décrit un type de comportement réactif).

2. C'est-à-dire non instanciable.

L'opérateur réactif de délégation (delegate)

Il spécifie qu'un comportement réactif ne comportant pas le message déclencheur de la règle réactive soit traité comme étant ce message déclencheur. Nous notons \mathcal{G}_D l'ensemble des comportements réactifs basés sur l'opérateur réactif de délégation.

L'opérateur réactif séquentiel (sequential)

Il spécifie que deux comportements réactifs doivent être exécutés séquentiellement, c'est-à-dire de manière synchrone. Nous notons \mathcal{G}_S l'ensemble des comportements réactifs basés sur l'opérateur réactif séquentiel.

L'opérateur réactif concurrentiel (concurrency)

Il spécifie que deux comportements réactifs doivent être exécutés concurrentiellement, c'est-à-dire de manière asynchrone (non déterministe). Nous notons \mathcal{G}_C l'ensemble des comportements réactifs basés sur l'opérateur réactif concurrentiel.

L'opérateur réactif d'envoi de messages (msg)

Il spécifie que la méthode qu'il contient doit être exécutée (par envoi de messages). Nous notons \mathcal{G}_M l'ensemble des comportements réactifs basés sur l'opérateur réactif d'envoi de messages.

L'opérateur réactif d'affectation (assignment)

Il spécifie que la valeur d'une variable est affectée à la valeur de retour d'un comportement réactif d'envoi de message ou d'affectation. Nous notons \mathcal{G}_A l'ensemble des comportements réactifs basés sur l'opérateur réactif d'affectation.

L'opérateur réactif d'attente (waiting)

Il spécifie que l'exécution d'un message (par envoi de messages), d'une affectation ou d'une attente est contrainte par la fin de l'exécution d'un message provenant d'un autre comportement réactif (cette fin d'exécution est stockée dans une variable de type ISL `mailbox`). Nous notons \mathcal{G}_W l'ensemble des comportements réactifs basés sur l'opérateur réactif d'attente.

Les opérateurs réactifs conditionnels (ifThen et ifThenElse)

Ils spécifient une exécution conditionnelle d'un comportement réactif en fonction du résultat (un booléen) d'une exécution d'un comportement réactif d'envoi de messages. Nous notons \mathcal{G}_N l'ensemble des comportements réactifs basés sur les opérateurs réactifs conditionnels.

L'opérateur réactif de traitement d'exception (try)

Il spécifie un traitement (sous la forme d'un comportement réactif) à exécuter lorsqu'une exception est détectée dans un comportement réactif. Nous notons \mathcal{G}_T l'ensemble des comportements réactifs basés sur l'opérateur réactif de traitement des exceptions.

L'opérateur réactif d'exception (exception)

Il spécifie un refus d'exécution du message déclencheur. Ce refus pourra être capturé et traité par un *comportement réactif de traitement d'exception*. Nous notons \mathcal{G}_E l'ensemble des comportements réactifs basés sur l'opérateur réactif d'exception et \mathcal{G}_{-E} l'ensemble des comportements réactifs basés sur un autre opérateur réactif. Nous avons $\mathcal{G}_{-E} = \mathcal{C}_{\mathcal{G}_E}$.

2.4 Structure des opérateurs des comportements réactifs

Un opérateur réactif spécifie la sémantique de l'exécution d'un comportement réactif. Nous modélisons donc un comportement réactif par un objet représentant un opérateur réactif. Cette modélisation des comportements réactifs par des objets de première classe permet une spécialisation de la sémantique de ces comportements et l'ajout de nouveaux comportements réactifs (sous la forme de nouveaux opérateurs réactifs). Ainsi la structure d'un comportement réactif est la même que celle d'un objet.

Chacun des comportements réactifs dispose de sa propre classe. Celle-ci décrit la sémantique d'exécution de l'opérateur réactif associé et la sémantique de la fusion comportementale (cette sémantique est décrite dans le chapitre suivant sous la forme d'un ensemble de règles de réécriture). Les concepts communs à tous les comportements réactifs sont décrits dans une classe abstraite, nommée *C_{reactive}*, dont les classes des comportements réactifs héritent. La classe *C_{reactive}* définit les comportements suivants :

- Une méthode d'exécution du comportement réactif. Cette méthode abstraite est appelée lorsque le comportement réactif doit être exécuté. La sémantique exacte de cette méthode est définie plus avant dans ce chapitre, et ce pour chacun des comportements réactifs définis par le modèle à interactions distribuées.

- Une méthode de fusion du comportement réactif avec un autre comportement réactif. Cette fusion comportementale est notamment appliquée lorsqu’un ensemble de comportements réactifs doit être exécuté suite à l’invocation d’un message. La sémantique de cette fusion est décrite au chapitre suivant.

NOTE. — Il est à noter que les attributs définis dans la classe décrivant un comportement réactif sont dépendant de l’opérateur réactif. Par exemple, pour le comportement réactif séquentiel, sa classe définit deux attributs dont les valeurs sont les deux comportements réactifs qui doivent être exécutés séquentiellement.

2.5 Règles réactives et comportements réactifs : une métaclasse commune

Une règle réactive ainsi que ses instances, les règles d’interaction, sont des objets définis par le modèle à interactions distribuées pour sa propre utilisation. Il ne s’agit donc pas d’objets applicatifs. De plus, la sémantique de ces objets fait partie intégrante de la sémantique de la gestion des interactions par le modèle. Ainsi, la modification de cette sémantique grâce à des comportements réactifs peut totalement « corrompre » le fonctionnement des interactions.

De ce fait une règle réactive, mais également une règle d’interaction, ne doit pas être en mesure de participer à une interaction. De même, et pour des raisons identiques, un comportement réactif ne doit pas être en mesure de participer à une interaction.

Pour ce faire, le modèle à interactions distribuées définit une métaclasse commune aux règles réactives et aux comportements réactifs qui empêche la déclaration d’interactions dont un objet règle réactive, règle d’interaction ou comportement réactif est l’un des participants. Nous appelons *c metareactive* cette classe.

3 Sémantique des comportements réactifs

La sémantique de l’envoi de messages définie par le modèle à objets doit être étendue pour prendre en compte l’exécution des comportements réactifs des règles d’interaction associées à un message. Dans le modèle à objets, cette sémantique consiste à exécuter la méthode spécifiée par le message en lui fournissant l’environnement d’exécution de l’objet sur lequel s’applique la méthode ainsi que la valeur de ses paramètres.

3.1 Exécution des comportements réactifs

Dans le modèle à interactions distribuées, la sémantique de l’envoi de messages est conservée lorsqu’aucune règle d’interaction n’est définie sur le message devant être exécuté mais, lorsqu’une ou plusieurs règles d’interaction sont définies sur le message devant être exécuté, la sémantique de ces fonctions d’exécution consiste à exécuter une combinaison des comportement réactifs (nous verrons en 6.4 le mécanisme de cette combinaison) définis par ces règles d’interaction.

Définition 5.1 : Fonction d’exécution

Soit θ la fonction déterminant le comportement réactif associé à un message et dont le domaine de définition est $\mathcal{O} \times \mathcal{B} \rightarrow \mathcal{G}$. Soit λ^r la fonction d’exécution d’un comportement réactif et dont le domaine de définition est $\mathcal{G} \times L^{fin}(\mathcal{O}) \rightarrow \mathcal{O}$. La fonction d’exécution λ (décrivant la sémantique de l’envoi de messages et définie par le modèle à objets) est redéfinie par le modèle à interactions distribuées, lorsqu’elle est appliquée à un message interagissant, comme suit :

$$\lambda : \left\{ \begin{array}{l} \mathcal{O} \times \mathcal{B} \times L^{fin}(\mathcal{O}) \rightarrow \mathcal{O} \\ (o, m, a) \mapsto \lambda(o, m, a) = \lambda^r(\theta(o, m), a) \end{array} \right. \quad (5.1)$$

□

EXEMPLE. — En reprenant l’exemple de l’égaliseur graphique, lorsque toutes les interactions sont présentes dans le système alors, lorsque le message `c2.modifierVolume` va être invoqué, la fonction θ va être appelée. Elle va tout d’abord déterminer l’ensemble des règles d’interaction qui doivent être exécutées (phase 1) puis fusionner ces règles d’interaction (phase 2).

Le résultat de la phase 1 est l’ensemble composé des deux règles d’interaction suivantes :

```
c2.modifierVolume (int delta) -> c2.modifierVolume (delta) ; b2.deplacer (c2.obtenirVolume ())
c2.modifierVolume (int delta) -> c2.modifierVolume (delta) ; g2.afficherVolume (c2.obtenirVolume ())
```

Le résultat de la phase 2 (et donc de la fonction θ) est le suivant :

```
c2.modifierVolume (int delta) -> c2.modifierVolume (delta) ; b2.deplacer (c2.obtenirVolume ()) //
g2.afficherVolume (c2.obtenirVolume ())
```

3.2 Sémantique de l'exécution des comportements réactifs

Les règles d'interaction étant des citoyens de première classe (elles sont des objets, instances des règles réactives), elles possèdent une méthode permettant l'exécution de leurs comportements réactifs. De même, les comportements réactifs sont des objets qui disposent d'une méthode d'exécution. Dans ce paragraphe, nous décrivons la sémantique de cette méthode, que nous nommons λ^r , pour chacun des comportements réactifs.

Lors de l'exécution de son comportement réactif, une règle d'interaction dispose, en plus des paramètres instanciés du message interagissant (qui sont les valeurs des arguments du message déclencheur), d'un ensemble de variables qui lui sont propres. Nous associons donc à l'exécution d'un comportement réactif un environnement d'exécution contenant les valeurs de ces paramètres et variables. Nous notons \mathcal{E} l'ensemble dénombrable des environnements d'exécution. Puisqu'un environnement d'exécution est une liste de paramètres et de variables (tous des objets) nous avons $\mathcal{E} \subset L^{fin}(\mathcal{O})$.

De plus, chaque comportement réactif dispose d'une valeur de retour. Si le message qui a déclenché l'exécution du comportement réactif fait partie de ce comportement réactif alors la valeur de retour de ce dernier correspond à la valeur de retour de l'exécution du message déclencheur. Si le message qui a déclenché l'exécution du comportement réactif n'est pas présent dans le comportement réactif alors ce dernier a comme valeur de retour celle de l'exécution du dernier message du comportement réactif (par dernier message nous entendons le dernier message qui est exécuté).

Les définitions ci-après décrivent, à l'aide de la sémantique naturelle de TYPOL [Des88], et, pour chaque comportement réactif, la méthode d'exécution réactive λ^r qui lui est associé.

Définition 5.2 : Fonction d'exécution réactive (λ^r)

La fonction λ^r est définie comme suit :

$$\lambda^r : \left\{ \begin{array}{ll} \mathcal{G} \times L^{fin}(\mathcal{O}) & \rightarrow \mathcal{O} \cup \{nil\} \\ (cr, a) & \mapsto \lambda^r(cr, a) \end{array} \right. \quad (5.2)$$

Où a est la liste des paramètres instanciés du message déclencheur. □

Définition 5.3 : Fonction d'exécution d'un comportement réactif d'envoi de messages

Nous appelons fonction d'exécution d'un comportement réactif d'envoi de messages la fonction associant à un comportement réactif d'envoi de messages (désigné par l'opérateur `msg`) le résultat de son exécution (ce résultat est renvoyé sous la forme d'un environnement d'exécution).

Sa sémantique est définie par la règle suivante³ :

$$\frac{env \vdash call(o, m, \zeta(o, m, env)) : o', env'}{env \vdash msg(o, m) : o', env'}$$

où `call` est la fonction d'envoi de messages du langage cible et ζ est une fonction associant à un message (o, m) la valeur de ses paramètres instanciés définis dans l'environnement d'exécution e :

$$\zeta : \left\{ \begin{array}{ll} \mathcal{O} \times \mathcal{M} \times P^{fin}(\mathcal{E}) & \rightarrow L^{fin}(\mathcal{O}) \\ (o, m, e) & \mapsto \zeta(o, m, e) \end{array} \right. \quad (5.3)$$

□

Définition 5.4 : Fonction d'exécution d'un comportement réactif d'affectation

Nous appelons fonction d'exécution d'un comportement réactif d'affectation la fonction associant à un comportement réactif d'affectation (désigné par l'opérateur `assignment`) le résultat de son exécution (ce résultat est renvoyé sous la forme d'un environnement d'exécution).

Sa sémantique est définie par la règle suivante :

$$\frac{env \vdash cr_1 : o, env' \quad env' \vdash assign(x, o) : -, -}{env \vdash assignment(x, cr_1) : o, env'}$$

où `assign` est la fonction d'affectation du langage applicatif. □

3. Cette règle signifie que l'exécution d'un comportement réactif d'envoi de messages (opérateur `reactifmsg`) dans un environnement d'exécution env implique l'exécution de la fonction `call` dans ce même environnement. Le résultat de l'exécution du comportement réactif d'envoi de messages est le résultat de la fonction `call` (l'objet o) et l'environnement d'exécution généré par l'exécution de l'opérateur réactif d'envoi de messages est celui généré par la fonction d'envoi de message du langage cible.

Définition 5.5 : Fonction d'exécution d'un comportement réactif d'attente

Nous appelons fonction d'exécution d'un comportement réactif d'attente la fonction associant à un comportement réactif d'attente (désigné par l'opérateur `waiting`) le résultat de son exécution (ce résultat est renvoyé sous la forme d'un environnement d'exécution).

Sa sémantique est définie par la règle suivante :

$$\frac{\begin{array}{l} env \vdash \text{wait}(mb) : -, env' \\ env' \vdash cr_1 : o, env'' \end{array}}{env \vdash \text{waiting}(cr_1, mb) : o, env''}$$

où `wait` est une fonction qui attend l'exécution d'un message (grâce à la variable « boîte aux lettres » `mb`) et renvoie l'environnement issu de cette exécution (cette fonction ne renvoie aucune valeur). □

Définition 5.6 : Fonction d'exécution d'un comportement réactif de délégation

Nous appelons fonction d'exécution d'un comportement réactif de délégation la fonction associant à un comportement réactif de délégation (désigné par l'opérateur `delegate`) le résultat de son exécution (ce résultat est renvoyé sous la forme d'un environnement d'exécution).

Sa sémantique est définie par la règle suivante :

$$\frac{env \vdash cr_1 : o, env'}{env \vdash \text{delegate}(cr_1) : o, env'}$$

□

Définition 5.7 : Fonction d'exécution d'un comportement réactif séquentiel

Nous appelons fonction d'exécution d'un comportement réactif séquentiel la fonction associant à un comportement réactif séquentiel (désigné par l'opérateur `sequential`) le résultat de son exécution (ce résultat est renvoyé sous la forme d'un environnement d'exécution).

Sa sémantique est définie par les trois règles suivantes :

$$\frac{\begin{array}{l} env \vdash cr_1 : o, env' \\ env' \vdash cr_2 : o', env'' \end{array}}{env \vdash \text{sequential}(cr_1, cr_2) : o, env'' \mid m \in cr_1} \qquad \frac{\begin{array}{l} env \vdash cr_1 : o, env' \\ env' \vdash cr_2 : o', env'' \end{array}}{env \vdash \text{sequential}(cr_1, cr_2) : o', env'' \mid m \in cr_2}$$

$$\frac{\begin{array}{l} env \vdash cr_1 : o, env' \\ env' \vdash cr_2 : o', env'' \end{array}}{env \vdash \text{sequential}(cr_1, cr_2) : o', env'' \mid m \notin cr_1 \wedge m \notin cr_2}$$

□

Définition 5.8 : Fonction d'exécution d'un comportement réactif concurrentiel

Nous appelons fonction d'exécution d'un comportement réactif concurrentiel la fonction associant à un comportement réactif concurrentiel (désigné par l'opérateur `concurrency`) le résultat de son exécution (ce résultat est renvoyé sous la forme d'un environnement d'exécution).

Sa sémantique est définie par les trois règles suivantes :

$$\frac{\begin{array}{l} env \vdash cr_1 : o, env' \\ env \vdash cr_2 : o', env'' \end{array}}{env \vdash \text{concurrency}(cr_1, cr_2) : o, env' \cup env'' \mid m \in cr_1}$$

$$\frac{\begin{array}{l} env \vdash cr_1 : o, env' \\ env \vdash cr_2 : o', env'' \end{array}}{env \vdash \text{concurrency}(cr_1, cr_2) : o', env' \cup env'' \mid m \in cr_2}$$

$$\frac{\begin{array}{l} env \vdash cr_1 : o, env' \\ env \vdash cr_2 : o', env'' \end{array}}{env \vdash \text{concurrency}(cr_1, cr_2) : o \vee o', env' \cup env'' \mid m \notin cr_1 \wedge m \notin cr_2}$$

□

Définition 5.9 : Fonction d'exécution d'un comportement réactif conditionnel

Nous appelons fonction d'exécution d'un comportement réactif conditionnel la fonction associant à un comportement réactif conditionnel (désigné par l'opérateur `ifThenElse`) le résultat de son exécution (ce résultat est renvoyé sous la forme d'un environnement d'exécution).

Sa sémantique est définie par les quatre règles suivantes (où la condition est le message $c = (o_c, m_c)$ et $null$ l'absence de valeur de retour) :

$$\frac{env \vdash \text{msg}(o_c, m_c) : \text{vrai}, env' \quad env' \vdash cr_1 : o, env''}{env \vdash \text{ifThenElse}(c, cr_1, cr_2) : o, env''} \quad \frac{env \vdash \text{msg}(o_c, m_c) : \text{vrai}, env' \quad env' \vdash cr_1 : o, env''}{env \vdash \text{ifThen}(c, cr_1) : o, env''}$$

$$\frac{env \vdash \text{msg}(o_c, m_c) : \text{faux}, env' \quad env' \vdash cr_2 : o, env''}{env \vdash \text{ifThenElse}(c, cr_1, cr_2) : o, env''} \quad \frac{env \vdash \text{msg}(o_c, m_c) : \text{faux}, env' \quad env' \vdash cr_1 : o, env''}{env \vdash \text{ifThen}(c, cr_1) : null, env''}$$

□

Définition 5.10 : Fonction d'exécution d'un comportement réactif gérant les exceptions

Le langage ISL permet de capturer des exceptions émises par un comportement réactif ou par un message invoqué depuis un comportement réactif (grâce à l'opérateur réactif `msg`). Ainsi, le support des exceptions est en grande partie dépendant du langage applicatif. En effet, dans un langage applicatif ne définissant pas le concept d'exception, la mise en place du support des exception du modèle à interactions distribuées ne peut être réalisée.

De ce fait, la sémantique de fonction d'un comportement réactif gérant les exceptions présentée ici n'est valable que pour un langage disposant d'un support des exceptions (C++ ou Java par exemple). Nous supposons que le langage propose une fonction, nommée `throw` servant à déclencher une exception, ainsi qu'une fonction `catch` servant à capturer une exception. Nous supposons également que lorsqu'une exception est déclenchée par une méthode, celle-ci retourne l'exception.

Cette sémantique est définie par les trois règles suivantes :

$$\frac{env \vdash \text{throw}(e) : o, env'}{env \vdash \text{exception}(e) : o, env'}$$

$$\frac{env \vdash cr_1 : o, env'}{env \vdash \text{try}(cr_1, e, cr_2) : o, env' \mid e \neq o}$$

$$\frac{env \vdash cr_1 : e, env' \quad env' \vdash cr_2 : o, env''}{env \vdash \text{try}(cr_1, e, cr_2) : o, env''}$$

□

4 Définition des schémas d'interactions

Un schéma d'interactions est la description des comportements réactifs d'un ensemble d'interactions. Ces comportements réactifs sont décrits, au sein d'un schéma d'interactions, sous la forme de règles réactives. Celles-ci correspondent aux comportements réactifs pouvant être exécutés lors d'un envoi de messages à l'un des objets interagissants participants à l'interaction.

Afin d'apporter aux interactions le même niveau d'abstraction qu'aux objets, les schémas d'interactions sont formalisés sous la forme de classes (vérification du critère C1.1). Ainsi, nous avons $\mathcal{S} \subset \mathcal{C}$.

4.1 Une métaclasse pour les schémas d'interactions

Les schémas d'interactions disposent d'un ensemble d'informations qui leur sont propres : la liste des classes et des noms des participants, l'ensemble des règles réactives (et donc des comportements réactifs), et l'ensemble des schémas d'interactions hérités. Par conséquent, tout comme pour les règles réactives, le modèle à interactions distribuées définit une classe spécifique pour tous les schémas d'interactions (concept de métaclasse puisqu'un schéma d'interactions est une classe). Nous appelons $c_{\text{metascheme}}$ cette classe « racine » du graphe d'héritage des métaclasses des schémas d'interactions.

La classe $c_{\text{metascheme}}$ des schémas d'interactions définit les deux attributs suivants (il s'agit d'attributs de classe dont la valeur est présente dans l'état des schémas d'interactions) :

- Un attribut $P \in \mathcal{A}$ décrivant la liste ordonnée des classes et des noms des participants. Son domaine de valeurs est $L^{fin}(\mathcal{C} \times \mathcal{N})$.
- Un attribut $R \in \mathcal{A}$ décrivant l'ensemble des règles réactives. Son domaine de valeurs est $P^{fin}(\mathcal{R})$.

La valeur de l'attribut P décrivant l'ensemble des classes des objets interagissants participants pour le schéma d'interactions VisualiserNiveauSonore est la liste ((Controleur, c), (Glissiere, g), (Bouton, b)).

Son ensemble de règles réactives (valeur de l'attribut R) est l'ensemble composé des quatre règles réactives suivantes :

```
b.enDeplacement (int delta) -> b.enDeplacement (delta) // c.modifierVolume (delta),
c.modifierVolume (int delta) -> c.modifierVolume (delta) ; b.deplacer (c.obtenirVolume ()),
c.modifierVolume (int delta) -> c.modifierVolume (delta) ; g.afficherVolume (c.obtenirVolume ()),
this.init () -> delegate [ g.afficherVolume (c.obtenirVolume ()) // b.deplacer (c.obtenirVolume ()) ]
```

4.2 Structure d'un schéma d'interactions

La structure de classe des schémas d'interactions définit, entre autres attributs, les deux attributs suivants (attributs d'instance) :

- Un attribut $p \in \mathcal{A}$, dont le domaine des valeurs est $L^{fin}(\mathcal{O})$, décrivant la liste ordonnée des objets interagissants sur lesquels est instancié le schéma d'interactions.
- Un attribut $r \in \mathcal{A}$, dont le domaine des valeurs est $P^{fin}(\mathcal{O})$, décrivant l'ensemble des règles d'interaction⁴.

De plus, la structure des schémas d'interactions définit les quatre comportements suivants :

- Une méthode d'initialisation de l'interaction.
- Une méthode « d'activation » des règles d'interaction de l'interaction dans le système.
- Une méthode d'inhibition des règles d'interaction définies par l'interaction.
- Une méthode de terminaison de l'interaction.

Pour ce faire, le modèle à interactions distribuées définit une classe abstraite C_{scheme} , instance de la classe $C_{metascheme}$ qui sera héritée par tous les schémas d'interactions et qui définit ces deux attributs d'instance. Ainsi tout schéma d'interactions hérite directement ou indirectement de la classe C_{scheme} et est une instance de la classe $C_{metascheme}$ (ou de l'une de ses sous-classes).

Ceci est présenté, à l'aide de la notation UML [BJR97], par la figure 5.2.

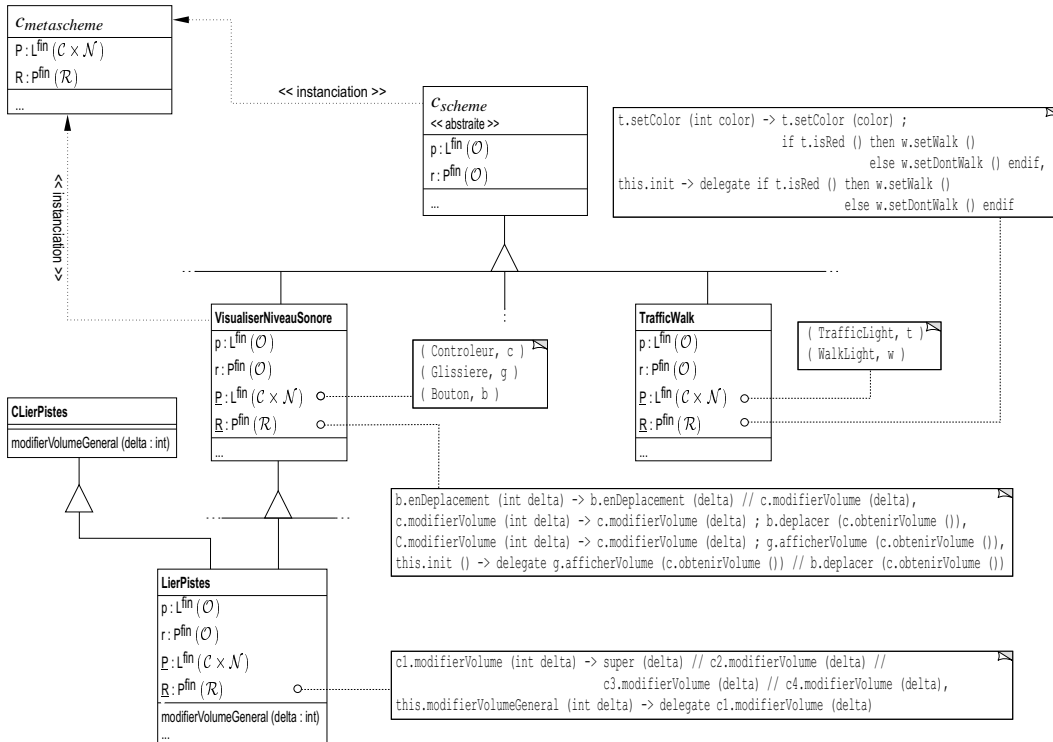


FIG. 5.2 – Description UML des schémas d'interactions

4. Nous rappelons qu'une règle d'interaction est une instance d'une règle réactive.

4.3 Spécialisation par raffinement

Bien que les schémas d'interactions soient représentés par des classes, l'héritage des règles réactives ne peut être réalisé par le mécanisme d'héritage proposé par le modèle à objets. En effet, les comportements réactifs décrits par les règles réactives sont des attributs définis dans la métaclasse et donc des attributs de classe. De ce fait, leur contenu ne peut donc pas être pris en charge par le concept d'héritage classique du modèle à objets.

Ainsi, la sémantique de cet héritage est définie par le modèle à interactions distribuées et a lieu lors de l'instanciation des schémas d'interactions, afin de valider la cohérence de l'interaction créée (cette sémantique est décrite dans le paragraphe 5.6). Dans ce paragraphe, nous nous restreignons à l'héritage des schémas d'interactions en tant que classes et montrons que la définition d'une classe dérivée d'un schéma d'interactions est un schéma d'interactions.

Définition 5.11 : Fonction d'héritage des schémas d'interactions

Le modèle à objets définit une fonction d'héritage, notée ϖ , associant à une classe (ou une métaclasse) $c \in \mathcal{C}$ l'ensemble des classes dont elle hérite [Jau00] :

$$\varpi : \begin{cases} \mathcal{C} & \rightarrow P^{fin}(\mathcal{C}) \\ c & \mapsto \{c' \in \mathcal{C} \mid c \in \beta(c')\} \end{cases} \quad (5.4)$$

Par analogie, le modèle à interactions distribuées définit une fonction d'héritage des schémas d'interactions, notée ϖ_{sch} , qui associe à un schéma d'interactions $s \in \mathcal{S}$ l'ensemble des schémas d'interactions dont il hérite⁵ :

$$\varpi_{sch} : \begin{cases} \mathcal{S} & \rightarrow P^{fin}(\mathcal{S}) \\ s & \mapsto \{s' \in \mathcal{S} \mid s \in \beta(s')\} = \varpi_{|_{\mathcal{S}}}(s) \end{cases} \quad (5.5)$$

Où β est la fonction de sous-classement, définie par le modèle à objets, associant à chaque classe l'ensemble de ses sous-classes et dont le domaine de définition est $\beta : \mathcal{C} \rightarrow P^{fin}(\mathcal{C})$. □

Soit les deux fonctions suivantes définies par le modèle à objets :

- La fonction d'instanciation, notée γ , qui associe à un objet $o \in \mathcal{O}$ sa classe. Son domaine de définition est $\gamma : \mathcal{O} \rightarrow \mathcal{C}$.
- La fonction de peuplement, notée π^* , qui associe à une classe $c \in \mathcal{C}$ l'ensemble des objets instances de cette classe ou de ses sous-classes. Son domaine de définition est $\pi^* : \mathcal{C} \rightarrow P^{fin}(\mathcal{O})$.

La métaclasse $c_{metascheme}$ étant la « racine » du graphe d'héritage des métaclasses des schémas d'interactions, un schéma d'interactions s vérifie les propriétés suivantes :

- $\forall s \in \mathcal{S}, s \in \pi^*(c_{metascheme})$
- $\forall s \in \mathcal{S}, c_{metascheme} \in \varpi(\gamma(s))$

Ces deux propriétés nous permettent de montrer que les sous-classes d'un schéma d'interactions sont des schémas d'interactions, c'est-à-dire que $\forall s \in \mathcal{S}, \beta(s) \in P^{fin}(\mathcal{S})$.

Propriété 5.1 : Une sous-classe d'un schéma d'interactions est un schéma d'interactions

Toute classe c , sous-classe d'un schéma d'interactions s , est un schéma d'interactions, c'est-à-dire de manière formelle, $\forall s \in \mathcal{S}, \beta(s) \in P^{fin}(\mathcal{S})$.

Preuve :

Soit $s \in \mathcal{S}$ et $\mathcal{E} = \beta(s)$. Soit $c \in \mathcal{E}$. Montrons que $c \in \mathcal{S}$:

- Si $\gamma(c) = c_{metascheme}$ alors $c \in \mathcal{S}$.
- Sinon montrons que $\gamma(c) \in \beta(c_{metascheme})$ et donc, d'après la définition de $c_{metascheme}$, que $c \in \mathcal{S}$.
Puisque $s \in \mathcal{S}$ nous avons $c_{metascheme} \in \varpi(\gamma(s))$.
De même puisque $c \in \beta(s)$ nous avons $s \in \varpi(c)$.
Nous pouvons en déduire (grâce à la propriété de transitivité de ϖ) que $c_{metascheme} \in \varpi(\gamma(c))$ et par conséquent que $c \in \mathcal{S}$.

Nous avons donc $\forall s \in \mathcal{S}, \beta(s) \in P^{fin}(\mathcal{S})$. □

5. Si E^f dénote l'ensemble des fonctions de E dans F alors $\forall f \in E^f$ et $E' \subseteq E$ alors $f|_{E'}$ dénote la restriction de la fonction f définie sur l'ensemble E' .

5 Instanciation des schémas d'interactions

Les schémas d'interactions sont des classes. On peut donc les instancier. Nous appelons ces instances les interactions. Ces dernières sont donc construites à partir des schémas d'interactions en étendant le mécanisme d'instanciation des classes du modèle à objets (puisque un schéma d'interactions est une classe). Nous allons donc décrire ce mécanisme et l'adapter aux schémas d'interactions.

5.1 Définition formelle de l'instanciation

Définition 5.12 : Fonction d'instanciation des schémas d'interactions

Le modèle à objets définit une fonction d'instanciation, notée γ , qui associe à un objet $o \in \mathcal{O}$ sa classe :

$$\gamma : \begin{cases} \mathcal{O} & \rightarrow \mathcal{C} \\ o & \mapsto \gamma(o) \end{cases} \quad (5.6)$$

Par analogie, le modèle à interactions distribuées définit une fonction d'instanciation, notée γ_{sch} , qui associe à une interaction $i \in \mathcal{I}$ son schéma d'interactions :

$$\gamma_{sch} : \begin{cases} \mathcal{I} & \rightarrow \mathcal{S} \\ i & \mapsto \gamma_{sch}(i) = \gamma_{\mathcal{I}}(i) \end{cases} \quad (5.7)$$

□

Définition 5.13 : Fonctions de peuplement des schémas d'interactions

Nous définissons une fonction de peuplement des schémas d'interactions, notée π_{sch} , associant à chaque schéma d'interactions l'ensemble de ses interactions (instances propres) :

$$\pi_{sch} : \begin{cases} \mathcal{S} & \rightarrow P^{fin}(\mathcal{I}) \\ s & \mapsto \pi_{sch}(s) = \{i' \in \mathcal{I} \mid s \in \gamma_{sch}(i')\} \end{cases} \quad (5.8)$$

De la fonction de peuplement π_{sch} l'on peut définir la fonction d'extension π_{sch}^* associant à chaque schéma d'interactions l'ensemble des interactions créées dans ce schéma d'interactions ou dans ses sous-schémas d'interactions :

$$\pi_{sch}^* : \begin{cases} \mathcal{S} & \rightarrow P^{fin}(\mathcal{I}) \\ s & \mapsto \bigcup_{s' \in \beta(s)} \pi_{sch}(s') \end{cases} \quad (5.9)$$

□

Définition 5.14 : Ensemble des interactions

Nous appelons fonction d'ensemble des interactions d'un objet la fonction, notée ι , qui associe à tout objet o l'ensemble des interactions auxquelles il participe :

$$\iota : \begin{cases} \mathcal{O} & \rightarrow P^{fin}(\mathcal{I}) \\ o & \mapsto \iota(o) \end{cases} \quad (5.10)$$

□

EXEMPLE. — En reprenant l'exemple de l'égaliseur graphique (paragraphe 4.2), lorsque les interactions $Visu2$, $Visu3$, $Visu4$ et $Master$ sont présentes dans le système alors l'appel de la fonction ι sur l'objet $c2$ renvoie l'ensemble $(Visu2, Master)$. L'appel de la fonction ι sur l'objet $b2$ renvoie l'ensemble $(Visu2)$.

Instanciation, héritage et typage

Lors de la définition des schémas d'interactions, nous avons indiqué qu'un schéma d'interactions pouvait être spécialisé par raffinement grâce à la notion d'héritage des comportements réactifs des schémas d'interactions. Nous avons également précisé que cet héritage avait lieu lors de l'instanciation d'un schéma d'interactions.

Cet héritage des comportements réactifs implique une vérification de la cohérence des liens d'héritage qui existent entre les schémas d'interactions afin de s'assurer que l'instanciation du schéma d'interactions construira une interaction ayant du sens (notamment au niveau du typage des participants).

Ainsi, le mécanisme d’instanciation doit s’assurer que les types des participants définis dans un schéma d’interactions s sont compatibles (au sens du modèle à objets) vis-à-vis des types des participants définis dans les super-schémas d’interactions de s . Cette vérification est dépendante du langage applicatif. Notre modèle à interactions distribuées délègue cette vérification à sa mise en œuvre

5.2 Structure des interactions

Les interactions sont des entités capables d’exprimer la communication entre un ensemble d’objets par des comportements réactifs. Ces comportements réactifs sont définis sous forme de règles d’interaction, instances des règles réactives définies dans le schéma d’interactions dont l’interaction est l’instance. Ainsi chaque interaction contient des objets qui représentent ses participants qui peuvent être contactés par envoi de messages à l’aide des règles d’interaction associées à l’interaction.

Toute interaction dispose des deux attributs suivants (définis dans le schéma d’interactions c_{scheme}) :

- Un attribut $p \in \mathcal{A}$, dont le domaine des valeurs est $L^{fin}(\mathcal{O})$, décrivant la liste ordonnée des objets interagissants sur lesquels est instancié le schéma d’interactions (ces objets sont appelés les *participants* de l’interaction). La cardinalité de cette liste est la même que celle de l’attribut de classe P décrivant la liste des classes des objets participants du schéma d’interactions. De plus, pour chaque objet participant, son type correspond à celui du même rang défini dans la liste de l’attribut P , ou à un type compatible (au sens du langage applicatif).
- Un attribut $r \in \mathcal{A}$, dont le domaine des valeurs est $P^{fin}(\mathcal{O})$, décrivant l’ensemble des règles d’interaction. Cet ensemble correspond à l’instanciation de la combinaison des règles réactives définies dans le schéma d’interactions de l’interaction et de celles définies dans les super-schémas d’interactions de l’interaction. La sémantique de la combinaison est décrite par l’héritage des règles réactives (détaillée dans le paragraphe suivant).

De plus, toute interaction dispose des quatre comportements suivants :

- Une méthode d’initialisation de l’interaction. Cette méthode est appelée juste après la construction de l’objet représentant l’interaction.
- Une méthode « d’activation » des règles d’interaction de l’interaction dans le système.
- Une méthode d’inhibition des règles d’interaction définies par l’interaction. Cette méthode supprime dans le système toutes les règles d’interaction définies par l’interaction sans pour autant détruire l’objet représentant l’interaction.
- Une méthode de terminaison de l’interaction. Cette méthode est appelée juste avant la destruction de l’objet représentant l’interaction.

EXEMPLE. — En reprenant l’exemple de l’égaliseur graphique, l’ensemble des participants pour l’interaction `Wisu2` (instance du schéma d’interactions `VisualiserNiveauSonore`) est la liste $(c2, g2, b2)$. L’ensemble des règles d’interaction de cette interaction est l’ensemble composé de l’instance de chacune des règles réactives définies dans le schéma d’interactions (car ce dernier n’hérite d’aucun schéma d’interactions).

Pour l’interaction `Master`, l’ensemble des participants est la liste $(c1, g1, b1, c2, c3, c4)$.

L’ensemble des règles d’interaction est le suivant (le paragraphe 5.6 explique comment obtenir cet ensemble) :

```
b.enDeplacement (int delta) -> b.enDeplacement (delta) // c.modifierVolume (delta),
c1.modifierVolume (int delta) -> [ c1.modifierVolume (delta) ; [ b.deplacer (c1.obtenirVolume ()) //
    g.afficherVolume (c1.obtenirVolume ()) ] ] // c2.modifierVolume (delta) //
    c3.modifierVolume (delta) // c4.modifierVolume (delta),
this.init () -> delegate [ g.afficherVolume (c1.obtenirVolume ()) // b.deplacer (c1.obtenirVolume ()) ]
```

La règle d’interaction dont le message déclencheur est `c1.modifierVolume` est celle définie dans le schéma d’interactions `LierPistes` à laquelle le mot-clef `super` a été remplacé par le comportement réactif résultant de la fusion comportementale (décrite en 6.4) des règles d’interaction définies dans le schéma d’interactions hérité et dont le message déclencheur est aussi `c1.modifierVolume` (au renommage près).

5.3 Structure des règles d’interaction

Une règle d’interaction est une instance d’une règle réactive (qui est une classe). Elle est contenue dans une interaction.

Toute règle d’interaction dispose du comportement et des trois attributs suivants :

- Un attribut $p \in \mathcal{A}$, dont le domaine de valeurs est \mathcal{O} , décrivant l’objet pour lequel l’invocation de la méthode définie par l’attribut de classe M va déclencher le comportement réactif.

- Un attribut $cr \in \mathcal{A}$, dont le domaine de valeurs est \mathcal{G} , correspondant à l’instanciation de l’attribut de classe CR (ce mécanisme d’instanciation est décrit ci-après).
- Un attribut $v \in \mathcal{A}$, dont le domaine de valeurs est $L^{fin}(O)$, correspondant aux valeurs des variables définies dans la règle d’interaction.
- Une méthode d’exécution du comportement réactif cr . Cette méthode est appelée lorsque le message déclencheur a été invoqué. Sa sémantique consiste à exécuter le comportement réactif cr .

6 Instanciation des règles réactives : héritage des comportements réactifs

Tout comme pour les comportements des objets, les comportements réactifs d’une interaction sont un concept complexe mettant en œuvre à la fois les paradigmes d’instanciation des règles réactives et d’héritage des schémas d’interactions.

Bien que les schémas d’interactions soient représentés par des classes, l’héritage des comportements réactifs ne peut être réalisé par le mécanisme classique d’héritage proposé par le modèle à objets. En effet, les comportements réactifs décrits par les règles réactives sont des attributs statiques dont la sémantique est définie par le modèle à interactions distribuées. De plus, ces règles réactives doivent être instanciées au sein de l’interaction. Ainsi, nous décrivons dans ce paragraphe ce mécanisme d’instanciation des règles réactives permettant l’héritage des comportements réactifs.

L’héritage des comportements réactifs contenus dans un schéma d’interactions s se fait en deux étapes :

- Une première étape qui consiste à déterminer l’ensemble des règles réactives qui doivent être instanciées.
- Une seconde étape qui consiste à instancier proprement dit (au sens instanciation du modèle à objets) chacune des règles réactives obtenues à l’étape précédente.

6.1 Détermination des règles réactives à instancier

Soit les quatre fonctions suivantes :

- μ_h^r la fonction retournant, pour un schéma d’interactions $s \in \mathcal{S}$, l’ensemble des règles réactives définies dans les schémas d’interactions hérités par s . Son domaine de définition est $\mu_h^r : \mathcal{S} \rightarrow P^{fin}(R)$.
- μ_i^r la fonction retournant, pour un schéma d’interactions $s \in \mathcal{S}$, l’ensemble des règles réactives définies dans le schéma d’interactions s . Son domaine de définition est $\mu_i^r : \mathcal{S} \rightarrow P^{fin}(R)$.
- ν la fonction retournant, pour un schéma d’interactions $s \in \mathcal{S}$, l’ensemble des règles réactives devant être instanciées. Son domaine de définition est $\nu : \mathcal{S} \rightarrow P^{fin}(R)$.
- η la fonction retournant, pour une règle réactive, son message déclencheur. Son domaine de définition est $\eta : \mathcal{R} \rightarrow \mathcal{B}$.

La détermination des règles réactives à instancier est définie par les trois règles suivantes (décrites sous la forme de trois propriétés).

Propriété 5.2 : Comportement réactif non hérité

S’il existe une règle d’interaction r définie dans un schéma d’interactions dont le message déclencheur est m et s’il n’existe pas de règle d’interaction définie dans un schéma d’interactions hérité par s dont le message déclencheur est aussi m , la règle d’interaction r fait partie de l’ensemble des règles d’interaction contenues dans les interactions instances du schéma d’interactions s .

Ceci se traduit formellement comme suit :

$$\text{Soit } s \in \mathcal{S}, a = \mu_h^r(s), b = \mu_i^r(s) \\ \forall r \in b, \nexists r' \in a \mid \eta(r') = \eta(r) \Rightarrow r \in \nu(s)$$

□

Propriété 5.3 : Héritage d’un comportement réactif non redéfini

S’il existe une règle d’interaction r définie dans un schéma d’interactions hérité par un schéma d’interactions s dont le message déclencheur est m et s’il n’existe pas de règle d’interaction définie dans le schéma d’interactions s dont le message déclencheur est aussi m , la règle d’interaction r fait partie de l’ensemble des règles d’interaction contenues par les interactions instances du schéma d’interactions s .

Ceci se traduit formellement comme suit :

Soit $s \in \mathcal{S}$, $a = \mu_h^r(s)$, $b = \mu_i^r(s)$
 $\forall r \in a, \nexists r' \in b \mid \eta(r') = \eta(r) \Rightarrow r \in \nu(s)$

□

Propriété 5.4 : Héritage d'un comportement réactif redéfini

S'il existe un ensemble de règles d'interaction e , dont le message déclencheur est m , définies dans les schémas d'interactions hérités par un schéma d'interactions s et s'il existe une règle d'interaction définie dans le schéma d'interactions s dont le message déclencheur est aussi m , une règle d'interaction r' , issue de r , fait partie de l'ensemble des règles d'interaction contenues par les interactions instances du schéma d'interactions s .

En fait cette règle d'interaction r' est identique à la règle d'interaction r à laquelle les comportements réactifs définis dans les règles d'interaction de l'ensemble e ont été inclus (par application de la fonction de fusion comportementale, décrite en 6.4) à la position, dans r , définie par le mot-clef `super` du langage ISL.

Notons $subst(a, b, c)$ la fonction substituant le comportement réactif c par le comportement réactif b dans le comportement réactif a et φ la fonction de fusion comportementale sur les ensembles (cette fonction est décrite au chapitre 6).

L'héritage de comportements réactifs redéfinis se traduit formellement comme suit :

Soit $s \in \mathcal{S}$, $a = \mu_h^r(s)$, $b = \mu_i^r(s)$, $m \in \mathcal{B}$
Soit $e \subseteq a \mid r_1 \in e \Rightarrow \eta(r_1) = m$,
 $\forall r \in b, \eta(r) = m \Rightarrow subst(r, \varphi(e), \text{super}) \in \nu(s)$

□

6.2 Algorithme d'héritage des règles réactives

Soient schéma un schéma d'interaction et sur-schémas l'ensemble des schémas d'interactions dont il hérite. Les règles réactives de schéma, après héritage des schémas d'interactions sur-schémas, sont spécifiées par la fonction HériteRéactif décrite ci-après.

```
HériteRéactif (schéma, sur-schémas)
H := ∅
HR := ∅
Pour tous S de sur-schémas
  HR := Renomme (S, Définition-du-renommage (schéma, S))
Pour tous R de schéma
  H := Ajoute (H, Fusion-ou-Ajout (R, HR))
H := Ajoute (HR, H)
```

Les règles réactives issues d'un schéma d'interactions hérité sont renommées (fonction `Renomme`) en fonction des noms des participants dans le schéma d'interactions et le super-schéma d'interactions (fonction `Définition-du-renommage`). Ensuite l'héritage des règles réactives définies dans les super-schémas d'interactions et redéfinies dans le schéma d'interactions est effectuée (fonction `Fusion-ou-Ajout`). Puis les règles réactives héritées non redéfinies dans le schéma d'interactions sont ajoutées à l'ensemble des règles réactives qui seront ensuite instanciées dans l'interaction, instance du schéma d'interactions `schema`.

```
Fusion-ou-Ajout (R, HR)
HF := ∅
Pour tous R' de HR
  si Déclenche (R') = Déclenche (R)
    alors HF := Ajoute (R, HF) et HR := Supprime (R', HR)
RF := Fusion (HF)
retourne Remplace (R, 'super', RF)
```

Lorsqu'une règle réactive redéfinit des règles réactives héritées alors ces règles réactives redéfinies sont fusionnées entre elles puis le résultat de cette fusion est insérée en lieu et place du mot-clef `super` dans la règle réactive du schéma d'interactions (fonction `Remplace`). Sinon la règle réactive est ajoutée aux règles réactives du schéma d'interactions.

EXEMPLE. — Dans l'exemple de l'égaliseur graphique, le schéma d'interactions `LierPistes` hérite du schéma d'interactions `VisualiserNiveauSonore`. Ainsi, l'ensemble des règles d'interaction de l'interaction `Master` (qui est une instance de `LierPistes`) est l'instanciation de la combinaison des règles réactives définies dans `VisualiserNiveauSonore` avec celles définies dans `LierPistes`.

Par exemple la règle réactive `b.enDeplacement (int delta) -> b.enDeplacement (delta) // c.modifierVolume (delta)` définie par le schéma d'interactions `VisualiserNiveauSonore` n'est pas redéfinie par le schéma d'interactions `LierPistes` et son instanciation fait donc partie de l'ensemble des règles d'interaction des instances de `LierPistes`.

Par contre le schéma d'interactions `VisualiserNiveauSonore` définit deux règles réactives dont le message déclencheur est `c.modifierVolume (int delta)` qui sont redéfinies dans le schéma d'interactions `LierPistes`.

Ainsi, le résultat de la fusion comportementale des deux règles réactives définies dans `VisualiserNiveauSonore` sera incorporé à la règle réactive redéfinie par le schéma d'interactions `LierPistes` en lieu et place du mot-clef `super` (propriété 5.4).

La règle réactive résultante est la suivante :

```
cl.modifierVolume (int delta) -> [ cl.modifierVolume (delta) ; [ b.deplacer (cl.obtenirVolume ()) //  
    g.afficherVolume (cl.obtenirVolume ()) ] ] // c2.modifierVolume (delta) //  
    c3.modifierVolume (delta) // c4.modifierVolume (delta),
```

7 Conclusion

Dans ce chapitre, nous avons présenté de manière formelle notre modèle à interactions distribuées qui est basé sur l'extension du mécanisme de l'envoi de messages, les concepts de schémas d'interactions, d'interactions et de comportements réactifs.

Nous avons notamment abordé les points suivants :

- L'abstraction offerte aux interactions grâce au concept de schémas d'interactions et sa place par rapport aux classes.
- Les avantages de la réification des interactions, des règles d'interactions et des comportements réactifs sous la forme de citoyens de première classe.
- La place des interactions par rapport aux objets interagissants.
- La définition et la déclaration des interactions et la manière dont les concepts du modèle à objets ont été utilisés et étendus aux schémas d'interactions et aux interactions.
- La sémantique de l'exécution des comportements réactifs et leur définition récursive sous la forme d'opérateurs réactifs.

Le chapitre suivant complète cette définition en décrivant de manière détaillée la sémantique de la fonction de fusion comportementale. Il présente les règles de réécriture (en sémantique naturelle) qui décrivent la sémantique de la fusion comportementale des règles d'interaction (comme nous l'avons vu, cette fusion comportementale est notamment appliquée lors de l'exécution d'un ensemble de règles d'interaction) puis démontre ses deux propriétés fondamentales, à savoir la commutativité et l'associativité.

- Schéma d'interactions décrivant un distributeur de boissons avec monnayeur et afficheur LCD ; ce schéma d'interactions décrit un distributeur de boissons disposant d'un afficheur LCD permettant d'afficher la somme introduite dans le monnayeur et un ensemble de messages prédéfinis (par exemple en cas de monnaie insuffisante).

```
interaction CompleteDrinkMachine (PushButton push, Container container, Money money, Display display)
  extend DrinkMachine (push, container, money)
{
  push.Push () -> if money.EnoughMoney () then push.Push () // money.Debit ()
                                     else delegate display.NotEnoughMoney () endif,
  money.Insert (int coin) -> money.Insert (coin) ; display.UpdateMoney (coin),
  money.Reset () -> money.Reset () ; display.ResetMoney ()
}
```

- Schéma d'interactions décrivant une connexion entre le distributeur de boissons et l'entreprise qui le gère ; ce schéma d'interactions permet à l'entreprise qui gère un distributeur de boissons d'être automatiquement notifiée lorsque le conteneur du distributeur a atteint un seuil critique (« presque vide » par exemple). De plus, il remplit le conteneur en fonction des stocks disponibles et prévient le service de comptabilité.

```
interaction FactoryManager (Container container, Factory factory, Comptable comptable)
{
  container.RemoveOne (), int nItems ->
    container.RemoveOne () ; if container.UnderLimit () then container.GetFreeCell (out nItems) ;
    [ container.Fill (nItems) // factory.Server (nItems) ] ; comptable.ItemsOut (nItems) endif
}
```

- Schéma d'interactions décrivant le schéma de conception Observateur permettant de faire une « trace » de l'utilisation d'un distributeur de boissons ; ce schéma d'interactions décrit le fait qu'à chaque fois qu'une action spécifique est réalisée sur le distributeur de boissons (ici le fait d'enlever une boisson du conteneur) un observateur est notifié.

```
interaction Observable (Container container, Observer observer)
{
  container.RemoveOne () -> container.RemoveOne () // observer.Update ()
  container.Fill (int nItems) -> container.Fill (nItems) // observer.Update ()
}
```

1.1 Fusion comportementale lors de l'héritage des comportements réactifs

Il a été vu dans le paragraphe 5.6 que l'héritage des comportements réactifs impliquait, dans certains cas, une combinaison des comportements réactifs qui, étant définis dans un schéma d'interactions *s*, sont redéfinis dans un schéma d'interactions héritant de *s*.

EXEMPLE. — Le schéma d'interactions *DrinkMachine* redéfinit le comportement réactif dont le message déclencheur est le message formel `push.Push ()`. Cette redéfinition fait appel au mot-clé `super` signifiant qu'il doit être remplacé, lors de l'héritage des règles d'interaction, par le comportement réactif issu de la fusion comportementale des règles d'interaction définies dans les schémas d'interactions hérités (ici, il n'y qu'une seule règle, définie dans *SimpleDrinkMachine*).

Ce mécanisme d'héritage des comportements réactifs est réalisé lors de l'instanciation des schémas d'interactions. Ainsi la fusion comportementale des règles d'interaction est appliquée, dans ce cas, sur des messages formels (c'est-à-dire que les paramètres ne sont pas instanciés).

De plus, le mécanisme de fusion comportementale doit prendre en compte une unification des paramètres et des noms des objets interagissants. En effet, comme cela est le cas entre les schémas d'interactions *SimpleDrinkMachine* et *DrinkMachine*, un schéma d'interactions héritant d'un autre schéma d'interactions peut nommer différemment les objets participants. Cette unification s'apparente à l'unification en logique [SS94].

EXEMPLE. — Le schéma d'interactions *DrinkMachine* utilise le nom `push` pour désigner l'objet interagissant bouton poussoir, alors que le schéma d'interactions *SimpleDrinkMachine*, dont hérite *DrinkMachine*, utilise le nom `button` pour désigner le même objet interagissant. Ainsi, l'une des deux règles d'interaction dont la partie sélecteur du message déclencheur est `push ()` et dont l'objet interagissant est le bouton poussoir doit être renommée pour unifier les noms utilisés entre les deux règles d'interaction. Au terme de cette unification, les deux règles d'interaction pourront être combinées.

1.2 Fusion comportementale lors de l'exécution des comportements réactifs

Un autre moment où la fusion comportementale est nécessaire est lors de l'exécution des comportements réactifs définis sur un message. En effet, supposons que sur un objet o soit définies trois règles d'interaction dont le message déclencheur est le même. Lorsque ce message va être invoqué sur cet objet les trois comportements réactifs associés aux trois règles d'interaction doivent être exécutés.

Cependant, si ces trois comportements réactifs impliquent l'exécution du message déclencheur, ce dernier ne doit être exécuté qu'une seule et unique fois, tout en s'assurant que les sémantiques d'exécution décrites dans chacune des règles d'interaction sont respectées. C'est justement le rôle de la fusion comportementale lorsqu'elle est appliquée à ce contexte d'exécution des comportements réactifs.

De plus, en raison de la définition dynamique des interactions, les comportements réactifs associés à un message déclencheur peuvent ne pas être les mêmes lors de deux appels au message déclencheur. Ainsi, la fusion comportementale doit être appliquée, au plus tard, à chaque fois qu'un message déclencheur est invoqué.

REMARQUE. — En fait, l'ensemble des règles d'interaction associées à une méthode pour un objet donné ne peut varier que si l'ensemble des interactions « posées » sur cet objet varie. Il est donc possible de n'appliquer la fusion comportementale entre un ensemble de règles d'interaction définies sur un message donné que si l'ensemble des interactions associées à l'objet qui contient le message a changé.

EXEMPLE. — Supposons qu'une instance i_1 du schéma d'interactions `FactoryManager` soit présente dans le système entre des objets c (de type `Container`), f (de type `Factory`) et p (de type `Comptable`) et qu'une instance du schéma d'interactions `Observable` soit présente entre l'objet c et un objet o (de type `Observer`). Nous avons donc les règles d'interaction suivantes :

```
c.RemoveOne (), int nItems ->
  container.RemoveOne (); if c.UnderLimit () then nItems := c.GetFreeCell ();
  [ c.Fill (nItems) // f.Server (nItems) ]; p.ItemsOut (nItems) endif
c.RemoveOne () -> c.RemoveOne () // o.Update ()
```

La première règle d'interaction spécifie que le message déclencheur doit être exécuté puis, une fois l'exécution de ce message terminée (opérateur `sequential`), qu'une réaction conditionnelle doit être exécutée. La seconde règle d'interaction spécifie que le message déclencheur et le message `Update` sur l'objet observateur (`observer`) peuvent être exécutés concurremment.

La fusion comportementale de ces deux règles d'interaction doit conserver ces sémantiques. Ainsi, dans la règle d'interaction obtenue par fusion, l'exécution du comportement réactif conditionnel devra toujours se faire après celle du message déclencheur, l'exécution du message `observer.Update ()` devant toujours être réalisée de manière concurrente à celle du message déclencheur.

Il est à noter qu'aucune contrainte d'exécution n'existe entre les deux règles d'interaction hormis celle définie par le message déclencheur. Par conséquent aucune contrainte d'exécution ne doit être introduite entre le comportement conditionnel et le message `observer.Update ()`. Ainsi ces deux comportements réactifs doivent s'exécuter concurremment.

On peut ainsi déduire que le résultat de la fusion comportementale appliquée à ces deux règles d'interaction est le suivant :

```
c.RemoveOne (), int nItems ->
  [ c.RemoveOne (); if c.UnderLimit () then nItems := c.GetFreeCell ();
  [ c.Fill (nItems) // f.Server (nItems) ]; p.ItemsOut (nItems) endif ] // o.Update ()
```

2 Règles d'équivalence

Nous définissons ci-dessous un ensemble de règles d'équivalence des comportements réactifs. Ces règles d'équivalence pourront être appliquées afin de transformer un comportement réactif en un autre comportement réactif sémantiquement équivalent.

REMARQUE. — Ceci signifie que l'exécution du comportement réactif produit par l'une des règles d'équivalence sera sémantiquement identique à celle du comportement réactif initial. Ainsi, des messages exécutés séquentiellement dans la règle d'interaction initiale le sont toujours une fois la transformation appliquée et l'ordre d'application est inchangé. Il en est de même pour les messages exécutés concurrentiellement¹.

Il est à noter que ceci n'implique nullement que les deux comportements réactifs fourniront le même résultat. En effet, l'opérateur `concurrency` introduit du non-déterminisme dans le mécanisme d'exécution des comportements réactifs. Ainsi, si deux méthodes sont exécutées concurrentiellement et qu'elles partagent en écriture une variable, la valeur de cette variable sera indéterminée.

1. Cependant, dans ce cas, l'ordre d'application peut être modifié. Ceci est, sémantiquement parlant, sans incidence.

NOTE. — Certaines de ces règles d'équivalence sont implicitement utilisées par la fonction de fusion comportementale (décrite en 6.4) pour rendre syntaxiquement correct le comportement réactif généré par son application ou pour permettre l'application de l'une des règles de fusion définies.

Transformation T1 :

Transformation d'un comportement réactif séquentiel en un comportement réactif concurrentiel

Cette transformation est appelée par l'une des règles de fusion comportementale (décrite en 6.4). Elle transforme une règle d'interaction décrivant un comportement réactif séquentiel par une règle d'interaction décrivant, grâce à un comportement réactif concurrentiel et un comportement réactif d'attente, la même sémantique comportementale.

- $\forall R \in \mathcal{G}_S : m \rightarrow \text{sequential}(cr_1, cr_2), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{concurrency}(\text{assignment}(cr_1, x), \text{waiting}(cr_2, x))$

Cette transformation génère un comportement réactif syntaxiquement incorrect. L'application des deux transformations suivantes (transformation T2 et T3) permet de le rendre syntaxiquement correct.

Transformation T2 :

Combinaison d'un comportement réactif d'attente avec un autre comportement réactif

Cette transformation est implicitement appelée par la transformation T1 afin de rendre syntaxiquement correct le comportement réactif qu'elle génère.

- $\forall R \in \mathcal{G} : m \rightarrow \text{waiting}(\text{delegate}(cr), x), cr \in \mathcal{G}.$
 $R \equiv m \rightarrow \text{delegate}(\text{waiting}(cr, x))$
- $\forall R \in \mathcal{G} : m \rightarrow \text{waiting}(\text{sequential}(cr_1, cr_2), x), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{sequential}(\text{waiting}(cr_1, x), cr_2)$
- $\forall R \in \mathcal{G} : m \rightarrow \text{waiting}(\text{concurrency}(cr_1, cr_2), x), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{concurrency}(\text{waiting}(cr_1, x), \text{waiting}(cr_2, x))$
- $\forall R \in \mathcal{G} : m \rightarrow \text{waiting}(\text{ifThen}(c, cr), x), c \in \mathcal{G}_M \cup \mathcal{G}_W, cr \in \mathcal{G}.$
 $R \equiv m \rightarrow \text{ifThen}(\text{waiting}(c, x), cr)$
- $\forall R \in \mathcal{G} : m \rightarrow \text{waiting}(\text{ifThenElse}(c, cr_1, cr_2), x), c \in \mathcal{G}_M \cup \mathcal{G}_W, (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{ifThenElse}(\text{waiting}(c, x), cr_1, cr_2)$
- $\forall R \in \mathcal{G} : m \rightarrow \text{waiting}(\text{try}(cr_1, e, cr_2), x), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{try}(\text{waiting}(cr_1, x), e, cr_2)$

Transformation T3 :

Combinaison d'un comportement réactif d'affectation avec un autre comportement réactif

Cette transformation est implicitement appelée par la transformation T1 afin de rendre syntaxiquement correct le comportement réactif qu'elle génère.

- $\forall R \in \mathcal{G} : m \rightarrow \text{assignment}(\text{delegate}(cr), x), cr \in \mathcal{G}.$
 $R \equiv m \rightarrow \text{delegate}(\text{assignment}(cr, x))$
- $\forall R \in \mathcal{G} : m \rightarrow \text{assignment}(\text{sequential}(cr_1, cr_2), x), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{sequential}(cr_1, \text{assignment}(cr_2, x))$
- $\forall R \in \mathcal{G} : m \rightarrow \text{assignment}(\text{concurrency}(cr_1, cr_2), x), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{concurrency}(\text{assignment}(cr_1, x'), \text{assignment}(cr_2, x'')) \mid x = x' \cup x''$
- $\forall R \in \mathcal{G} : m \rightarrow \text{assignment}(\text{ifThen}(c, cr), x), c \in \mathcal{G}_M \cup \mathcal{G}_W, cr \in \mathcal{G}.$
 $R \equiv m \rightarrow \text{ifThen}(\text{assignment}(c, x), cr)$
- $\forall R \in \mathcal{G} : m \rightarrow \text{assignment}(\text{ifThenElse}(c, cr_1, cr_2), x), c \in \mathcal{G}_M \cup \mathcal{G}_W, (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{ifThenElse}(c, \text{assignment}(cr_1, x), \text{assignment}(cr_2, x))$
- $\forall R \in \mathcal{G} : m \rightarrow \text{assignment}(\text{try}(cr_1, e, cr_2), x), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{try}(\text{assignment}(cr_1, x), e, \text{assignment}(cr_2, x))$

Transformation T4 :

Commutativité et associativité des comportements réactifs

- Commutativité de l'opérateur concurrency :
 $\forall R \in \mathcal{G}_C : m \rightarrow \text{concurrency}(cr_1, cr_2), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{concurrency}(cr_2, cr_1)$
- Associativité de l'opérateur assignment :
 $\forall R \in \mathcal{G}_A : m \rightarrow \text{assignment}(x_1, \text{assignment}(x_2, cr)), cr \in \mathcal{G}_A \cup \mathcal{G}_M.$
 $R \equiv m \rightarrow \text{assignment}(x_2, \text{assignment}(x_1, cr))$
- Associativité de l'opérateur waiting :
 $\forall R \in \mathcal{G}_W : m \rightarrow \text{waiting}(\text{waiting}(cr, mb_1), mb_2), cr \in \mathcal{G}_W \cup \mathcal{G}_A \cup \mathcal{G}_M.$
 $R \equiv m \rightarrow \text{waiting}(\text{waiting}(cr, mb_2), mb_1)$
- Associativité de l'opérateur concurrency :
 $\forall R \in \mathcal{G}_C : m \rightarrow \text{concurrency}(\text{concurrency}(cr_1, cr_2), cr_3), (cr_1, cr_2, cr_3) \in \mathcal{G}^3.$
 $R \equiv m \rightarrow \text{concurrency}(cr_1, \text{concurrency}(cr_2, cr_3))$
- Associativité de l'opérateur sequential :
 $\forall R \in \mathcal{G}_S : m \rightarrow \text{sequential}(\text{sequential}(cr_1, cr_2), cr_3), (cr_1, cr_2, cr_3) \in \mathcal{G}^3.$
 $R \equiv m \rightarrow \text{sequential}(cr_1, \text{sequential}(cr_2, cr_3))$
- Associativité de l'opérateur try :
 $\forall R \in \mathcal{G}_S : m \rightarrow \text{try}(\text{try}(cr_1, e_1, cr_2), e_2, cr_3), (cr_1, cr_2, cr_3) \in \mathcal{G}^3.$
 $R \equiv m \rightarrow \text{try}(\text{try}(cr_1, e_2, cr_3), e_1, cr_2)$

Transformation T5 :

Simplification d'une règle d'interaction contenant un comportement réactif d'exception

L'application de cette transformation permet de simplifier une règle d'interaction contenant un comportement réactif d'exception car ce dernier est « absorbant » vis-à-vis des autres comportements réactifs.

- $\forall R \in \mathcal{G}_S : m \rightarrow \text{sequential}(\text{exception}(e), cr_2), cr_2 \in \mathcal{G}.$
 $R \equiv m \rightarrow \text{exception}(e)$
- $\forall R \in \mathcal{G}_D : m \rightarrow \text{delegate}(\text{exception}(e)).$
 $R \equiv m \rightarrow \text{exception}(e)$
- $\forall R \in \mathcal{G}_T : m \rightarrow \text{try}(\text{exception}(e_1), e_2, cr_1), cr_1 \in \mathcal{G}, e_1 \neq e_2.$
 $R \equiv m \rightarrow \text{exception}(e_1)$
- $\forall R \in \mathcal{G}_T : m \rightarrow \text{try}(\text{exception}(e), e, cr_1), cr_1 \in \mathcal{G}.$
 $R \equiv m \rightarrow cr_1$

Transformation T6 :

Distributivité d'un comportement réactif de traitement des exceptions par rapport aux autres comportements réactifs

L'application de cette transformation « distribue » un comportement réactif de traitement des exceptions par rapport à un comportement réactif conditionnel. Nous supposons que l'exécution de la condition (une méthode) ne provoque pas l'exception capturée par l'opérateur try.

- Distributivité de l'opérateur try par rapport à l'opérateur ifThen :
 $\forall R \in \mathcal{G}_T : m \rightarrow \text{try}(\text{ifThen}(c, cr_1), e, cr_2), (cr_1, cr_2) \in \mathcal{G}^2.$
 $R \equiv m \rightarrow \text{ifThen}(c, \text{try}(cr_1, e, cr_2))$
- Distributivité de l'opérateur try par rapport à l'opérateur ifThenElse :
 $\forall R \in \mathcal{G}_T : m \rightarrow \text{try}(\text{ifThenElse}(c, cr_1, cr_2), e, cr_3), (cr_1, cr_2, cr_3) \in \mathcal{G}^3.$
 $R \equiv m \rightarrow \text{ifThenElse}(c, \text{try}(cr_1, e, cr_3), \text{try}(cr_2, e, cr_3))$

3 Substitutions et unifications

Nous appelons **substitution** la fonction substituant, pour un ensemble de couple de noms de variable (o, n) , le nom de la variable o par le nom de la variable n . Ainsi, appliquer une substitution S , formée de couples (old_name, new_name) , à un comportement réactif revient à substituer les noms des variables old_name du comportement réactif par leurs valeurs associées dans S , à savoir new_name .

Il est à noter que les variables des comportements réactifs sont des variables libres (c'est-à-dire qu'il n'y a pas de lien direct entre elles). Par conséquent, l'application d'une substitution S doit conserver cette propriété.

Nous dirons que deux messages m_1 et m_2 peuvent être **unifiés** si, et seulement si, ils ont la même signature et que leurs arguments peuvent être unifiés deux à deux, c'est-à-dire s'il existe une substitution S telle que $S(m_1) = S(m_2)$ ne produise pas de conflit de conversion des variables.

De même, nous dirons que deux comportements réactifs cr_1 et cr_2 peuvent être unifiés si, et seulement si, il existe une substitution S telle que $S(cr_1) = S(cr_2)$ ne produisant pas de conflit de conversion des variables.

EXEMPLE. — Soit la règle d'interaction suivante :

```
windows.move (int dx, int x) -> menu.move (dx, x) ; scrollbars.move (dx, x)
```

L'application de la substitution $S = \{(dx, x)\}$ à cette règle d'interaction renvoie la règle d'interaction suivante :

```
windows.move (int x, int x) -> menu.move (x, x) ; scrollbars.move (x, x)
```

Nous pouvons constater que, avec cette substitution, les deux variables dx et x provoquent un conflit de conversion (elles portent désormais le même nom). Cette substitution ne peut donc pas être appliquée à un comportement réactif ou à un message.

Définition 6.1 : Fonction d'unification

Nous appellerons fonction d'unification de deux messages, notée v_{msg} la fonction suivante (\mathcal{U} est l'ensemble infini dénombrable des substitutions) qui associe à deux messages m_1 et m_2 un message unifié m ainsi qu'une substitution minimale s permettant de passer des messages m_1 et m_2 à m :

$$v_{msg} : \left\{ \begin{array}{l} \mathcal{B}^2 \rightarrow \mathcal{B} \times \mathcal{U} \\ (m_1, m_2) \mapsto \begin{array}{l} v_{msg}(m_1, m_2) = (m, s) \\ m = s(m_1) \wedge m = s(m_2) \end{array} \end{array} \right. \quad (6.1)$$

□

Propriété 6.1 : Commutativité de la fonction d'unification

La fonction d'unification v_{msg} est commutative.

Preuve :

Soit $(m, m') \in \mathcal{B}^2$ et $(s, s') \in \mathcal{U}^2$ tels que $v_{msg}(m_1, m_2) = (m, s)$ et $v_{msg}(m_2, m_1) = (m', s')$. Nous devons montrer que $m = m'$ et $s = s'$.

Nous avons $m' = s'(m_1) = s'(s^{-1}(m))$ et $m = s(m_1) = s(s'^{-1}(m'))$ donc $\exists s'' \in \mathcal{U} \mid s''(m) = s''(m')$ et, par conséquent, $\exists (m'', s'') \in \mathcal{B} \times \mathcal{U} \mid v_{msg}(m, m') = (m'', s'')$.

Nous avons par définition de la fonction $v_{msg} : m'' = s''(m) = s''(s(m_1))$ et $m'' = s''(m') = s''(s'(m_1))$. Donc $s'' \circ s = s'' \circ s'$ d'où $\underbrace{s''^{-1} \circ s''}_{=Id} \circ s = \underbrace{s''^{-1} \circ s''}_{=Id} \circ s'$, d'où, finalement, $s = s'$, et par conséquent $m = m'$.

□

4 Fusion comportementale des règles d'interaction

Cette section décrit la fusion comportementale d'un ensemble de règles d'interaction. Cette fusion génère une unique règle d'interaction dont le comportement réactif est soit inclus en lieu et place du mot-clef `super` lors de l'héritage des règles d'interaction, soit exécutée en lieu et place de l'ensemble des règles d'interaction définies et qui a la même sémantique que cet ensemble de règles d'interaction.

Nous présentons tout d'abord la fonction de fusion comportementale entre deux règles d'interaction. Nous étendons cette définition à une fonction de fusion comportementale n-aire. Nous décrivons ensuite les règles de réécriture, à l'aide de la sémantique naturelle `TYPOL` [Des88], définissant la sémantique de la fonction de fusion comportementale.

Définition 6.2 : Fusion comportementale

Nous définissons la fonction de fusion comportementale, nommée φ , comme suit :

$$\varphi : \left\{ \begin{array}{l} \mathcal{R}^2 \rightarrow \mathcal{R} \\ (r_1, r_2) \mapsto r \end{array} \right. \quad (6.2)$$

□

Définition 6.3 : Fusion comportementale n-aire

De la fonction de fusion comportementale φ et de ses propriétés de commutativité et d'associativité (décrites plus en avant dans ce chapitre) nous pouvons en déduire la fonction de fusion comportementale n-aire, notée φ^n , définie récursivement comme suit :

$$\varphi^2 : \left\{ \begin{array}{l} \mathcal{R}^2 \rightarrow \mathcal{R} \\ (r_1, r_2) \mapsto \varphi(r_1, r_2) \end{array} \right. \quad (6.3)$$

$$\forall n > 2 \quad \varphi^n : \left\{ \begin{array}{l} \mathcal{R}^n \rightarrow \mathcal{R} \\ (r_1, r_2, \dots, r_n) \mapsto \varphi(r_1, \varphi^{n-1}(r_2, \dots, r_n)) \end{array} \right. \quad (6.4)$$

□

Les règles de réécriture définissant la sémantique de la fusion comportementale sont décrites ci-dessous. Elles sont présentées par ordre décroissant de priorité. Nous les avons classées en deux catégories : les règles de réécriture terminales (elles renvoient le résultat de la fusion comportementale sans appel récursif et sont donc des axiomes), et les règles de réécriture non terminales (elles sont basées sur un appel récursif). Pour toutes ces règles (hormis la règle R1) nous notons s la substitution permettant d'unifier les comportements réactifs des deux règles d'interaction à fusionner et m le message unifié. De plus, sauf mention contraire, chacun des comportements réactifs à fusionner contient une unique invocation au message déclencheur².

4.1 Règles de réécriture terminales : axiomes

Règle de fusion R1 : Échec de la fusion des sous-comportements réactifs

Lors de la fusion comportementale de deux comportements réactifs, si la fusion comportementale de leurs sous-comportements réactifs échoue alors la fusion comportementale de ces deux comportements réactifs échoue également. Il y a échec lorsqu'aucune règle de fusion comportementale ne peut être appliquée, ou s'il n'existe aucune substitution permettant d'unifier les deux comportements réactifs à fusionner.

□

Règle de fusion R2 : Fusion des exceptions

SÉMANTIQUE INFORMELLE. — Un comportement réactif d'exception est absorbant vis-à-vis des autres comportements réactifs.

SOLUTION INTUITIVE. — Soit la règle d'interaction suivante décrivant la sémantique d'enregistrement d'un document (objet `doc`) dans une application (objet `appli`) :

```
R1: doc.save () -> if (appli.alreadySaved (doc)) then doc.save () else delegate doc.saveAs () endif
```

L'éditeur de cette application souhaite réaliser une version d'évaluation offrant toutes les fonctionnalités de son application mais ne permettant pas la sauvegarde des documents édité. Il ajoute donc la règle d'interaction suivante à la version d'évaluation :

```
R2: doc.save () -> exception Evaluation
```

Ainsi, pour la version d'évaluation, les deux règles d'interaction ci-dessus sont définies. Par conséquent, la règle d'interaction résultante de la fusion comportementale de ces deux règles doit être :

```
doc.save () -> exception Evaluation
```

Ainsi, la sémantique de la fusion comportementale dont au moins l'un des comportements réactifs est un comportement réactif d'exception est décrite par les deux règles de sémantique naturelle suivantes :

$$(m_1 \rightarrow \text{exception}(e_1), m_2 \rightarrow \text{exception}(e_2)) : m \rightarrow \text{concurrency}(\text{exception}(e_1), \text{exception}(e_2))$$
$$(m_1 \rightarrow cr_1, m_2 \rightarrow \text{exception}(e)) : m \rightarrow \text{exception}(e)$$

□

Règle de fusion R3 : Fusion des délégations

SÉMANTIQUE INFORMELLE. — Un comportement réactif de délégation se substitue au message déclencheur et est considéré comme étant ce dernier.

SOLUTION INTUITIVE. — Cet exemple décrit la fusion comportementale de deux règles d'interaction dont l'une est une délégation. La règle d'interaction désignant la délégation est issue d'un schéma d'interactions décrivant le schéma de conception `Adaptateur`. Elle spécifie que lorsque le message déclencheur est invoqué sur l'objet `adaptateur` (nommé `adapter`) alors c'est une méthode de l'objet `adapté` (nommé `adaptee`) qui est réellement exécutée.

² Le comportement réactif d'exception ne contient pas d'invocation au message déclencheur. Le comportement réactif conditionnel contient 2 invocations : une pour la branche `then` et une autre pour la branche `else`. L'unique cas où ceci n'est pas vérifié se produit lorsqu'une des branches (ou les deux) contient un comportement réactif de délégation ou d'exception.

Nous avons donc les deux règles d'interaction suivantes :

```
R1: adapter.Request (int param) -> delegate adaptee.TrueRequest (adapter.Convert (param))
R2: adapter.Request (int param) -> adapter.Request (param) // observer.notify () ; adapter.OtherRequest ()
```

Intuitivement le résultat de la fusion comportementale de ces deux règles d'interaction est la règle d'interaction suivante :

```
adapter.Request (int param) -> delegate adaptee.TrueRequest (adapter.Convert (param)) // observer.notify () ;
adapter.OtherRequest ()
```

Ainsi, l'invocation du message déclencheur dans la règle d'interaction R2 est remplacée par le comportement réactif de délégation de la règle d'interaction R1.

Ainsi, la sémantique de la fusion comportementale d'un comportement réactif de délégation avec un autre comportement réactif est décrite par les deux règles de sémantique naturelle suivantes :

$$(m_1 \rightarrow \text{delegate}(cr_1), m_2 \rightarrow cr_2) : m \rightarrow \text{subst}(m, s(\text{delegate}(cr_1)), s(cr_2))$$

$$(m_1 \rightarrow cr_1, m_2 \rightarrow \text{delegate}(cr_2)) : m \rightarrow \text{subst}(m, s(\text{delegate}(cr_2)), s(cr_1))$$

où $\text{subst}(m, cr_1, cr_2)$ est une fonction de substitution dont le domaine de définition est $\text{subst} : \mathcal{B} \times \mathcal{R}^2 \rightarrow \mathcal{R}$. Elle substitue dans le comportement réactif cr_2 le message m par le comportement réactif cr_1 .

NOTE. — Cette règle de fusion comportementale est la seule décrivant la fusion comportementale d'un comportement réactif de délégation. Ainsi la fusion comportementale de deux comportements réactifs différents exprimant une délégation sur le même message déclencheur échoue car aucune substitution ne peut être trouvée et aucune règle de fusion comportementale (hormis la règle décrivant un échec de la fusion) ne peut être appliquée. □

Règle de fusion R4 : Fusion des envois de messages

SÉMANTIQUE INFORMELLE. — Un comportement réactif désignant le message déclencheur est neutre vis-à-vis de la fonction de fusion comportementale.

SOLUTION INTUITIVE. — Soit les deux règles d'interaction suivantes (issues d'un appel récursif à la fonction de fusion comportementale) :

```
R1: square.move (int x, int y) -> square.move (x, y)
R2: square.move (int dx, int dy) -> square.move (dx, dy)
```

Intuitivement, le résultat de la fusion comportementale appliquée à ces deux règles d'interaction doit être le suivant :

```
square.move (int x, int y) -> square.move (x, y)
```

Ainsi, la sémantique de la fusion comportementale d'un comportement réactif d'envoi de messages avec un autre comportement réactif est décrite par la règle de sémantique naturelle suivante :

$$(m_1 \rightarrow \text{msg}(o, m), m_2 \rightarrow cr) : m \rightarrow s(cr)$$

□

4.2 Règles de réécriture non terminales

Ces règles de réécriture de la fusion comportementale réappliquent la fonction de fusion comportementale sur une sous partie des comportements réactifs (principe de récursivité).

Règle de fusion R5 : Fusion d'un comportement réactif d'affectation

SÉMANTIQUE INFORMELLE. — Les comportements réactifs d'affectation s'imbriquent les uns dans les autres et, vis-à-vis des autres comportements réactifs, se comportent comme le comportement réactif d'envoi de messages.

SOLUTION INTUITIVE. — Soit les deux règles d'interaction suivantes (issues d'un appel récursif à la fonction de fusion) :

```
R1: canvas.scale (), int s -> s := canvas.scale ()
R2: canvas.scale (), int f -> f := canvas.scale ()
```

Comme on le ferait avec un langage comme C++, le résultat intuitif de l'application de la fonction de fusion comportementale sur ces deux règles d'interaction est la suivante :

```
R1.2: canvas.scale (), int s, int f -> f := s := canvas.scale ()
```

Supposons maintenant que cette dernière règle d'interaction doive être fusionnée avec la règle d'interaction suivante :

```
R3: canvas.scale () -> try canvas.scale () ; rule.scale () catch (invalidScale) Reaction
```

Intuitivement, l'application de la fonction de fusion comportementale sur ces deux règles d'interaction doit remplacer le message déclencheur dans la règle d'interaction R3 par les affectations de la règle d'interaction R1.2 :

```
canvas.scale (), int s, int f -> try f := s := canvas.scale () ; rule.scale () catch (invalidScale) Reaction
```

Ainsi, un comportement réactif d'affectation se comporte comme un comportement réactif d'envoi de messages.

Ainsi, la sémantique de la fusion comportementale d'un comportement réactif d'affectation avec un autre comportement réactif d'affectation est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_2) : m \rightarrow cr_{1,2}}{(m_1 \rightarrow \text{assignment}(x_1, cr_1), m_2 \rightarrow \text{assignment}(x_2, cr_2)) : m \rightarrow \text{assignment}(x_1, \text{assignment}(x_2, cr_{1,2}))}$$

La sémantique de la fusion comportementale d'un comportement réactif d'affectation avec un comportement réactif séquentiel est décrite par les deux règles de sémantique naturelle suivantes :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow cr_{1,3}}{(m_1 \rightarrow \text{sequential}(cr_1, cr_2), m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow \text{sequential}(cr_{1,3}, s(cr_2)) \mid m \in cr_1}$$

$$\frac{(m_1 \rightarrow cr_2, m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow cr_{2,3}}{(m_1 \rightarrow \text{sequential}(cr_1, cr_2), m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow \text{sequential}(s(cr_1), cr_{2,3}) \mid m \in cr_2}$$

La sémantique de la fusion comportementale d'un comportement réactif d'affectation avec un comportement réactif de traitement des exceptions est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow cr_{1,3}}{(m_1 \rightarrow \text{try}(cr_1, e, cr_2), m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow \text{try}(cr_{1,3}, e, s(cr_2))}$$

□

Règle de fusion R6 : Fusion d'un comportement réactif d'attente

SÉMANTIQUE INFORMELLE. — Les comportements réactifs d'attente s'imbriquent les uns dans les autres et encapsulent les comportements réactifs d'affectation. De plus, tout comme les comportements réactifs d'affectation, ils se comportent comme des comportements réactifs d'envoi de messages vis-à-vis des autres comportements réactifs.

SOLUTION INTUITIVE. — Soit les deux règles d'interaction suivantes (issues d'un appel récursif à la fonction de fusion comportementale) :

```
R1: canevas.scale (), mailbox mb1 -> wait canevas.scale () for mb1
R2: canevas.scale (), mailbox mb2 -> wait canevas.scale () for mb2
```

L'application de la fonction de fusion comportementale doit s'assurer que l'exécution du message `canevas.scale ()` ne débute que lorsque les deux attentes sont terminées :

```
canevas.scale (), mailbox mb1, mailbox mb2 -> wait [ wait canevas.scale () for mb2 ] for mb1
```

Ainsi, la sémantique de la fusion comportementale d'un comportement réactif d'attente avec un autre comportement réactif d'attente est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_2) : m \rightarrow cr_{1,2}}{(m_1 \rightarrow \text{waiting}(cr_1, mb_1), m_2 \rightarrow \text{waiting}(cr_2, mb_2)) : m \rightarrow \text{waiting}(\text{waiting}(cr_{1,2}, mb_2), mb_1)}$$

La sémantique de la fusion comportementale d'un comportement réactif d'attente avec un comportement réactif d'affectation est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment}(x, cr_2)) : m \rightarrow cr_{1,2}}{(m_1 \rightarrow \text{waiting}(cr_1, mb), m_2 \rightarrow \text{assignment}(x, cr_2)) : m \rightarrow \text{waiting}(cr_{1,2}, mb)}$$

La sémantique de la fusion comportementale d'un comportement réactif d'attente avec un comportement réactif séquentiel est décrite par les deux règles de sémantique naturelle suivantes :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{waiting}(mb, cr_3)) : m \rightarrow cr_{1,3}}{(m_1 \rightarrow \text{sequential}(cr_1, cr_2), m_2 \rightarrow \text{waiting}(mb, cr_3)) : m \rightarrow \text{sequential}(cr_{1,3}, s(cr_2)) \mid m \in cr_1}$$

$$\frac{(m_1 \rightarrow cr_2, m_2 \rightarrow \text{waiting}(mb, cr_3)) : m \rightarrow cr_{2,3}}{(m_1 \rightarrow \text{sequential}(cr_1, cr_2), m_2 \rightarrow \text{waiting}(mb, cr_3)) : m \rightarrow \text{sequential}(s(cr_1), cr_{2,3}) \mid m \in cr_2}$$

La sémantique de la fusion comportementale d'un comportement réactif d'attente avec un comportement réactif de traitement des exceptions est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{waiting}(cr_3, mb)) : m \rightarrow cr_{1,3}}{(m_1 \rightarrow \text{try}(cr_1, e, cr_2), m_2 \rightarrow \text{waiting}(cr_3, mb)) : m \rightarrow \text{try}(cr_{1,3}, e, s(cr_2))}$$

□

Règle de fusion R7 : Fusion d'un comportement réactif concurrentiel

SÉMANTIQUE INFORMELLE. — La fusion comportementale favorise l'exécution concurrente de comportements réactifs ne disposant pas, avant application de la fonction de fusion comportementale, de relation d'ordre sur l'exécution de ces comportements réactifs.

SOLUTION INTUITIVE. — Soit les deux règles d'interaction suivantes :

```
R1: square.move (int dx, int dy), int dz -> dz := square.move (dx, dy) // trace.update (dz)
R2: square.move (int x, int y) -> square.move (x, y) // properties.updatePos (x, y)
```

Aucune relation d'ordre n'est définie entre les messages `trace.update` et `properties.updatePos`. Par conséquent, le résultat de la fusion comportementale appliquée à ces deux règles d'interaction ne doit pas en définir une.

Ainsi, après application de la fonction de fusion comportementale, ces deux messages doivent être exécutés concurrentiellement :

```
square.move (int x, int y), int dz -> [ dz := square.move (x, y) // properties.updatePos (x, y) ] //
    trace.update (dz)
```

Ainsi, la sémantique de la fusion comportementale d'un comportement réactif concurrentiel avec un autre comportement réactif est décrite par la règle de sémantique naturelle suivante ³ :

$$\frac{(m_1 \rightarrow cr_2, m_2 \rightarrow cr_3) : m \rightarrow cr_{2,3}}{(m_1 \rightarrow \text{concurrency}(cr_1, cr_2), m_2 \rightarrow cr_3) : m \rightarrow \text{concurrency}(s(cr_1), cr_{2,3}) \mid m_1 \in cr_2}$$

□

Règle de fusion R8 : Fusion d'un comportement réactif séquentiel

SÉMANTIQUE INFORMELLE. — La fusion comportementale favorise l'exécution concurrente mais doit conserver les relations d'ordre établies au sein des règles d'interaction par les comportements réactifs séquentiels.

SOLUTION INTUITIVE. — Soit les deux règles d'interaction suivantes :

```
R1: window.display (screen s) -> [ window.display (s) ; title.display (s) ] ; border.display (s)
R2: window.display (screen s) -> [ window.display (s) ; content.display (s) ] ; scrollbar.display (s)
```

Aucune relation d'ordre n'est définie entre les deux messages `title.display` et `scrollbar.display`, ni entre les deux messages `content.display` et `border.display`. Par conséquent, le résultat de la fusion comportementale appliquée à ces deux règles d'interaction ne doit pas en définir une.

Ainsi, après application de la fonction de fusion comportementale, ces deux messages doivent être exécutés concurrentiellement. Ceci peut être réalisé en remplaçant (par application de la règle de transformation T1) un comportement réactif séquentiel par un comportement réactif concurrentiel.

Ainsi, le résultat de la fusion comportementale des deux règles d'interaction définies ci-dessus est la règle d'interaction suivante :

```
window.display (screen s), mailbox mb -> [ windows.display (s) ; [ [ content.display (s) ; scrollbar.display (s) ] //
    mb := title.display (s) ] ] // wait border.display (s) for mb
```

Ainsi, la sémantique de la fusion comportementale de deux comportements réactifs séquentiels est décrite par la règle de sémantique naturelle suivante :

$$\frac{T1(m_1 \rightarrow \text{sequential}(cr_1, cr_2)) : R}{(m_1 \rightarrow \text{sequential}(cr_1, cr_2), m_2 \rightarrow \text{sequential}(cr_3, cr_4)) : (R, m_2 \rightarrow \text{sequential}(cr_3, cr_4))}$$

où $T1$ est l'application de la règle de transformation T1 définie en 6.2.

□

Règle de fusion R9 : Fusion d'un comportement réactif conditionnel

SÉMANTIQUE INFORMELLE. — Un comportement réactif conditionnel « encapsule » les autres comportements réactifs.

SOLUTION INTUITIVE. — Soit les deux règles d'interaction suivantes :

```
R1: math.calculus (int op, floatVect vars) -> if ctrl.hasSideEffect (op)
    then
        math.calculus (op, vars) ; debug.mathop (op, vars)
    else
        math.calculus (op, vars) // debug.mathop (op, vars) endif
R2: math.calculus (int op, floatVect vars) -> math.calculus (op, vars) ; ctrl.update (op)
```

Intuitivement, la fusion de ces deux règles d'interaction implique de fusionner la seconde règle d'interaction avec, d'une part, la partie `then` de la première règle d'interaction et avec, d'autre part, la partie `else` de cette même règle d'interaction.

Ainsi, les deux règles d'interaction suivantes doivent être fusionnées (partie `then`) :

```
R1': math.calculus (int op, floatVect vars) -> math.calculus (op, vars) ; debug.mathop (op, vars)
R2: math.calculus (int op, floatVect vars) -> math.calculus (op, vars) ; ctrl.update (op)
```

Le résultat de cette fusion est la règle d'interaction suivante :

```
math.calculus (int op, floatVect vars), mailbox mb -> [ mb := math.calculus (op, vars) ; ctrl.update (op) ] //
    wait debug.mathop (op, vars) for mb
```

3. Nous notons $m \in cr$ l'appartenance du message m dans le comportement réactif cr .

Les deux règles d'interaction suivantes doivent, elles aussi, être fusionnées (partieelse):
R1: `math.calculus (int op, floatVect vars) -> math.calculus (op, vars) // debug.mathop (op, vars)`
R2: `math.calculus (int op, floatVect vars) -> math.calculus (op, vars) ; ctrl.update (op)`

Le résultat de leur fusion comportementale est la règle d'interaction suivante :

```
math.calculus (int op, floatVect vars) -> [ math.calculus (op, vars) ; ctrl.update (op) ] //
      debug.mathop (op, vars)
```

De tout ceci, on peut en déduire que le résultat de la fusion comportementale de R1 et R2 est la règle d'interaction suivante :

```
math.calculus (int op, floatVect vars), mailbox mb -> if ctrl.hasSideEffect (op) then
  [ mb := math.calculus (op, vars) ; ctrl.update (op) ] //
  wait debug.mathop (op, vars) for mb
else
  [ math.calculus (op, vars) ; ctrl.update (op) ] //
  debug.mathop (op, vars) endif
```

Ainsi, la sémantique de la fusion comportementale d'un comportement réactif conditionnel avec un autre comportement réactif est décrite par les deux règles de sémantique naturelle suivantes :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) : m \rightarrow cr_{1,3} \quad (m_1 \rightarrow cr_2, m_2 \rightarrow cr_3) : m \rightarrow cr_{2,3}}{(m_1 \rightarrow \text{ifThenElse}(c, cr_1, cr_2), m_2 \rightarrow cr_3) : m \rightarrow \text{ifThenElse}(s(c), cr_{1,3}, cr_{2,3})}$$

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_2) : m \rightarrow cr_{1,2}}{(m_1 \rightarrow \text{ifThen}(c, cr_1), m_2 \rightarrow cr_2) : m \rightarrow \text{ifThen}(s(c), cr_{1,2}, cr_2)}$$

□

Règle de fusion R10 : Fusion d'un comportement réactif de traitement des exceptions

SÉMANTIQUE INFORMELLE. — Un comportement réactif de traitement des exceptions est un « bloc » indivisible.

SOLUTION INTUITIVE. — Soit les deux règles d'interaction suivantes :

```
R1: master.request (string text) -> try master.request (out text) catch ( NetworkException ) slave.request (out text)
R2: master.request (string text) -> try master.request (out text) ; observer.update (text)
      catch ( InvalidOperation ) Reaction
```

Intuitivement, le résultat de la fusion comportementale de ces deux règles d'interaction est la règle d'interaction suivante :

```
master.request (string text) -> try [ try master.request (out text)
      catch ( NetworkException ) slave.request (out text) ;
      observer.update (text) ]
      catch ( InvalidOperation ) Reaction
```

Ainsi, la sémantique de la fusion comportementale de deux comportements réactifs de traitements des exceptions est décrite par les trois règles de sémantique naturelle suivantes :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) : m \rightarrow cr_{1,3} \quad (m_1 \rightarrow cr_2, m_2 \rightarrow cr_4) : m \rightarrow cr_{2,4}}{(m_1 \rightarrow \text{try}(cr_1, e, cr_2), m_2 \rightarrow \text{try}(cr_3, e, cr_4)) : m \rightarrow \text{try}(cr_{1,3}, e, cr_{2,4})}$$

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{try}(cr_3, e_2, cr_4)) : m \rightarrow cr_{1,3,4}}{(m_1 \rightarrow \text{try}(cr_1, e_1, cr_2), m_2 \rightarrow \text{try}(cr_3, e_2, cr_4)) : m \rightarrow \text{try}(m \rightarrow cr_{1,3,4}, e_1, s(cr_2)) \mid cr_3 \in \mathcal{G}_M \cup \mathcal{G}_A \cup \mathcal{G}_W}$$

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) : m \rightarrow cr_{1,3}}{(m_1 \rightarrow \text{try}(cr_1, e_1, cr_2), m_2 \rightarrow \text{try}(cr_3, e_2, cr_4)) : m \rightarrow \text{try}(\text{try}(cr_{1,3}, e_2, s(cr_4)), e_1, s(cr_2))}$$

La sémantique de la fusion comportementale d'un comportement réactif de traitement des exceptions avec un comportement réactif séquentiel est décrite par les deux règles de sémantique naturelle suivantes :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{try}(cr_3, e, cr_4)) : R}{(m_1 \rightarrow \text{sequential}(cr_1, cr_2), m_2 \rightarrow \text{try}(cr_3, e, cr_4)) : m \rightarrow \text{sequential}(R, s(cr_2)) \mid m \in cr_1}$$

$$\frac{(m_1 \rightarrow cr_2, m_2 \rightarrow \text{try}(cr_3, e, cr_4)) : R}{(m_1 \rightarrow \text{sequential}(cr_1, cr_2), m_2 \rightarrow \text{try}(cr_3, e, cr_4)) : m \rightarrow \text{sequential}(s(cr_1), R) \mid m \in cr_2}$$

□

4.3 Complétude des règles de la fusion comportementale

Le tableau 6.1 présente la règle de fusion comportementale à appliquer en fonction du type des comportements réactifs et précise le type de comportement réactif de la règle d'interaction résultante de la fusion comportementale. Ce tableau montre également la fermeture du système de fusion.

delegate	$\frac{R1}{\text{échech}}$	$\frac{R3}{\text{sequential}}$	$\frac{R3}{\text{concurrency}}$	$\frac{R3}{\text{ifThenElse}}$	$\frac{R3}{\text{delegate}}$	$\frac{R3}{\text{assignment}}$	$\frac{R3}{\text{waiting}}$	$\frac{R3}{\text{try}}$	$\frac{R2}{\text{exception}}$
sequential	$\frac{R3}{\text{sequential}}$	$\frac{T1, R8}{\text{concurrency}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R4}{\text{sequential}}$	$\frac{R5}{\text{sequential}}$	$\frac{R6}{\text{sequential}}$	$\frac{R10}{\text{sequential}}$	$\frac{R2}{\text{exception}}$
concurrency	$\frac{R3}{\text{concurrency}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R4}{\text{concurrency}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R2}{\text{exception}}$
ifThenElse	$\frac{R3}{\text{ifThenElse}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R4}{\text{ifThenElse}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R2}{\text{exception}}$
msg	$\frac{R3}{\text{delegate}}$	$\frac{R4}{\text{sequential}}$	$\frac{R4}{\text{concurrency}}$	$\frac{R4}{\text{ifThenElse}}$	$\frac{R4}{\text{msg}}$	$\frac{R4}{\text{assignment}}$	$\frac{R4}{\text{waiting}}$	$\frac{R4}{\text{try}}$	$\frac{R2}{\text{exception}}$
assignment	$\frac{R3}{\text{assignment}}$	$\frac{R5}{\text{sequential}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R4}{\text{assignment}}$	$\frac{R5}{\text{assignment}}$	$\frac{R6}{\text{waiting}}$	$\frac{R5}{\text{try}}$	$\frac{R2}{\text{exception}}$
waiting	$\frac{R3}{\text{waiting}}$	$\frac{R6}{\text{sequential}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R4}{\text{waiting}}$	$\frac{R6}{\text{waiting}}$	$\frac{R6}{\text{waiting}}$	$\frac{R6}{\text{try}}$	$\frac{R2}{\text{exception}}$
try	$\frac{R3}{\text{try}}$	$\frac{R10}{\text{sequential}}$	$\frac{R7}{\text{concurrency}}$	$\frac{R9}{\text{ifThenElse}}$	$\frac{R4}{\text{try}}$	$\frac{R5}{\text{try}}$	$\frac{R6}{\text{try}}$	$\frac{R10}{\text{try}}$	$\frac{R2}{\text{exception}}$
exception	$\frac{R2}{\text{exception}}$	$\frac{R2}{\text{exception}}$	$\frac{R2}{\text{exception}}$	$\frac{R2}{\text{exception}}$	$\frac{R2}{\text{exception}}$	$\frac{R2}{\text{exception}}$	$\frac{R2}{\text{exception}}$	$\frac{R2}{\text{exception}}$	$\frac{R2}{\text{exception}}$

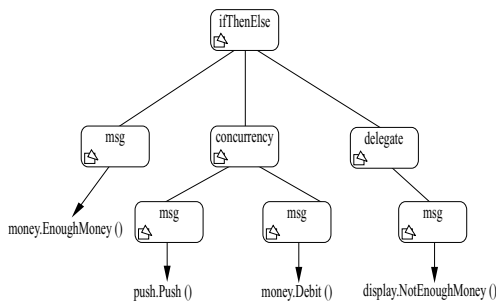
TAB. 6.1 – Complétude des règles de la fusion comportementale

5 Propriété de commutativité de la fusion comportementale

Propriété 6.2 : Commutativité de la fonction de fusion comportementale

La fonction de fusion comportementale est commutative, c'est-à-dire qu'elle vérifie la propriété suivante (on rappelle que deux règles d'interaction sont équivalentes si et seulement si la sémantique de leur exécution est identique) :

$$\forall (r_1, r_2) \in \mathcal{R}^2 \quad \varphi(r_1, r_2) \equiv \varphi(r_2, r_1)$$



Pour cette preuve, nous allons supposer que les comportements réactifs décrits par les règles d'interaction sont représentés par des arbres (figure ci-contre) dont les noeuds sont les opérateurs réactifs non terminaux (sequential ou ifThenElse par exemple), et les feuilles les opérateurs terminaux (msg ou assign par exemple).

L'arbre de syntaxe abstraite du langage ISL est une représentation possible.

Nous allons prouver cette propriété par récurrence sur la hauteur des arbres de syntaxe abstraite décrivant les règles d'interaction à fusionner. Nous avons donc l'hypothèse de récurrence suivante ⁴ :

$$\forall n \in \mathbb{N}^*, \forall (r_1, r_2, r_3) \in \mathcal{R}^3 \mid \begin{aligned} &h(r_1) = n - 1 \wedge h(r_3) = n \wedge r_1 \in r_3 \\ &\varphi(r_1, r_2) \equiv \varphi(r_2, r_1) \Rightarrow \varphi(r_3, r_2) \equiv \varphi(r_2, r_3) \end{aligned}$$

4. Nous notons $r_1 \in r_2$ le fait que l'arbre de r_1 est un sous-arbre de r_2 . De même nous notons h la fonction associant à une règle d'interaction la hauteur de son arbre de syntaxe abstraite. Son domaine de définition est : $h : \mathcal{R} \rightarrow \mathbb{N}$

De cette hypothèse de récurrence, nous pouvons en déduire l'hypothèse de récurrence étendue suivante :

$$\forall n \in \mathbb{N}^*, \forall (r_1, r_2, r_3) \in \mathcal{R}^3 \mid \bar{h}(r_1) \leq n - 1 \wedge \bar{h}(r_3) = n \wedge r_1 \in r_3 \\ \varphi(r_1, r_2) \equiv \varphi(r_2, r_1) \Rightarrow \varphi(r_3, r_2) \equiv \varphi(r_2, r_3)$$

5.1 Preuve pour les axiomes (règles terminales) de la fusion comportementale

Preuve de la commutativité de la règle de fusion des exceptions (R2) :

Nous avons les deux axiomes suivants :

$$(R1 : m_1 \rightarrow \text{exception}(e_1), R2 : m_2 \rightarrow \text{exception}(e_2)) : m \rightarrow \text{concurrency}(\text{exception}(e_1), \text{exception}(e_2))$$

$$(m_1 \rightarrow cr_1, m_2 \rightarrow \text{exception}(e)) : m \rightarrow \text{exception}(e)$$

Dans le cas du premier axiome, nous avons les équivalences suivantes :

$$\begin{aligned} \varphi(R_1, R_2) &= m \rightarrow \text{concurrency}(\text{exception}(e_1), \text{exception}(e_2)) \\ &\equiv m \rightarrow \text{concurrency}(\text{exception}(e_2), \text{exception}(e_1)) && \text{(par application de T4)} \\ &\equiv \varphi(R_2, R_1) \end{aligned}$$

Le second axiome stipule qu'un comportement réactif d'exception est absorbant vis-à-vis des autres comportements réactifs. Ainsi, par définition, ce second axiome indique que la fusion d'un comportement réactif d'exception avec un autre comportement réactif est commutative.

Ainsi la règle de fusion comportementale R2 est commutative. ◇

Preuve de la commutativité de la règle de fusion de délégation (R3) :

La commutativité de cette règle de fusion comportementale est vérifiée par définition. ◇

Preuve de la commutativité de la règle de fusion d'envoi de messages (R4) :

Nous avons l'axiome suivant :

$$(m_1 \rightarrow \text{msg}(o, m), m_2 \rightarrow cr) : m \rightarrow s(cr)$$

Cet axiome stipule qu'un comportement réactif d'envoi de messages est neutre vis-à-vis des autres comportements réactifs. Ainsi, par définition, cet axiome indique que la fusion d'un comportement réactif d'envoi de messages avec un autre comportement réactif est commutative. ◇

Nous venons de prouver que la propriété de commutativité de la fonction de fusion comportementale est vérifiée pour chacun des axiomes. Nous pouvons maintenant utiliser l'hypothèse de récurrence étendue et montrer la propriété de commutativité pour les autres règles de fusion comportementale.

5.2 Preuve pour les règles non terminales de la fusion comportementale

NOTE. — La preuve de la commutativité de la fonction de fusion comportementale, pour chacune des règles de fusion comportementale non terminale, est basée sur le même schéma de preuve. Celui-ci consiste à utiliser l'hypothèse de récurrence ainsi que les règles d'équivalence (présentées au paragraphe 6.2) afin d'obtenir du résultat de la fusion de deux règles d'interaction r_1 avec r_2 le résultat de la fusion de r_2 avec r_1 .

Par conséquent, nous ne détaillons ce schéma de preuve que pour la première règle de fusion comportementale (R5) et, pour les autres règles de fusion, ne présentons que les étapes essentielles à la compréhension de la preuve.

Preuve de la commutativité de la règle de fusion d'affectation (R5) :

La fusion de deux comportements réactifs d'affectation est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_2) : m \rightarrow cr_{1.2}}{(R1 : m_1 \rightarrow \text{assignment}(x_1, cr_1), R2 : m_2 \rightarrow \text{assignment}(x_2, cr_2)) : m \rightarrow \text{assignment}(x_1, \text{assignment}(x_2, cr_{1.2}))}$$

Grâce à l'hypothèse de récurrence, nous avons ⁵ :

$$\varphi (m_1 \rightarrow cr_1, m_2 \rightarrow cr_2) = m \rightarrow cr_{1.2} \equiv \varphi (m_2 \rightarrow cr_2, m_1 \rightarrow cr_1) = m \rightarrow cr_{2.1}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned} \varphi (R_1, R_2) &= m \rightarrow \text{assignment} (x_1, \text{assignment} (x_2, cr_{1.2})) \\ &\equiv m \rightarrow \text{assignment} (x_1, \text{assignment} (x_2, cr_{2.1})) && \text{(par hypothèse de récurrence)} \\ &\equiv m \rightarrow \text{assignment} (x_2, \text{assignment} (x_1, cr_{2.1})) && \text{(par application de T4)} \\ &\equiv \varphi (R_2, R_1) \end{aligned}$$

La fusion comportementale d'un comportement réactif d'affectation avec un comportement réactif séquentiel est décrite par deux règles de sémantique naturelle. La preuve de commutativité est en tout point identique pour chacune de ces deux règles. Par conséquent, nous ne détaillons ici que la preuve de commutativité pour la première règle, à savoir :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment} (x, cr_3)) : m \rightarrow cr_{1.3}}{(R1 : m_1 \rightarrow \text{sequential} (cr_1, cr_2), R2 : m_2 \rightarrow \text{assignment} (x, cr_3)) : m \rightarrow \text{sequential} (cr_{1.3}, s (cr_2)) \mid m \in cr_1}$$

Grâce à l'hypothèse de récurrence, nous avons :

$$\varphi (m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment} (x, cr_3)) = m \rightarrow cr_{1.3} \equiv \varphi (m_2 \rightarrow \text{assignment} (x, cr_3), m_1 \rightarrow cr_1) = m \rightarrow cr_{3.1}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned} \varphi (R_1, R_2) &= m \rightarrow \text{sequential} (cr_{1.3}, s (cr_2)) \\ &\equiv m \rightarrow \text{sequential} (cr_{3.1}, s (cr_2)) && \text{(par hypothèse de récurrence)} \\ &\equiv \varphi (R_2, R_1) \end{aligned}$$

La fusion comportementale d'un comportement réactif d'affectation avec un comportement réactif de traitement des exceptions est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment} (x, cr_3)) : m \rightarrow cr_{1.3}}{(R1 : m_1 \rightarrow \text{try} (cr_1, e, cr_2), R2 : m_2 \rightarrow \text{assignment} (x, cr_3)) : m \rightarrow \text{try} (cr_{1.3}, e, s (cr_2))}$$

Grâce à l'hypothèse de récurrence définie ci-dessus nous pouvons en déduire que :

$$\begin{aligned} \varphi (R_1, R_2) &= m \rightarrow \text{try} (cr_{1.3}, e, s (cr_2)) \\ &\equiv m \rightarrow \text{try} (cr_{3.1}, e, s (cr_2)) && \text{(par hypothèse de récurrence)} \\ &\equiv \varphi (R_2, R_1) \end{aligned}$$

Ainsi, la règle de fusion comportementale R5 est commutative. ◇

Preuve de la commutativité de la règle de fusion d'attente (R6) :

La fusion comportementale d'un comportement réactif d'attente avec un comportement réactif d'affectation est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment} (x, cr_2)) : m \rightarrow cr_{1.2}}{(m_1 \rightarrow \text{waiting} (cr_1, mb), m_2 \rightarrow \text{assignment} (x, cr_2)) : m \rightarrow \text{waiting} (cr_{1.2}, mb)}$$

Grâce à l'hypothèse de récurrence, nous avons :

$$\varphi (m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment} (x, cr_2)) = m \rightarrow cr_{1.2} \equiv \varphi (m_2 \rightarrow \text{assignment} (x, cr_2), m_1 \rightarrow cr_1) = m \rightarrow cr_{2.1}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned} \varphi (R_1, R_2) &= m \rightarrow \text{waiting} (cr_{1.2}, mb) \\ &\equiv m \rightarrow \text{waiting} (cr_{2.1}, mb) && \text{(par hypothèse de récurrence)} \\ &\equiv \varphi (R_2, R_1) \end{aligned}$$

Les preuves de la propriété de commutativité de la fonction de fusion comportementale concernant les autres règles de sémantique naturelle de la règle de fusion R6 sont en tout point identiques aux preuves de la commutativité de la fonction de fusion comportementale présentées pour la règle de fusion R5. Nous ne les détaillons donc pas.

Ainsi, la règle de fusion comportementale R6 est commutative. ◇

5. Nous notons $cr_{x,y}$ le comportement réactif résultant de la fusion comportementale du comportement réactif cx avec le comportement réactif cy .

Preuve de la commutativité de la règle de fusion d'un comportement réactif concurrentiel (R7) :
La fusion comportementale d'un comportement réactif concurrentiel avec un autre comportement réactif est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_2, m_2 \rightarrow cr_3) : m \rightarrow cr_{2.3}}{(R1 : m_1 \rightarrow \text{concurrency}(cr_1, cr_2), R2 : m_2 \rightarrow cr_3) : m \rightarrow \text{concurrency}(s(cr_1), cr_{2.3}) \mid m_1 \in cr_2}$$

Grâce à l'hypothèse de récurrence, nous avons :

$$\varphi(m_1 \rightarrow cr_2, m_2 \rightarrow cr_3) = m \rightarrow cr_{2.3} \equiv \varphi(m_2 \rightarrow cr_3, m_1 \rightarrow cr_2) = m \rightarrow cr_{3.2}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned} \varphi(R_1, R_2) &= m \rightarrow \text{concurrency}(s(cr_1), cr_{2.3}) \\ &\equiv m \rightarrow \text{concurrency}(s(cr_1), cr_{3.2}) \\ &\equiv \varphi(R_2, R_1) \end{aligned}$$

Ainsi la règle de fusion comportementale R7 est commutative. ◇

Preuve de la commutativité de la règle de fusion d'un comportement réactif séquentiel (R8) :

La fusion comportementale de deux comportements réactifs séquentiels est définie par la fusion comportementale de l'un des deux comportements réactifs séquentiels avec la transformation de l'autre comportement réactif séquentiel en un comportement réactif concurrentiel. Or la fusion comportementale d'un comportement réactif concurrentiel avec un autre comportement réactif est commutative (preuve ci-dessus) donc la fusion comportementale de deux comportements réactifs séquentiels est commutative.

Ainsi la règle de fusion comportementale R8 est commutative. ◇

Preuve de la commutativité de la règle de fusion d'un comportement réactif conditionnel (R9) :

La fusion comportementale d'un comportement réactif conditionnel avec un autre comportement réactif est notamment décrite par la règle de sémantique naturelle suivante :

$$\frac{\begin{array}{l} (m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) : m \rightarrow cr_{1.3} \\ (m_1 \rightarrow cr_2, m_2 \rightarrow cr_3) : m \rightarrow cr_{2.3} \end{array}}{(R1 : m_1 \rightarrow \text{ifThenElse}(c, cr_1, cr_2), R2 : m_2 \rightarrow cr_3) : m \rightarrow \text{ifThenElse}(s(c), cr_{1.3}, cr_{2.3})}$$

Grâce à l'hypothèse de récurrence, nous avons :

$$\begin{aligned} \varphi(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) &= m \rightarrow cr_{1.3} \equiv \varphi(m_2 \rightarrow cr_3, m_1 \rightarrow cr_1) = m \rightarrow cr_{3.1} \\ \varphi(m_1 \rightarrow cr_2, m_2 \rightarrow cr_3) &= m \rightarrow cr_{2.3} \equiv \varphi(m_2 \rightarrow cr_3, m_1 \rightarrow cr_2) = m \rightarrow cr_{3.2} \end{aligned}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned} \varphi(R_1, R_2) &= m \rightarrow \text{ifThenElse}(s(c), cr_{1.3}, cr_{2.3}) \\ &\equiv m \rightarrow \text{ifThenElse}(s(c), cr_{3.1}, cr_{3.2}) \\ &\equiv \varphi(R_2, R_1) \end{aligned}$$

La preuve de la propriété de commutativité de la fonction de fusion comportementale concernant la seconde règle de sémantique naturelle de la règle de fusion R9 est en tout point identique. Nous ne la détaillons donc pas.

Ainsi la règle de fusion comportementale R9 est commutative. ◇

Preuve de la commutativité de la règle de fusion d'un comportement réactif de traitement des exceptions (R10) :

La fusion comportementale de deux comportements réactifs de traitement de la même exception est décrite par la règle de sémantique naturelle suivante :

$$\frac{\begin{array}{l} (m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) : m \rightarrow cr_{1.3} \\ (m_1 \rightarrow cr_2, m_2 \rightarrow cr_4) : m \rightarrow cr_{2.4} \end{array}}{(m_1 \rightarrow \text{try}(cr_1, e, cr_2), m_2 \rightarrow \text{try}(cr_3, e, cr_4)) : m \rightarrow \text{try}(cr_{1.3}, e, cr_{2.4})}$$

Grâce à l'hypothèse de récurrence, nous avons :

$$\begin{aligned} \varphi(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) &= m \rightarrow cr_{1.3} \equiv \varphi(m_2 \rightarrow cr_3, m_1 \rightarrow cr_1) = m \rightarrow cr_{3.1} \\ \varphi(m_1 \rightarrow cr_2, m_2 \rightarrow cr_4) &= m \rightarrow cr_{2.4} \equiv \varphi(m_2 \rightarrow cr_4, m_1 \rightarrow cr_2) = m \rightarrow cr_{4.2} \end{aligned}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned}\varphi(R_1, R_2) &= m \rightarrow \text{try}(cr_{1.3}, e, cr_{2.4}) \\ &\equiv m \rightarrow \text{try}(cr_{3.1}, e, cr_{4.2}) \\ &\equiv \varphi(R_2, R_1)\end{aligned}$$

La fusion comportementale de deux comportements réactifs de traitement de deux exceptions différentes est décrite par les deux règles de sémantique naturelle suivantes :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{try}(cr_3, e_2, cr_4)) : m \rightarrow cr_{1.3.4}}{(R1 : m_1 \rightarrow \text{try}(cr_1, e_1, cr_2), R2 : m_2 \rightarrow \text{try}(cr_3, e_2, cr_4)) : m \rightarrow \text{try}(m \rightarrow cr_{1.3.4}, e_1, s(cr_2)) \mid cr_3 \in \mathcal{G}_M \cup \mathcal{G}_A \cup \mathcal{G}_W}$$

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) : m \rightarrow cr_{1.3}}{(R3 : m_1 \rightarrow \text{try}(cr_1, e_1, cr_2), R4 : m_2 \rightarrow \text{try}(cr_3, e_2, cr_4)) : m \rightarrow \text{try}(\text{try}(cr_{1.3}, e_2, s(cr_4)), e_1, s(cr_2))}$$

Grâce à l'hypothèse de récurrence, nous avons :

$$\begin{aligned}\varphi(m_1 \rightarrow cr_1, m_2 \rightarrow \text{try}(cr_3, e_2, cr_4)) &= m \rightarrow cr_{1.3.4} \equiv \varphi(m_2 \rightarrow \text{try}(cr_3, e_2, cr_4), m_1 \rightarrow cr_1) = m \rightarrow cr_{3.4.1} \\ \varphi(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) &= m \rightarrow cr_{1.3} \equiv \varphi(m_2 \rightarrow cr_3, m_1 \rightarrow cr_1) = m \rightarrow cr_{3.1}\end{aligned}$$

De ce fait, pour la première règle de sémantique naturelle, nous pouvons en déduire que :

$$\begin{aligned}\varphi(R_1, R_2) &= m \rightarrow \text{try}(cr_{1.3.4}, e_1, s(cr_2)) \\ &\equiv m \rightarrow \text{try}(cr_{3.4.1}, e_1, s(cr_2)) \\ &\equiv \varphi(R_2, R_1)\end{aligned}$$

De même, pour la seconde règle de sémantique naturelle, nous pouvons en déduire que :

$$\begin{aligned}\varphi(R_3, R_4) &= m \rightarrow \text{try}(\text{try}(cr_{1.3}, e_2, s(cr_4)), e_1, s(cr_2)) \\ &\equiv m \rightarrow \text{try}(\text{try}(cr_{3.1}, e_2, s(cr_4)), e_1, s(cr_2)) \\ &\equiv m \rightarrow \text{try}(\text{try}(cr_{3.1}, e_1, s(cr_2)), e_2, s(cr_4)) \quad (\text{par application de T4}) \\ &\equiv \varphi(R_4, R_3)\end{aligned}$$

Les preuves de la propriété de commutativité de la fonction de fusion comportementale concernant les autres règles de sémantique naturelle de la règle de fusion R10 sont en tout point identique aux preuves de la commutativité de la fonction de fusion comportementales présentées ci-dessus. Nous ne les détaillons donc pas.

Ainsi la règle de fusion comportementale R10 est commutative. \diamond

Nous venons par conséquent de vérifier que toutes les règles de fusion comportementale vérifiaient la propriété de commutativité :

$$\forall (r_1, r_2) \in \mathcal{R}^2 \quad \varphi(r_1, r_2) \equiv \varphi(r_2, r_1)$$

Ainsi, la fonction de fusion comportementale est commutative. \square

6 Propriété d'associativité de la fusion comportementale

Propriété 6.3 : Associativité de la fonction de fusion comportementale

La fonction de fusion comportementale est associative, c'est-à-dire qu'elle vérifie la propriété suivante :

$$\forall (r_1, r_2, r_3) \in \mathcal{R}^3 \quad \varphi(\varphi(r_1, r_2), r_3) \equiv \varphi(r_1, \varphi(r_2, r_3))$$

Tout comme pour prouver la commutativité de la fonction de fusion comportementale, nous allons prouver cette propriété d'associativité par récurrence sur la hauteur des arbres de syntaxe abstraite décrivant les règles d'interaction à fusionner. Nous avons donc l'hypothèse de récurrence suivante :

$$\begin{aligned}\forall n \in \mathbb{N}^*, \forall (r_1, r_2, r_3, r_4) \in \mathcal{R}^4 \mid \bar{h}(r_1) = n - 1 \wedge \bar{h}(r_3) = n \wedge r_1 \in r_3 \\ \varphi(\varphi(r_4, r_2), r_3) \equiv \varphi(r_4, \varphi(r_2, r_3)) \Rightarrow \varphi(\varphi(r_1, r_2), r_3) \equiv \varphi(r_1, \varphi(r_2, r_3))\end{aligned}$$

De cette hypothèse de récurrence nous pouvons en déduire l'hypothèse de récurrence étendue suivante :

$$\begin{aligned}\forall n \in \mathbb{N}^*, \forall (r_1, r_2, r_3, r_4) \in \mathcal{R}^4 \mid \bar{h}(r_1) \leq n - 1 \wedge \bar{h}(r_3) = n \wedge r_1 \in r_3 \\ \varphi(\varphi(r_4, r_2), r_3) \equiv \varphi(r_4, \varphi(r_2, r_3)) \Rightarrow \varphi(\varphi(r_1, r_2), r_3) \equiv \varphi(r_1, \varphi(r_2, r_3))\end{aligned}$$

6.1 Preuve pour les axiomes (règles terminales) de la fusion comportementale

Preuve de l'associativité de la règle de fusion des exceptions (R2) :

Nous avons les deux axiomes suivants :

$$(m_1 \rightarrow \text{exception}(e_1), m_2 \rightarrow \text{exception}(e_2)) : m \rightarrow \text{concurrency}(\text{exception}(e_1), \text{exception}(e_2))$$

$$(m_1 \rightarrow cr_1, m_2 \rightarrow \text{exception}(e)) : m \rightarrow \text{exception}(e)$$

Nous pouvons en déduire que :

$$\begin{aligned} & - R_1 \in \mathcal{G}_E, (R_2, R_3) \in \mathcal{G}_{-E}^2 \\ & \quad \varphi(R_1, \varphi(R_2, R_3)) = R_1 \quad \text{par application du second axiome de la règle R2} \\ & \quad \text{et } \varphi(\varphi(R_1, R_2), R_3) = \varphi(R_1, R_3) \\ & \quad \quad = R_1 \\ & - (R_1, R_2) \in \mathcal{G}_E^2, R_3 \in \mathcal{G}_{-E} \\ & \quad \varphi(R_1, \varphi(R_2, R_3)) = \underbrace{\varphi(R_1, R_2)}_{\in \mathcal{G}_E} \text{ car } \varphi(R_2, R_3) = R_2 \\ & \quad \quad = \varphi(\varphi(R_1, R_2), R_3) \text{ puisque } R_3 \in \mathcal{G}_{-E} \\ & - R_1 \in \mathcal{G}_E : m_1 \rightarrow \text{exception}(e_1), \quad R_2 \in \mathcal{G}_E : m_2 \rightarrow \text{exception}(e_2), \quad R_3 \in \mathcal{G}_E : m_3 \rightarrow \text{exception}(e_3) \\ & \quad \varphi(\varphi(R_1, R_2), R_3) = \varphi(m \rightarrow \text{concurrency}(\text{exception}(e_1), \text{exception}(e_2)), R_3) \\ & \quad \quad = m \rightarrow \text{concurrency}(\text{concurrency}(\text{exception}(e_1), \text{exception}(e_2)), \text{exception}(e_3)) \\ & \quad \text{et } \varphi(R_1, \varphi(R_2, R_3)) = \varphi(R_1, m \rightarrow \text{concurrency}(\text{exception}(e_2), \text{exception}(s(e_3)))) \\ & \quad \quad = m \rightarrow \text{concurrency}(\text{exception}(e_1), \text{concurrency}(\text{exception}(e_2), \text{exception}(e_3))) \\ & \quad \quad = \varphi(\varphi(R_1, R_2), R_3) \end{aligned}$$

Ainsi la règle de fusion comportementale R2 est associative. ◇

Preuve de l'associativité de la règle de fusion de délégation (R3) :

La fusion comportementale d'un comportement réactif de délégation avec un autre comportement réactif est notamment décrite par les deux axiomes de sémantique naturelle suivants :

$$(m_1 \rightarrow \text{delegate}(cr_1), m_2 \rightarrow cr_2) : m \rightarrow \text{subst}(m, s(\text{delegate}(cr_1)), s(cr_2))$$

$$(m_1 \rightarrow cr_1, m_2 \rightarrow \text{delegate}(cr_2)) : m \rightarrow \text{subst}(m, s(\text{delegate}(cr_2)), s(cr_1))$$

où subst est une fonction de substitution.

Soit $R_1 : m_1 \rightarrow \text{delegate}(cr_1)$, $R_2 : m_2 \rightarrow cr_2$, et $R_3 : m_3 \rightarrow cr_3$.

Nous pouvons en déduire que :

$$\begin{aligned} \varphi(R_1, \varphi(R_2, R_3)) & \equiv \varphi(m_1 \rightarrow \text{delegate}(cr_1), \varphi(R_2, R_3)) \\ & \equiv m \rightarrow \text{subst}(m, s(\text{delegate}(cr_1)), \varphi(R_2, R_3)) \\ \text{et } \varphi(\varphi(R_1, R_2), R_3) & \equiv \varphi(m \rightarrow \text{subst}(s(\text{delegate}(cr_1)), s(cr_2)), R_3) \end{aligned}$$

Or la sémantique d'un comportement réactif de délégation est strictement le même que celui du message interagissant. Ainsi le résultat de la fusion de la règle R_2 , à laquelle le message interagissant a été substitué par le comportement réactif de délégation de la règle R_1 , avec la règle R_3 est strictement identique au résultat de la fusion des règles R_2 et R_3 auquel le message interagissant est substitué par le comportement réactif de délégation de la règle R_1 .

Nous avons par conséquent l'équivalence suivante :

$$m \rightarrow \text{subst}(m, s(\text{delegate}(cr_1)), \varphi(R_2, R_3)) = \varphi(m \rightarrow \text{subst}(s(\text{delegate}(cr_1)), s(cr_2)), R_3)$$

Nous pouvons en déduire que :

$$\varphi(R_1, \varphi(R_2, R_3)) \equiv \varphi(\varphi(R_1, R_2), R_3)$$

Ainsi la règle de fusion comportementale R3 est associative. ◇

Preuve de l'associativité de la règle de fusion des envois de messages (R4) :

La fusion comportementale d'un comportement réactif d'envoi de messages avec un autre comportement réactif est décrite par la règle de sémantique naturelle suivante :

$$(m_1 \rightarrow \text{msg}(o, m), m_2 \rightarrow cr) : m \rightarrow s(cr)$$

Soit $R_1 : m_1 \rightarrow \text{msg}(o, m)$ et $(R_2, R_3) \in \mathcal{G}_{-E}^2$.

Nous pouvons en déduire que :

$$\begin{aligned} \varphi(\varphi(R_1, R_2), R_3) &= \varphi(\varphi(m_1 \rightarrow \text{msg}(o, m), R_2), R_3) \\ &\equiv \varphi(R_2, R_3) \\ \text{et } \varphi(R_1, \varphi(R_2, R_3)) &= \varphi(m_1 \rightarrow \text{msg}(o, m), \varphi(R_2, R_3)) \\ &\equiv \varphi(R_2, R_3) \end{aligned}$$

Ainsi la règle de fusion comportementale R4 est associative. ◇

Nous venons de prouver que la propriété d'associativité de la fonction de fusion comportementale est vérifiée pour les cas de base. Nous pouvons maintenant utiliser l'hypothèse de récurrence étendue et montrer la propriété d'associativité pour les autres règles de fusion comportementale.

6.2 Preuve pour les règles non terminales de la fusion comportementale

NOTE. — Les preuves de l'associativité de la fonction de fusion comportementale concernant les règles non terminales de la fusion comportementale sont toutes basées sur le même principe. De ce fait, nous le présentons en détail pour la première règle non terminale (règle R5) et ne présentons que les étapes essentielles à la compréhension des autres preuves.

Preuve de l'associativité de la règle de fusion d'affectation (R5) :

La fusion de deux comportements réactifs d'affectation est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow cr_2) : m \rightarrow cr_{1,2}}{(R1 : m_1 \rightarrow \text{assignment}(x_1, cr_1), R2 : m_2 \rightarrow \text{assignment}(x_2, cr_2)) : m \rightarrow \text{assignment}(x_1, \text{assignment}(x_2, cr_{1,2}))}$$

Soit $R_1 : m_1 \rightarrow \text{assignment}(x_1, cr_1)$, $R_2 : m_2 \rightarrow \text{assignment}(x_2, cr_2)$, et $R_3 : m_3 \rightarrow \text{assignment}(x_3, cr_3)$.

Grâce à l'hypothèse de récurrence, nous avons :

$$\varphi(\varphi(m_1 \rightarrow cr_1, m_2 \rightarrow cr_2), m_3 \rightarrow cr_3) = cr_{(1,2),3} \equiv \varphi(m_1 \rightarrow cr_1, \varphi(m_2 \rightarrow cr_2, m_3 \rightarrow cr_3)) = cr_{1,(2,3)}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned} \varphi(\varphi(R_1, R_2), R_3) &\equiv \varphi(m \rightarrow \text{assignment}(x_1, \text{assignment}(x_2, cr_{1,2})), R_3) \\ &\equiv m \rightarrow \text{assignment}(x_1, \text{assignment}(x_2, \text{assignment}(x_3, cr_{(1,2),3}))) \\ \text{et } \varphi(R_1, \varphi(R_2, R_3)) &\equiv \varphi(R_1, m \rightarrow \text{assignment}(x_2, \text{assignment}(x_3, cr_{2,3}))) \\ &\equiv m \rightarrow \text{assignment}(x_1, \text{assignment}(x_2, \text{assignment}(x_3, cr_{1,(2,3)}))) \\ &\equiv m \rightarrow \text{assignment}(x_1, \text{assignment}(x_2, \text{assignment}(x_3, cr_{(1,2),3}))) \\ &\equiv \varphi(\varphi(R_1, R_2), R_3) \end{aligned}$$

La fusion comportementale d'un comportement réactif d'affectation avec un comportement réactif séquentiel est décrite par deux règles de sémantique naturelle. La preuve de l'associativité est en tout point identique pour chacune de ces deux règles. Par conséquent, nous ne détaillons ici que la preuve de l'associativité pour la première règle, à savoir :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow cr_{1,3}}{(R1 : m_1 \rightarrow \text{sequential}(cr_1, cr_2), R2 : m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow \text{sequential}(cr_{1,3}, s(cr_2)) \mid m \in cr_1}$$

Soit $R_1 : m_1 \rightarrow \text{sequential}(cr_1, cr_2)$, $R_2 : m_2 \rightarrow \text{assignment}(x_1, cr_3)$, et $R_3 : m_3 \rightarrow \text{assignment}(x_2, cr_4)$.

Grâce à l'hypothèse de récurrence, nous avons :

$$\varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3) = cr_{(1,2),3} \equiv \varphi(m_1 \rightarrow cr_1, \varphi(R_2, R_3)) = cr_{1,(2,3)}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned}\varphi(\varphi(R_1, R_2), R_3) &\equiv \varphi(m \rightarrow \text{sequential}(cr_{1.3}, s(cr_2)), R_3) \\ &\equiv m \rightarrow \text{sequential}(cr_{(1.2).3}, s(cr_2)) \\ \text{et } \varphi(R_1, \varphi(R_2, R_3)) &\equiv m \rightarrow \text{sequential}(\varphi(m_1 \rightarrow cr_1, \varphi(R_2, R_3)), s(cr_2)) \\ &\equiv m \rightarrow \text{sequential}(cr_{(1.2).3}, s(cr_2)) \\ &\equiv \varphi(\varphi(R_1, R_2), R_3)\end{aligned}$$

La fusion comportementale d'un comportement réactif d'affectation avec un comportement réactif de traitement des exceptions est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow cr_{1.3}}{(R_1 : m_1 \rightarrow \text{try}(cr_1, e, cr_2), R_2 : m_2 \rightarrow \text{assignment}(x, cr_3)) : m \rightarrow \text{try}(cr_{1.3}, e, s(cr_2))}$$

Soit $R_1 : m_1 \rightarrow \text{try}(cr_1, e, cr_2)$, $R_2 : m_2 \rightarrow \text{assignment}(x_1, cr_3)$, $R_3 : m_3 \rightarrow \text{assignment}(x_2, cr_4)$.

Grâce à l'hypothèse de récurrence définie ci-dessus nous pouvons en déduire que :

$$\begin{aligned}\varphi(\varphi(R_1, R_2), R_3) &\equiv m \rightarrow \text{try}(\varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3), e, s(cr_2)) \\ \text{et } \varphi(R_1, \varphi(R_2, R_3)) &\equiv m \rightarrow \text{try}(\varphi(m_1 \rightarrow cr_1, \varphi(R_2, R_3)), e, s(cr_2)) \\ &\equiv m \rightarrow \text{try}(\varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3), e, s(cr_2)) \\ &\equiv \varphi(\varphi(R_1, R_2), R_3)\end{aligned}$$

Ainsi la règle de fusion comportementale R5 est associative. ◇

Preuve de l'associativité de la règle de fusion d'attente (R6) :

La fusion comportementale d'un comportement réactif d'attente avec un comportement réactif d'affectation est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_1, m_2 \rightarrow \text{assignment}(x, cr_2)) : m \rightarrow cr_{1.2}}{(m_1 \rightarrow \text{waiting}(cr_1, mb), m_2 \rightarrow \text{assignment}(x, cr_2)) : m \rightarrow \text{waiting}(cr_{1.2}, mb)}$$

Soit $R_1 : m_1 \rightarrow \text{waiting}(cr_1, mb)$, $R_2 : m_2 \rightarrow \text{assignment}(x_1, cr_2)$, et $R_3 : m_3 \rightarrow \text{assignment}(x_2, cr_3)$.

Grâce à l'hypothèse de récurrence, nous avons :

$$\varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3) \equiv \varphi(m_1 \rightarrow cr_1, \varphi(R_2, R_3))$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned}\varphi(\varphi(R_1, R_2), R_3) &\equiv \varphi(m \rightarrow \text{waiting}(\varphi(m_1 \rightarrow cr_1, R_2), mb), R_3) \\ &\equiv m \rightarrow \text{waiting}(\varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3), mb) \\ \text{et } \varphi(R_1, \varphi(R_2, R_3)) &\equiv m \rightarrow \text{waiting}(\varphi(m_1 \rightarrow cr_1, \varphi(R_2, R_3)), mb) \\ &\equiv m \rightarrow \text{waiting}(\varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3), mb) \\ &\equiv \varphi(\varphi(R_1, R_2), R_3)\end{aligned}$$

Les preuves de la propriété d'associativité de la fonction de fusion comportementale concernant les autres règles de sémantique naturelle de la règle de fusion R6 sont en tout point identique aux preuves de l'associativité de la fonction de fusion comportementale présentées pour la règle de fusion R5. Nous ne les détaillons donc pas.

Ainsi la règle de fusion comportementale R6 est associative. ◇

Preuve de l'associativité de la règle de fusion d'un comportement réactif concurrentiel (R7) :

La fusion comportementale d'un comportement réactif concurrentiel avec un autre comportement réactif est décrite par la règle de sémantique naturelle suivante :

$$\frac{(m_1 \rightarrow cr_2, m_2 \rightarrow cr_3) : m \rightarrow cr_{2.3}}{(m_1 \rightarrow \text{concurrency}(cr_1, cr_2), m_2 \rightarrow cr_3) : m \rightarrow \text{concurrency}(s(cr_1), cr_{2.3}) \mid m_1 \in cr_2}$$

Soit $R_1 : m_1 \rightarrow \text{concurrency}(cr_1, cr_2)$ et $(R_2, R_3) \in \mathcal{G}_{-E}^2$.

Grâce à l'hypothèse de récurrence, nous avons :

$$\varphi(\varphi(m_1 \rightarrow cr_2, R_2), R_3) \equiv \varphi(m_1 \rightarrow cr_2, \varphi(R_2, R_3))$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned} \varphi(\varphi(R_1, R_2), R_3) &\equiv \varphi(m \rightarrow \text{concurrency}(s(cr_1), \varphi(m_1 \rightarrow cr_2, R_2)), R_3) \\ &\equiv m \rightarrow \text{concurrency}(s(cr_1), \varphi(\varphi(m_1 \rightarrow cr_2, R_2), R_3)) \\ \text{et } \varphi(R_1, \varphi(R_2, R_3)) &\equiv \varphi(m_1 \rightarrow \text{concurrency}(cr_1, cr_2), \varphi(R_2, R_3)) \\ &\equiv m \rightarrow \text{concurrency}(s(cr_1), \varphi(m_1 \rightarrow cr_2, \varphi(R_2, R_3))) \\ &\equiv m \rightarrow \text{concurrency}(s(cr_1), \varphi(\varphi(m_1 \rightarrow cr_2, R_2), R_3)) \\ &\equiv \varphi(\varphi(R_1, R_2), R_3) \end{aligned}$$

Ainsi la règle de fusion comportementale R7 est associative. \diamond

Preuve de l'associativité de la règle de fusion d'un comportement réactif séquentiel (R8) :

La fusion comportementale de deux comportements réactifs séquentiels est définie par la fusion comportementale de l'un des deux comportements réactifs séquentiels avec la transformation de l'autre comportement réactif séquentiel en un comportement réactif concurrentiel.

Or la fusion comportementale d'un comportement réactif concurrentiel avec un autre comportement réactif est associative (preuve ci-dessus). Ainsi la règle de fusion comportementale R8 est associative. \diamond

Preuve de l'associativité de la règle de fusion d'un comportement réactif conditionnel (R9) :

La fusion comportementale d'un comportement réactif conditionnel avec un autre comportement réactif est notamment décrite par la règle de sémantique naturelle suivante :

$$\frac{\begin{array}{l} (m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) : m \rightarrow cr_{1,3} \\ (m_1 \rightarrow cr_2, m_2 \rightarrow cr_3) : m \rightarrow cr_{2,3} \end{array}}{(m_1 \rightarrow \text{ifThenElse}(c, cr_1, cr_2), m_2 \rightarrow cr_3) : m \rightarrow \text{ifThenElse}(s(c), cr_{1,3}, cr_{2,3})}$$

Soit $R_1 : m_1 \rightarrow \text{ifThenElse}(c, cr_1, cr_2)$ et $(R_2, R_3) \in \mathcal{G}_{-E}^2$.

Grâce à l'hypothèse de récurrence, nous avons :

$$\begin{aligned} \varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3) &\equiv \varphi(m_1 \rightarrow cr_1, \varphi(R_2, R_3)) \\ \varphi(\varphi(m_1 \rightarrow cr_2, R_2), R_3) &\equiv \varphi(m_1 \rightarrow cr_2, \varphi(R_2, R_3)) \end{aligned}$$

De ce fait, nous pouvons en déduire que :

$$\begin{aligned} \varphi(\varphi(R_1, R_2), R_3) &\equiv \varphi(m \rightarrow \text{ifThenElse}(s(c), \varphi(m_1 \rightarrow cr_1, R_2), \varphi(m_1 \rightarrow cr_2, R_2)), R_3) \\ &\equiv m \rightarrow \text{ifThenElse}(s(c), \varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3), \varphi(\varphi(m_1 \rightarrow cr_2, R_2), R_3)) \\ \text{et } \varphi(R_1, \varphi(R_2, R_3)) &\equiv \varphi(m_1 \rightarrow \text{ifThenElse}(c, cr_1, cr_2), \varphi(R_2, R_3)) \\ &\equiv m \rightarrow \text{ifThenElse}(s(c), \varphi(m_1 \rightarrow cr_1, \varphi(R_2, R_3)), \varphi(m_1 \rightarrow cr_2, \varphi(R_2, R_3))) \\ &\equiv m \rightarrow \text{ifThenElse}(s(c), \varphi(\varphi(m_1 \rightarrow cr_1, R_2), R_3), \varphi(\varphi(m_1 \rightarrow cr_2, R_2), R_3)) \\ &\equiv \varphi(\varphi(R_1, R_2), R_3) \end{aligned}$$

La preuve de la propriété d'associativité concernant la seconde règle de fusion comportementale d'un comportement réactif conditionnel avec un autre comportement réactif est en tout point identique et n'est donc pas présentée.

Ainsi la règle de fusion comportementale R9 est associative. \diamond

Preuve de l'associativité de la règle de fusion d'un comportement réactif de traitement des exceptions (R10) :

La fusion comportementale de deux comportements réactifs de traitements de la même exception est décrite par la règle de sémantique naturelle suivante :

$$\frac{\begin{array}{l} (m_1 \rightarrow cr_1, m_2 \rightarrow cr_3) : m \rightarrow cr_{1,3} \\ (m_1 \rightarrow cr_2, m_2 \rightarrow cr_4) : m \rightarrow cr_{2,4} \end{array}}{(m_1 \rightarrow \text{try}(cr_1, e, cr_2), m_2 \rightarrow \text{try}(cr_3, e, cr_4)) : m \rightarrow \text{try}(cr_{1,3}, e, cr_{2,4})}$$

Soit $R_1 : m_1 \rightarrow \text{try}(cr_1, e, cr_2)$, $R_2 : m_2 \rightarrow \text{try}(cr_3, e, cr_4)$, et $R_3 : m_3 \rightarrow \text{try}(cr_5, e, cr_6)$.

Grâce à l'hypothèse de récurrence, nous avons :

$$\begin{aligned} \varphi(m_1 \rightarrow cr_1, \varphi(m_2 \rightarrow cr_3, m_3 \rightarrow cr_5)) &\equiv \varphi(\varphi(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3), m_3 \rightarrow cr_5) \\ \varphi(m_1 \rightarrow cr_2, \varphi(m_2 \rightarrow cr_4, m_3 \rightarrow cr_6)) &= cr_{(2,4),6} \equiv \varphi(\varphi(m_1 \rightarrow cr_2, m_2 \rightarrow cr_4), m_3 \rightarrow cr_6) = cr_{2,(4,6)} \end{aligned}$$

De ce fait nous pouvons en déduire que :

$$\begin{aligned}\varphi(\varphi(R_1, R_2), R_3) &\equiv \varphi(m \rightarrow \text{try}(\varphi(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3), e, cr_{2.4}), R_3) \\ &\equiv m \rightarrow \text{try}(\varphi(\varphi(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3), m_3 \rightarrow cr_5), e, cr_{(2.4).6})\end{aligned}$$

$$\begin{aligned}\text{et } \varphi(R_1, \varphi(R_2, R_3)) &\equiv \varphi(R_1, m \rightarrow \text{try}(\varphi(m_2 \rightarrow cr_3, m_3 \rightarrow cr_5), e, cr_{4.6})) \\ &\equiv m \rightarrow \text{try}(\varphi(m_1 \rightarrow cr_1, \varphi(m_2 \rightarrow cr_3, m_3 \rightarrow cr_5)), e, cr_{2.(4.6)}) \\ &\equiv m \rightarrow \text{try}(\varphi(\varphi(m_1 \rightarrow cr_1, m_2 \rightarrow cr_3), m_3 \rightarrow cr_5), e, cr_{(2.4).6}) \\ &\equiv \varphi(\varphi(R_1, R_2), R_3)\end{aligned}$$

Les preuves de la propriété d'associativité de la fonction de fusion comportementale concernant les autres règles de sémantique naturelle de la règle de fusion R10 sont en tout point similaire à la preuve de l'associativité de la fonction de fusion comportementales présentées ci-dessus. Nous ne les détaillons donc pas.

Ainsi la règle de fusion comportementale R10 est associative. ◇

Nous venons par conséquent de vérifier que toutes les règles de fusion comportementale vérifiaient la propriété d'associativité :

$$\forall (r_1, r_2, r_3) \in \mathcal{R}^3 \quad \varphi(\varphi(r_1, r_2), r_3) \equiv \varphi(r_1, \varphi(r_2, r_3))$$

□

7 Conclusion

Dans ce chapitre, nous avons présenté :

- Le mécanisme de combinaison des comportements réactifs, la fusion comportementale, permettant de générer un comportement réactif exécutable disposant de la même sémantique d'exécution que l'ensemble des comportements réactifs combinés. Ce mécanisme est utilisée lors de l'héritage des schémas d'interactions et lors de l'exécution des comportements réactifs associés à un message déclencheur.
- Les deux propriétés essentielles de la fusion comportementale lui conférant son fort pouvoir d'expression de la sémantique des comportements réactifs : la commutativité et l'associativité.

Apports de notre modèle à interactions distribuées

Les interactions disposent d'un niveau d'abstraction et de conception similaire à celui des objets grâce à la notion de schéma d'interactions. Ainsi les interactions sont des entités de même statut et importance que les objets. Elles sont *indépendantes* et *autonomes* vis-à-vis des objets interagissants sur lesquels elle s'appliquent. Ceci permet notamment l'expression des interactions au niveau des instances sans pour autant impliquer que toutes les instances d'une classe soient concernées.

De plus, cette séparation logique entre les objets et les interactions permet de préserver le principe d'encapsulation, les comportements réactifs étant exclusivement exprimés sur l'interface des objets participants. Cette séparation permet de fortement réduire la complexité des objets interagissants et améliore la réutilisation et la maintenance de ces objets. La mise en œuvre des interactions sous la forme d'objets (étendus) permet de les doter d'informations propres (aussi bien au niveau structurel que fonctionnel) et de définir une interaction sur une autre interaction.

Les interactions sont n-aires et non orientées : chaque participant pouvant aussi bien être l'instigateur du déclenchement d'un comportement réactif (par le biais de l'une de ses méthodes) que l'un des acteurs de l'exécution d'un comportement réactif. De plus, afin d'améliorer l'expressivité des comportements réactifs, ces derniers sont réifiés sous la forme de règles d'interactions. L'interprétation de ces règles est spécifiée par des opérateurs. Chaque opérateur, également réifié, définit le sens des actions et du contrôle des messages. Ainsi l'expression des comportements réactifs est adaptable, par la modification de la sémantique des opérateurs, et extensible, par l'ajout de nouveaux opérateurs ⁶.

⁶. Ceci implique bien évidemment de vérifier que la sémantique définie pour ces nouveaux opérateurs conserve toujours les propriétés de commutativité et d'associativité de la fonction de fusion comportementale.

Mise en œuvre du modèle

Notre modèle à interactions distribuées est basé d'une part sur le contrôle de l'envoi de messages (puisque sa sémantique est modifiée par la prise en compte des comportements réactifs) et d'autre part sur une approche dynamique du support des interactions, et notamment sur un mécanisme de découverte dynamique des schémas d'interactions.

Le modèle présenté décrit de manière idéale le support des interactions dans un environnement compilé et fortement typé. Cependant, certains aspects, comme la gestion des exceptions, sont très dépendants du langage utilisé ou difficilement réalisables. Étant conscient de ces limitations imposées par la mise en œuvre, elles ont été prises en compte afin que les concepts primordiaux du modèle restent valides dans toute mise en œuvre. Nous pensons donc que le modèle à interactions distribuées présenté peut être mis en œuvre dans différents langages à objets compilés et typés.

Références

- [AOP01] Overview of aspect-oriented programming, March 2001.
- [BJR97] G. Booch, I. Jacobson, and J. Rumbaugh. Unified modeling language (uml), January 1997. *Rational Software Corporation*, version 1.0.
- [Des88] T. Despeyroux. Typol : a formalism to implement natural semantics. Technical Report 94, INRIA, 1988.
- [INR89] INRIA. *The CENTAUR User's Manual*. Sophia-Antipolis, France, 1989.
- [Jau00] O. Jautzy. *Intégration de source de données hétérogènes : Une Approche Langage*. PhD thesis, École Nationale des Ponts et Chaussées, March 2000.
- [KLM83] G. Kahn, B. Lang, and B. Mélése. Metal : a formalism to specify formalisms. In *Science of Computer Programming*, volume 3, North-Holland, 1983.
- [KLM⁺97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceeding of ECOOP'97*, number 1241 in LNCS, pages 220–242, Jyväskylä (Finlande), June 1997. Springer Verlag.
- [San95] C. Souza Dos Santos. *Un Mécanisme de Vues pour les Systèmes de Gestion de Bases de Données Objet*. PhD thesis, Université de Paris-Sud, Paris (France), November 1995.