RÈGLES DE COHÉRENCE UML 2.0

HUGUES MALGOUYRES
JEAN-PIERRE SEUMA-VIDAL
GILLES MOTET

Version 1.1

Laboratoire d'Étude des Systèmes Informatiques et Automatiques,
Institut National des Sciences Appliquées de Toulouse,
Département Génie Électrique,
135, avenue de Rangueil, 31077 Toulouse cedex 4.

courriel : malgouyres@insa-toulouse.fr
jpseuma@insa-toulouse.fr
motet@insa-toulouse.fr
url : http ://www.lesia.insa-tlse.fr

Table des matières

Chapitre 1

Introduction

1.1 Objectif

Incohérences du langage UML 2.0

Les méthodes graphiques offrent un moyen prometteur pour maîtriser la complexité des logiciels en offrant une description en plusieurs "vues". La langage UML permet une telle description et est adopté comme un standard industriel de fait. Cependant, les vues de description d'un logiciel ne sont pas disjointes car elles contiennent des informations redondantes ou complémentaires et il est donc essentiel de s'assurer de leur cohérence.

Dans la suite de notre document, une incohérence est définie par :

incohérence	*Une incohérence est la violation d'une propriété associée au langage UML qui doit être respectée par tout modèle UML.*

Une incohérence est la conséquence de constructions redondantes ou complémentaires incompatibles entre-elles. Les termes de la compatibilité sont exprimés par des « règles de cohérence ».

Le but de ce document est d'énumérer l'ensemble des règles de cohérence associées au langage UML et que doivent respecter tout modèle UML d'un système quelconque. Il se base sur la spécification d'UML 2.0 du 02/08/2003 (adopted specification) [7].

Remarquons que l'absence d'incohérence ne prouve pas le bon fonctionnement d'un système mais que la présence d'une incohérence a de grandes chances de déboucher sur un système défectueux. De plus, l'expérience montre que de très nombreuses fautes de modélisation sont perceptibles à travers la détection d'incohérences.

Les incohérences reposent sur la sémantique de vérification des constructions du langage UML, c'est-à-dire sur la bonne façon de constituer un modèle UML, et non sur leur sémantique opérationnelle, c'est-à-dire sur leurs apports descriptifs dans un modèle. Nous nous intéressons donc aux règles de cohérence de l'emploi d'une construction et non à la fonction qu'elle remplit.

Voici un exemple qui illustre la différence de ces sémantiques dans le cas d'une affectation écrite en langage ADA. La sémantique opérationnelle associée à l'instruction « `A := B ;` » pourrait être « *on copie la valeur de l'expression B dans la variable A* »

alors que la sémantique de vérification serait « *le type de la variable A et le type de l'expression B doivent être identiques* ». Cette propriété formule un usage cohérent de l'affectation vis-à-vis de la variable affectée et de l'expression dont la valeur sera affectée.

Ce document est une étude de la sémantique de vérification de chaque élément de base du langage UML, de chaque relation entre ces éléments, de chaque diagramme décrit par le langage UML et enfin d'un modèle dans son ensemble en tenant compte de la cohérence inter-diagramme.

Type des incohérences considérées

Nous nous plaçons dans le cadre de l'expression des modèles complets, par opposition aux vues incomplètes qui peuvent être utiles lors d'une itération du processus de développement.

La figure 1.1 tirée de [2] présente une classification des types d'incohérences. Dans cette classification, nos travaux se situent dans les couches 2 et 3, c'est-à-dire que nous ne traitons que des incohérences sur les éléments et les relations entre ces éléments (niveau 3) et sur les diagrammes et les relations entre diagrammes (niveau 2).

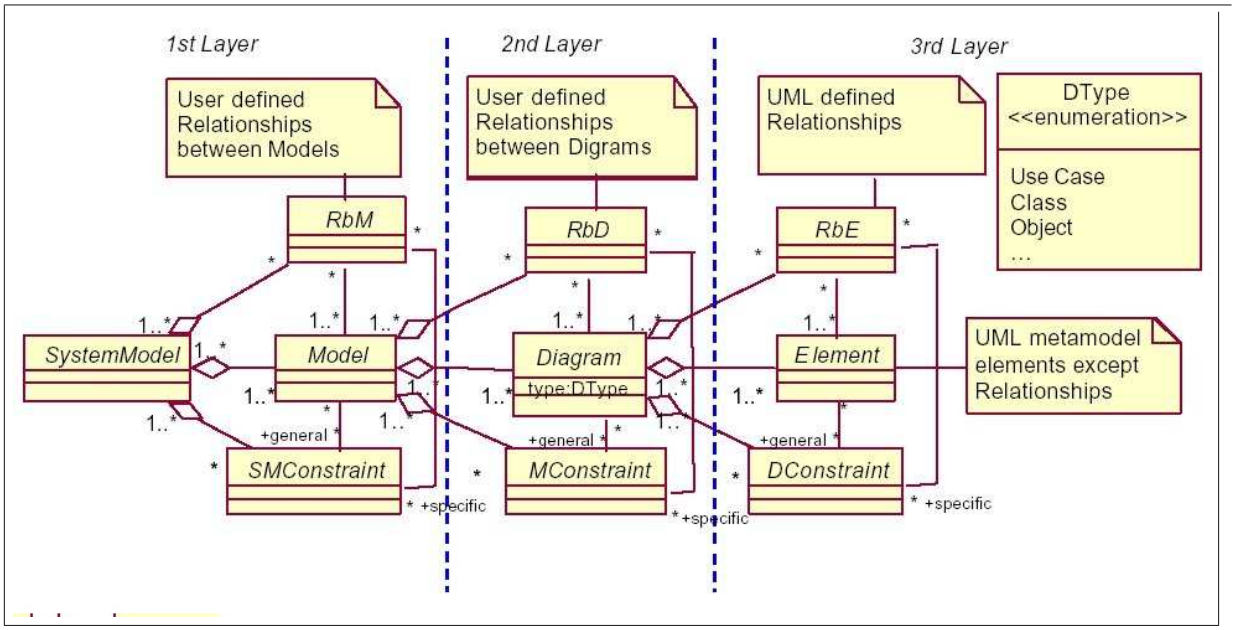

FIG. 1.1 – Type d'incohérences

Remarquons cependant que certaines des règles énoncées pourraient aussi être classées dans des relations entre modèles. C'est le cas de règles qui mettent en jeu des constructions décrivant le même système avec des niveaux de raffinement différents (cf. la règle 251 par exemple).

1.2 Structure du document

Structuration en diagramme

Les règles de cohérence que doivent respecter les modèles UML ont été structurées afin de faciliter leur utilisation lors d'analyse de modèles particuliers (par exemple par des revues). Les différentes parties abordent successivement les diagrammes suivants :

1. diagramme de classes,
2. diagramme de composants,
3. diagramme de structures composites,
4. diagramme de déploiements,
5. diagramme d'activités,
6. diagramme d'interactions,
7. diagramme de machines à états,
8. diagramme des cas d'utilisation.

Une dernière partie aborde les aspects inter-diagrammes.

Structuration intra-diagramme

Chaque partie est composée de 3 chapitres qui traitent respectivement d'un point de vue sur les constituants du diagramme étudié :

1. Les éléments de base exprimés dans le diagramme et donc les règles qu'ils doivent respecter en les considérant séparément les uns des autres.
2. Les relations couplant ces éléments de base exprimées dans le diagramme et donc les règles que ces relations doivent respecter.
3. Les règles concernant globalement le diagramme étudié qui expriment des contraintes sur les usages conjoints de plusieurs éléments et de plusieurs relations dans ce diagramme.

Canevas pour chaque construction considérée

Chacun de ces chapitres est ensuite découpé en sections qui correspondent à un élément, une relation ou un diagramme. Dans chacune de ces sections, nous adopterons le même canevas d'étude :

- en premier lieu, nous présenterons le « **contexte** », c'est-à-dire les circonstances dans lesquelles une erreur peut être commise lors de la modélisation UML ;
- ensuite, nous définirons les « **règles de cohérence** » (*consistency rules* en anglais) qui expriment par intention la sémantique de vérification que l'élément doit respecter ; il peut exister une ou plusieurs règles qui seront numérotées ;
- lorsque cela sera possible, nous donnerons des « **guides** » de style pour faciliter la détection des erreurs ;
- la rubrique « **justification** » apporte une justification pour chaque guide, les guides et les justifications sont numérotés ce qui permet de faire le lien entre un guide et sa justification.

Typologie des règles

Enfin, deux types d'informations sont associés aux règles :

- l'origine de la règle qui peut être :

1. une contrainte directement tirée de la norme auquel cas le marqueur `[[7] p.XXX]` suit la règle ;
2. directement déduite du méta-modèle auquel cas le marqueur `[Règle dérivée du méta-modèle]` suit la règle ;
3. une nouvelle règle, auquel cas le marqueur `[Nouvelle règle]` suit la règle ;
4. une règle tirée de la littérature auquel cas la référence du document et une page éventuelle suit la règle (marqueur de type `[tirée de [7] p.XXX]`), ce marqueur est également utilisé pour les règles provenant de la spécification d'UML 2.0 mais qui n'y apparaissent pas en tant que contrainte.

– le niveau de d'écriture et d'application de la règle :

1. le premier niveau correspond aux règles pouvant être écrites au niveau langage (ou méta-modèle) et s'appliquant sur le méta-modèle uniquement, ces règles sont marquées par `[Règle sur le méta-modèle]` ;
2. le deuxième niveau correspond aux règles pouvant être écrites au niveau langage et qui contraignent les modèles, ces règles sont les plus intéressantes et les plus nombreuses c'est pourquoi aucun marqueur ne leur est associé ;
3. le dernier niveau correspond aux règles ne pouvant pas être écrites au niveau méta, ceci est dû au fait qu'aucune méta-classe ne correspond à l'élément graphique sur lequel s'applique la règle, ces règles sont identifiées par `[Règle utilisateur]`.

Un exemple avec le langage ada

Considérant que les langages de programmation sont mieux connus que ceux de modélisation et afin de bien faire comprendre le contenu de chaque section de ce document, nous illustrons ici le canevas présenté précédemment pour une affectation d'une variable par une expression en langage ADA (`A := B ;`).

Contexte Le langage ADA permet d'affecter la valeur d'une expression à une variable.

Règle de cohérence

• Règle 1 : Le type de la variable A et le type de l'expression B doivent être identiques.

Guide

• Guide 1 : Nous conseillons de créer des types différents pour chaque concept et en particulier de ne pas utiliser les types prédéfinis mais de les redéfinir (opérateur `new`).

Justification

• Justification 1 : Ceci permet de détecter toute affectation involontaire entre éléments qui expriment des notions différentes. Par exemple, ceci permettrait de détecter une affectation entre une expression qui exprime une température et une variable qui représente une vitesse même si ces deux éléments sont codés par des flottants.

Remarques finales

Certaines règles peuvent être difficiles à comprendre. Afin d'éviter au maximum toute mauvaise interprétation de ces règles, un travail d'illustration est en cours. Ce travail consiste pour chacune des règles à donner un modèle qui respecte et un modèle qui enfreint la règle considérée.

Un lexique Anglais/Français regroupe en fin de document l'ensemble des termes employés.

Des notes de bas de page présentes dans le texte ne doivent pas être considérées par le lecteur, mais sont des notes qui servent de base de réflexion aux auteurs.

Première partie

Diagramme de classes

Chapitre 2

Éléments

2.1 Element

Contexte `Element` est une méta-classe abstraite qui représente un constituant du modèle. En tant que tel, un élément a la capacité de contenir d'autres éléments.

Règles de cohérence

• Règle 2 : Un élément ne peut pas directement ou indirectement se contenir lui-même. [[7] p.30]

• Règle 3 : Tous les éléments excepté les paquetages doivent avoir un possesseur. [[7] p.30]

Remarque Tous les éléments d'UML doivent obligatoirement être contenus dans un autre élément sauf les paquetages.

2.2 Namespace

Contexte Un espace de nommage (*namespace* en anglais) est un élément qui peut contenir en ensemble d'éléments nommés pouvant être identifiés par leurs noms.

Un espace de nommage a la capacité d'importer des éléments individuels afin que ceux-ci puissent être référencés sans utilisation du nom qualifié à l'intérieur de l'espace de nommage importateur. Dans le cas de conflits de noms, il faut avoir recours aux alias.

L'annexe A.4 montre la hiérarchie d'héritage de la méta-classe `Namespace` et donc l'ensemble des éléments qui sont considérés comme étant des espaces de nommage.

Règles de cohérence

• Règle 4 : Tous les membres d'un espace de nommage doivent pouvoir être distingués. [[7] p.36]

Remarque Par défaut, un élément A se distingue d'un élément B si A n'est pas du même type que B, un sous-type de B, ou, si A et B n'ont pas le même nom. Cette définition du qualificatif « distingué » peut être redéfinie, c'est notamment le cas pour les opérations qui sont distinguables en fonction de leur signature.

• Règle 5 : Les membres importés d'un espace de nommage dérivent des relations d'importation d'éléments (cf. section 3.1) et d'importation de paquetage (cf. section 3.2). [[7] p.36] [Règle sur le méta-modèle]

Remarque Dans la méta-modèle la relation de composition entre un élément nommé et un espace de nommage est spécifiée par le rôle `ownedMember`.

2.3 Named Element

Contexte [1] Un élément nommé (*named element* en anglais) est un élément qui peut avoir un nom. Le nom est utilisé pour identifier l'élément. Un élément nommé peut avoir un nom qualifié (*qualified name* en anglais) qui lui permet d'être identifié de façon non ambiguë dans une hiérarchie d'espaces de nommage imbriqués.

Les éléments nommés peuvent être contenus dans un espace de nommage (cf. section 2.2).

Règles de cohérence

• Règle 6 : Si un élément nommé n'est pas inclus dans un espace de nommage il n'a pas de visibilité. [[7] p.34] [Règle sur le méta-modèle]

• Règle 7 : Si un élément nommé n'a pas de nom ou qu'au moins un de ses espaces de nommage qui le contient n'a pas de nom, alors l'élément n'a pas de nom qualifié. [[7] p.34] [Règle sur le méta-modèle]

• Règle 8 : Quand l'élément et tous les espaces de nommage qui le contiennent sont nommés, le nom qualifié est construit à partir des noms des espaces de nommage successifs. [[7] p.34] [Règle sur le méta-modèle]

Remarque Le nom qualifié est construit de la façon suivante :

```
namespace1 :: ... :: namespaceN :: element-name
```

Remarque Les règles 9 à 11 s'appliquent lors de l'utilisation des noms qualifiés dans les modèles.

• Règle 9 : Soit `namespace1 : : ... : : namespaceN : : element-name` un nom qualifié, l'espace de nommage `namespace1` doit être l'espace de nommage le plus général contenant indirectement (ou non) `element-name`.[Règle dérivée du méta-modèle]

• Règle 10 : Soit `namespace1 : : ... : : namespaceN : : element-name` un nom qualifié, pour tout P appartenant à 2..N, `namespaceP` doit être inclus dans `namespaceP-1`. [Règle dérivée du méta-modèle]

• Règle 11 : Soit `namespace1 : : ... : : namespaceN : : element-name` un nom qualifié, l'élément `element-name` doit appartenir à l'espace de nommage `namespaceN`. [Règle dérivée du méta-modèle]

[2]

• Règle 12 : Un élément ne peut appartenir qu'à un seul espace de nommage en même temps.[Règle dérivée du méta-modèle]

Remarque La figure 2.1 montre un exemple de non respect de la règle 12. En effet, cette règle implique que tout paquetage ne peut être contenu que par un autre paquetage.

[1] faire une section namedElement dans la section template pour mettre les règles de la page 560 de la spec

[2] Guide de prévention : Le nom d'un élément ne doit pas correspondre à un mot prédéfini du langage UML. **Remarque** Les mots prédéfinis du langage sont les stéréotypes, les contraintes prédéfinies, les propriétés et les mots clés. Se référer à l'annexe B.2 pour avoir la liste de ces mots et l'élément du langage sur lequel ils peuvent s'appliquer.

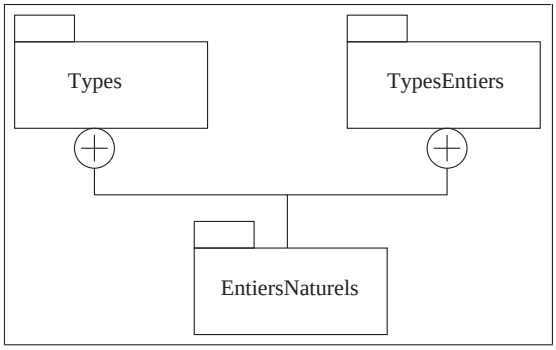

FIG. 2.1 – Paquetage appartenant à plusieurs paquetages

[3]

2.4 Comment

Contexte Un commentaire (*comment* en anglais) est une annotation textuelle qui peut être attachée à un ensemble d'éléments.

Un commentaire n'ajoute pas forcément de sémantique aux éléments annotés. Dans ce cas, il représente des informations utiles aux lecteurs du modèle.

Règles de cohérence

Pas de règles supplémentaires.

2.5 Multiplicity

Contexte La multiplicité (*multiplicity* en anglais) spécifie les valeurs de cardinalité possibles pour un ensemble d'éléments. La cardinalité d'un ensemble est le nombre d'éléments contenus dans cet ensemble.

Règles de cohérence

• RÈGLE 13 : L'expression d'une multiplicité doit suivre la syntaxe :[[7] p.43]

```
multiplicity ::= <multiplicity_range> ['{'<order_designator>'}']
                                      ['{'<uniqueness_designator>'}']
multiplicity_range ::= [ lower  ..  ] upper
lower ::= integer | value_specification
upper ::= unlimited_natural |  *  | value_specification
<order_designator> ::= ordered | unordered
<uniqueness_designator> ::= unique | nonunique
```

[3] RÈGLE 4 du doc commun : guide prévention ? il est conseillé d'utiliser des caractères imprimables pour nommer les éléments

où `unlimited_natural` doit être un entier naturel.

Remarque La syntaxe interdit d'associer à la même multiplicité les propriétés `{ordered}` et `{unordered}` et les propriétés `{unique}` et `{nonunique}`.

Remarque Le terme optionnel `'{'<uniqueness_designator>'}'` n'est pas présent dans la norme et a été rajouté.

• Règle 14 : Le champ `value_specification` doit être évaluable et être un spécification de valeur (cf. section 2.6) de type entier. [Nouvelle règle]

• Règle 15 : Une multiplicité doit définir au moins une cardinalité valide (c'est-à-dire que `upper>0`). [[7] p.41]

• Règle 16 : La borne inférieure doit être un entier non négatif. [[7] p.41]

• Règle 17 : La borne supérieure doit être supérieure ou égale à la borne inférieure. [[7] p.41]

• Règle 18 : Si des valeurs non littérales sont utilisées pour décrire les bornes des multiplicités, leur évaluation ne doit pas avoir d'effet de bord. [[7] p.41]

• Règle 19 : Si des valeurs non littérales sont utilisées pour décrire les bornes des multiplicités, alors leurs spécifications doivent être des constantes. [[7] p.41]

• Règle 20 : Le méta-attribut dérivé `lower` doit être égal à la borne inférieure. [[7] p.41] [Règle sur le méta-modèle]

• Règle 21 : Le méta-attribut dérivé `upper` doit être égal à la borne supérieure. [[7] p.41] [Règle sur le méta-modèle]

2.6 Value Specification and Expression

Contexte En UML, une spécification de valeur (*value specification* en anglais) peut être évaluable ou non, dans ce dernier cas, on parle d'expression opaque (*opaque expression* en anglais).

Une expression est évaluée selon un structure arborescente où les nœuds sont des opérateurs et les branches des opérandes. L'évaluation d'une expression contenant une expression opaque (comme opérande) est laissée à la responsabilité de l'outil. Le langage dans lequel est écrit une expression opaque peut être spécifié. Les expressions OCL sont considérées comme des expressions opaques.

Règles de cohérence

• Règle 22 : Dans le cas où aucune expression opaque n'apparaît dans la formulation de l'expression, l'expression doit être syntaxiquement correcte. [Nouvelle règle]

Guide

• Guide 2 : Nous conseillons de ne pas utiliser d'expression opaque quand cela est possible.

Justification

• Justification 2 : Lorsque le guide 2 est suivi, la règle 22 peut être totalement vérifiée.

2.7 Constraint

Contexte Une contrainte est une condition ou une restriction exprimée en langage naturel ou en langage compréhensible par l'AGL (Atelier de Génie Logciel) qui ajoute des informations sémantiques à l'élément.

Règles de cohérence

• Règle 23 : L'expression d'une contrainte doit suivre la syntaxe suivante :[[7] p.55]

```
constraint ::= '{'[<name>':'] <boolean expression>'}'
```

• Règle 24 : Les valeurs spécifiées par une contrainte doivent être évaluées à vrai. [[7] p.54]

• Règle 25 : L'évaluation d'une contrainte ne doit pas avoir d'effets de bord. [[7] p.54]

• Règle 26 : Une contrainte ne peut pas s'appliquer à elle-même. [[7] p.54]

• Règle 27 : Une contrainte doit s'appliquer à l'élément pour lequel elle a été définie. [Nouvelle règle]

Remarque Une liste des contraintes prédéfinies est disponible en annexe B.3.

• Règle 28 : L'élément qui contient la contrainte doit avoir accès aux éléments mis en jeu par la contrainte. [tirée de [7] p.54]

2.8 Keyword

Contexte La notation UML fait usage de mots clefs (*keywords* en anglais) pour faire la distinction entre des variations d'un même motif graphique. Ceci permet de décrire différentes classes du méta-modèle avec le même motif graphique. En règle générale, la méta-classe d'un élément représentée avec un motif et un mot clé est une spécialisation de la méta-classe représentée avec uniquement le motif graphique.

Remarque « Keyword » ne fait pas partie des méta-classes du méta-modèle.

Règles de cohérence

• Règle 29 : Toute utilisation de mot clé suppose que le mot clé soit prédéfini. [Nouvelle règle][Règle utilisateur]

• Règle 30 : Toute utilisation d'un mot clé prédéfini doit être réalisée sur l'élément adéquat. [Nouvelle règle][Règle utilisateur]

Remarque La liste des mots clés prédéfinis ainsi que les éléments sur lesquels ils peuvent être utilisés se situe en annexe B.2.

• Règle 31 : Lorsqu'un mot clé est associé à un élément n'ayant pas de motif graphique spécial, le mot clé doit figurer à toutes les apparitions de l'élément dans le modèle. [Nouvelle règle][Règle utilisateur]

Remarque Cette règle est valide seulement si l'élément ne dispose pas d'un motif graphique spécial. Dans ce cas et si la représentation graphique spéciale est utilisée le mot clé n'apparaît pas nécessairement.

2.9 Element Properties

Contexte Dans UML, chaque élément possède un ensemble de propriétés qui peut s'appliquer sur l'élément. Les propriétés servent à ajouter des informations supplémentaires aux éléments du modèle. Dans de nombreux cas, une propriété sert à exprimer la valeur d'un attribut du méta-modèle pour l'élément concerné.

Remarque « Element Properties » ne fait pas partie des méta-classes du méta-modèle.

Règles de cohérence

• Règle 32 : Excepté les propriétés `subsets` et `redefines`, les propriétés doivent respecter la syntaxe : [Nouvelle règle]

```
property ::= {name ['=' value]}
```

où :

- name est le nom de la propriété (le marqueur de la propriété) ;
- le symbole '=' est un séparateur ;
- value est la valeur (celle du marqueur) à laquelle doit se trouver la propriété ; ce champ est optionnel.

• Règle 33 : Le champ `name` doit correspondre à une propriété prédéfinie du langage UML. [Nouvelle règle][Règle utilisateur]

• Règle 34 : Toute propriété doit s'appliquer sur un élément pour lequel elle a été définie. [Nouvelle règle][Règle utilisateur]

Remarque La liste des propriétés et des éléments sur lesquels elles peuvent s'appliquer est présentée en annexe B.1.

• Règle 35 : Si le champ `value` est présent, celui-ci doit exprimer une valeur qui est compatible avec les valeurs que peut prendre la propriété. [Nouvelle règle][Règle utilisateur]

Remarque Par exemple pour une propriété booléenne la valeur doit être vrai ou faux.

• Règle 36 : Lorsque le champ `value` n'est pas présent celui-ci est pris par défaut à vrai. Il faut donc que la propriété soit de type booléen. [Nouvelle règle][Règle utilisateur]

• Règle 37 : Chaque propriété associée à un élément du modèle doit être compatible avec les autres propriétés associées à cet élément. [Nouvelle règle][Règle utilisateur]

Remarque Voici la liste des propriétés qui sont incompatibles lorsqu'elles s'appliquent sur le même élément :

- `{subsets prop-name1}` et `{redefines prop-name2}` si prop-name1=prop-name2 ;
- `{unrestricted}` et `{readOnly}` ;
- `{ordered}` et `{bag}` ;
- `{sequence}` (ou `{seq}`) et `{bag}` ;
- `{property-name=value1}` et `{property-name=value2}` si value1≠value2. [4]

[5]

[4] reprendre la liste des propriétés lorsqu'elle sera finie

[5] LA section sur les stéréotypes est à intégrer dans profile => cf. fichier source (Sec-element-classes-commun.tex) :

2.10 Instance Specification

Contexte Une spécification d'instance (*instance specification* en anglais) représente l'existence d'une entité dans le système modélisé.

La représentation de l'instance dépend du type de son classificateur : un objet (*object* en anglais) pour une classe, un lien (*link* en anglais) pour une association, etc.

Lorsque l'on fait apparaître des instances, il est possible d'attribuer des valeurs aux caractéristiques structurelles. Il est possible de ne représenter qu'un sous-ensemble des caractéristiques structurelles. En UML, la valeur de la caractéristique structurelle de l'instance est appelée « slot ». Le type de la caractéristique structurelle peut être montré lors de son affectation.

Il est possible de spécifier le nom d'une instance et le ou les classificateurs qui typent l'instance en question.

Une spécification d'instance est décrite en utilisant le même motif que le classificateur. À la place du nom du classificateur doit apparaître une chaîne de caractère.

Règles de cohérence

• Règle 38 : La chaîne de caractère identifiant la spécification d'instance doit respecter la syntaxe : [[7] p.59]

```
instance ::= [InstanceName] [':' ClassifierName]['=' ValueSpecification]
```

[6]

• Règle 39 : Chaque slot doit définir une caractéristique structurelle contenue (directement contenue ou héritée) par un classificateur de la spécification d'instance. [[7] p.58] [7]

Remarque En règle générale, une spécification d'instance est typée par un seul classificateur.

• Règle 40 : Une caractéristique structurelle ne peut se voir attribuer de valeur qu'une seule fois dans la spécification d'instance. [[7] p.58]

Remarque Ceci tient compte du fait qu'une même caractéristique peut être héritée de plusieurs classificateurs.

• Règle 41 : La caractéristique structurelle d'une instance et la valeur qui lui est éventuellement donnée doivent être du même type. [Nouvelle règle]

• Règle 42 : Si le type de la caractéristique structurelle est indiqué dans la spécification de l'instance, il doit correspondre au type de cette caractéristique structurelle dans le classificateur. [Nouvelle règle][Règle utilisateur]

• Règle 43 : La multiplicité des caractéristiques structurelles dans le classificateur doit être respectée par le nombre de caractéristiques structurelles dans la spécification d'instance. [Nouvelle règle]

• Règle lien 1 : Le ou les classificateurs types de le spécification d'instance (s'ils sont spécifiés) peuvent être référencés par leur nom qualifié. Ces noms qualifiés doivent respecter les règles 9 à 11.

Remarque Le type d'une instance est forcément un classificateur.

[6] Ce dernier champ optionnel n'est pas présent dans la norme.

[7] La définition OCL de la norme me semble mauvaise

Guide

• Guide 3 : D'un point de vue purement syntaxique le champ `ClassifierName` peut être omis mais il est vivement conseillé de le faire figurer.

• Guide 4 : Le champ `InstanceName` peut être omis. Il est cependant conseillé de faire figurer le nom de l'objet.

Justification

• Justification 3 : Nous conseillons de faire apparaître le champ `ClassifierName` car ceci permet de vérifier qu'une instance est utilisée conformément à la description qu'en fait son classificateur.

• Justification 4 : Nous conseillons de faire figurer le champ `InstanceName` afin de vérifier la cohérence de l'utilisation de l'objet dans le modèle.

[8]

[9]

2.11 Classifier

Contexte Les classificateurs (*classifier* en anglais) sont représentés par la méta-classe abstraite `Classifier` (cf. annexe A.1).

Un classificateur est un espace de nommage dont les membres peuvent être des caractéristiques. Par exemple une classe peut contenir des opérations et des attributs.

Un classificateur est également un type. Un classificateur peut avoir des relations de généralisation avec d'autres classificateurs. Les règles relatives aux relations de généralisation sont présentées en section 3.3.

Un classificateur est un élément redéfinissable. Cela signifie que lorsqu'un classificateur A est spécialisé par un classificateur B, les classificateurs contenus dans A et hérités par B peuvent être redéfinis dans B.

Règles de cohérence

Pas de règles supplémentaires à celles énumérées à la section 3.3.

2.12 Class

Contexte Une classe décrit un ensemble d'objets qui partagent les mêmes caractéristiques, contraintes et la même sémantique. Les caractéristiques d'une classe sont des attributs, des opérations et des réceptions.

Les attributs d'une classe sont représentés par des instances de la méta-classe `Property` (au sens possession, cf. section 2.15). Certains de ces attributs représentent des fins d'associations navigables.

On peut décomposer la classe en deux parties principales, appelées compartiments :

[8] Guide de prévention : Toutes les caractéristiques structurelles du classificateur qui n'ont pas de valeur par défaut doivent avoir une valeur explicite dans l'objet ? ? ?.

[9] GUIDE 11 ancien doc commun à mettre ? Dans le cas des attributs nous conseillons de sécifier le type de l'attribut car ceci permet de vérifier que celui-ci est identique au type défini dans la classe.

– un compartiment du nom ;
– un compartiment de liste qui contient une liste de chaînes de caractères ; de façon générale, il est séparé en deux compartiments qui correspondent aux attributs et aux opérations ; ce compartiment est optionnel.

UML offre la possibilité d'utiliser des points de suspension (...) pour spécifier que tous les éléments ne sont pas présents dans la liste, que d'autres attributs ou opérations sont présents mais non exposés dans cette vue particulière. [10]

Règles de cohérence

• Règle 44 : Une classe est forcément nommée. [Nouvelle règle]

• Règle 45 : Le compartiment du nom ne doit contenir qu'une seule chaîne de caractères qui correspond au nom, il ne contient donc qu'une seule chaîne de caractères qui n'est ni entre guillemets ni entre accolades. [Nouvelle règle][Règle utilisateur]

• Règle lien 2 : Lorsqu'une classe possède des propriétés, celles-ci doivent vérifier les règles exprimées dans la section 2.9.
[11]

• Règle 46 : L'emploi de points de suspension dans une liste d'attributs ou d'opérations suppose que des attributs ou des opérations non explicités dans cette vue sont présents dans la classe. D'autres vues du modèle doivent donc les faire apparaître pour que le modèle soit cohérent. [Nouvelle règle][Règle utilisateur]

• Règle 47 : Si les points de suspension sont attachés à une propriété ou à un stéréotype, les éléments non présents doivent également être attachés à la propriété ou au stéréotype lorsqu'ils apparaissent explicitement. [Nouvelle règle][Règle utilisateur]

2.13 Redefinable Element

Contexte Un élément redéfinissable (*redefinable element* en anglais) est un élément qui, quand il est défini dans le contexte d'un classificateur, peut être redéfini plus précisément dans le contexte d'un autre classificateur qui spécialise (directement ou non) le classificateur contexte.

L'arbre de spécialisation de la classe abstraite `RedefinableElement` est montré en annexe A.5.

Des règles de redéfinition spécifiques à chaque sous-classe seront édictées plus loin dans le document.

Règles de cohérence

• Règle 48 : Au moins un des contextes d'un élément qui en redéfinit un autre doit être une spécialisation d'au moins un des contextes de l'élément redéfini. [[7] p.70]

Remarque Dans le cas le plus courant où un élément redéfinissable n'a qu'un contexte, la règle 48 peut s'exprimer de la façon suivante : le contexte d'un élément qui en

[10] Guide de prévention ? (rq de hugues : pas compris)

[11] guide de prévention : Si la classe possède des propriétés, celles-ci doivent figurer à chaque apparition de la classe dans le modèle.

redéfinit un autre doit être une spécialisation du contexte de l'élément redéfini.

• Règle 49 : Un élément et l'élément qu'il redéfinit doivent être cohérents entre eux. [[7] p.70]

Remarque Cette règle est une règle abstraite qui sera redéfinie dans chaque élément redéfinissable.[12]

2.14 Operation

Contexte Une opération (*operation* en anglais) est l'implantation d'un service qui peut être demandé à toutes les instances d'un même classificateur dans le but de déclencher un comportement. Pour cela, une opération est spécifiée par un nom, un type, des paramètres et des contraintes.

Remarque Une opération est souvent utilisée dans le contexte des classes mais peut être contenue par une classe, un type de données, une interface ou un artefact.

Dans cette section nous exprimons, en 2.14.1 les règles qui s'appliquent uniquement au méta-modèle, et en 2.14.2 les règles qui s'appliquent également aux modèles.

2.14.1 Règles sur le méta-modèle

Contexte Nous présentons ici les règles qui s'appliquent sur les opérations et qui ne concernent que le méta-modèle.

Règles de cohérence

• Règle 50 : Si une opération a un seul résultat, l'attribut `isOrdered` est égal à l'attribut `isOrdered` de ce résultat. Autrement, `isOrdered` est faux. [[7] p.77] [Règle sur le méta-modèle]

• Règle 51 : Si une opération a un seul résultat, l'attribut `isUnique` est égal à l'attribut `isUnique` de ce résultat. Sinon `isUnique` est vrai. [[7] p.77] [Règle sur le méta-modèle]

• Règle 52 : Si une opération a un seul résultat, l'attribut `lower` est égal à l'attribut `lower` de ce résultat. Sinon `lower` n'est pas défini. [[7] p.77] [Règle sur le méta-modèle]

• Règle 53 : Si une opération a un seul résultat, l'attribut `upper` est égal à l'attribut `upper` de ce résultat. Sinon `upper` n'est pas défini. [[7] p.77] [Règle sur le méta-modèle]

• Règle 54 : Si une opération a un seul résultat, l'attribut `type` est égal à l'attribut `type` de ce résultat. Sinon `type` n'est pas défini. [[7] p.77] [Règle sur le méta-modèle]

2.14.2 Règles impliquant les modèles

Contexte Nous présentons ici les règles qui s'appliquent sur les opérations et qui concernent directement le niveau modèle.

[12] vérifier que c'est bien le cas

Règles de cohérence

• Règle 55 : L'expression d'une opération doit suivre la syntaxe suivante : [[7] p.78]

```
[visibility] name ( parameter-list ) : property-string
```

• Règle 56 : Le champ `parameter-list` de la règle 55 doit suivre la syntaxe : [[7] p.79]

```
direction name : type-expression '['multiplicity']'
                 = default-value [{ property-string }]
```

où `direction` a comme valeur par défaut `in`.

• Règle 57 : Une opération ne peut être contenue que par une classe, une interface, un artifact ou un type de données. [Règle dérivée du méta-modèle]

• Règle 58 : On ne peut spécifier une condition sur le corps d'une opération que si l'opération n'a pas d'effet de bord (c'est-à-dire lorsque le méta-attribut `{isQuery}` de l'opération vaut `true`). [[7] p.78]

• Règle 59 : Une opération redéfinissant une autre opération doit respecter les propriétés suivantes : [[7] p.78]

– les deux opérations ont le même nombre de paramètres et le même nombre de résultats ;
– le type de chaque paramètre et résultat de l'opération est conforme au type du paramètre ou résultat correspondant.

Remarque La règle 59 est une redéfinition de la règle 49.

• Règle 60 : Deux opérations qui appartiennent à une même classe doivent avoir des signatures différentes. [[7] p.72]

Remarque Cette règle redéfinit la règle 4 pour les opérations.

Remarque Deux signatures d'opération ont la même signature si et seulement si le nom des deux opérations est le même et les paramètres sont de même types (en prenant en compte le nombre et l'ordre des paramètres).

• Règle 61 : Deux paramètres d'une opération ne doivent pas avoir le même nom (même s'ils sont de types différents). [Nouvelle règle]

Remarque La règle 61 redéfinit la règle 4 pour les paramètres.

• Règle 62 : Si le champ `parameter-list` est présent, celui-ci doit contenir tous les paramètres de l'opération. [Nouvelle règle][Règle utilisateur]

• Règle 63 : Dans un diagramme de machines à états, une opération sans effet de bord ne doit pas être le déclencheur d'une transition qui relie deux états différents du classificateur contenant l'opération. [Nouvelle règle][13]

• Règle 64 : La méthode associée à une opération ne doit pas modifier les paramètres dont le mode de passage est `in`. [Nouvelle règle][14]

Guides

• Guide 5 : Il est conseillé de faire apparaître le type des paramètres d'une opération.

• Guide 6 : Il est conseillé de spécifier une opération qui ne modifie pas l'état du système par la propriété "{query}".

[13] inter-diagramme : diagramme d'état et de classe

[14] inter-diagramme : diagramme de Classes / diag. Activités et StateMachine

Justification

• Justification 5 : Ceci se justifie par la possibilité de vérifier la cohérence de leur utilisation (par exemple lors de la manipulation d'attributs dans les actions associées aux transitions d'un diagramme de machine à état).

• Justification 6 : Ceci se justifie par la possibilité de vérifier qu'effectivement cette opération ne modifie pas l'état du système.

2.15 Property (au sens possession)

Contexte Une possession (*property* en anglais) est une caractéristique structurelle.

Quand une possession est contenue par une classe, elle représente un attribut. Un attribut représente soit un attribut "classique" (cf. section 2.15.2) soit une fin d'association navigable.

Une possession contenue par une association représente une fin d'association non navigable.

Afin d'expliquer ces notions, nous présentons la partie du méta-modèle concernée en figure 2.2. La figure 2.3 montre un modèle UML où deux classes sont mises en relation par une association navigable de `Classe1` vers `Classe2` uniquement. Enfin, la figure 2.4 correspond à son application dans le méta-modèle. Cette dernière figure fait ressortir le fait que :

- au niveau du méta-modèle, une fin d'association navigable est équivalente à un attribut (en effet les associations entre `Classe1` et `Classe2` et entre `Classe1` et `Classe3` sont identiques) ;
- `Classe1` a accès à `Classe2` via `ownedAttribute` et `type` alors que `Classe2` n'a aucun chemin d'accès à `Classe1`.

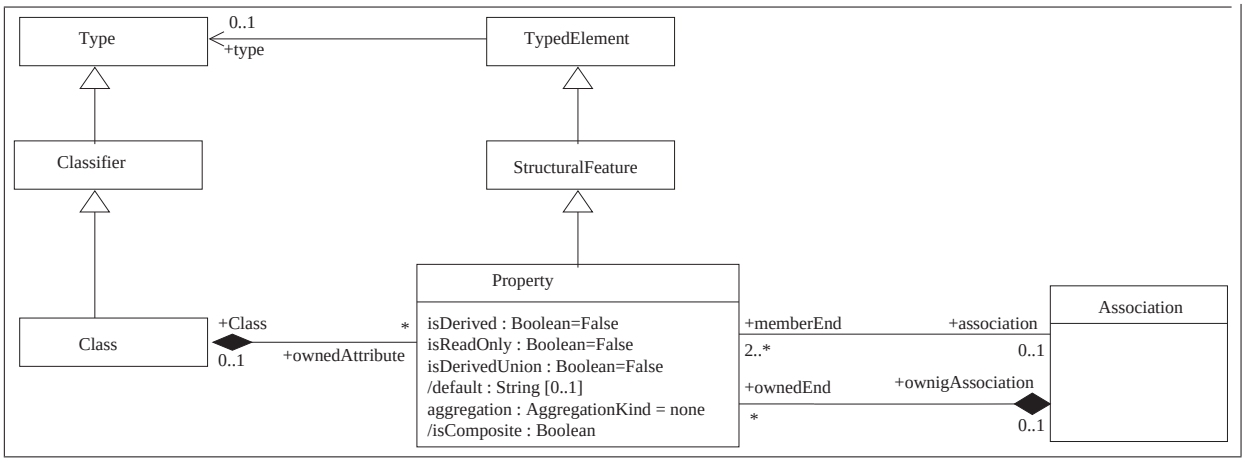

Fig. 2.2 – Partie du méta-modèle relative aux classes, à ses attributs et ses associations

Remarque La partie du méta-modèle présentée par la figure 2.2 explique aussi pourquoi les attributs et les fins d'association (qui sont représentés par la méta-classe `Property`) peuvent être associés aux mêmes propriétés.

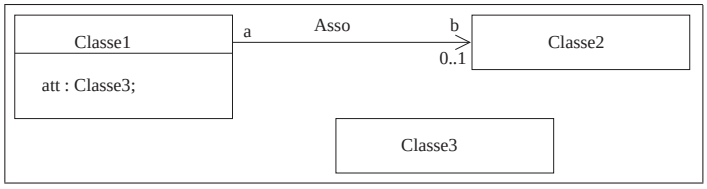

FIG. 2.3 – Modèle de deux classes

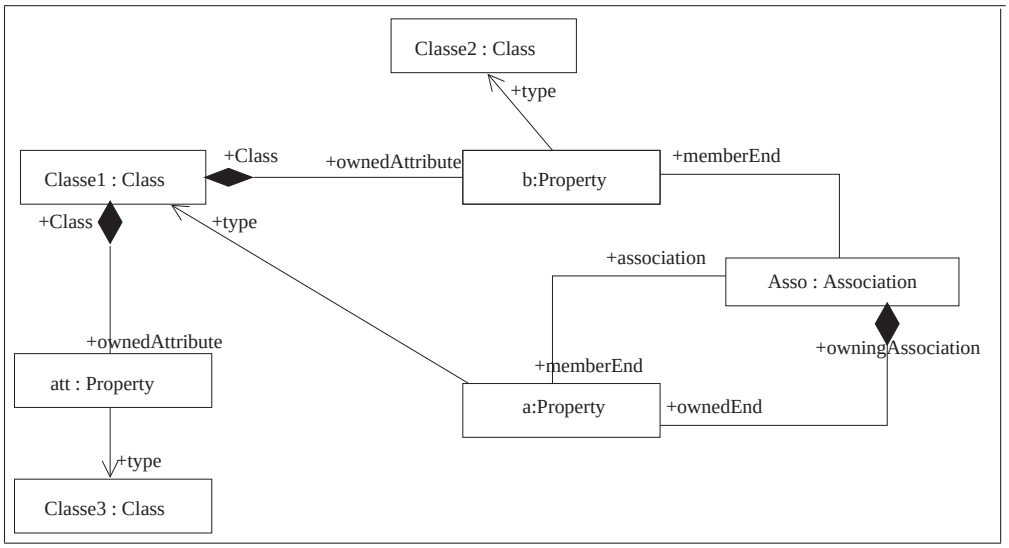

FIG. 2.4 – Application au méta-modèle du modèle de la figure 2.3

Bien que les attributs et les possessions soient représentées par la même méta-classe, certaines différences dans leur utilisation apparaissent. Nous présentons donc dans la section 2.15.1 les règles qui s'appliquent aux fins d'association et aux attributs, dans la partie 2.15.2 les règles spécifiques aux attributs et dans les parties 3.9.1.3 et 3.9.1.4 les règles spécifiques aux fins d'association.

2.15.1 Règles générales

Règles de cohérence

• RÈGLE 65 : Une possession marquée avec la propriété `{union}` doit être dérivée. [[7] p.91]

Remarque Graphiquement ceci revient à mettre le caractère '/' devant le nom de l'attribut.

• RÈGLE 66 : Une possession dérivée d'une union (propriété `{union}`) doit avoir son méta-attribut `isReadOnly` à la valeur `true`. [[7] p.91] [Règle sur le méta-modèle]

Remarque Le fait qu'un attribut soit marqué comme étant une `union` implique donc que cet attribut doit respecter les règles 80 et 81.

• RÈGLE 67 : Une possession marquée avec la propriété `{union}` doit être soit redéfinie par une autre possession, soit au moins deux possessions sont des sous-ensembles de l'ensemble représenté par l'attribut. [Nouvelle règle]

Remarque Des règles de redéfinition d'attributs et de fins d'association seront présentées plus loin.

• Règle 68 : La propriété `{subsets <property-name>}` ne peut apparaître que quand le contexte de la possession sous-ensemble est conforme au contexte de la possession sur-ensemble. [[7] p.90]

• Règle 69 : Lorsqu'un attribut est marqué par la propriété `{subsets <property-name>}`, le type de la possession sous-ensemble doit être conforme au type de la possession sur-ensemble, et la borne supérieure de la multiplicité de la possession sous-ensemble doit être inférieure à la borne supérieure de la multiplicité de la possession sur-ensemble.[[7] p.91]

• Règle 70 : Lorsqu'une possession est marquée par la propriété `{subsets <property-name>}`, il doit exister une possession héritée telle que cette possession soit nommée `property-name`. [Nouvelle règle][Règle utilisateur]

Remarque Cette possession doit également respecter les règles 68 et 69.

• Règle 71 : La somme des bornes inférieures des multiplicités des possessions qui sont le sous-ensemble d'une possession (nommée A) doit être inférieure à la borne supérieure de la multiplicité de A. [Nouvelle règle]

• Règle 72 : Lorsqu'une possession est marquée par la propriété `{redefines <property-name>}`, il doit exister une possession héritée telle que cette possession soit nommée `property-name`. [Nouvelle règle][Règle utilisateur]

Remarque Cette possession doit également respecter la règle 73. D'autre part, la figure 2.5 montre un exemple de respect de la règle 72 (sous-figure i)) et un exemple de non respect de cette règle (sous-figure ii)).

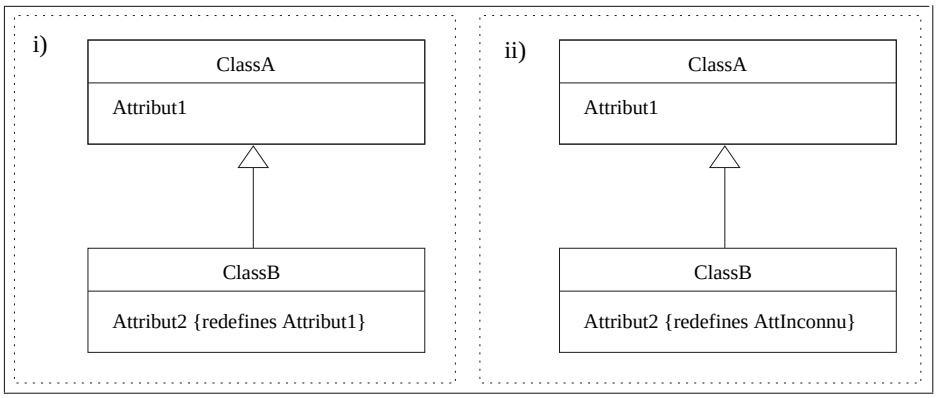

Fig. 2.5 – Exemple de respect et de non respect de la règle 72

• Règle 73 : Lorsqu'une possession A est marquée par la propriété `{redefines <property-name>}`, la multiplicité de la possession redéfinie doit être un sur-ensemble de la multiplicité de la possession A. [Nouvelle règle]

• Règle 74 : Une possession qui représente un attribut "classique" ne peut ni redéfinir ni être le sous-ensemble d'une possession qui représente une fin d'association. [Nouvelle règle]

• Règle 75 : Une possession qui représente une fin d'association ne peut ni redéfinir ni être le sous-ensemble d'une possession qui représente un attribut "classique". [Nouvelle règle]

Remarque Au niveau méta-modèle, une possession qui représente un attribut "classique" est une possession qui n'est pas associée à la méta-classe `association`. Une fin d'association est une possession qui est associée à une association.

2.15.2 Attribute

Contexte Un attribut est une caractéristique structurelle d'une classe. Dans cette section nous nous intéressons aux attributs d'une classe dans le sens "classique", c'est-à-dire aux attributs qui ne représentent pas une fin d'association navigable.

Règles de cohérence

• Règle 76 : Les attributs doivent être présentés en suivant la syntaxe suivante : [[7] p.64]

```
[visibility][/]name[:type][multiplicity][= default]
                                    [{ property-string }]
```

• Règle 77 : Un attribut doit avoir un nom. [Nouvelle règle]

• Règle 78 : Le champ `type` doit être soit une classe ou être un type primitif (cf. section 2.18). [Nouvelle règle]

• Règle lien 3 : Le type peut être exprimé via le nom qualifié du classificateur et doit donc respecter les règles 9 à 11.

• Règle 79 : Dans le cas où l'attribut a une valeur par défaut, cette valeur doit avoir le même type que celui de l'attribut. [Nouvelle règle]

• Règle 80 : Les méthodes associées à des opérations ne doivent pas modifier d'attributs qui ont la propriété `readOnly`. [Nouvelle règle][15]

• Règle 81 : Si un attribut est marqué avec la propriété `{readOnly}`, aucune action d'une transition (champ `action-expression` du label) du diagramme d'état du classificateur ne doit modifier cet attribut. [Nouvelle règle][16]

Guide

• Guide 7 : Il est conseillé d'exprimer le type de chaque attribut.

• Guide 8 : Il est conseillé d'exprimer la visibilité de l'attribut.

Justification

• Justification 7 : Nous conseillons d'exprimer le type de chaque attribut afin de pouvoir détecter toute utilisation incorrecte de celui-ci et la règle 79.

• Justification 8 : Nous conseillons d'exprimer la visibilité de l'attribut pour vérifier la correction de l'accès à un attribut.

[17]

[15] inter-diagramme -> avec le diagramme de classe et le diagramme de machine à état ou le diagramme d'activité

[16] inter-diagramme classes - machine à états

[17] prévention : spécifier de façon explicite la multiplicité de l'attribut

2.16 Data Type

Contexte Un type de données (*data type* en anglais) est une sorte de classificateur dont les instances sont des valeurs (pas des objets). Une valeur n'a pas d'identité de sorte que deux valeurs identiques ne peuvent pas être différenciées. Habituellement, un type de donnée sert à spécifier le type d'un attribut.

Un type de données est représenté avec le mot clé « data type ».

Règles de cohérence

• Règle 82 : Les opérations contenues dans les types de données sont des fonctions pures, c'est-à-dire qu'elles ne doivent modifier aucun paramètre (pas de mode `inout`), et qu'elles doivent avoir exactement une valeur de retour (un paramètre en mode `out` ou `return`). [Nouvelle règle]

• Règle 83 : Les instances d'un type de données sont des valeurs et ne peuvent donc pas être nommées. [Nouvelle règle]

2.17 Enumeration

On peut identifier deux grandes familles de règles de cohérence concernant les énumérations, l'une associée à la déclaration et l'autre à l'utilisation de l'énumération.

Une énumération est spécifiée avec le mot clé « enumeration ».

2.17.1 Déclaration

Contexte Une énumération est un type de données dont les instances sont un ensemble de littéraux nommés et ordonnés.

On spécifie une énumération en associant le mot clé « enumeration » au symbole d'une classe.

Règles de cohérence

• Règle 84 : Si une énumération apparaît plusieurs fois dans un modèle en spécifiant ses littéraux, tous les littéraux doivent apparaître et dans le même ordre. [Nouvelle règle][Règle utilisateur]

• Règle lien 4 : Les littéraux qui appartiennent à une même énumération doivent tous avoir des noms différents (découle de la règle 4 pour les littéraux d'une énumération).

• Règle 85 : Une énumération contient au moins un littéral nommé. [Nouvelle règle]

• Règle lien 5 : Les énumérations étant des sortes de types de données, elles doivent respecter les règles de la section 2.16.

2.17.2 Utilisation

Contexte Les littéraux sont nommés et ordonnés entre eux mais aucune autre algèbre n'est définie par défaut. Des opérations sur l'ensemble ordonné des littéraux peuvent être définies dans la classe de stéréotype « enumeration ».

Règles de cohérence

• Règle 86 : Toute opération utilisée sur les littéraux autre que la comparaison de littéraux et la comparaison de leur rang doit être définie par l'énumération. [Nouvelle règle]

2.18 Primitive Type

Contexte Un type primitif est un type de données prédéfini. Un type primitif de données peut avoir une algèbre et des opérations définies en dehors du langage UML, en mathématique par exemple.

Les types primitifs définis en UML sont les types `Boolean`, `Integer`, `UnlimitedNatural` et `String`. Cette liste peut être étendue.

Un type primitif est spécifié avec le mot clé « primitive ».

Règles de cohérence

• Règle lien 6 : Les types primitifs étant des types de données, ils doivent respecter les règles de la section 2.16.

2.19 Package

Contexte En UML, un paquetage (*package* en anglais) est un mécanisme qui permet d'organiser des éléments de modélisation en groupes. On peut accéder aux éléments qu'il contient grâce aux noms qualifiés.

La visibilité d'un élément contenu dans un paquetage peut être spécifiée en mettant le signe '+' (visibilité publique) ou '-' (visibilité privée) devant le nom de l'élément. Lorsque la visibilité n'est pas spécifiée, un élément est visible en dehors du paquetage. [18]

Règles de cohérence

• Règle 87 : Les éléments contenus dans les paquetages doivent être des `PackageableElement` (cf. annexe A.6). [Règle dérivée du méta-modèle]

• Règle 88 : Si un élément contenu dans un paquetage a une visibilité, alors cette visibilité est `public` ou `private`. [[7] p.100]

2.20 Model

Contexte Un modèle (*model* en anglais) est la description d'un système physique avec un certain objectif, comme la description d'aspects logiques ou comportementaux du système physique pour une certaine catégorie de lecteurs.

Un modèle est noté avec le mot clé « model » associé au symbole du paquetage.

Les stéréotypes « metamodel » et « systemModel » peuvent s'appliquer aux modèles.

[18] implique un guide de prévention-> mettre une visibilité et ne pas laisser d'élément sans visibilité spécifiée explicitement

Règles de cohérence

• Règle lien 7 : Un modèle étant une sorte de paquetage, un modèle doit respecter les règles énoncées en 2.19.

2.21 Interface

Contexte Une interface est une sorte de classificateur qui représente une déclaration d'un ensemble cohérent de caractéristiques et d'exigences publiques. Ainsi, elle représente une sorte de contrat qui doit être satisfait par n'importe quelle instance de classificateur qui réalise l'interface.

Les exigences qui peuvent être associées à une interface prennent la forme de contraintes (comme des préconditions et des postconditions), ou des spécifications de protocoles, qui imposent des restrictions d'ordre pour les interactions qui passent au travers de l'interface.

Remarque Une interface peut contenir des attributs, des opérations et des réceptions. Une réception est la spécification de la réaction comportementale d'un objet à un signal.

La spécification d'une interface se fait avec le mot clé « interface » ou avec la notation graphique des interfaces.

Règles de cohérence

• Règle 89 : La visibilité de toutes les caractéristiques d'une interface doit être publique. [[7] p.114]

• Règle 90 : Le(s) classificateur(s) qui réalise(nt) une interface doit (doivent) respectivement avoir une opération ou une réception pour chaque opération ou réception de l'interface. [Nouvelle règle]

Remarque Les opérations ou réceptions peuvent aussi être contenues par des possessions du classificateur.

• Règle 91 : Une interface ne possède pas de méthodes. [Règle dérivée du métamodèle]

Remarque Une méthode est l'implantation d'une opération (cf. section 2.30).

• Règle 92 : Une interface ne possède pas d'association. [Nouvelle règle][19]

2.22 Signal

Contexte Un signal est la spécification d'un type d'instances d'envoi de requête entre des objets communiquants. L'objet récepteur traite l'instance du signal comme décrit par ses réceptions (cf. section 2.23).

Les données transférées lors de l'envoi d'un signal sont représentées par les attributs du signal. Un signal est défini indépendamment du classificateur qui va le traiter.

Un signal est une sorte de classificateur.

Règles de cohérence

Pas de règles trouvées.

[19] pas sûr

2.23 Reception

Contexte Une réception (*reception* en anglais) est une déclaration qui indique qu'une classe est capable de traiter un signal.

La réception d'une instance de signal par l'instance du classificateur qui contient une réception correspondante provoque l'invocation d'un comportement spécifié par la méthode de la réception. Une réception correspond à un signal si le signal reçu est un sous-type du signal référencé par la réception.

Une réception ne peut appartenir qu'à une interface ou une classe.

Comme le montre la figure 2.6 une réception se note avec le mot clé « signal ».

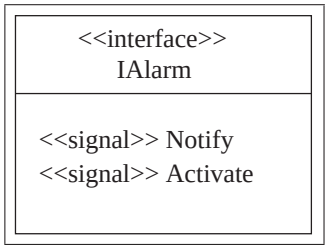

FIG. 2.6 – Notation graphique des réceptions

Règles de cohérence

• RÈGLE 93 : Une réception ne peut pas être "query". [[7] p.394] [20]

• RÈGLE 94 : Une classe passive ne peut pas avoir de réception. [[7] p.394]

• RÈGLE 95 : Une réception ne peut appartenir qu'à une classe ou à une interface. [Règle dérivée du méta-modèle]

2.24 Association Class

Contexte Une classe d'association (*association class* en anglais) est un élément qui possède à la fois les propriétés d'une association et d'une classe.

Règles de cohérence

• RÈGLE 96 : Une classe d'association ne peut pas être définie entre elle-même et quelque chose d'autre. Il ne faut pas non plus que la classe d'association soit un parent ou un enfant d'une des classes mises en jeu. [[7] p.118]

• RÈGLE LIEN 8 : Les classes association étant des associations, elles doivent respecter les propriétés des associations présentées en section 3.9.

• RÈGLE LIEN 9 : Les classes association étant des classes, elles doivent respecter les propriétés des classes présentées en section 2.12.

[20] problème avec query => query n'est pas un attribut ni une association de reception

2.25 Parameterable Element

Contexte Les éléments pouvant jouer le rôle de paramètre générique (*parameterable element* en anglais) sont des éléments qui peuvent être exposés comme des paramètres formels d'une signature de généricité.

Les éléments pouvant jouer le rôle de paramètre générique sont présentés en annexe A.7.

Règles de cohérence

Pas de règles trouvées.

2.26 Template Parameter

Contexte Un paramètre générique (*template parameter* en anglais) est contenu par une signature de généricité dans laquelle il expose un élément pouvant jouer le rôle de paramètre générique (cf. section 2.25) comme un paramètre formel de cette signature. Cet élément pouvant jouer le rôle de paramètre générique n'a de sens qu'à l'intérieur de l'élément générique (ou à l'intérieur de ses spécialisations).

Nous présentons donc dans la section 2.26.1 les règles générales qui s'appliquent aux paramètres génériques, dans la partie 2.26.2 les règles spécifiques aux classificateurs jouant le rôle de paramètres génériques et dans la partie 2.26.3 les règles spécifiques aux opérations passées en paramètre générique.

2.26.1 Règles générales

Règles de cohérence

• Règle 97 : La déclaration d'un paramètre générique doit suivre la syntaxe : [[7] p.549]

```
template-parameter ::= template-parameter-name
                       [ ‘:‘ parameter-kind ] [‘=’ default]
```

Remarque La règle 97 est redéfinie par certaines sous-classes de `ParameterableElement`.

• Règle 98 : Le champ `parameter-kind` doit correspondre à une métaclasse existante. [Nouvelle règle]

• Règle 99 : La valeur par défaut (champ `default`) doit être compatible avec l'élément pouvant jouer le rôle de paramètre générique. [[7] p.548]

• Règle 100 : La valeur par défaut ne peut être contenue que si l'élément pouvant jouer le rôle de paramètre générique n'est pas contenu. [[7] p.548] [Règle sur le métamodèle]

• Règle 101 : Le paramètre générique ne doit pas être utilisé ailleurs dans le modèle qu'au sein de l'élément générique. [Nouvelle règle]

Remarque La figure 2.7 donne un exemple de non-respect d'utilisation d'un paramètre générique (règle 101). Dans ce cas, la classe `ClasseExterieure` n'a pas le droit d'utiliser le paramètre générique `TempClass1`. En revanche, la classe `AutreClasse` a tout à fait le droit d'avoir un attribut de type `TempClass2` car elle est contenue par l'élément générique (`P2`).

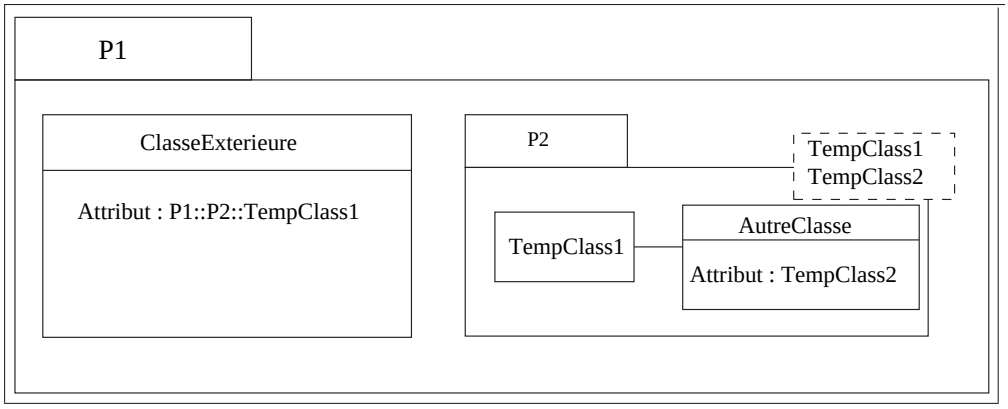

FIG. 2.7 – Utilisation interdite d'un paramètre générique

2.26.2 Classificateur

Contexte Les classificateurs sont des éléments pouvant jouer le rôle de paramètres génériques spéciaux pour lesquels d'autres règles s'appliquent.

Règles de cohérence

• RÈGLE 102 : Dans le cas où le paramètre générique est un classificateur, la syntaxe à respecter est la suivante : [[7] p.556]

```
classifier-template-parameter ::= parameter
     [ ‘:‘ parameter-kind ] [‘>’ constraint] [‘=’ default]
parameter ::= parameter-name
constraint ::= [‘{contract }’] classifier-name1
default ::= classifier-name2
```

• RÈGLE 103 : Les champs `classifier-name1` et `classifier-name2` de la règle 102, doivent correspondre à un classificateur visible par le classificateur générique. [Nouvelle règle]

• RÈGLE 104 : Les métaclasses des classificateurs référencées par `classifier-name1` et par `classifier-name2` ainsi que la métaclasse référencée par `parameter-kind` doivent être les mêmes. [Nouvelle règle]

Remarque Le champ `parameter-kind` de la règle 102 doit respecter la règle 98.

• RÈGLE 105 : Si le champ optionnel `[‘>’ constraint]` apparaît sans `{contract}` alors le paramètre de substitution (lors de la délimitation de l'élément générique) devra être une spécialisation du classificateur spécifié par `classifier-name1`. [Nouvelle règle]

• RÈGLE 106 : Si le champ optionnel `[‘>’ constraint]` apparaît sans `{contract}` alors `default` doit être une spécialisation du classificateur spécifié par `classifier-name1`. [Nouvelle règle]

2.26.3 Opération

Contexte Les opérations peuvent être passées en tant que paramètre générique d'un élément générique.

Règles de cohérence

• Règle 107 : Une opération passée en tant que paramètre générique doit être notée en respectant la syntaxe suivante : [[7] p.565]

```
operation-template-parameter ::= parameter [‘:‘parameter-kind]
                                           [‘=’default]
parameter ::= operation-name ( parameter-list )
default ::= operation-name ( parameter-list )
```

2.27 Template Signature

Contexte Une signature de généricité (*template signature* en anglais) montre les paramètres génériques d'un élément générique.

D'autre part, quand une signature de généricité appartient à un classificateur et que ce classificateur est spécialisé, il est possible de redéfinir la signature de généricité au sein du classificateur spécialisé. Ceci permet d'ajouter des paramètres génériques au classificateur spécialisé.

Règles de cohérence

• Règle 108 : Au niveau méta-modèle, la signature de généricité doit contenir tous ses paramètres. [[7] p.550] [Règle sur le méta-modèle]

• Règle 109 : Les paramètres hérités sont les paramètres de la signature étendue. [[7] p.557] [Règle sur le méta-modèle]

• Règle 110 : Lors de spécialisations de classificateurs génériques, tous les paramètres génériques des classificateurs généraux doivent être des paramètres génériques du classificateur spécifique. La redéfinition d'une signature générique est donc cohérente si des paramètres génériques sont uniquement ajoutés. [[7] p.557]

Remarque Lors d'héritage simple (ce qui est le plus courant), la règle 110 peut s'exprimer de la manière suivante : lors d'une spécialisation d'un classificateur générique, tous les paramètres génériques du classificateur général doivent être des paramètres génériques du classificateur spécifique.

Remarque Les spécialisations d'éléments génériques sont donc des éléments génériques.

2.28 Templateable Element

Contexte Un élément supportant la généricité (*templateable element* en anglais) est un élément qui peut être défini comme générique et peut ensuite être borné. Les éléments supportant la généricité peuvent donc contenir des signatures de généricité.

2.28.1 Règles générales

Règles de cohérence

• Règle 111 : Les paramètres génériques d'un élément supportant la généricité doivent appartenir (directement ou non) à l'élément générique. [Nouvelle règle][Règle

sur le méta-modèle]

• Règle 112 : La métaclasse d'un élément supportant la généricité doit être un `TemplateableElement` (cf. annexe A.8). [Règle dérivée du méta-modèle]

2.28.2 Opération

Contexte Les opérations supportent la généricité, c'est-à-dire qu'une opération peut avoir des paramètres génériques.

Règles de cohérence

• Règle 113 : Un opération générique doit suivre la syntaxe suivante : [[7] p.563]

```
visibility name ‘<‘ template-parameter-list ‘>’
                ‘<<‘ binding-expression-list ‘>>’
                ‘(‘ parameter-list ‘)’ ‘:’ property-string
```

2.29 Behaviored Classifier

Contexte Un classificateur comportemental (*behaviored classifier* en anglais) est un classificateur qui peut contenir des comportements (cf. section 2.30).

Règles de cohérence

Pas de règles supplémentaires.

2.30 Behavior

Contexte Un comportement (*behavior* en anglais) est la description du comportement d'une entité en réponse à des événements. En UML il existe trois sortes de comportements, les machines à états, les activités et les interactions.

Un comportement est soit associé à la description du classificateur comportemental qui le contient soit associé comme étant une méthode correspondant à une caractéristique comportementale. Dans le premier cas, le comportement décrit la séquence des états dans lequel une instance peut se trouver au cours de sa vie. Dans le deuxième cas, un comportement décrit le déroulement du calcul de la caractéristique comportementale.

Lors de l'invocation d'un comportement, l'exécution peut recevoir un ensemble de paramètres d'entrée et les résultats sont décrits par un ensemble de paramètres de sortie.

Règles de cohérence

• Règle 114 : Les paramètres du comportement doivent correspondre aux paramètres de la caractéristique comportementale. [[7] p.381]

• Règle 115 : La caractéristique comportementale implémentée par le comportement doit être une caractéristique (éventuellement héritée) du classificateur contexte du comportement. [[7] p.381]

• Règle 116 : Si la caractéristique comportementale implantée a été redéfinie par un ancêtre du classificateur qui contient ce comportement, alors le comportement doit

réaliser la dernière caractéristique comportementale redéfinie. [[7] p.381]

Remarque Les règles 114, 115 et 116 s'appliquent dans le cas où le comportement correspond à une méthode d'une caractéristique comportementale.

• Règle 117 : Il peut y avoir au plus un comportement pour une paire formée par un classificateur (possesseur du comportement) et la caractéristique comportementale (spécification du comportement). [[7] p.381]

• Règle 118 : Dans le cas où un comportement décrit le comportement du classificateur, il ne peut pas avoir de caractéristique comportementale comme spécification. [[7] p.383]

Chapitre 3

Relations

3.1 Element Import

Contexte L'importation d'un élément (*element import* en anglais) est une relation entre un espace de nommage (*namespace* en anglais) et un élément contenu dans un paquetage. Ceci permet aux éléments de l'espace de nommage importateur d'accéder à l'élément importé sans utiliser son nom qualifié.

Afin d'éviter les conflits de noms ou de préciser les noms, il est possible de donner un alias correspondant au nom de l'élément importé dans l'espace de nommage source de la relation. [1]

La figure 3.1 (tirée de [7]) montre un exemple d'importation d'un élément avec un alias.

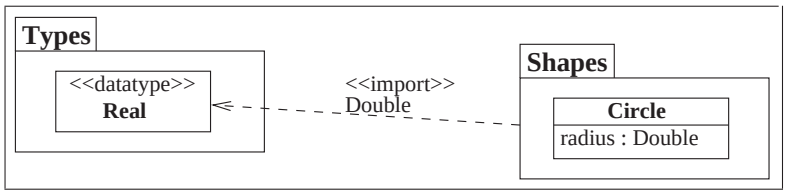

FIG. 3.1 – Exemple d'une importation d'élément avec alias

Une importation d'élément est marquée en associant le mot clé « import » à une relation de dépendance.

Remarque Dans le cas où la cible de la relation est un paquetage, la relation est une relation d'importation de paquetage (cf. section 3.2).

Règles de cohérence

• RÈGLE 119 : La source d'une importation d'élément doit être un espace de nommage. [Règle dérivée du méta-modèle]

• RÈGLE 120 : La cible d'une importation d'élément doit être un élément empaquetable (cf. annexe A.6). [Règle dérivée du méta-modèle]

[1] peut donner lieu à un guide de prévention : renommer pour donner des noms explicites

[2]

• Règle 121 : Un espace de nommage ne doit pas être la source de plusieurs importations d'éléments qui ont le même nom. [Nouvelle règle]

• Règle 122 : L'élément importé doit être visible par l'espace de nommage qui importe l'élément. [Nouvelle règle]

• Règle 123 : Si un alias est spécifié, aucun nom d'élément appartenant à l'espace de nommage importateur ne doit être identique à l'alias spécifié. [Nouvelle règle]

• Règle 124 : La visibilité d'une relation d'importation d'un élément doit être soit publique soit privée (« public » ou « private » en anglais) et donc ne doit pas être « protected » ni « package ». [[7] p.32] [Règle sur le méta-modèle]

• Règle 125 : Un élément importé doit posséder une visibilité publique ou pas de visibilité du tout. [[7] p.32]

3.2 Package Import

Contexte Une importation de paquetage est une relation entre un espace de nommage et un paquetage, qui signifie que l'espace de nommage ajoute tous les membres du paquetage dans son propre espace de nommage.

Une importation de paquetage est spécifiée par une relation de dépendance associée au mot clé « import » ou « access ». Le mot clé « import » spécifie une importation publique (c'est-à-dire que c'est une relation transitive) et le mot clé « access » une importation privée.

Remarque La notation d'une importation de paquetage et d'une importation d'élément est identique. La distinction entre ces deux types d'importation est réalisée en fonction de la cible : dans le cas où la cible de la relation d'importation est un paquetage, la relation est une importation de paquetage.

Règles de cohérence

• Règle 126 : La source d'une relation d'importation de paquetage doit être un espace de nommage. [Règle dérivée du méta-modèle]

• Règle lien 10 : Il ne doit pas y avoir de collision de noms entre les éléments importés et les éléments contenus dans le paquetage "importateur". Cette règle exprime la règle 4 dans le cas où l'espace de nommage importe des éléments.

• Règle 127 : Dans un modèle complet, l'importation doit être utile. C'est-à-dire qu'en supposant que l'importation du paquetage n'ait pas lieu, un accès erroné à au moins un élément de la cible doit être détecté. [Nouvelle règle]

Remarque Dans le cas contraire, le modèle n'est pas incohérent mais l'importation est aberrante.

[2] guide de prévention RÈGLE 234 : "Dans le cas où aucun alias n est spécifié, aucun nom d élément appartenant à l espace de nommage importateur ne doit être identique au nom de l élément importé." est en fait un guide de prévention cf. spec p32 : c possible mais dans ce cas il faut utiliser le nom qualifié : ça sert à rien donc.

3.3 Generalization

Contexte Une relation d'héritage est une relation de classification entre un élément général (le parent) et un élément plus spécifique (l'enfant). L'enfant combine la structure et le comportement du parent et y ajoute des informations spécifiques.

3.3.1 Éléments mis en relation

Contexte Les règles de cohérence qui suivent portent sur les éléments mis en relation par la relation d'héritage.

Règles de cohérence

• Règle 128 : Les types des classificateurs généraux et spécialisés doivent être compatibles. [[7] p.62] [3]

• Règle 129 : Le méta-type du classificateur spécial de la relation d'héritage doit être le même ou être une spécialisation du méta-type du classificateur général. [Nouvelle règle][4]

Remarque Un classificateur A peut donc spécialiser un classificateur B si A et B sont de même type ou si B est un ancêtre de A. Par exemple, une classe d'association peut spécialiser une classe mais pas l'inverse, une association peut spécialiser une association, une classe peut spécialiser une classe, etc.

• Règle 130 : Les éléments mis en relation par une relation d'héritage doivent être des classificateurs. [Règle dérivée du méta-modèle]

• Règle 131 : La hiérarchie des généralisations doit être orientée (*directed* en anglais) et acyclique. [[7] p.62]

Remarque Ainsi, un classificateur ne doit pas être à la fois un ancêtre et un descendant d'un même classificateur.

• Règle 132 : Une relation d'héritage doit mettre en jeu exactement un parent et un enfant. [Règle dérivée du méta-modèle]

• Règle 133 : Seuls des classificateurs peuvent être mis en relation par une généralisation. [Règle dérivée du méta-modèle]

• Règle 134 : Si on utilise des points de suspension `(...)` dans le modèle pour montrer l'existence d'enfants additionnels, ces enfants doivent apparaître dans d'autres vues du modèle. [Nouvelle règle][Règle utilisateur]

• Règle 135 : Les classificateurs généraux sont les classificateurs référencés par la relation de généralisation. [[7] p.62] [Règle sur le méta-modèle]

• Règle 136 : L'association `inheritedMember` est calculée en faisant l'héritage des membres héritables des parents. [[7] p.62] [Règle sur le méta-modèle]

3.3.2 Generalization Set

Contexte Un ensemble de généralisation (*GeneralizationSet* en anglais) définit un ensemble de partitions d'une relation d'héritage. Un ensemble de généralisation décrit la

[3] pb dans la def de maySpecializeType dans la norme

[4] Cette règle est-elle redondante de la règle précédente ? ? ? -> voir version finale de la norme

façon de partitionner l'ensemble des classificateurs qui sont enfants d'un même parent via une relation d'héritage.

La propriété `{complete}` peut s'appliquer à un ensemble de généralisation. Ceci indique que toute instance du parent est aussi une instance d'un des enfants de la relation d'héritage.[5]

La propriété `{disjoint}` peut s'appliquer à un ensemble de généralisation. Celle-ci indique que les classificateurs spécifiques appartenant à un ensemble de généralisation disjoint ne peuvent avoir d'instances en commun.

Remarque Le contraire de `{complete}` est `{incomplete}` et le contraire de `{disjoint}` est `{overlapping}`. Par défaut, un ensemble de généralisation est `{incomplete, disjoint}`. Ces deux propriétés figurent toujours en couple (cf. règle 137).

Règles de cohérence

• Règle 137 : Si l'on veut contraindre les ensembles de généralisation, seuls les couples suivants de propriétés sont possibles : `{complete, disjoint}`, `{incomplete, disjoint}`, `{complete, overlapping}` et `{incomplete, overlapping}`. [tirée de [7] p.123] [Règle utilisateur]

Remarque Le couple par défaut est `{incomplete, disjoint}`.

• Règle 138 : Chaque généralisation associée à un ensemble de généralisation particulier doit avoir le même classificateur général. [[7] p.67]

• Règle 139 : Le parent d'un ensemble de généralisation qui a la propriété `{complete}` ne peut pas être instancié. [Nouvelle règle]

• Règle 140 : Lorsque l'ensemble de généralisation a la propriété `{disjoint}`, deux enfants (E1 et E2) qui appartiennent à cet ensemble de généralisation ne peuvent pas avoir de descendant en commun. Ainsi, aucun classificateur ne doit hériter à la fois de E1 et de E2. [Nouvelle règle]

• Règle 141 : Le classificateur qui est en association avec un ensemble de généralisation (qui est donc un méta-type (*power-type* en anglais)) ne peut être ni un classificateur spécifique ni un classificateur général pour toutes les relations de généralisation définies pour cet ensemble de généralisation. En d'autres termes, un méta-type ne peut pas être une instance de lui-même et ces instances ne peuvent pas faire partie de ses sous-classes. [[7] p.62]

Remarque La figure 3.2 donne deux exemples de non-respect de la règle 141.

3.3.3 Éléments abstraits

Contexte Tout classificateur peut être abstrait, ce qui est représenté par la propriété `{abstract}`. Un élément est dit abstrait quand il ne peut pas être directement instancié.

Règles de cohérence

• Règle 142 : Un élément abstrait doit être l'ancêtre (direct ou non) d'au moins un élément non abstrait. [Nouvelle règle]

[5] à regarder dans la dernière spec si ça n'a pas changé car c'est étonnant...

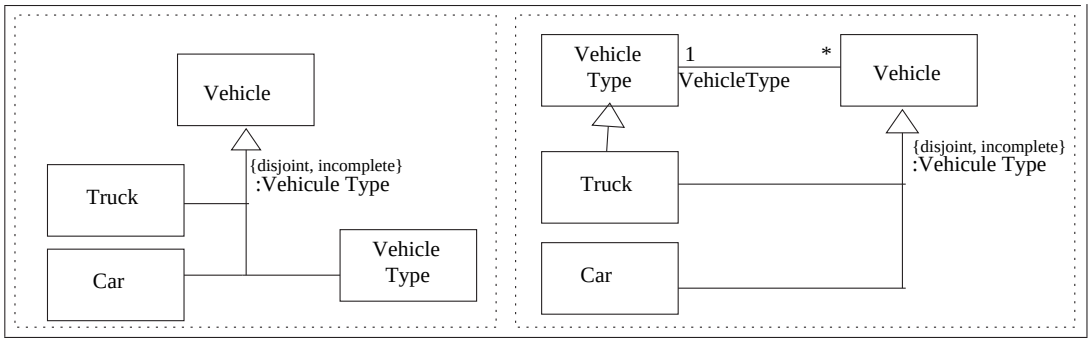

FIG. 3.2 – Exemples de non-respect de la règle 141

Remarque Cette règle ne s'applique pas aux premières phases de développement du modèle.

• RÈGLE 143 : Un élément abstrait ne doit pas avoir d'instance. [tirée de [7] p.61] [6]

3.4 Dependency

Contexte Une relation de dépendance (*Dependency* en anglais) est une relation sémantique entre deux éléments ou deux ensembles d'éléments pour lesquels la modification de l'un peut affecter l'autre sans que l'inverse ne soit nécessairement vrai.

Remarque Graphiquement, le fait qu'un élément dépende de plusieurs autres éléments se traduit par plusieurs relations de dépendances.

Règles de cohérence

• RÈGLE 144 : Une relation de dépendance doit posséder un élément ou un ensemble d'éléments source (*client*) et un élément ou un ensemble d'éléments cible (*supplier*). Ces ensembles ne peuvent pas être vides. [Règle dérivée du méta-modèle]

Guide

• GUIDE 9 : Si plusieurs dépendances conviennent au problème, il est conseillé d'utiliser la dépendance la plus précise, c'est-à-dire la plus basse dans l'arbre de spécialisation des dépendances (cf. figure A.2 en A.2).

Justification

• JUSTIFICATION 9 : Si plusieurs stéréotypes de dépendance conviennent au problème, il est conseillé d'utiliser le stéréotype le plus précis, de façon à détecter avec le plus de précision le non-respect des règles propres à la dépendance.

[6] Guide de prévention : Lorsqu'un ensemble de généralisation possède plus de deux fils, nous conseillons de tous les regrouper et de ne représenter qu'une seule occurrence du nom de l'ensemble de généralisation.

3.5 Abstraction

Contexte Une relation d'abstraction est une sorte de relation de dépendance.

Une abstraction est une relation qui établit un rapport entre deux éléments ou ensembles d'éléments qui représentent le même concept à des différents niveaux d'abstraction ou selon des points de vue différents.

Une relation d'abstraction peut être associée aux stéréotypes « derive », « refine » et « trace ».

3.5.1 Règles générales

Règles de cohérence

• Règle 145 : Une relation de dépendance d'abstraction doit mettre en jeu deux ensembles d'éléments disjoints. [Nouvelle règle]

• Règle 146 : Aucun cycle formé par des relations d'abstraction ne doit apparaître. [Nouvelle règle]

3.5.2 Derive Stereotype

Contexte Une dépendance de stéréotype « `derive` » indique que le client de la relation (source de la flèche) peut être calculé à partir du fournisseur (cible de la flèche). On fait apparaître le client pour des raisons d'efficacité ou de clarté bien qu'il soit logiquement redondant.

Il est possible de spécifier de manière formelle quelle expression permet de calculer la source à partir de la cible.

Remarque On peut représenter un élément dérivé en plaçant une oblique (*slash* en anglais) devant son nom. Il est alors possible de ne pas faire apparaître la relation de dépendance.

Règles de cohérence

• Règle lien 11 : Si l'expression qui sert à calculer l'élément dérivé est explicitée, cette expression doit répondre aux règles énoncées en 2.6. [Nouvelle règle]

• Règle 147 : Le type de l'élément source et le type de l'expression éventuellement utilisée pour décrire la dérivation doivent être identique. [Nouvelle règle]

Guides

• Guide 10 : Nous conseillons de faire figurer l'expression de calcul de l'élément dérivé.

Justification

• Justification 10 : Le guide 10 permet de vérifier le fait que l'élément est effectivement un élément dérivé. Ainsi la vérification de la règle lien 11 et de la règle 147 permet de s'assurer que tous les éléments nécessaires à son calcul sont connus et qu'il existe une manière cohérente de les assembler pour obtenir une valeur cohérente de l'élément dérivé.

3.5.3 Refine Stereotype

Contexte Une relation de dépendance « `refine` » indique que la cible décrit la source de façon plus détaillée.

Règles de cohérence

• Règle 148 : Les types des éléments mis en jeu par une relation de dépendance de stéréotype « `refine` » doivent être cohérents. [Nouvelle règle]

• Règle 149 : L'ensemble des caractéristiques structurelles et comportementales de l'ensemble des éléments sources doivent être présents dans l'ensemble des éléments cibles. [Nouvelle règle]

• Règle 150 : Aucun circuit formé par une relation de dépendance de stéréotype « `refine` » ne doit apparaître. [Nouvelle règle]

3.5.4 Trace Stereotype

Contexte Une dépendance stéréotypée « `trace` » indique une relation historique ou relative au processus de développement entre deux éléments qui représentent le même concept, sans qu'il existe des règles spécifiques pour dériver l'un à partir de l'autre.

Règles de cohérence Pas de règle trouvée

3.6 Realization

Contexte Une relation de réalisation est une sorte de relation d'abstraction et donc une sorte de relation de dépendance.

Une relation de dépendance associée au mot clé « `realize` » est une relation entre deux ensembles d'éléments, dont l'un représente une spécification (la cible) et l'autre représente l'implantation (la source).

Règles de cohérence

• Règle 151 : Aucun circuit formé par une relation de dépendance de stéréotype « realize » ne doit apparaître. [Nouvelle règle]

• Règle 152 : Les relations de réalisation apparaissent uniquement à trois occasions : entre les interfaces et les classes ou les composants qui les réalisent, entre les cas d'utilisation et les collaborations des classes qui les réalisent et entre les composants et les classes qui les réalisent. [tirée de [6] p.25] [7]

3.7 Substitution

Contexte Une relation de substitution entre deux classificateurs indique que le classificateur source est conforme à la spécification du classificateur cible. Ceci implique que les instances du classificateur source peuvent remplacer à l'exécution les instances du classificateur cible quand des instances du classificateur cible sont attendues.

[7] voir d'où vient cette règle -> il y a rien page 25 du guide utilisateur

Une relation de substitution est notée par une relation de dépendance associée au mot clé « `substitute` ».

Règles de cohérence

• Règle 153 : Une relation de substitution a exactement un classificateur source. [Règle dérivée du méta-modèle]

• Règle 154 : Une relation de substitution a exactement un classificateur cible. [Règle dérivée du méta-modèle]

• Règle 155 : Les interfaces implantées par le classificateur cible doivent être implantées par le classificateur source (le substituant). [Nouvelle règle]

• Règle 156 : Tout port que le classificateur cible possède doit avoir un port correspondant dans le classificateur source. [Nouvelle règle]

• Règle 157 : Dans le cas où la relation de substitution met en relation deux classes, l'ensemble des attributs, des opérations et de réceptions de la classe cible doivent apparaître dans la classe source. [Nouvelle règle]

3.8 Usage

Contexte Une relation d'utilisation est une sorte de relation de dépendance.

Une utilisation (*usage* en anglais) est une relation dans laquelle l'élément source requiert la présence de l'élément cible (ou ensemble d'éléments cibles).

Une relation d'utilisation est notée avec le mot clé « use » associé à une relation de dépendance.

Les stéréotypes « call », « create », « instantiate », « responsibility » et « send » peuvent s'appliquer à une relation d'utilisation (auquel cas le mot clé « use » n'apparaît pas).

3.8.1 Call Stereotype

Contexte Une relation de dépendance d'utilisation de stéréotype « call » spécifie que l'opération source ou une opération de la classe source invoque l'opération cible ou une opération de la classe cible.

Règles de cohérence

• Règle 158 : La source d'une relation d'appel doit être soit une opération soit une classe. [tirée de [7] p.593]

• Règle 159 : La cible d'une relation d'appel doit être soit une opération soit une classe. [tirée de [7] p.593]

• Règle 160 : L'opération ou la classe cible doit être visible par la source de la relation. [Nouvelle règle]

• Règle 161 : Dans le cas où la source de la relation est une classe, cette classe doit posséder au moins une opération. [Nouvelle règle]

• Règle 162 : Dans le cas où la cible de la relation est une classe, cette classe doit posséder au moins une opération visible par la source de la relation. [Nouvelle règle]

• Règle 163 : Une dépendance de stéréotype « `call` » implique que la source de la relation fait appel au moins une fois à la cible de la relation. Cet appel doit être visible

dans un des diagrammes d'interaction (diagrammes de séquence, de communication). [Nouvelle règle][8]

Remarque Dans le cas contraire, le modèle n'est pas incohérent mais la relation « d'appel » est aberrante.

3.8.2 Create Stereotype

Contexte Une relation de dépendance d'utilisation de stéréotype « `create` » spécifie qu'une instance du classificateur source de l'opération crée au moins une instance du classificateur cible.

Règles de cohérence

• Règle 164 : Ce type de relation met en jeu exactement un classificateur source et un classificateur cible. [Nouvelle règle]

• Règle 165 : Le classificateur cible ne doit pas être abstrait. [Nouvelle règle]

• Règle 166 : Le classificateur cible ne doit ni être un signal, ni un acteur, ni un cas d'utilisation. [Nouvelle règle]

• Règle 167 : Si l'objet créé (instance d'une classe B) appartient à une classe A (spécifié par une relation de composition entre les classes A et B), alors l'objet créateur doit être une instance de A ou doit pouvoir appartenir à la classe A. [Nouvelle règle][9]

3.8.3 Instantiate Stereotype

Contexte Une dépendance d'utilisation de stéréotype « `instantiate` » indique que des opérations du classificateur source créent des instances du classificateur cible.

Règles de cohérence

• Règle 168 : Le classificateur cible d'une relation de dépendance « `instantiate` » ne doit pas être abstrait. [Nouvelle règle]

• Règle 169 : Le classificateur source doit posséder au moins un opération. [Nouvelle règle]

3.8.4 Send Stereotype

Contexte Une relation de dépendance d'utilisation de stéréotype « `send` » spécifie que l'opération source envoie le signal cible.

Règles de cohérence

• Règle 170 : La source d'une dépendance stéréotypée « send » doit être une opération. [tirée de [7] p.595]

• Règle 171 : La cible d'une dépendance stéréotypée « send » doit être un signal. [tirée de [7] p.595]

• Règle 172 : Le signal cible doit être visible par l'opération source. [Nouvelle règle]

[8] inter-diagramme classes-interactions

[9] inter-diag, diag classe<=> diag séquence

3.9 Association

Contexte UML permet de réaliser des associations entre plusieurs classificateurs. Ce sont des relations sémantiques entre deux classificateurs ou plus qui impliquent des connexions entre leurs instances.

Nous pouvons distinguer les associations n-aires des associations binaires (qui sont un cas particulier des associations n-aires pour n=2). Les associations binaires mettent en relation deux classificateurs exactement alors que les associations n-aires peuvent mettre en relation un nombre indéfini de classificateurs.

En 3.9.1, nous présentons les modèles d'erreurs ou de fautes qui s'appliquent à toutes les associations, en 3.9.2 ceux qui s'appliquent uniquement aux associations n-aires avec n>2, en 3.9.3 les modèles d'erreurs ou de fautes qui affectent uniquement les associations binaires et en 3.9.4 les règles concernant la spécialisation d'association.

Remarque Il est possible de créer des classes d'association (cf. section 2.24).

3.9.1 Règles communes à toutes les associations

Contexte Une association permet d'exprimer une relation potentielle entre instances de classificateurs. Ceci est exprimé en mettant en association des classificateurs.

Règles de cohérence

• Règle 173 : Si l'association est une classe d'association (cf. 2.24), le nom optionnel de l'association doit être le même que celui de la classe d'association. [Nouvelle règle]

• Règle 174 : Une association met en relation des classificateurs exclusivement. [Règle dérivée du méta-modèle]

• Règle 175 : Toute association a au moins deux possessions (qui au niveau modèle sont représentées par des fins d'association). [Règle dérivée du méta-modèle]

• Règle 176 : Toute possession qui est une fin d'association doit avoir un type et ce type doit être un classificateur. D'un point de vue modèle ceci veut dire qu'une fin d'association est forcément reliée à un classificateur. [Nouvelle règle]

Remarque Il est possible qu'une association mette en jeu uniquement un classificateur avec lui-même.

• Règle lien 12 : Si l'association est une classe d'association, elle doit respecter les règles des classes (cf. 2.12) et des associations (cf. 3.9).

• Règle 177 : Si une possession (*property* en anglais) (qui représente une fin d'association) est contenue par une classe, associée avec une association binaire, et que l'autre fin de l'assocation est aussi contenue par une classe, alors `opposite` donne l'autre fin. [[7] p.90] [Règle sur le méta-modèle]

• Règle 178 : Si une fin d'association est une classe alors les autres fins de cette association doivent aussi être des classes. [Nouvelle règle]

• Règle 179 : L'attribut `endType` est dérivé des types reliés par les fins d'associations. [[7] p.81] [Règle sur le méta-modèle]

3.9.1.1 Contrainte XOR

Contexte La contrainte {Xor} est une contrainte prédéfinie entre deux associations. Elle exprime le fait que les deux associations ne peuvent pas être instanciées en lien (*link* en anglais) en même temps sur le même objet.

Règles de cohérence

• Règle 180 : Toute contrainte {Xor} doit relier au moins deux associations. [Nouvelle règle]

• Règle 181 : Lorsque plusieurs associations sont reliées par une même contrainte {Xor}, ces associations doivent toutes avoir un classificateur en commun. [Nouvelle règle]

• Règle 182 : Les bornes inférieures des multiplicités des associations du coté du classificateur qui n'est pas commun aux deux associations doivent être 0. [tirée de [4]]

Remarque La figure 3.3 montre un exemple de non respect de de la règle 182.

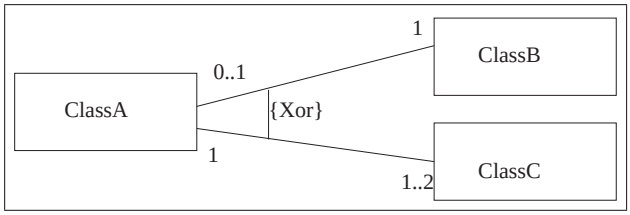

Fig. 3.3 – Exemple de non respect de la règle 182

• Règle 183 : Deux associations mutuellement exclusives, c'est-à-dire contraintes par Xor ne doivent pas être la spécification de deux liens reliés à un même objet. [Nouvelle règle]

Remarque Afin d'expliciter cette règle, la figure 3.4 montre comment spécifier une contraire XOR entre deux associations, la figure 3.5 montre un exemple où la contrainte XOR n'est pas respectée alors que la figure 3.6 montre un exemple où la contrainte est respectée.

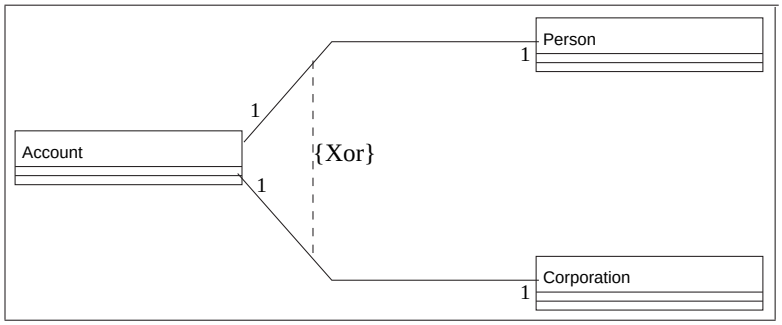

Fig. 3.4 – Spécification d'une contrainte XOR entre deux associations

3.9.1.2 Relation association - lien

Contexte Les instances des associations sont des liens. Il existe donc une relation classificateur - instance entre une association et un lien.

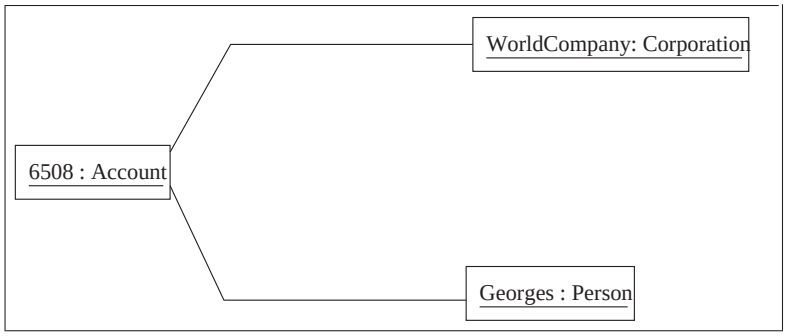

FIG. 3.5 – Exemple de non-respect de la contrainte XOR

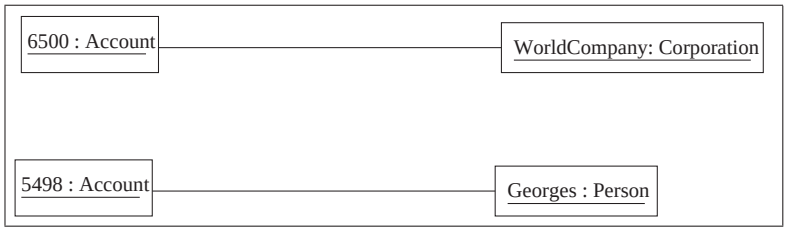

FIG. 3.6 – Exemple de respect de la contrainte XOR

Règles de cohérence

• RÈGLE 184 : Le nombre d'instances mises en relation par des liens doit respecter la multiplicité spécifiée pour l'association. [Nouvelle règle]

Remarque Dans certains cas, cette règle pose problème. Comme expliqué en [5], il n'est pas toujours possible d'implanter un modèle en respectant les contraintes. Considérons l'exemple de la figure 3.7.

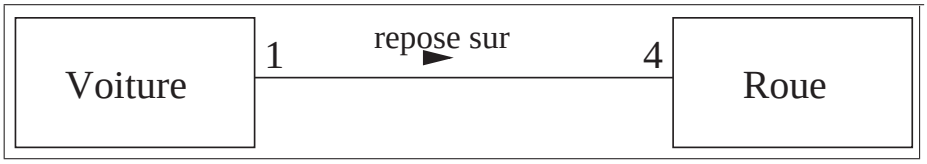

FIG. 3.7 – Multiplicité

Cet exemple met en valeur l'impossibilité lors de certaines étapes dynamiques de respecter les contraintes imposées par le modèle. En effet, créer une Voiture requiert d'avoir créé quatre Roues alors que créer une Roue requiert d'avoir créé une Voiture.

La solution envisagée par [5] est de modifier la multiplicité associée aux Roues en mettant 0..4. Une autre possibilité serait de ne prendre en compte cette règle que dans les états stables de l'objet car elle peut s'avérer fausse dans les étapes dynamiques d'initialisation, de terminaison du système et lors de changements de liaisons. Dans notre cas, cette règle pourrait par exemple être transgressée durant l'exécution des méthodes `CreerVoiture()` ou `ChangerRoue()`.

3.9.1.3 Décorations des fins d'association

Contexte Les fins d'association peuvent se voir attribuer un certain nombre de décorations. Cette partie traite des décorations qui peuvent être présentes aussi bien sur des associations binaires que n-aires (avec n>2), c'est-à-dire les multiplicités, les noms de rôles attribués aux fins d'associations et les propriétés (traitées en 3.9.1.4).

Règles de cohérence

• Règle 185 : Tout nom de rôle ne doit apparaître qu'une seule fois dans une association. [Nouvelle règle]

• Règle lien 13 : Toute multiplicité qui figure sur une fin d'association doit respecter les règles énoncées en 2.5.

3.9.1.4 Propriétés associées aux fins d'association

Contexte Les propriétés suivantes peuvent être associées aux fins d'association : `{subsets <property-name>}`, `{redefines <end-name>}`, `{union}`, `{ordered}`, `{bag}` ou `{sequence}` (équivalent à `{seq}`).

La propriété `{subsets <property-name>}` associée à une fin d'association indique que l'ensemble des classificateurs décrit par cette fin d'association est un sous-ensemble de l'ensemble décrit par une association plus générale. La figure 3.8 montre un exemple où une fin d'association est associée à cette propriété.

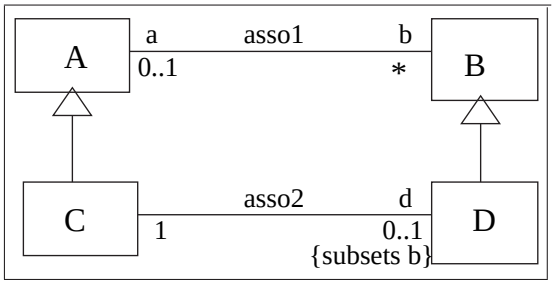

Fig. 3.8 – Fin d'association associé à la propriété {subsets <property-name>}

La propriété `{redefines <end-name>}` associée à une fin d'association indique que l'ensemble des classificateurs décrit par cette fin d'association redéfinit l'ensemble décrit par une association plus générale. La figure 3.9 montre un exemple d'utilisation de cette propriété tiré du méta-modèle d'UML. Dans cet exemple, il est spécifié que tous les éléments contenus dans la méta-classe `Package` sont des `PackageableElement`.

La propriété `{ordered}` spécifie qu'un classement peut être établi entre les instances en relation.

La propriété `{union}` spécifie que l'ensemble décrit par la fin d'association est calculé en faisant l'union de ses sous-ensembles.

La propriété `{bag}` spécifie que l'ensemble décrit par la fin d'association est une collection qui permet à un élément d'apparaître plusieurs fois et qui n'est pas ordonnée.

Les propriétés `{sequence}` et `{seq}` spécifient que l'ensemble décrit par la fin d'association est une séquence, c'est-à-dire un ensemble ordonné où les doublons peuvent apparaître.

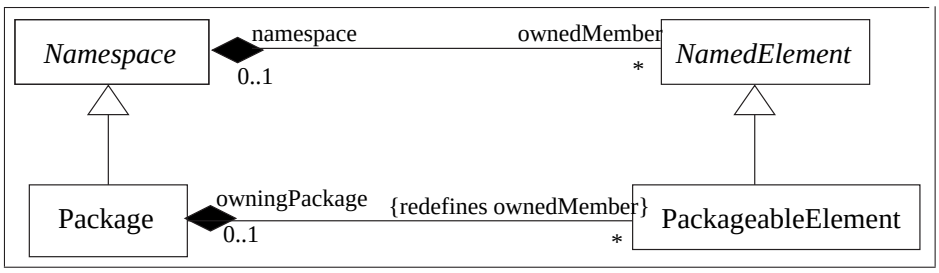

FIG. 3.9 – Exemple d'utilisation de la propriété {redefines} dans le méta-modèle d'UML

Par défaut un ensemble spécifié par une fin d'association n'est pas ordonné et ne contient pas de doublon.

Remarque Les règles communes aux attributs et aux fin d'associations sont présentées en section 2.15.
[10]

Règles de cohérence

• RÈGLE 186 : Seules les fins d'associations qui sont navigables peuvent être marquées comme `readOnly`. [[7] p.91] [Règle sur le méta-modèle]

• RÈGLE 187 : Lorsqu'une possession (nommée A) représentant une fin d'association est associée à la propriété `{subsets <property-name>}`, alors l'association plus "générale" doit avoir le même nombre de possessions que l'association de la possession A. [Nouvelle règle]

• RÈGLE 188 : Lorsqu'une possession représentant une fin d'association est associée à la propriété `{subsets <property-name>}`, alors chaque possession de l'association doit correspondre à une possession de l'association sur-ensemble et avoir des décorations d'agrégation ou de composition identiques. [Nouvelle règle]

• RÈGLE 189 : Lorsqu'une possession représentant une fin d'association est associée à la propriété `{subsets <property-name>}`, alors les autres possessions de l'association doivent toutes correspondre à une possession de l'association sur-ensemble et être associées à une multiplicité dont la borne supérieure est inférieure à la borne supérieure de la multiplicité associée à la possession plus générale. [Nouvelle règle]

• RÈGLE 190 : Lorsqu'une possession représentant une fin d'association est associée à la propriété `{subsets <property-name>}`, alors chaque possession de l'association doit correspondre à une possession de l'association sur-ensemble et être typée par un sous-type du type de la possession générale. [Nouvelle règle]

• RÈGLE 191 : Lorsqu'une possession (nommée A) représentant une fin d'association est associée à la propriété `{redefines <property-name>}`, alors l'association plus "générale" doit avoir le même nombre de possessions que l'association de la possession A. [Nouvelle règle]

• RÈGLE 192 : Lorsqu'une possession représentant une fin d'association est associée à la propriété `{redefines <property-name>}`, alors les autres possessions de l'association

[10] Guide pour empêcher l'utilisation du par défaut ?

doivent correspondre à une possession de l'association sur-ensemble et être typées par un sous-type du type de la possession générale. [Nouvelle règle]

• Règle 193 : Lorsqu'une possession représentant une fin d'association est associée à la propriété `{redefines <property-name>}`, alors chaque possession de l'association doit correspondre à une possession de l'association sur-ensemble et être associée à une multiplicité qui est un sous-ensemble de la multiplicité associée à la possession plus générale. [Nouvelle règle]

• Règle 194 : Lorsqu'une possession représentant une fin d'association est associée à la propriété `{redefines <property-name>}`, alors chaque possession de l'association doit correspondre à une possession de l'association sur-ensemble et avoir des décorations d'agrégation ou de composition identique ou plus forte. [Nouvelle règle]

• Règle 195 : Une fin d'association navigable ne peut être redéfinie que par une fin d'association navigable et ne peut être le sur-ensemble que d'une fin d'association navigable. [[7] p.91]

3.9.1.5 Qualifier

Contexte Un qualificateur (*qualifier* en anglais) est un attribut ou une liste d'attributs dont les valeurs servent à partitionner l'ensemble des instances reliées à une autre instance par l'intermédiaire d'une association. Les qualificateurs sont des attributs de l'association.

Dans le méta-modèle, un qualificateur est représenté par une possession qui appartient à une autre possession qui représente une fin d'association.

Règles de cohérence

• Règle 196 : Une possession qui représente un qualificateur ne peut appartenir qu'à une possession qui représente une fin d'association. [Nouvelle règle][Règle sur le méta-modèle]

• Règle 197 : Lorsqu'une fin d'association possède un qualificateur, l'autre fin doit être navigable. [Nouvelle règle]

Remarque Puisque les fins d'association ne peuvent être navigables que lorsque l'association est binaire (cf. règle 199), les qualificateurs ne peuvent apparaître que dans le cas d'associations binaires.

3.9.2 Décorations des fins d'associations n-aires quand n>2

Contexte Les règles supplémentaires qui s'appliquent dans le cas d'associations N-aires quand N>2 portent sur les décorations des fins d'associations. En effet, certaines décorations peuvent apparaître pour des fins d'associations binaires mais sont interdites pour des fins d'associations n-aires (avec n>2). Les règles qui suivent portent sur ces restrictions.

Règles de cohérence

• Règle 198 : Dans le cas d'association n-aire où n>2, les agrégations composites (cf. section 3.9.3.2) ne peuvent pas apparaître. [tirée de [7] p.90]

• Règle 199 : Dans le cas d'association n-aire où n>2, aucune fin d'association ne doit être navigable. [tirée de [7] p.81] [11]

3.9.3 Association Binaire

Les règles qui suivent ne concernent que les associations binaires.

3.9.3.1 Navigabilité

Contexte La navigabilité d'une association indique le sens de passage autorisé d'un classificateur à un autre.

Règles de cohérence

• Règle 200 : Seuls des échanges de messages dans les sens navigables sont possibles. [Nouvelle règle][12]

• Règle 201 : Seules des classes peuvent être reliées par des associations navigables. [Règle dérivée du méta-modèle]

Guides

• Guide 11 : Nous conseillons de spécifier au maximum les navigabilités.

Justification

• Justification 11 : Le guide 11 permet de s'assurer qu'il n'y a d'échange de message pour des associations non navigables (cf. règle 200).

3.9.3.2 Aggregation

Contexte Une agrégation est une forme d'association spéciale qui définit une relation de tout-partie entre agrégat et composant.

Remarque Il est possible qu'un même composant fasse partie de plusieurs agrégats. Ici le terme composant exprime l'élément agrégé et n'est pas la traduction de *component*.

Règles de cohérence

• Règle 202 : Pour une association, seule une fin d'association peut avoir une agrégation de type `composite` ou `share`. [Nouvelle règle]

• Règle 203 : Il est interdit de spécifier une relation d'agrégation pour des associations n-aires lorsque n>2. [Nouvelle règle][13]

• Règle 204 : Aucun circuit faisant intervenir des liens qui correspondent à des associations dont le type d'agrégation est `share` ou `composite` ne doit apparaître au

[11] Vérifier que ce n'est pas rectifié dans la nouvelle norme

[12] inter-diagramme classe - interactions

[13] déjà dans association n-aire, on le remet ?->j'ai un doute sur la validité de cette règle : est-ce toujours vrai et si oui d'où vient-elle ?

niveau objet. [Nouvelle règle][14]

• Règle 205 : Aucun circuit faisant intervenir des associations dont le type d'agrégation est `share` ou `composite` et dont la multiplicité est 1 ne doit apparaître. [Nouvelle règle]

3.9.3.3 Composition

Contexte La composition (*composition* en anglais) est une forme d'agrégation forte. Une relation de composition suppose que la partie composante (cible) appartient exclusivement à la partie composée (source) et entraîne des contraintes sur la vie du composant par rapport à la vie du composé.

Règles de cohérence

• Règle lien 14 : La composition est une forme d'agrégation forte et doit donc répondre aux règles exposées en 3.9.3.2.

• Règle 206 : Une multiplicité d'une agrégation composite du coté du classificateur composé ne doit pas avoir sa borne supérieure plus grande que 1. [[7] p.90]

• Règle 207 : Toute instance de la classe composante doit appartenir à une et une seule instance de la classe composée. [Nouvelle règle][15]

• Règle 208 : Lorsque plusieurs classes composent une même classe et que ces classes sont mises en relation via une association, les objets (instances des classes composantes) qui sont mis en relation par un lien (instance de l'association) doivent tous appartenir au même objet composite (qui est l'instance de la classe composée). [16] En d'autres termes, dans ce cas, le lien doit mettre en relation des objets (instances des classes composantes) qui appartiennent au même objet (instance de la classe composée). [Nouvelle règle][17]

• Règle 209 : Lorsque deux classes sont reliées par une relation de composition, seul l'objet composé ou un des objets qu'il englobe (par héritage) peuvent créer ou détruire les objets composants. [Nouvelle règle][18]

• Règle 210 : La valeur de `isComposite` est vraie uniquement si `aggregation` est `composite`. [[7] p.91] [Règle sur le méta-modèle]

3.9.4 Spécialisation d'association

Contexte Il est possible de spécifier des relations de généralisation (héritage) entre des associations.

Une relation de généralisation entre des associations peut être représentée en reliant deux associations par une flèche de généralisation.

[14] inter-diag : classes - objets

[15] inter-diag : classes - objets

[16] !!! à approfondir : Est-ce que qu'il est possible qu'une classe composante soit en relation avec une classe qui ne fait pas partie de la classe composée ?

[17] inter-diag : classes - objets

[18] inter-diagramme : diag de classe <=> diag d'interaction

Règles de cohérence

• Règle 211 : Une association qui spécialise une autre association doit avoir le même nombre de fins d'association. [[7] p.81]

• Règle 212 : Quand une association spécialise une autre association, chaque fin de l'association spécifique correspond à une fin de l'association générale. Les fins d'association spécifiques doivent atteindre des éléments de même type que les fins d'association générales ou des sous-types de ces fins d'association générales. [[7] p.81]

3.10 Package Merge

Contexte Un interclassement de paquetage (*package merge* en anglais) correspond à un inter-classement où le contenu du paquetage cible est interclassé avec le contenu du paquetage source.

C'est un mécanisme qui est utilisé lorsque des éléments du même nom sont censés représenter le même concept même s'ils ne se trouvent pas dans le même paquetage.

Un interclassement de paquetage est représenté par une relation de dépendance associée au mot clé « `merge` ».

Règles de cohérence

• Règle 213 : Une relation d'interclassement de paquetage doit avoir comme source un (et un seul) paquetage. [Règle dérivée du méta-modèle]

• Règle 214 : Une relation d'interclassement de paquetage doit avoir comme cible un (et un seul) paquetage. [Règle dérivée du méta-modèle]

• Règle 215 : Aucun circuit de ce type de relation ne doit apparaître. [Nouvelle règle]

3.11 Template Binding

Contexte Une délimitation d'éléments génériques (*template binding* en anglais) consiste à recopier le contenu de l'élément générique dans l'élément délimité en remplaçant tous les éléments génériques par les paramètres de substitution. Une telle relation fait donc la correspondance entre les paramètres génériques et les paramètres de substitution (utilisés dans l'élément borné).

Dans le cas où aucun paramètre de substitution n'est spécifié pour un paramètre générique, l'élément par défaut (si spécifié) est utilisé. Si aucune valeur par défaut n'est spécifiée, l'élément borné est lui aussi générique.

Une relation de délimitaion d'élément générique est notée par le symbole d'une relation de dépendance associé au mot clé « `bind` ».

Règles de cohérence

• Règle 216 : Chaque paramètre de substitution doit se référer à un paramètre de la signature générique (contenue par l'élément générique). [[7] p.547]

• Règle 217 : Une délimitation d'élément générique contient au maximum un paramètre de substitution pour chaque paramètre générique. [[7] p.547]

• Règle 218 : Le paramètre de substitution doit être compatible avec le paramètre générique. [[7] p.550]

• Règle 219 : Une relation de délimitation d'élément générique est une relation qui doit mettre en jeu exactement un élément générique (cible de la relation) et un élément borné (source de la relation). [Règle dérivée du méta-modèle]

Remarque Un point de variation sémantique fait ressortir le fait que dans le cas où tous les paramètres génériques ne sont pas bornés, un élément borné peut aussi être un élément générique.

Chapitre 4

Diagrammes

4.1 Class Diagram

Contexte Les diagrammes de structure se focalisent sur des aspects structurels particuliers tels que les relations entre paquetages (diagramme de paquetages, *package diagram* en anglais), en montrant des spécifications d'instances (diagramme d'objets, *object diagram* en anglais) et les relations entre classes (diagramme de classes, *class diagram* en anglais).

Règles de cohérence

• Règle 220 : Les nœuds graphiques contenus dans un diagramme de classes ne peuvent être que : [[7] p.131]

- une classe ;
- une interface ;

• Règle 221 : Les chemins graphiques pouvant être contenus dans un diagramme de classes ne peuvent être que : [[7] p.131]

- une association de type agrégation, composite ou simple ;
- une dépendance ;
- une généralisation ;
- une réalisation.

• Règle 222 : Le seul nœud graphique pouvant être contenu dans un diagramme de paquetages est le paquetage. [[7] p.131]

• Règle 223 : Les chemins graphiques pouvant être contenus dans un diagramme de paquetages ne peuvent être que : [[7] p.131]

- une importation de paquetage (*packageImport* en anglais), dépendance avec le mot clé « import » ou « access » ;
- une importation d'un élément (*elementImport* en anglais), dépendance avec le mot clé « import » ;
- un interclassement de paquetage (*packageMerge* en anglais), dépendance avec le mot clé « merge » ;
- une dépendance ;

• Règle 224 : Le seul nœud graphique pouvant être contenu dans un diagramme d'objets est une instance de spécification. [[7] p.131]

• Règle 225 : Les seuls chemins graphiques pouvant être contenus dans un diagramme d'objets ne peuvent être que des liens. [[7] p.131]

Deuxième partie

Diagramme de composants

Chapitre 5

Éléments

5.1 Component

Contexte Un composant représente une partie modulaire d'un système qui encapsule son contenu et qui est remplaçable au sein de son environnement.

Un composant définit un comportement en terme d'interfaces fournies ou requises. Un composant est donc défini par ses interfaces requises et fournies. Des fonctionnalités plus importantes peuvent être réalisées par la réutilisation de composants au sein d'un composant englobant ou d'un ensemble de composants, et en connectant leurs interfaces requises ou fournies entre elles.

La définition d'interfaces requises ou fournies d'un composant peut mettre en jeu des ports.

Un composant peut être substitué par un autre composant en représentant une dépendance marquée avec le mot clé « substitute ».
[1]

Règles de cohérence

• Règle 226 : Un composant peut être substitué par un autre composant seulement s'il est de même type, c'est-à-dire si les interfaces fournies et requises par le nouveau composant sont conformes aux interfaces de l'ancien composant. [tirée de [7] p.136] [2]

• Règle 227 : Une interface fournie par un composant doit être soit implantée par le composant lui-même, soit par un des classificateurs qui le réalisent, ou être le type d'un port fourni du composant. [tirée de [7] p.136]

[1] Il existe des composants logiques et des composants physiques, regarder en détail, et surtout quel lien entre composant physique et artifact ?

[2] dépendance « substitute » entre 2 composants

Remarque Le type d'un port est l'interface ou les interfaces fournies par le port.

• Règle 228 : Si cela est précisé, un composant ne peut être réalisé que par des classes ou des collaborations.[Nouvelle règle][3]

• Règle 229 : Les éléments possédés par un composant doivent être des éléments empaquetables. [Règle dérivée du méta-modèle][4]

• Règle 230 : Les artifacts qui implantent des composants doivent leur être connectés soit par contenance physique soit par une relation « implement », qui est une instance de la méta-association entre composant et artifact. [tirée de [7] p.141] [5]

• Règle 231 : Tout composant a au moins une interface fournie. [Nouvelle règle]

• Règle 232 : Dans le cas où un comportement tel une machine à état de protocole est attaché à une interface, un port ou au composant lui-même pour définir la vue externe du composant plus précisément en rendant explicites les contraintes dynamiques dans la séquence des appels d'opérations, les événements correspondants à une opération ou à un signal doivent pouvoir être réalisés par le composant ou délégués via une interface requise. [Nouvelle règle][6]

Guides

[7]

[3]voir si ce sous-ensemble de classificateur est complet

[4]sous-ensemble des éléments empaquetables non défini dans la spec

[5]voir dans la derniere version de la spec comment restreindre la relation entre artifact et composant

[6]inter-diag Machine a états de protocole - composant

[7]guide de prévention : Pour représenter qu'une interface est requise par un composant, il est conseillé de faire figurer une relation de dépendance d'usage depuis le composant ou depuis un des classificateurs qui le réalisent vers l'interface, ou d'expliciter le type d'un port requis du composant.
justif : cela permet de rendre explicite le lien entre le composant et l'interface

Chapitre 6

Relations

6.1 Connector (Component Diagram)

Contexte Les connecteurs vus dans cette section sont une spécialisation des connecteurs vus dans la section 9.2.

Dans les diagrammes de composants, il existe deux types de connecteurs, les connecteurs de délégation (*Delegation Connector* en anglais) détaillés en section 6.1.1 et les connecteurs d'assemblage (*Assembly Connector* en anglais) détaillés en section 6.1.2.

6.1.1 Delegation connector

Contexte Un connecteur de délégation est un connecteur qui relie le contrat externe d'un composant (spécifié par ses ports) à la réalisation de ce comportement par les parties internes du composant.

Remarque La délégation suggère que les messages concrets et le flot de signaux apparaitra entre les ports connectés, possiblement sur plusieurs niveaux de décomposition hiérarchique. Il est à noter que ce flot de données n'est pas toujours réalisé dans tous les environnements ou toutes les implantations (c'est-à-dire qu'il ne peut être considéré que seulement en phase de conception).

Règles de cohérence

• Règle 233 : Un connecteur de délégation doit uniquement être défini entre des interfaces utilisées ou des ports de même type, c'est-à-dire entre deux ports ou interfaces fournis (*provided* en anglais) par le composant ou entre deux ports ou interfaces requis (*required* en anglais) par le composant. [[7] p.143]

• Règle 234 : Si un connecteur de délégation est défini entre une interface fournie ou un port et un classificateur interne au composant, alors ce classificateur doit avoir une relation « implement » avec l'interface de ce port.[[7] p.143] [1]

• Règle 235 : Si un connecteur de délégation est défini entre une interface source ou un port et une interface cible ou un port, alors l'interface cible doit supporter un sous-ensemble des signatures compatible des opérations de l'interface source ou du port.

[1] pas clair, dans la spec actuelle, il s'agit d'interface utilisée

[[7] p.144]

• Règle 236 : Dans un modèle complet, si un port source possède des connecteurs de délégation vers un ensemble de ports cibles délégués, alors l'union des interfaces de ces ports cibles doit être compatible (au niveau des signatures) avec l'interface qui type le port source. [[7] p.144]

• Règle 237 : Un connecteur de délégation doit relier le port d'un composant à une partie interne du composant.[Nouvelle règle]

6.1.2 Assembly Connector

Contexte Un connecteur d'assemblage est un connecteur entre deux composants qui définit qu'un composant fournit le service qu'un autre composant requiert.

Règles de cohérence

• Règle 238 : Un connecteur d'assemblage doit uniquement être défini à partir d'une interface requise ou d'un port vers une interface fournie ou un port. [[7] p.144]

• Règle 239 : L'ensemble des services fournis par le port ou l'interface doit être un sur-ensemble de l'ensemble des services requis. [Nouvelle règle]

Remarque Les services requis ou fournis sont la liste des opérations présentes dans les interfaces.

Chapitre 7

Diagrammes

7.1 Components Diagram

Contexte Le diagramme de composant spécifie un ensemble de constructions qui peuvent être utilisées pour définir des systèmes logiciels de taille et de complexité arbitraire. Un composant est une unité modulaire dont les interfaces sont bien définies et qui est remplaçable dans son environnement. Les concepts mis en jeu dans ce diagramme traitent du domaine du développement basé composant (*component-based development* en anglais) et de la structuration de système basée composant. Le composant est modélisé au travers du cycle de vie et est successivement raffiné.

Un aspect important du développement basé composant est la réutilisation de composant construits précédemment. Un composant peut être considéré comme une unité autonome à l'intérieur d'un système ou d'un sous-système. Il a une ou plusieurs interfaces requises ou fournies (potentiellement exposée au travers de ports), et son intérieur est caché et inaccessible autrement qu'au travers de ses interfaces.

Les règles suivantes portent sur les éléments qui peuvent être contenus dans un diagramme de composants.

Règles de cohérence

• Règle 240 :

Les nœuds contenus dans un diagramme de composants ne peuvent être que :

- des composants ;
- des interfaces ;
- des classes ;
- des artifacts ;
- des ports. [[7] p.149]

• Règle 241 : Les chemins graphiques qui peuvent apparaître dans un diagramme de composants sont : [[7] p.149]

- des connecteurs d'assemblage ;
- des connecteurs de délégation ;
- des réalisations ;
- des relations de d'implantation ;
- des dépendances d'utilisation.
- des dépendances ;

[1]

[1]distinguer lorsque on représente un diagramme de composants avec ou sans structure interne ? Cela aboutirait a 2 règles :
regle1 : Les seuls chemins graphiques pouvant être contenus dans un diagramme de composants sont le connecteur d'assemblage et des relations de dépendance de type implementation, realization et usage. (spec149) et

regle2 : Lorsque la structure interne d'un composant est détaillée, les seuls chemins graphiques pouvant être contenus à l'intérieur du composant sont l'association, le connecteur d'assemblage et de délégation. (new)

Troisième partie

Diagramme de structures composites

Chapitre 8

Éléments

8.1 Structured Classifier

Contexte Un classificateur structuré est une métaclasse abstraite qui représente tous les classificateurs dont le comportement peut être entièrement ou partiellement décrit par la collaboration d'instances contenues ou référencées.

Les métaclasses qui sont des classificateurs structurés sont les collaborations et les classificateurs encapsulés qui incluent les classes (cf. annexe A.9).

Les multiplicités sur les caractéristiques structurelles et les fins de connecteurs indiquent le nombre d'instances (objets et liens) qui peuvent être créés.

La figure 8.1 (tirée de [7]) donne deux moyens de modéliser le fait qu'une voiture a quatre roues et que ces quatre roues sont reliées deux par deux par des axes ; contrairement à la figure 8.1 (i), la figure 8.1 (ii) utilise les multiplicités.

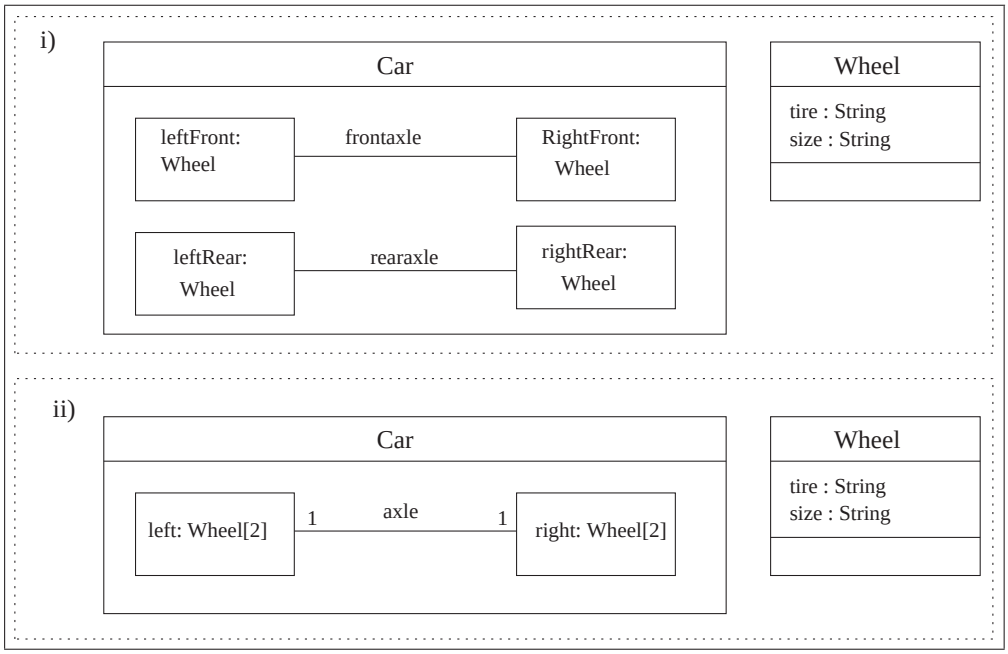

Fig. 8.1 – Utilisation ou non des multiplicités dans un classificateur structuré

Les multiplicités sur les possessions indiquent le nombre d'instances qui sont créées et les multiplicités sur les fins de connecteur indiquent le nombre de liens qui peuvent être créés pour chacune de ces instances.

D'autre part, UML offre la possibilité de représenter des instances de classes structurées.

Règles de cohérence

• Règle 242 : La chaîne de caractères d'un rôle dans une spécification d'instance doit suivre la syntaxe suivante : [[7] p.175]

```
{{ {[ name ['/'rolename]] | '/'rolename}':'classifiername
[{','classifiername}*]} | { name ['/'rolename] | '/'rolename}}
```

• Règle 243 : Les multiplicités des éléments connectés doivent être cohérentes, c'est-à-dire que le connecteur doit être instanciable. [[7] p.174]

Remarque La figure 8.2 montre une incohérence dans les multiplicités d'un classificateur structuré. En effet, ce schéma spécifie que chaque instance qui joue le rôle `a` doit être connectée à deux instances qui jouent le rôle `c` et que deux instances du rôle `a` doivent être créées. Il faut donc au minimum deux instances de rôle `c` pour que ces deux contraintes soient respectées. Or, ce schéma spécifie également qu'une seule instance qui joue le rôle `c` ne peut être créée, d'où une incohérence dans les multiplicités.

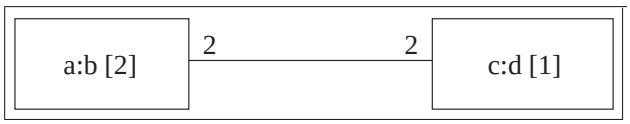

Fig. 8.2 – Incohérence dans l'utilisation des multiplicités dans un classificateur structuré

8.2 Port

Contexte Un port est une caractéristique structurelle d'un classificateur encapsulé (seules les classes et les composants peuvent contenir des ports) qui spécifie un point d'interaction entre le classificateur et son environnement ou entre un classificateur et ses parties internes. Les ports sont connectés aux parties internes du classificateur par des connecteurs au travers desquels des requêtes peuvent être faites pour invoquer les caractéristiques comportementales du classificateur.

Des interfaces requises ou fournies peuvent être associées aux ports afin de spécifier les opérations ou les réceptions (*reception* en anglais) respectivement attendues de l'environnement ou offertes par le classificateur.

Un port peut être redéfini lorsque le classificateur qui le contient est spécialisé.

Règles de cohérence

• Règle 244 : Les ports ne peuvent être créés ou détruits excepté pendant la création ou la destruction du classificateur contenant. [[7] p.168]

• Règle 245 : Les interfaces requises d'un port doivent être fournies par les éléments auxquels le port est connecté. [[7] p.168]

• Règle 246 : Un port qui en redéfinit un autre doit être associé à toutes les interfaces du port redéfini ou à des sous-types de ces interfaces. [tirée de [7] p.168]

• Règle 247 : Lorsqu'un port comportemental est associé à une interface fournie, alors, le classificateur composant doit pouvoir satisfaire l'interface avec ses propres opérations et réceptions. [Nouvelle règle]

Remarque Un port comportemental est noté en reliant le port à un symbole d'un état comme montré dans la figure 8.3. Dans ce cas, le port comportemental fournit l'interface `powertrain`.

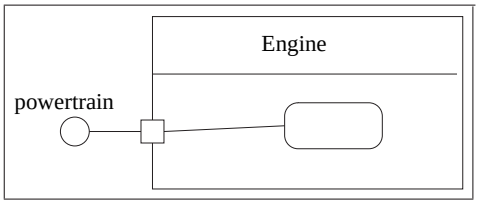

Fig. 8.3 – Représentation d'un port comportemental

• Règle 248 : Les interfaces requises ou fournies par le port doivent être des interfaces connues du classificateur qui contient le port. [Nouvelle règle]

• Règle 249 : Un port ne peut être possédé que par des classificateurs encapsulés (cf. annexe A.9). [Règle dérivée du méta-modèle]

Remarque préliminaire à la règle 250 : Lorsqu'une instance d'un classificateur est créée, les instances correspondantes à chacun de ses ports sont créées et gardées dans des *slots* spécifiés par les ports, en accord avec leurs multiplicités. Ces instances sont désignées par le terme de points d'interaction.

• Règle 250 : L'objet contenant un point d'interaction doit être une instance du classificateur qui réalise les interfaces fournies par le port dont le pint d'interactio est l'instance.[tirée de [7] p.168]

Guides • Guide 12 : Nous conseillons de préciser au maximum le fait qu'un port fournit ou requiert un interface.

• Justification 12 : Le guide 12 permet de détecter les erreurs associées aux règles 245 à 248.

8.3 Property (Composite Structures Diagram)

Contexte Dans les diagrammes de structure composite (*composite structure diagram* en anglais), une possession (*property* en anglais) représente un ensemble d'instances qui appartiennent à une instance de classificateur.

Vis-à-vis du classificateur qui la contient, une possession peut être de deux types. En effet, soit la possession est une partie (*part* en anglais) ce qui indique une composition, soit la possession représente un attribut non composite.

La figure 8.4 tirée de [7] montre un exemple de possession. La sous-figure (i) montre un classe `Car` qui a une association de composition de nom de rôle `rear` avec une classe `Wheel` et une association de nom de rôle `e` avec une classe `Engine`. La sous-figure (ii) exprime la même chose. Cependant, la sous-figure (ii) spécifie en plus que, dans le contexte particulier de la classe `Car`, les instances qui jouent le rôle `e` ne peuvent être connectées qu'à deux instances qui jouent le rôle `rear`, et que les instances qui jouent le rôle `e` et `rear` ne peuvent être liées que si elles jouent un rôle dans la même instance de la classe `Car`.

Remarque Dans la sous-figure (ii), le fait que la possession `Engine` n'est pas composite est indiqué par un rectangle aux cotés pointillés.

Les possessions sont contenues par des classificateurs structurés.

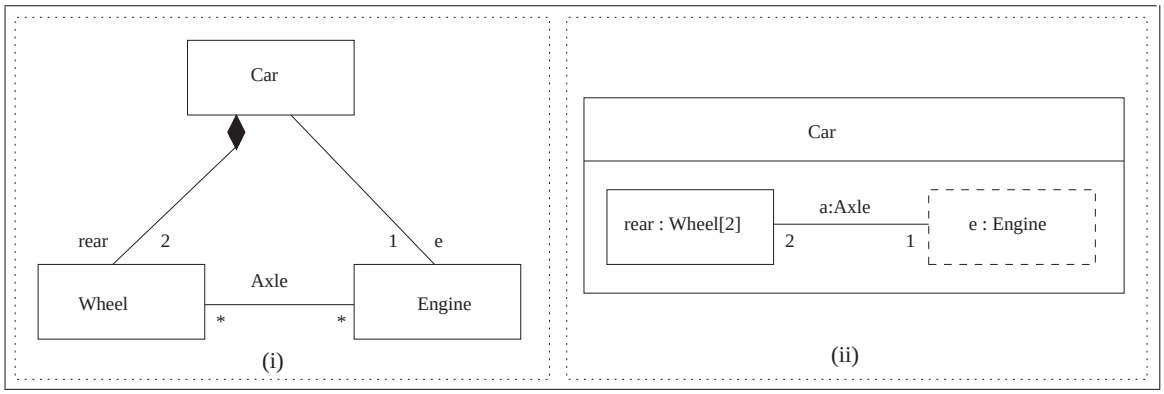

FIG. 8.4 – Possession composite ou non

Règles de cohérence

• RÈGLE 251 : Dans un diagramme de structures composites, si une possession A apparaît comme composite (A est une partie (*part* en anglais) de B), alors le méta-attribut `isComposite` de la possession A doit être vrai. Ceci implique dans le diagramme de classes, que si la classe A est reliée par une association à la classe B alors cette association doit être composite (la classe englobante étant la classe B). [Nouvelle règle][1]

• RÈGLE 252 : Une possession A qui n'est pas une partie de la classe B dans un diagramme de structures composites, doit être une possession dont le méta-attribut `isComposite` est faux. Ceci implique dans le diagramme de classes, que si la classe A est reliée par une association à la classe B (la classe englobante) alors cette association doit être une association navigable. [Nouvelle règle][2]

[1] inter-diagramme structures composites - classes

[2] inter-diagramme structures composites - classes

8.4 Collaboration

Contexte Les collaborations sont généralement utilisées pour expliquer comment des instances en coopération peuvent réaliser une tâche ou un ensemble de tâches. Chaque instance en collaboration joue un certain rôle au sein de la collaboration. Une collaboration ne fait apparaître que les aspects nécessaires à la compréhension et supprime le reste. Par exemple un objet peut jouer des rôles dans différentes collaborations qui vont chacune montrer des aspects différents de l'objet.
[3]

Les caractéristiques, les contenus des instances participantes à la collaboration, et les liens entre ces instances ne sont jamais tous requis dans une collaboration particulière. Ainsi, une collaboration peut être définie en termes de rôles typés par des interfaces.

Dans le cas de spécialisation de collaboration, les rôles de la collaboration générale peuvent être étendus dans la collaboration spécialisée.

Règles de cohérence

• Règle 253 : Lorsqu'une relation d'héritage met en relation deux collaborations, il faut que les types des différents rôles de la collaboration spécialisée soient conformes aux types correspondants de la collaboration générale. Pour cela, les classificateurs présents dans la collaboration spécialisée doivent posséder les caractéristiques des classificateurs de même rôle de la collaboration générale. [tirée de [7] p.158]

Remarque Ceci n'implique pas forcément qu'entre les classificateurs qui réalisent ces rôles, on ait des relations d'héritage correspondantes.

• Règle 254 : Tout élément connectable qui entre en jeu dans une collaboration doit avoir un rôle. [Nouvelle règle]

• Règle lien 15 : Tout rôle au sein d'une collaboration doit être unique. [Nouvelle règle](voir règle 4)

Guides

• Guide 13 : Nous conseillons de faire figurer dans la collaboration l'ensemble des propriétés nécessaires pour chacun des rôles.

• Justification 13 : Ce guide permet de détecter les erreurs associées à la règle 260.
Remarque Les propriétés nécessaires au rôle sont définies dans la collaboration.
[4]

Remarque Sur la figure 8.5 tirée de [7], la collaboration `Sale` est reliée au classificateur `BrokeredSale`, qui se trouve être ici aussi une collaboration. Avec cette représentation, les délimitations de rôle (*role bindings* en anglais, cf. section 9.1), sont montrées explicitement par des dépendances.

[3] A voir avec nouvelle spec et mots clés « represents » et « occurrence » : Une collaboration peut être attachée à une opération ou à un classificateur par une occurrence de collaboration (cf. section 8.5). Mais une collaboration n'est pas directement instantiable.

[4] guide prévention : Nous conseillons, chaque fois que cela est possible, de faire figurer la flèche creuse avec ligne pointillée de la collaboration vers le classificateur qu'elle représente avec le mot clé « occurrence »
justif : Ceci permet de préciser que la collaboration est utilisée dans le classificateur.

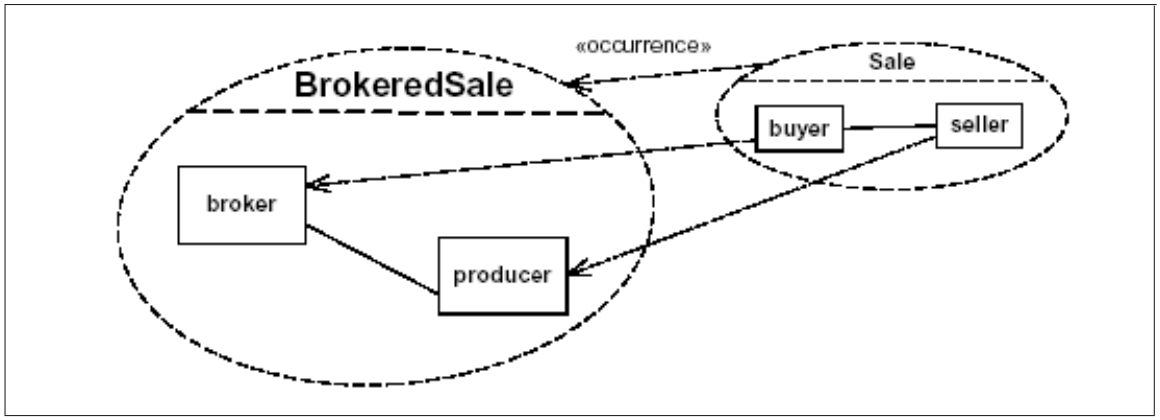

FIG. 8.5 – La collaboration `Sale` utilisée par le classificateur `BrokeredSale`

8.5 Collaboration Occurrence

Contexte Une occurrence de collaboration (*collaboration occurrence* en anglais) représente l'application du patron décrit par une collaboration à une situation spécifique qui implique des classes ou des instances spécifiques qui jouent les rôles de la collaboration.

L'occurrence de collaboration indique un ensemble de rôles et de connecteurs qui coopèrent à l'intérieur du classificateur rattaché à une collaboration donnée, indiquée par le type de l'occurrence de collaboration.

Il peut y avoir plusieurs occurrences d'une collaboration donnée pour un classificateur, chacune impliquant un ensemble différent de rôles et de connecteurs. Un rôle ou un connecteur donné peut être impliqué dans plusieurs occurrences de la même ou de différentes collaborations.

Règles de cohérence

• RÈGLE 255 : Tous les éléments clients (sources) d'une relation de délimitation de rôle (cf. section 9.1) doivent être dans un classificateur et tous les éléments fournisseurs (cibles) doivent être dans la collaboration et être compatibles. [[7] p.160]

Remarque Le terme "être compatible" issu de la specification 2.0 d'UML nous semble exprimé par la règle 256.

• RÈGLE 256 : Les classes ou instances attachées à un rôle dans une occurrence de collaboration doivent posséder l'ensemble des caractéristiques structurelles et comportementales spécifiées pour l'élément connectable auquel on a attribué ce rôle dans la collaboration. [Nouvelle règle]

• RÈGLE 257 : Tout rôle de la collaboration doit être délimité dans l'occurrence de collaboration à un élément connectable du classificateur ou de l'opération. [[7] p.160]

• RÈGLE 258 : Les connecteurs dans le classificateur se connectent selon les connecteurs de la collaboration. [[7] p.160]

Remarque Nous interprétons cela par le respect de le topologie.

• RÈGLE 259 : Dans une occurrence de collaboration, les classes qui jouent des rôles qui sont connectés dans la collaboration doivent se connaître soit par association navi-

gable, soit par des attributs. [Nouvelle règle]

• Règle 260 : Les classes ou instances attachées à un rôle doivent posséder l'ensemble des propriétés spécifiées pour ce rôle. [Nouvelle règle]

Guides

Remarque Sur la figure 8.6 tirée de [7], l'occurrence de collaboration anonyme `:RealizeDisplayBehavior` est reliée au classificateur `Window`.
[5]

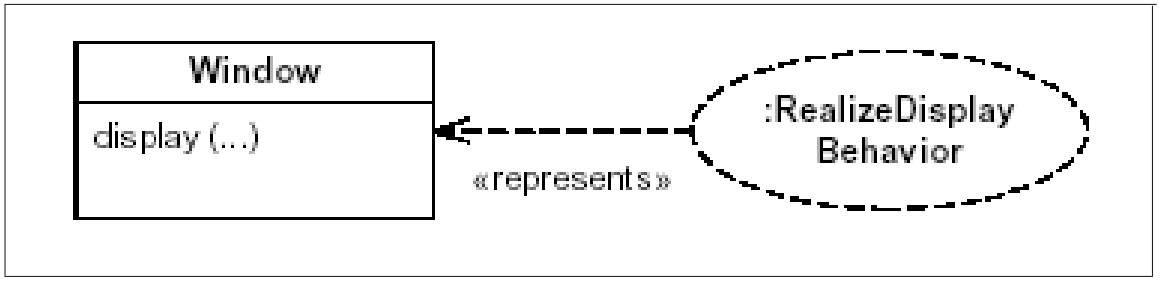

Fig. 8.6 – Une occurrence de collaboration reliée à un classificateur

8.6 Parameter (Collaboration Diagram)

Contexte Un paramètre spécifie comment des arguments peuvent être passés en entrée ou en sortie d'une caractéristique comportementale comme une opération. Le type et la multiplicité du paramètre restreint les valeurs qui peuvent être passées, leur nombre, et l'ordre des valeurs passées.

Dans le diagramme de collaboration, un paramètre a la possibilité d'être relié à un connecteur, c'est un élément connectable.

Règles de cohérence

• Règle 261 : Un paramètre ne peut être relié à une fin de connecteur que dans le contexte d'une collaboration. [[7] p.167]

8.7 Invocation Action

Contexte En plus d'avoir pour cible un objet, les actions d'invocation (*invocation action* en anglais) peuvent aussi invoquer des caractéristiques comportementales sur des ports, à partir desquels les requêtes d'invocation sont routées sur des liens qui dérivent des connecteurs attachés. Les actions d'invocation peuvent aussi être envoyées vers une cible via un port donné.

[5] guide de prévention : Nous conseillons, chaque fois que cela est possible, de faire figurer la flèche creuse avec ligne pointillée de l'occurrence de collaboration vers le classificateur par lequel elle est utilisée avec le mot clé « represents »
justif : Ceci permet de préciser que la collaboration est utilisée par le classificateur et de bien préciser le type de la relation de dépendance utilisée entre l'occurrence de collaboration et le classificateur relié et permet de vérifier la règle 258.

Règles de cohérence

• Règle 262 : Le méta-attribut `onPort` doit être un port de l'objet récepteur. [Règle sur le méta-modèle]

Chapitre 9

Relations

9.1 Role Binding

[1]

Contexte Une relation de dépendance de type délimitation de rôle (*Role Binding* en anglais) est une mise en correspondance entre des caractéristiques du type de la collaboration et les caractéristiques du classificateur ou de l'opération dans le cadre d'une occurrence de collaboration. Cette mise en correspondance indique quel élément connectable du classificateur ou de l'opération joue quel(s) rôle(s) dans la même occurrence de collaboration.

Remarque Un élément connectable peut être lié (délimité) à plusieurs rôles dans la même occurrence de collaboration (voir section 8.5).

Les figures 9.1 et 9.2 montrent des exemples de collaboration et d'occurrence de collaboration avec des affectations de rôle.

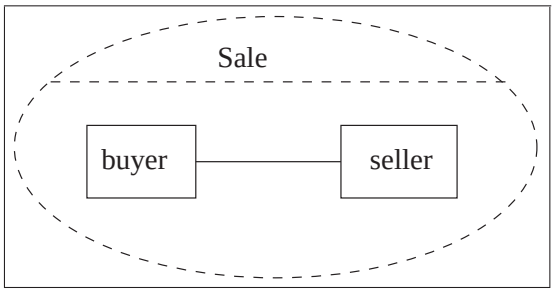

FIG. 9.1 – The sale collaboration

[1] ce n'est pas encore une métaclasse d'UML 2.0, voir avec la nouvelle spec

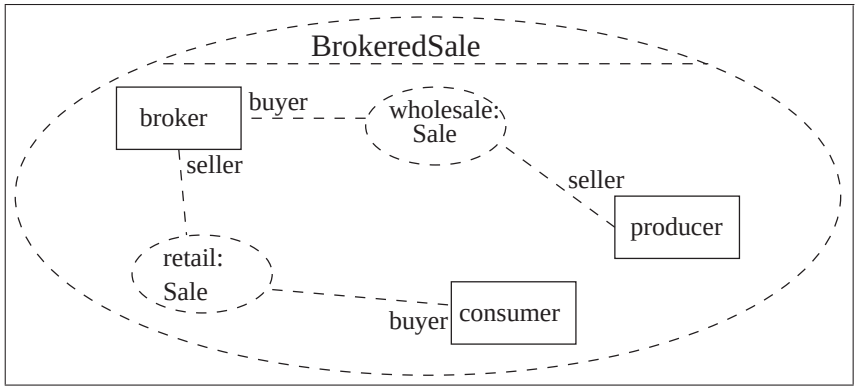

FIG. 9.2 – The BrokeredSale collaboration

Règles de cohérence

• RÈGLE 263 : Toute affectation de rôle doit appartenir à une occurrence de collaboration. [Règle dérivée du méta-modèle]

• RÈGLE 264 : Une affectation de rôle doit mettre en relation un rôle d'une collaboration et un élément connectable. [Nouvelle règle]

• RÈGLE 265 : Tout rôle de la collaboration doit être affecté à une classe ou à une instance dans l'occurrence de la collaboration. [[7] p.160]

• RÈGLE 266 : Tout nom de rôle de l'occurrence de la collaboration doit être présent dans la collaboration. [Nouvelle règle]

Guides

Remarque Comme à la figure 9.2, les rôles `buyer` et `seller` de la collaboration `Sale` apparaissent dans la spécification de la collaboration `BrokeredSale`. En effet, on voit les délimitations de rôle entre les occurrences de la collaboration `Sale` et les trois rôles de la collaboration `BrokeredSale`.

[2]

9.2 Connector (Composite Structures Diagram)

La section 9.2 concerne les règles propres aux connecteurs dans les diagrammes de structures. Nous détaillons les règles sur les connecteurs en général et les règles propres aux fins de connecteurs (cf. 9.2.2).

9.2.1 Généralités

Contexte Un connecteur (*connector* en anglais) spécifie un lien qui rend possible la communication entre deux ou plusieurs instances dans une structure composite. À la différence des associations qui spécifient des liens entre n'importe quelles instances des classificateurs associés, les connecteurs spécifient des liens entre des instances qui appartiennent à un classificateur spécifique. Il est possible de nommer le connecteur.

[2]guide prévention : Pour chaque délimitation de rôle (*Role Binding* en anglais), nous conseillons de relier par une ligne en pointillés l'ellipse représentant l'occurrence de collaboration à l'élément client.

[3]

Règles de cohérence

• Règle 267 : Si le connecteur est nommé, il doit suivre la syntaxe suivante :

```
{{ [ name ]  ':'  classname } | name }
```

où `name` est le nom du connecteur et `classname` le nom de l'association qui type le connecteur. [tirée de [7] p.165]

• Règle 268 : Le champ `classname` d'un connecteur nommé doit être le nom d'exactement une association. [Nouvelle règle][4]

• Règle 269 : Un connecteur doit être relié à au moins deux éléments connectables par des fins de connecteur (cf. section 9.2.2). [Règle dérivée du méta-modèle]

• Règle 270 : Si un connecteur est attaché à un élément connectable qui a des interfaces requises, alors les éléments connectables attachés aux autres fins de connecteur doivent réaliser des interfaces qui sont compatibles avec les interfaces requises. [[7] p.164]

• Règle 271 : Si un connecteur est attaché à un élément connectable qui a des interfaces requises, alors tous ports attachés aux autres fins de connecteur doivent fournir des interfaces qui sont compatibles avec ces interfaces requises, ou alors d'autres éléments connectables doivent réaliser des interfaces qui sont compatibles avec ces interfaces requises. [[7] p.164]

• Règle 272 : Les types des éléments connectables auxquels sont attachés les fins de connecteur doivent être conformes aux types des fins d'association de l'association qui type le connecteur, si spécifiée. [[7] p.164]

• Règle 273 : Dans le cas où un connecteur est typé par une association, la multiplicité de chaque fin de connecteur doit être un sous-ensemble de la multiplicité de la fin d'association correspondante. [Nouvelle règle][5]

Guide

• Guide 14 : Lorsqu'un connecteur est l'instance d'une association nous conseillons de l'indiquer.

• Justification 14 : Le guide 14 permet de vérifier les règles 268, 272 et 273.

9.2.2 Connector End

Contexte Une fin de connecteur (*connector end* en anglais) représente l'extrémité d'un connecteur qui attache le connecteur à un élément connectable. Chaque fin de connecteur est une partie du connecteur.

Remarque Par partie, on entend une possession composite d'un classificateur (cf. section 8.3).

[3] guide de prévention :préciser les cas où le diagramme de structure composée est plus adapté que le diagramme de classe

[4] inter-diagramme structures composite - classes

[5] inter-diagramme

Règles de cohérence

• Règle 274 : Toutes les fins des connecteurs doivent être reliées à un élément connectable (*connectable element* en anglais) (cf. annexe A.3). [Règle dérivée du méta-modèle]

• Règle 275 : Dans le cas où la fin de connecteur est attachée à un port du classificateur contenant, le méta-attribut `partWithPort` doit être vide. [Règle sur le méta-modèle][[7] p.165]

• Règle 276 : Si aucun des méta-attributs d'une fin de connecteur `partWithPort` et `role` n'est vide alors `role` doit être un port qui est défini par le type de `partWithPort`. [[7] p.166]

[6]

[7]

9.3 Keyword « represents »

[8]

Contexte Le mot clé « represents » sur une relation de dépendance indique qu'une collaboration est utilisée dans un classificateur.

Règles de cohérence

• Règle 277 : La source d'une telle relation est une collaboration. [9] [Nouvelle règle]

• Règle 278 : La cible d'une telle relation est un classificateur. [Nouvelle règle]

[6] prévention guide : Nous conseillons de définir une multiplicité pour les fins d'association de l'association qui type le connecteur.
justif : Si aucune multiplicité n'est définie, par défaut elle sera égale à la multiplicité du rôle auquel la fin est attachée.

[7] prévention guide : Si la fin de connecteur est attachée à un port d'une partie de la structure interne de l'élément connectable, nous conseillons de bien définir la multiplicité.
justif Si la fin de connecteur est attachée à un port d'une partie de la structure interne de l'élément connectable, et qu'aucune multiplicité n'est définie, par défaut elle sera égale à la multiplicité du port multipliée par la multiplicité de la partie.

[8] a regarder dans derniere spec ?

[9] ou occurrence de collaboration ?

Chapitre 10

Diagrammes

10.1 Composite Structures Diagram

Contexte Les diagrammes de structures composites servent à montrer comment plusieurs éléments interconnectés, représentant des instances exécutables (*run-time* en anglais) collaborent à travers des liens de communication pour atteindre des objectifs communs.

Les règles suivantes portent sur les éléments qui peuvent être contenus dans un diagramme de structures composites.

Règles de cohérence

• Règle 279 : Les nœuds contenus dans un diagramme de structures composites ne peuvent être que :

- des ports ;
- des possessions ;
- des collaborations ;
- des occurrences de collaboration. [[7] p.178]

• Règle 280 : Les chemins graphiques qui peuvent apparaître dans un diagramme de structure composite doivent être :

- des connecteurs ;
- des délimitations de rôle (*Role Binding*).

[[7] p.178-179]

Remarque Tous les nœuds et chemins des diagrammes de structure composite peuvent être montrés dans les diagrammes de strutures.

Quatrième partie

Diagramme de déploiement

Chapitre 11

Éléments

11.1 Artifact

Contexte Un artefact (*artifact* en anglais) est la spécification d'une partie d'information physique utilisée ou produite lors du processus de développement d'un logiciel. Un artefact représente par exemple des fichiers de modélisation, des fichiers sources, des scripts, etc.

Un artefact est un classificateur et peut avoir des instances qui sont déployées sur un nœud particulier. [1]

Un artefact définit par l'utilisateur un élément concret du monde physique. Un artefact peut avoir des associations de composition avec d'autres artefacts qu'il contient.

Un artefact peut avoir des attributs, des associations navigables et des opérations.

Remarque Les stéréotypes suivants peuvent être appliqués aux artifacts (voir Annexe B. de [7]) : « script », « document », « executable », « file », « library », « source ».

Règles de cohérence

• Règle lien 16 : Un artefact est une sous-classe de la méta-classe `Classifier` et de ce fait doit respecter les règles de la section 2.11.[Règle dérivée du méta-modèle]

• Règle 281 : Un artifact doit être la manifestation d'un et un seul élément empaquetable.[Règle dérivée du méta-modèle]

• Règle 282 : Un artifact déployé doit s'exécuter sur au moins une instance de nœud (possiblement plusieurs). [Nouvelle règle]

11.2 Node

Contexte Un nœud est une ressource d'exécution sur laquelle des artefacts peuvent être déployés en vue d'être exécutés.

Les nœuds peuvent être interconnectés via des chemins de communication (*communication path* en anglais).

La figure 11.1 montre la représentation d'un nœud et d'une instance de nœud.

[1] Attention ne pas confondre les nœuds présents dans les diagrammes d'activité et les nœud présents dans les diagramme de déploiement

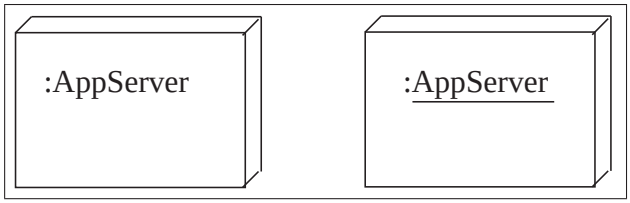

FIG. 11.1 – Spécification d'un nœud et d'une de ses instances

Règles de cohérence

• RÈGLE LIEN 17 : Un nœud est une sous-classe de la méta-classe `Class`, donc elle doit respecter les règles énoncées à la section 2.12.

• RÈGLE 283 : La structure interne d'un nœud, si définie, ne peut contenir que d'autres nœuds. [[7] p.196]

Remarque Lorsque des artefacts sont représentés dans les nœuds, c'est pour indiquer une relation de déploiement.

• RÈGLE 284 : Les nœuds ne peuvent être associés que par deux types d'association, des associations de composition d'une part (dans lesquelles toutes les parties sont des nœuds), et des chemins de communication d'autre part. [Nouvelle règle]

Guides

[2]

11.3 Device

Contexte [3] Un support d'exécution (*device* en anglais) est une sorte de nœud qui décrit une ressource physique de calcul sur laquelle des artefacts peuvent être déployés.

Règles de cohérence

• RÈGLE LIEN 18 : Un support d'exécution étant une sorte de nœud, un support d'exécution doit respecter les règles énoncées en section 11.2. [Règle dérivée du méta-modèle]

11.4 Execution Environment

Contexte Un environnement d'exécution (*execution environment* en anglais) est une sorte de nœud qui décrit un environnement d'exécution pour un type spécifique de composant.

[2] guide prévention Lorsque l'on veut montrer la capacité d'un nœud à supporter un composant, nous conseillons d'utiliser l'une des deux façons de le faire en faisant figurer le mot clé « deploy » sur une relation de dépendance entre le composant et le nœud. rmq : Il est aussi possible de le faire en imbriquant les symboles des composants dans le symbole du nœud. justif : Si on a choisi la première représentation, ceci permet de bien préciser la sémantique de la relation de dépendance.

[3] traduction OK ? ou dispositif

De manière générale, un environnement d'exécution est soit contenu par un autre environnement d'exécution soit par un support d'exécution.

Un environnement d'exécution est noté comme un nœud auquel on associe le mot clé « `ExecutionEnvironnement` ».

Règles de cohérence

• Règle lien 19 : Un environnement d'exécution étant une sorte de nœud, un environnement d'exécution doit respecter les règles énoncées en section 11.2.

• Règle 285 : Un environnement d'exécution ne peut pas contenir de support d'exécution. [Nouvelle règle]

11.5 Deployment Target

Contexte Une cible de déploiement est la spécification d'un endroit où il est possible de déployer des artefacts.

Il existe trois types de cible de déploiement, les nœuds, les spécifications d'instances (si celle-ci est l'instance d'un nœud), ou les possessions (cf. section 8.3). Les spécifications d'instances peuvent jouer le rôle de cible de déploiement afin de pouvoir spécifier des déploiements au niveau instance. Les possessions peuvent jouer le rôle d'une cible de déploiement afin de permettre la modélisation de déploiement sur des nœuds qui ont des structures internes et qui ont donc des possessions qui décrivent son fonctionnement interne.

Règles de cohérence

• Règle 286 : Une spécification d'instance ne peut jouer le rôle d'une cible de déploiement que si c'est une spécification d'instance d'un nœud et si elle fonctionne comme une partie dans la structure interne du nœud englobant. [[7] p.194]

• Règle 287 : Une possession ne peut être une cible de déploiement que si elle a pour type un nœud et si elle fonctionne comme une partie dans la structure interne du nœud englobant. [[7] p.197]

Remarque Dans les deux règles qui précèdent "fonctionner comme une partie dans la structure interne du nœud englobant" signifie qu'il existe une méta-association `part` entre le nœud (`StructuredClassifier`) et la spécification d'instance ou la possession (cf. p.153 de [7]).

11.6 Deployed Artifact

Contexte Un artefact déployé (*deployed artifact* en anglais) est un artefact ou une instance d'artefact qui a été déployé sur une cible de déploiement.

Règles de cohérence

• Règle 288 : Une spécification d'instance ne peut être un artefact déployé que si c'est une spécification d'instance d'un artefact. [[7] p.194]

11.7 Deployment Specification

Contexte Une spécification de déploiement (*deployment specification* en anglais) spécifie un ensemble de propriétés qui déterminent les paramètres d'exécution d'un artefact (qui représente un composant) qui est déployé sur un nœud.

Une spécification de déploiement est notée avec le mot clé « deployment spec ».

Règles de cohérence

• Règle 289 : La cible de déploiement d'une spécification de déploiement doit être un environnement d'exécution. [[7] p.190]

• Règle 290 : Les éléments qui jouent le méta-rôle `deployedElements` d'une cible de déploiement qui sont impliqués dans une relation de déploiement associée à une spécification de déploiement doivent être de type composants. [[7] p.190]

Remarque La règle 290signifie que lorsqu'un artefact est déployé en utilisant une spécification de déploiement, si cet artefact est la manifestation d'éléments, ces éléments doivent être des composants. [4]

Guides

[5]

[4] pas clair regarder dans la norme finale

[5] guide de prévention : Lorsque l'on veut montrer qu'une spécification de déploiement est attachée à un artifact qu'elle paramètre, on conseille de faire figurer explicitement le lien de dépendance du classificateur (spécification de déploiement) vers l'artifact

Chapitre 12

Relations

[1]

12.1 Manifestation

Contexte Une manifestation (*manifestation* en anglais) est une relation qui montre qu'un élément du modèle est incorporé dans un artefact (qui est la spécification d'un élément physique). Une relation de manifestation est une spécialisation d'une dépendance d'abstraction.

La figure 12.1 donne un exemple de manifestation entre un artefact et un composant.

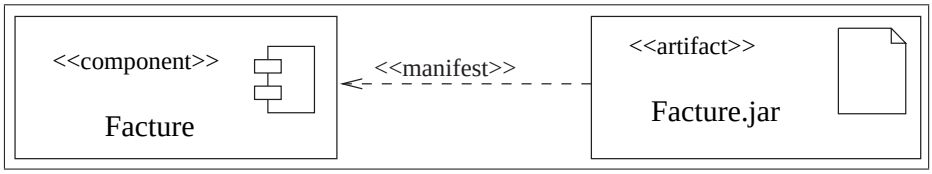

FIG. 12.1 – Exemple de manifestation

Règles de cohérence

• RÈGLE 291 : Une dépendance de manifestation doit avoir comme source un artefact et un élément empaquetable comme cible (cf. annexe A.6). [Règle dérivée du métamodèle]

Remarque Un artefact peut être la manifestation de plusieurs éléments empaquetables.

• RÈGLE LIEN 20 : Une manifestation est une spécialisation d'une dépendance d'abstraction et doit donc respecter les règles énoncées en section 3.5.

Guides

[2]

[1] Dans la classification des classificateurs il manque "component"

[2] guide de prévention : Lorsque l'on veut lier un artifact à l'élément empaquetable qu'il manifeste, on conseille de faire figurer explicitement le lien de dépendance avec le mot clé « manifest » de l'artifact

12.2 Communication Path

Contexte Un chemin de communication est une association entre deux nœuds au travers duquel les nœuds peuvent communiquer par l'échange de messages et de signaux.

Règles de cohérence

• Règle lien 21 : Un chemin de communication est une sous-classe de la méta-classe `Association` et de ce fait, il doit respecter les règles de cohérence énoncées en 3.9.

• Règle 292 : Les fins d'associations d'un chemin de communication sont typés par des nœuds exclusivement. [[7] p.186]

• Règle 293 : Les fins d'associations d'un chemin de communication doivent avoir leur méta-attributs `aggregation` à la valeur `none`. [Nouvelle règle][Règle sur le méta-modèle]

• Règle 294 : Un chemin de communication ne peut relier que deux nœuds, i.e. c'est une sorte d'association binaire. [tirée de [7] p.186] [3]

[4]

12.3 Deployment

Contexte Un déploiement (*deployment* en anglais) est l'allocation d'un artefact ou d'une instance d'artefact sur une cible de déploiement (cf. section 11.5).

Il est possible de décrire un déploiement au niveau type et au niveau instance.

Il est possible de décrire les propriétés qui paramètrent le déploiement et donc l'exécution d'un ou plusieurs artefacts (cf. section 11.7).

La figure 12.2 montre les deux notations possibles pour exprimer un déploiement.

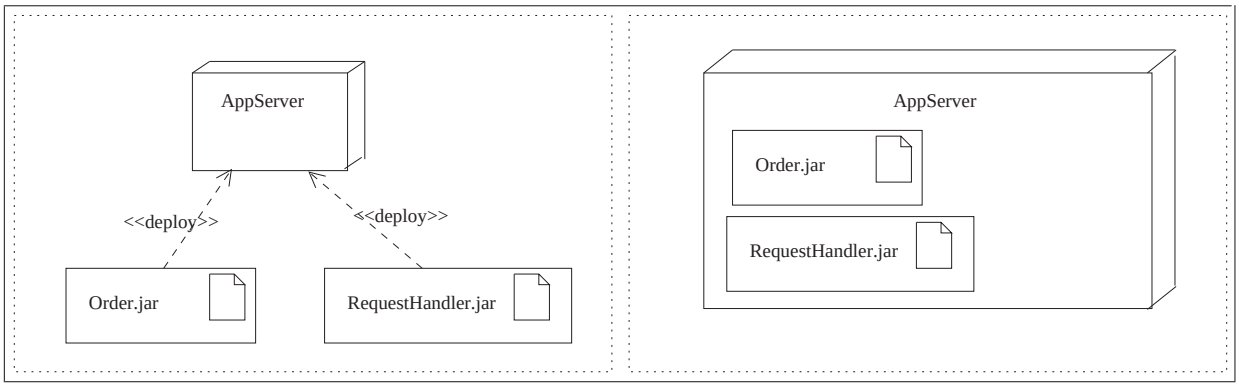

Fig. 12.2 – Déploiement

vers l'élément empaquetable. rmq : auparavant, on utilisait le mot clé « implementation », mais comme il y avait plusieurs emplois possibles, un mot clé singulier a été proposé.

[3] c ce qui semble être dit dans la norme mais je ne vois pas d'explications sémantiques

[4] guide de prévention : chaque fois que cela est possible, bien préciser ce stéréotype ? Un stéréotype peut être employé pour identifier le protocole de communication ou le réseau utlisé entre deux nœuds. provient de UML2.0 Toolkit p.275

Règles de cohérence

• Règle lien 22 : Comme un déploiement est une sous-classe de la méta-classe Dependency, il doit respecter les règles de cohérence énoncées en 3.4.

• Règle 295 : Un déploiement est une relation entre exactement une cible de déploiement et un ou plusieurs artefacts déployés. [Règle dérivée du méta-modèle]

Chapitre 13

Diagrammes

13.1 Deployment Diagram

Contexte Les diagrammes de déploiement (*deployment diagrams* en anglais) servent à définir l'architecture d'exécution d'un système. Cette architecture est représentée par l'affectation d'artefacts logiciels à des nœuds. Les nœuds sont connectés via des chemins de communication pour créer des systèmes en réseau. Les nœuds sont définis de manière encapsulée et représentent des cibles matérielles ou des environnements d'exécution logiciels. Les artefacts représentent des morceaux d'information résultants d'un processus de développement logiciel.

Règles de cohérence

• Règle 296 : Les nœuds graphiques qui peuvent apparaître dans un diagramme de déploiements sont : [[7] p.198-199]

- des artefact ;
- des nœuds ;
- des spécifications de déploiements (avec éventuellement des propriétés et leurs valeurs).

• Règle 297 : Les chemins graphiques qui peuvent apparaître dans un diagramme de déploiement sont : [[7] p.199-200]

- des relations d'association ;
- des relations de dépendance ;
- des relations de généralisation ;
- des relations de déploiement ;
- des relations de manifestation.

Cinquième partie

Diagramme d'activités

Chapitre 14

Éléments

14.1 Activity

Contexte Une activité (*activity* en anglais) définit un comportement paramétré par un séquencement organisé d'unités subordonnées dont les éléments simples sont les actions (cf. section 14.3). Le flot d'exécution est modélisé par des nœuds reliés par des arcs.

Les activités peuvent former une hiérarchie d'invocations en invoquant d'autres activités. Dans les modèles orientés objets, les activités sont généralement liées à des opérations qui sont directement invoquées.

Une activité est un comportement (*behavior* en anglais) et à ce titre peut être associé à des paramètres.

La sémantique des activités repose sur le flot de jetons. Par flot, on entend que l'exécution d'un nœud affecte et est affectée par l'exécution d'autres nœuds, et de telles dépendances sont représentées par les arcs (*edge* en anglais) dans le diagramme d'activités.

Chaque fois qu'une activité est invoquée, le méta-attribut `isSingleExecution` indique si la même exécution de l'activité manipule les mêmes jetons pour toutes les invocations, ou si une exécution distincte de l'activité est créée pour chaque invocation. S'il l'on veut spécifier qu'il s'agit de la même exécution, on peut faire figurer le mot clé « singleExecution » à l'intérieur de l'activité.

Des nœuds et des arcs hérités d'activités plus générales peuvent être remplacées (surchargées). Des contraintes peuvent être définies et s'appliquent à toutes les utilisations de l'activité. S'ils apparaissent à l'intérieur de l'activité, les mots clés « precondition » et « postcondition » doivent être suivis d'une contrainte.

La figure 14.1 tirée de [7] montre un exemple d'activité.

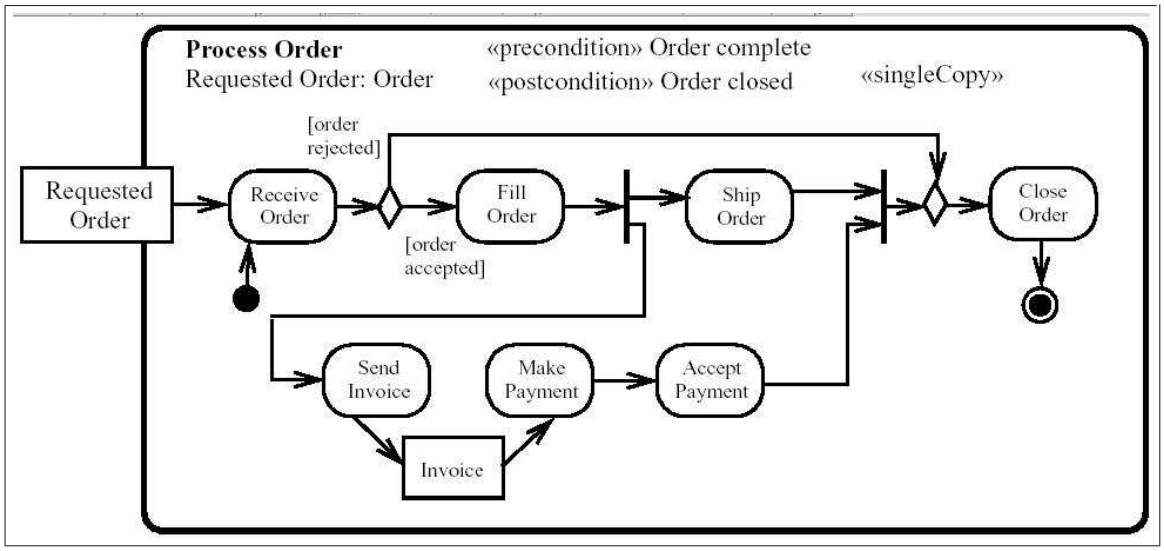

FIG. 14.1 – Exemple d'activité avec paramètre d'entrée

Règles de cohérence

• RÈGLE 298 : Les nœuds d'une activité doivent inclure un nœud paramètre d'activité pour chaque paramètre. [[7] p.285]

• RÈGLE 299 : Une activité ne peut pas être simultanément autonôme et avoir comme contexte un classificateur ou une caractéristique comportementale. [[7] p.285]

• RÈGLE 300 : Les noms des différents paramètres textuels doivent concorder avec les noms des paramètres graphiques. [Nouvelle règle][Règle utilisateur]

• RÈGLE 301 : Si le méta-attribut `isReadOnly` est vrai, l'activité ne doit faire aucune modification de variables ou d'objets externes à l'activité [tirée de [7] p.285] [1]

Remarque Comme montré par la figure 14.1, UML offre la possibilité d'exprimer les types des nœuds paramètres de manière textuelle.

14.2 Activity Node

Contexte Un nœud d'activité (*activity node* en anglais) est une classe abstraite pour représenter les étapes le long du flot d'une activité.

Il existe trois familles de nœuds d'activité : les nœuds d'exécutions (*executable node* en anglais), les nœuds objets (*object node* en anglais) et les nœuds de contrôle (*control nodes* en anglais).

La figure 14.2 présente les différents types de nœuds d'activité. De la gauche vers la droite, on trouve : le nœud représentant une action, qui est une sorte de nœud exécutable, un nœud objet, un nœud de décision ou d'interclassement, un nœud de bifucation ou d'union, un nœud initial, un nœud final et un nœud final de flot.

[1] Règle contraignant l'implantation, propriété non testable au niveau modèle mais vérifiable sur le code généré

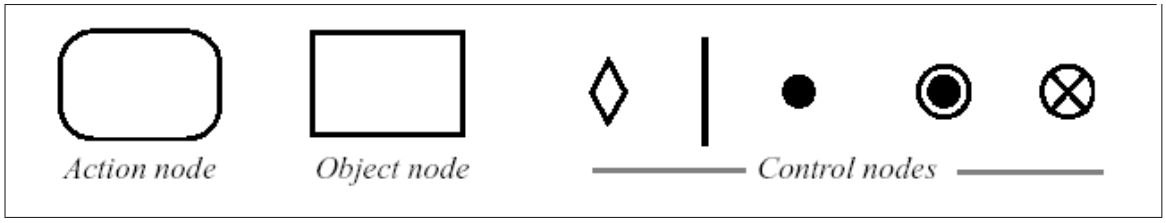

FIG. 14.2 – Notation des nœuds d'activité

Règles de cohérence

• RÈGLE 302 : Les nœuds d'activité ne peuvent être possédés que par des activités ou des groupes d'activités. [[7] p.302]

• RÈGLE 303 : Si le méta-attribut `mustIsolate` est vrai pour un nœud d'activité, les accès à un objet par l'intermédiaire d'une action à l'intérieur du nœud ne doivent pas rentrer en conflit avec l'accés au même objet par une action à l'extérieur du nœud. [tirée de [7] p.302]

Remarque Un conflit est défini comme une tentative d'écriture de l'objet par une action ou par les deux actions simultanément.

[2]

14.3 Action

Contexte Une action (*action* en anglais) est un nœud d'activité

exécutable qui est l'unité fondamentale de fonctionnalité exécutable dans une activité. L'exécution d'une action représente une transformation ou un calcul quelconques dans le système modélisé. Une exécution d'action représente le comportement d'une action à l'intérieur de l'exécution d'une activité spécifique. La méta-classe `Action` est abstraite, donc toutes les exécutions d'actions seront des exécutions d'un type particulier d'action.

Des pré et post-conditions peuvent être attachées à une action dans une note reliée à l'action avec les mots-clés « localPrecondition » et « localPostcondition », respectivement.

Toutes les règles propres aux différents types d'actions présentés dans le paquetage `Actions` de la norme ne sont pas détaillées ici (cf. Annexe **??**).

Règles de cohérence

• RÈGLE 304 : Un nœud d'action doit avoir au moins un arc entrant. [Nouvelle règle]

14.4 Object Node

Contexte Un nœud d'objet (*object node* en anglais) est une méta-classe abstraite. Se référer à l'annexe A.10 pour en avoir l'arbre de spécialisation. Cette classe sert à définir les flots d'objets dans les diagrammes d'activité.

[2]regle contraignant l'implantation, propriété non testable au niveau modèle mais vérifiable sur le code généré

Un nœud objet indique que l'instance d'un classificateur peut être disponible à un certain point de l'activité.

L'état que l'instance doit avoir en atteignant le nœud objet peut être indiqué.

Il est possible d'associer aux nœuds objets un comportement de sélection via le mot clé « selection ». Une sélection associée à un nœud objet prend en entrée l'ensemble des jetons présents dans le nœud et en sélectionne un pour les arcs sortants.

On peut aussi spécifier l'ordre dans lequel les objets sont sélectionnés, via le méta-attribut `ordering` et la propriété qui lui est associée notée `{ordering=X}` où X doit être `FIFO, LIFO, ordered` ou `unordered` (par défaut `ordering=FIFO`).

Il est également possible d'associer à un nœud objet un poids, appelé valeur limite haute qui détermine le nombre de jetons maximal qu'il peut contenir. Par défaut ce nombre est infini.

Règles de cohérence

• Règle lien 23 : Un nœud objet est un nœud d'activité et de ce fait doit respecter les règles de la section 14.2.

• Règle 305 : Si l'état de l'objet est indiqué, cet état doit correspondre à un état connu de l'objet. [Nouvelle règle][3]

Remarque Cette règle ne peut s'appliquer que si le type du nœud objet est spécifié. De plus, on appelle état connu d'un objet, un attribut suivi de sa valeur ou un état du diagramme d'état du classificateur qui type le nœud objet.

• Règle 306 : Tout arc qui entre ou sort d'un nœud d'objet doit être un arc de flot d'objets (*object flow edge* en anglais). [[7] p.349] [Règle sur le méta-modèle]

• Règle 307 : La valeur limite haute doit être égale à la valeur limite haute du plus proche nœud objet de flux qui n'a pas de nœud d'action qui intervient. [[7] p.349]

• Règle 308 : Si elle est spécifiée, la valeur de la limite haute doit être strictement positive. [Nouvelle règle]

• Règle 309 : Si un nœud objet a un comportement de sélection, alors le méta-attribut `ordering` doit être `ordered` et réciproquement. [[7] p.349]

• Règle 310 : Un comportement de sélection a un paramètre d'entrée et un paramètre de sortie. Le paramètre d'entrée doit être un "bag" d'éléments du même type que le nœud objet ou d'un type plus général. Le paramètre de sortie doit être du même type ou d'un sous-type du nœud objet. Le comportement de sélection ne doit pas avoir d'effets de bord. [[7] p.349]

Remarque Un "bag" est une collection non ordonnée qui peut contenir deux fois le même élément.

• Règle 311 : Si l'ordre de sélection des jetons est spécifié par `{ordering=X}` alors X doit être égale à `FIFO, LIFO, ordered` ou `unordered`. [Règle dérivée du méta-modèle][Règle utilisateur]

Guide

• Guide 15 : Nous conseillons de typer les nœuds objets.

• Justification 15 : L'application du guide 15 permet de détecter la règle 305 et 393.

[3] inter-diagramme

14.5 Pin

Contexte général Une broche (*pin* en anglais) est un nœud objet. Les broches sont connectées en entrées ou en sorties des actions. Elles fournissent des valeurs aux actions et acceptent des valeurs provenant d'actions. Dorénavant, pour désigner les broches, nous utiliserons le mot anglais *pin*.

Nous détaillons dans les sections 14.5.2 et 14.5.3 les règles propres aux pins d'entrée et sortie, et pins valeurs.

14.5.1 Règles générales

Règles de cohérence

• Règle 312 : Si l'action est une action d'invocation, le nombre et le type des pins doivent être les mêmes que le nombre de paramètres et le type des comportements ou caractéristiques comportementales invoqués. [[7] p.356]

Remarque Les paramètres sont mis en correspondance avec les pins de manière ordonnée.

• Règle 313 : Un pin est toujours soit en entrée d'une action soit en sortie d'une action. [Règle dérivée du méta-modèle]

Remarque La figure 14.3 représente deux notations équivalentes de flux d'objets entre deux actions. La sous-figure i) représente un flux d'objets en utilisant la notation des pins, alors que la sous-figure ii) utilise la notation d'un nœud objet. Conceptuellement, la notation de nœud objet ne devrait pas être utilisée vu que `ObjectNode` est une méta-classe abstraite (et ne devrait donc pas avoir d'instances dans les modèles). Il est cependant aisé de traduire cette notation en utilisant les pins.

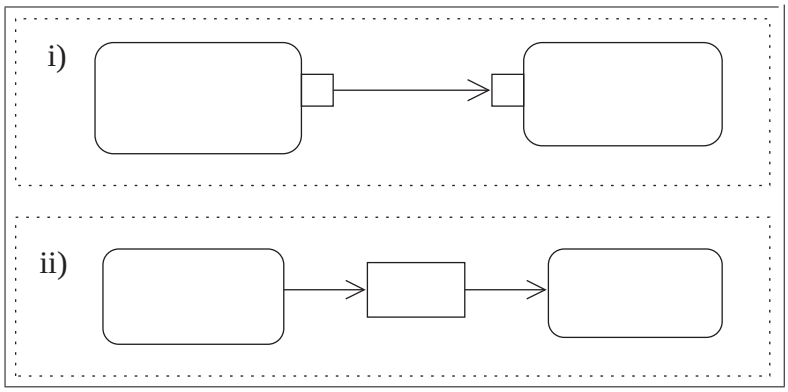

Fig. 14.3 – Représentations équivalentes de flux d'objets

• Règle 314 : Lorsque le flux d'objets est représenté sous forme de nœuds objets, l'arc entrant doit venir d'une action et l'arc sortant doit aller vers une action. [Nouvelle règle][Règle utilisateur]

• Règle lien 24 : Tout pin (pin d'entrée et pin de sortie) étant un nœud objet, celui-ci doit respecter les règles énoncées en section 14.4

Guides

[4]

14.5.2 Input Pin and Output Pin

Contexte Un pin d'entrée (*input pin* en anglais) est un pin qui supporte les valeurs d'entrée qui doivent être consommées par une action. C'est un nœud objet et il reçoit des valeurs de la part d'actions. Un pin de sortie (*output pin* en anglais) est un pin qui supporte les valeurs de sortie qui sont produites par une action. C'est un nœud objet et il délivre des valeurs à d'autres actions.

Règles de cohérence

• Règle 315 : Un pin d'entrée doit uniquement posséder des arcs entrants. [[7] p.336]

• Règle 316 : Un pin de sortie doit uniquement posséder des arcs sortants. [[7] p.352]

• Règle 317 : L'arc sortant d'un pin de sortie doit aller vers un nœud objet. [Nouvelle règle]

• Règle 318 : L'arc entrant d'un pin d'entrée doit venir d'un nœud objet. [Nouvelle règle]

Remarque Dans les règles 317 et 318, il peut y avoir des nœuds de contrôle intermédiaires. Le mot arc peut aussi être arc composé. Nous appelons arc composé un chemin constitué d'un ensemble d'arcs, de nœuds de contrôle, et de nœuds objets. Un arc composé a comme source et comme origine un ensemble d'actions.

14.5.3 Value Pin

Contexte Un pin valeur (*value pin* en anglais) est un pin d'entrée qui fournit une valeur à une action qui ne provient pas d'un arc de flot d'objets. Un pin valeur est toujours associé à une spécification de valeur.

Remarque Un pin valeur utilise la même notation qu'un pin d'entrée avec la spécification de la valeur écrite en dessous. D'autre part, il est également possible d'utiliser la notation utilisant les nœuds objets (cf. section 14.5).

Règles de cohérence

• Règle 319 : Les pins valeurs ne doivent pas avoir d'arcs entrants. [[7] p.363]

• Règle 320 : Le type de la spécification de valeur que le pin fournit doit être compatible avec le type du pin de valeur. [[7] p.363]

• Règle 321 : Un pin valeur est toujours associé à une spécification de valeur. [Règle dérivée du méta-modèle]

• Règle lien 25 : Un pin valeur doit respecter les règles sur les pins et les pins d'entrée énoncées en 14.5 et 14.5.2.

14.6 Activity Parameter Node

Contexte Un nœud paramètre d'activité (*activity parameter node* en anglais) est un nœud objet qui décrit les entrées ou sorties des activités. D'autre part, tout nœud

[4] guide prévention : Ne pas utiliser la notation broche tout seule (*standalone pin*) entre deux actions si les broches de sortie et d'entrée ne sont pas du même type.

paramètre d'activité est associé avec un paramètre de l'activité, correspondant au paramètre que le nœud objet va représenter au sein de la description de l'activité.

Règles de cohérence

• Règle 322 : Tout nœud paramètre d'activité doit être associé à un paramètre de l'activité. [[7] p.304]

• Règle 323 : Le type d'un nœud paramètre d'activité doit être le même que celui du paramètre de l'activité auquel il est associé. [[7] p.304]

• Règle 324 : Les nœuds paramètres d'activité ne doivent avoir aucun arc entrant (s'ils sont en début), et aucun arc sortant (s'ils sont en fin). [[7] p.304]

• Règle 325 : Un nœud paramètre d'activité sans nœud entrant et un ou plusieurs nœuds sortants doit être associé à un paramètre de direction `in` ou `inout`. [[7] p.304]

• Règle 326 : Un paramètre d'activité sans nœud sortant et un ou plusieurs nœuds entrants doit être associé à un paramètre de direction `out`, `inout` ou `return`. [[7] p.304] [5]

• Règle lien 26 : Un nœud paramètre d'activité est une sorte de nœud objet et doit donc respecter les règles énoncées en section 14.4.

14.7 Central Buffer Node

Contexte Un nœud central de mémoire tampon (*central buffer node* en anglais) ou nœud buffer central, accepte des flux de jetons objets en entrée qu'il transmet à un flux aval de jetons objets.

Ce type de nœud objet accepte la concurrence des jetons objets et peut donc avoir de multiples arcs entrants et de multiples arcs sortants.

La notation d'un nœud central de mémoire tampon utilise le mot clé « centralBuffer » associé à la notation d'un nœud objet.

Règles de cohérence

• Règle 327 : Un nœud central de mémoire tampon doit avoir au moins un arc entrant et au moins un arc sortant. [Nouvelle règle]

• Règle 328 : Un nœud central de mémoire tampon ne doit pas être connecté directement à des actions. Il doit passer par l'intermédiaire de pins. [Nouvelle règle]

14.8 Data Store Node

Contexte Un nœud de stockage de données (*data store node* en anglais) est un nœud de buffer central qui sert à stocker des informations non-transitoires. Un nœud de stockage de données garde tous les jetons qui entrent, tout en les copiant lorsqu'ils sont choisis pour continuer dans le flot aval. Les jetons entrants contenant un objet particulier remplacent n'importe quel jeton dans le nœud objet qui contient cet objet.

La spécification d'un nœud de stockage de données se fait par l'utilisation du mot clé « datastore ».

[5] pb dans la norme pour cette dernière règle : "in" au lieu de "out"

Règles de cohérence

• Règle lien 27 : Un nœud de stockage de données étant un type de nœud central de mémoire tampon, il doit respecter les règles énoncées en section 14.7.

14.9 Control Node

Contexte Un nœud de contrôle (*control node* en anglais) est un nœud d'activités abstrait utilisé pour coordonner les flots entre les nœuds d'une activité.

L'annexe A.11 présente l'arbre de spécialisation des nœuds de contrôle. Un nœud de contrôle peut être :

- un nœud initial (*initial node* en anglais) ;
- un nœud final (*final node* en anglais) ;
- un nœud de décision (*decision node* en anglais) ;
- un nœud d'interclassement ou fusion (*merge node* en anglais) ;
- un nœud de bifurcation (*fork node* en anglais) ;
- un nœud d'union (*join node* en anglais).

Remarque Il existe deux sortes de nœuds finaux.

La figure 14.4 présente les notations des différents nœuds de contrôle.

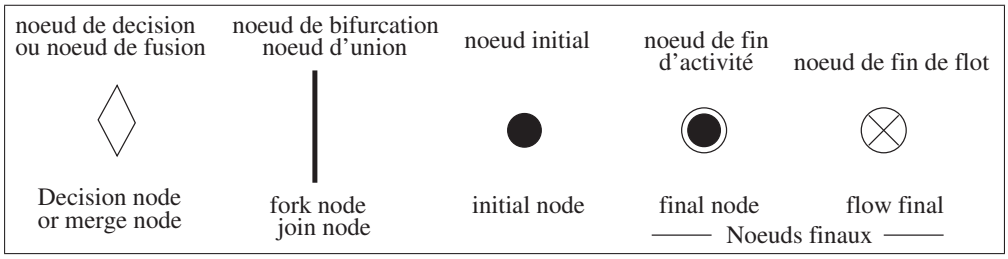

Fig. 14.4 – Les nœuds de contrôles et leur représentation graphique

Règles de cohérence

• Règle 329 : Tous les arcs entrants ou sortants d'un nœud de contrôle doivent être soit tous des flots d'objets, soit tous des flots de contrôle.[[7] p.317]

14.10 Initial Node

Contexte Un nœud initial (*initial node* en anglais) est un nœud de contrôle à partir duquel le flot débute lorsque l'activité est invoquée. Une activité peut avoir plusieurs nœuds initiaux.

Règles de cohérence

• Règle 330 : Un nœud initial ne doit pas avoir d'arc entrant.[[7] p.335]

• Règle 331 : Tout nœud initial a au moins un arc sortant. [Nouvelle règle]

• Règle 332 : Il est interdit d'avoir une relation directe entre un nœud initial et un nœud final. [Nouvelle règle]

• Règle lien 28 : Un nœud initial est un nœud de contrôle et doit donc respecter les règles de la section 14.9.

14.11 Final Node, Activity Final Node and Flow Final Fode

Contexte Un nœud final (*final node* en anglais) est un nœud de contrôle abstrait auquel s'arrêtent des flots de jetons d'activité.

Il existe deux sortes de nœuds finaux : les nœuds finaux d'activité (*activity final node* en anglais) et les nœuds finaux de flot (*flow final node* en anglais). Lorsqu'un jeton atteint un nœud final d'activité, tous les flots de l'activité sont stoppés. Au contraire, un nœud final de flot détruit les jetons qui lui arrivent mais n'a aucun effet sur les autres jetons de l'activité.

Règles de cohérence

• Règle 333 : Un nœud final ne doit pas avoir d'arcs sortants. [[7] p.332]

• Règle 334 : Un nœud final a au moins une transition entrante. [Nouvelle règle]

• Règle lien 29 : Un nœud final étant un nœud de contrôle, un nœud final doit respecter les règles de la section 14.9.

Guides

[6]

14.12 Merge Node

Contexte Un nœud d'interclassement (*merge node* en anglais) est un nœud de contrôle qui rassemble plusieurs flots alternatifs entrants en un seul flot sortant. Il n'est pas utilisé pour synchroniser des flots concurrents mais pour accepter un flot parmi plusieurs.

Règles de cohérence

• Règle 335 : Un nœud d'interclassement possède un arc sortant. [[7] p.343]

[6]guide de modélisation S'il n'est pas souhaitable d'arrêter tous les flots dans une activité, il est préférable d'utiliser un nœud final de flot d'activité plutôt qu'un nœud final d'activité
justif : dans le cas où la même exécution d'une activité est en train d'être utilisée pour toutes ses invocations, les flots multiples de jetons vont traverser le long de la même activité, et il n'est pas souhaitable de stopper tous les jetons simplement parce que l'un d'entre eux a atteint un nœud final d'activité

Remarque Graphiquement il est possible de fusionner un nœud de fusion et un nœud de décision. Il est donc possible que graphiquement un nœud d'interclassement ait plusieurs arcs sortants.

• Règle 336 : Un nœud d'interclassement possède au moins deux arcs entrants. [tirée de [7] p.343]

• Règle lien 30 : Un nœud de fusion étant un nœud de contrôle, un nœud de fusion doit respecter les règles de la section 14.9.

14.13 Decision Node

Contexte Un nœud de décision (*decision node* en anglais) est un nœud de contrôle qui fait un choix entre plusieurs flots sortants. Les flots sortants sont sélectionnés en fonction de la condition de garde qui est associée à chaque arc sortant.

Un nœud de décision peut être associé à un comportement grâce au mot clé « decisionInput ». La sortie du comportement est alors disponible par les conditions de gardes. Le comportement ne peut pas avoir d'effets de bords mais il peut par exemple naviguer d'un objet à un autre, ou prendre la valeur d'un attribut. Un comportement peut être exécuté plusieurs fois.

Un nœud de décision peut offrir un jeton à plusieurs arcs sortants mais ce jeton ne peut être accepté que par un seul. Cela peut conduire à un choix indéterministe.

Règles de cohérence

• Règle 337 : Tout nœud de décision doit avoir exactement un arc entrant. [[7] p.320]

Remarque Graphiquement il est possible de fusionner un nœud de fusion et un nœud de décision. Il est donc possible que graphiquement un nœud de décision ait plusieurs arcs entrants.

• Règle 338 : Un comportement associé à un nœud de décision a un paramètre d'entrée et un paramètre de sortie. Le paramètre d'entrée doit être de même type ou d'un type plus général que le type des jetons objets arrivant par le nœud entrant. De plus, le comportement ne doit pas avoir d'effets de bords. [[7] p.320]

• Règle 339 : Dans le cas où un comportement est associé au nœud de décision, l'utilisation de la sortie du comportement dans les conditions de garde implique que le type de cette sortie soit respecté. [Nouvelle règle]

Remarque Soit une classe `Disk` ayant un attribut `Status` qui peut prendre les valeurs `Full` ou `Empty`. La figure 14.5 montre un exemple en i) où la règle 339 est respectée alors qu'en ii) elle n'est pas respectée.

• Règle 340 : L'ensemble des valeurs que peuvent prendre la ou les variables qui entrent en jeu dans les conditions de gardes doit être couvert. [Nouvelle règle]

Remarque La condition de garde prédéfinie "else" peut être utilisée. Cette condition passe à vrai uniquement quand toutes les autres conditions de gardes ont été évaluées à faux.

• Règle lien 31 : Un nœud de décision étant un nœud de contrôle, un nœud de décision doit respecter les règles de la section 14.9.

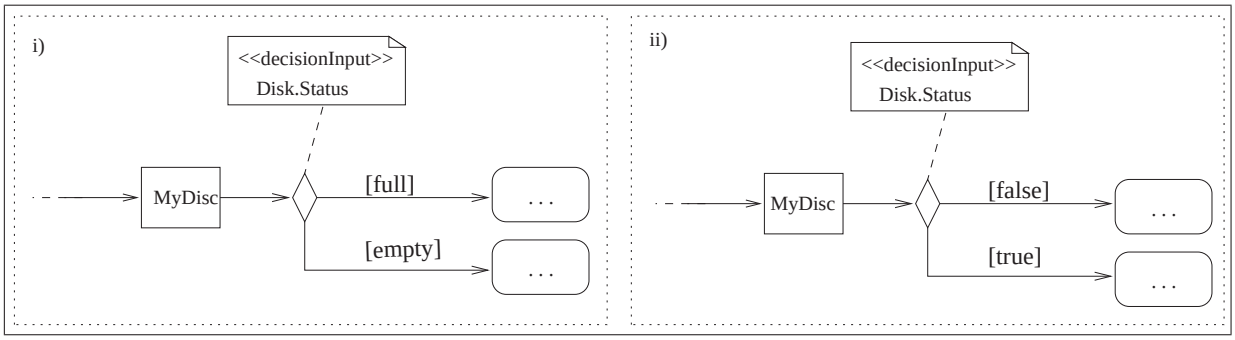

FIG. 14.5 – Exemples d'utilisation d'un comportement lié à un nœud de décision

Guides

7

8

14.14 Fork Node

Contexte Un nœud de bifurcation (*fork node* en anglais) est un nœud de contrôle qui sépare un flot en plusieurs flots concurrents. Les jetons arrivant à un nœud de bifurcation sont dupliqués à travers les arcs sortants. Les jetons offerts par l'arc entrant sont tous offerts aux arcs sortants. Quand un jeton offert est accepté par tous les arcs sortants, des duplicatas du jeton sont faits et une copie traverse chaque arc.

Règles de cohérence

• RÈGLE 341 : Un nœud de bifurcation doit possèder exactement un arc entrant. [[7] p.334]

Remarque Graphiquement il est possible de fusionner un nœud de bifurcation et un nœud d'union. Il est donc possible que graphiquement un nœud de bifurcation ait plusieurs arcs entrants.

• RÈGLE 342 : Un nœud de bifurcation doit possèder au moins deux arcs sortants. [tirée de [7] p.334]

Guides

9

[7]guide prévention : Pour les nœuds de décision, nous conseillons de définir une condition de garde `else` pour un et un seul arc sortant. justif : quand toutes les autres conditions de garde sont évaluées à faux, cela permet de sortir du nœud de décision

[8]guide pour prévenir l'utilisation conjointe des nœuds de décision et des nœuds d'interclassement (*merge*), peu lisible ?

[9]guide prévention : Si des conditions de garde sont spécifiées sur les arcs sortants d'un nœud de bifurcation, il est souhaitable de s'assurer qu'aucun nœud d'union en aval dépend de l'arrivée de jetons devant sortir d'un arc avec condition de garde. Pour éviter le problème, on peut introduire un nœud de décision comme p297, figure 212

14.15 Join Node

Contexte Un nœud d'union (*join node* en anglais) est un nœud de contrôle qui synchronise des flots multiples. S'il y a un jeton offert sur tous les arcs entrants du nœud d'union, alors des jetons sont offerts sur les arcs sortants selon ces règles :

- Si tous les jetons offerts sur les arcs entrants sont des jetons de contrôle, alors un jeton de contrôle est offert sur l'arc sortant.
- Si quelques jetons entrants sont des jetons de contrôle et d'autres des jetons de données, alors seuls les jetons de données sont offerts sur l'arc sortant.

Par défaut, un nœud d'union offre un jeton à sa sortie si tous ses arcs d'entrée offrent un jeton. Il est possible de changer cette spécification en utilisant la propriété `{joinSpec}=X` où X décrit la loi à suivre pour qu'un jeton soit offert en sortie du nœud d'union (par défaut `X` vaut `and`).

Règles de cohérence

• Règle 343 : Un nœud d'union doit avoir exactement un arc sortant. [[7] p.339]

Remarque Graphiquement il est possible de fusionner un nœud de bifurcation et un nœud d'union. Il est donc possible que graphiquement un nœud d'union ait plusieurs arcs sortants.

• Règle 344 : Un nœud d'union doit avoir au moins un arc entrant. [Nouvelle règle]

Remarque Pour que cela ait du sens, en pratique, un nœud d'union doit avoir deux arcs entrants.

Guides

[10]

14.16 Activity Group

Contexte Un groupe d'activité (*activity group* en anglais) est une construction de regroupement de nœuds et d'arcs. Les nœuds et les arcs peuvent appartenir à plus d'un groupe. Ils peuvent être utilisés pour différents propos et n'ont pas de sémantique propre.

La meta-classe `ActivityGroup` est abstraite, ses sous-types qui peuvent être rencontrés dans les modèles UML sont : `ActivityPartition` (cf. section 14.17), `InterruptibleActivityRegion` (cf. section 14.18), `StructuredActivityNode` (cf. section 14.22).

Règles de cohérence

• Règle 345 : Tous les nœuds et arcs d'un groupe doivent appartenir à la même activité que le groupe. [[7] p.301]

• Règle 346 : Aucun nœud ni arc d'un groupe ne doit être contenu par ses sous-groupes ou ses groupes contenants, et ce de manière transitive. [[7] p.301]

• Règle 347 : Les groupes peuvent seulement être possédés par des activités ou des groupes. [[7] p.301]

[10] guide pour prévenir l'utilisation conjointe des nœuds de bifurcation et des nœuds d'union, qui est peu lisible ?

14.17 Activity Partition

Contexte Les partitions d'activité (*activity partition* en anglais) divisent l'espace des nœuds et des arcs afin de montrer explicitement dans quelle entitée les actions peuvent être effectuées. Les partitions peuvent partager leur contenu. Elles correspondent souvent à des unités organisationnelles dans un modèle de business.

Une partition peut être décomposée en sous-partitions et regrouper d'autres partitions selon une autre dimension.

Les partitions n'affectent pas le flots de jetons du modèle, elles contraignent et fournissent une vue des comportements invoqués dans les activités. Des contraintes différentes s'appliquent selon le type de l'élément que la partition représente.

Pour modéliser le fait que les activités se produisent à l'extérieur du domaine d'un modèle particulier, on labellise la partition avec le mot clé « external ».

Les mots clés « class » et « attribute » peuvent être utilisés pour indiquer qu'une partition représente une classe ou un attribut d'une classe.

Règles de cohérence

• Règle 348 : Une partition qui représente une dimension pour des sous-partitions ne peut pas être contenue par d'autres partitions. [[7] p.307]

• Règle 349 : Aucun nœud ou arc qui appartient à une partition ne peut être contenu par une autre partition dans la même dimension. [[7] p.307]

• Règle 350 : Si une partition d'activité représente une partie, alors toutes les partitions non externes qui appartiennent à la même dimension doivent représenter des parties directement contenues par la structure interne du même classificateur. [[7] p.307] [11]

• Règle 351 : Si une partition d'activité non-externe représente un classificateur et est contenue par une autre partition, alors la partition contenante doit représenter un classificateur. De plus, le classificateur de la sous-partition doit être contenu par le classificateur représenté par la partition contenante, ou être associé à ce classificateur par une association de forte composition (et être du coté qui est contenu). [[7] p.307] [12]

• Règle 352 : Si une partition représente une partie et est contenue par une autre partition, alors la partie doit appartenir au classificateur représenté par la partition contenante ou appartenir à un classificateur qui est le type de la partie représentant la partition contenante. [[7] p.307]. [13]

• Règle 353 : Dans le cas où la partition représente une classe ou un attribut, ceux-ci doivent être déclarés dans le diagramme de structure. [Nouvelle règle][Règle utilisateur][14]

• Règle 354 : Une partition ne peut représenter qu'un classificateur (une classe en fait), une instance, une partie ou un attribut et/ou une valeur. [Nouvelle règle][15]

• Règle 355 : Dans le cas où une partition représente une classe, chaque action qui se trouve dans la partition (et qui n'est pas marquée comme « externe ») doit correspondre

[11] inter-diagramme activités - classes/composants
[12] inter-diagramme activités - classes
[13] regle pas bien comprise
[14] voir dans la nouvelle spec si « class » affecté à la partition est traduit dans le méta-modèle ?
[15] complet ?

à une opération de la classe ou à une réception de la classe. [Nouvelle règle][16]

• Règle lien 32 : Si la partition représente une instance d'un classificateur connu, les règles 351, 352 et 355 doivent être respectées (en prenant le classificateur qui type l'instance).

• Règle 356 : Si la partition représente une partie d'un classificateur, alors les comportements invoqués sont de la responsabilité des instances qui jouent le rôle de la partie représentée par la partition. Les procédures invoquées contenant un appel d'une opération ou envoyant un signal doivent avoir pour cible des objets à l'exécution qui jouent le rôle de la partie au moment où le message est envoyé.[tirée de [7] p.308] [17]

Remarque Tous les objets cibles d'une opération ou d'un passage de signal, invoqués par la même exécution de l'activité doivent jouer le rôle des parties de la même instance du classificateur structuré.

• Règle 357 : Dans la cas où une partition représente un attribut et ses sous-partitions représentent les valeurs de cet attribut, les valeurs doivent être du type de l'attribut (i.e. appartenir à l'ensemble des valeurs possibles pour l'attribut). [Nouvelle règle]

• Règle lien 33 : Une partition d'activité est une sorte de groupe d'activité et d'éléments nommés et doit donc respecter les règles présentes en sections 14.16 et 2.3.

14.18 Interruptible Activity Region

Contexte Une région interruptible d'activité (*interruptible activity region* en anglais) est un type de groupe d'activité.

Une région interruptible d'activité peut contenir un arc qui joue le rôle d'interrupteur pour cette région. Lorsqu'un tel arc est traversé l'ensemble des jetons et comportements de la région sont arrêtés.

Un arc d'interruption a sa propre notation graphique.

Règles de cohérence

• Règle 358 : Les arcs d'interruption d'une région d'activité interruptible doivent avoir leur nœud source dans la région et leur nœud cible en dehors de la région, dans la même activité qui contient la région. [[7] p.337]

Remarque Le type de cible n'est pas précisé dans la spécification. [18]

• Règle 359 : Toute région interruptible d'activité doit avoir au moins un arc d'interruption. [Nouvelle règle]

• Règle 360 : Les cibles d'un arc d'interruption doivent aboutir à un nœud exécutable qui doit contenir un handler d'exception. [Nouvelle règle]

• Règle lien 34 : Une région interruptible d'activité est une sorte de groupe d'activité et doit donc respecter les règles présentes en 14.16.

[16] inter-diag activités - classes

[17] regle contraignant l'implantation, pas bien comprise

[18] est-ce un object node ? ?

14.19 Executable Node

Contexte Un nœud exécutable (*executable node* en anglais) est une classe abstraite pour les nœuds d'activité qui peuvent être exécutés. Il est utilisé comme un point d'attachement pour les handlers d'exception.

Règles de cohérence

Pas de règles trouvées.

14.20 Parameter (Activities Diagram)

Contexte Au sein des activités, les paramètres (*parameter* en anglais) peuvent être associés aux concepts d'exception et de flot. De plus, les paramètres d'activité peuvent appartenir à des ensembles de paramètres (*parameter set* en anglais)(cf. section 14.21).

Le concept d'exception s'applique aux paramètres de sortie. Lorsqu'un jeton est placé dans un paramètre de sortie marqué comme étant une exception, une exception est levée. Tous les flots de l'activité sont alors interrompus. Les objets postés dans des paramètres de sortie qui ne sont pas un flot sont perdus. Les objets postés dans des paramètres de sortie qui représentent un flot ne sont pas affectés.

Les paramètres d'entrée qui représentent un flot (*streaming parameter* en anglais) sont accessibles par l'action même si celle-ci est en cours d'exécution. De même les paramètres de sorties qui représentent un flot peuvent se voir attribuer des jetons même si l'action n'a pas fini son exécution. En fait, les paramètres qui représentent des flots donnent à une action des flots de jetons (externes à l'action) alors même que l'action s'exécute.

Règles de cohérence

• Règle 361 : Un paramètre ne peut pas représenter un flot et une exception en même temps. [[7] p.352]

• Règle 362 : Un paramètre d'entrée ne peut pas être une exception. [[7] p.353]

• Règle 363 : Les comportements réentrants ne peuvent pas avoir de paramètres représentant des flots. [[7] p.353]

• Règle 364 : Une activité non associée au mot clé « singleExecution » (i.e. dont le méta-attribut `isSingleExecution` est à faux) ne peut pas avoir de paramètre d'entrée qui représente un flot. [Nouvelle règle]

Guides

[19]

14.21 Parameter Set

Contexte Un ensemble de paramètres (*parameter set* en anglais) est un élément qui fournit des ensembles alternatifs d'entrées et de sorties qu'un comportement peut né-

[19] guide prévention : Il est conseillé d'utiliser les paramètres d'exception dans les activités uniquement si l'on désire arrêter tous les flots de l'activité

cessiter. Chaque ensemble de paramètres agit comme un ensemble complet d'entrées et sorties pour un comportement et est exclusif des autres ensembles de paramètres du comportement.

Pour débuter une action, il suffit que tous les paramètres d'un ensemble de paramètres contiennent un jeton. Seuls les jetons présents dans l'ensemble de paramètres en question sont consommés. Ainsi, une action peut débuter même si des pins d'entrée qui se trouvent dans un autre ensemble de paramètres ne contiennent pas de jetons.

Un comportement avec des ensembles de paramètres d'entrée accepte des entrées provenant de paramètres de seulement l'un des ensembles à chaque exécution. Un comportement avec des ensembles de paramètres de sortie transmet des sorties aux paramètres de seulement un des ensembles de paramètres à chaque exécution.

Règles de cohérence

• Règle 365 : Les paramètres dans un ensemble de paramètres doivent tous être des paramètres d'entrée ou de sortie de la même entité paramétrée, et l'ensemble de paramètres est possédé par cette entité. [[7] p.354]

Remarque Dans la règle 365, le mot entité désigne soit une action soit une activité.

• Règle 366 : Si un comportement possède des paramètres d'entrée qui sont dans un ensemble de paramètres, alors n'importe quelle entrée qui n'appartient pas à un ensemble de paramètres doit être acheminée par flux. [[7] p.354]

• Règle 367 : Si un comportement possède des paramètres de sortie qui sont dans un ensemble de paramètres, alors n'importe quelle sortie qui n'appartient pas à un ensemble de paramètres doit être acheminée par flux. [[7] p.354]

14.22 Structured Activity Node

Contexte Un nœud d'activité structuré (*structured activity node* en anglais) représente une portion structurée de l'activité qui n'est pas partagée avec un autre nœud structuré, excepté pour une imbrication éventuelle. Il peut avoir des arcs de contrôle lui étant connectés et des pins dans les diagrammes d'activités structurés complets (cf. section 16.1).

Un nœud structuré se note avec le mot clé « structured ».

Règles de cohérence

• Règle 368 : Les arcs possédés par un nœud structuré doivent avoir leurs nœuds source et cible dans le même nœud d'activité structuré. [[7] p.362]

• Règle 369 : Les nœuds et les arcs contenus par un nœud structuré ne peuvent pas être contenus par un autre nœud structuré. [Règle dérivée du méta-modèle]

• Règle lien 35 : Les nœuds structuré d'activité sont une sorte de groupe d'activité et une sorte d'action. Ils doivent donc répondre aux règles énoncées en 14.16 et en 14.3.

14.23 Conditional Node

Contexte Un nœud conditionnel (*conditional node* en anglais) est un nœud structuré d'activité qui représente un choix exclusif entre plusieurs alternatives.

Un nœud conditionnel est composé d'une ou plusieurs clauses (cf. section 14.24). Chaque clause consiste en une section de test et une section de corps. Quand le nœud conditionnel commence son exécution, les sections de test des clauses sont exécutées. Même si plusieurs tests mènent à une valeur vraie, seul un corps est exécuté. Le choix est non déterministe sauf si des contraintes de séquencement dans le test des clauses sont spécifiées.

Une clause "else" est fournie. Cette clause est évaluée en dernier (donc si aucune autre clause n'est vraie) et retourne toujours la valeur vraie.

Les valeurs de sorties du nœud conditionnel peuvent être disponibles en sortie du nœud conditionnel. Ces valeurs de sorties sont créées par les différentes clauses.

Le méta-attribut booléen `isAssured` indique (s'il est vrai) qu'au moins une section de test d'une clause sera vraie.

Le méta-attribut booléen `isDeterminate` indique (s'il est vrai) qu'au plus une section de test de clause sera vraie de façon concurrente et donc que le choix de la clause est déterministe.

Règles de cohérence

• Règle 370 : Si le méta-attribut `isAssured` d'un nœud conditionnel est vrai, on doit absolument trouver au moins une section de test qui sera évaluée à vrai. [tirée de [7] p.314] [20]

• Règle 371 : Si le méta-attribut `isDeterminate` d'un nœud conditionnel est vrai, on doit absolument trouver au plus une section de test qui sera évaluée à vrai. [tirée de [7] p.314] [21]

Remarque Deux tests de clause peuvent s'exécuter en parallèle si les clauses qui les contiennent n'ont pas de relation de précédence l'une par rapport à l'autre.

• Règle 372 : Si le nœud conditionnel doit créer une valeur (i.e. a au moins un pin de sortie) alors au moins un corps de clause doit être exécuté. La règle 370 doit alors être appliquée. [Nouvelle règle]

• Règle 373 : Si des valeurs de sorties créées dans la partie de test ou de corps d'une clause sont utilisées en dehors du nœud conditionnel, chacune doit être créée (ou mise à jour) dans chaque clause. [tirée de [7] p.314]

• Règle 374 : Si le nœud conditionnel a des pins de sortie, toutes les sorties des corps des différentes clauses doivent placer des jetons dans chaque pin de sortie du nœud conditionnel (cas où les ensembles de paramètres ne sont pas utilisés par le nœud conditionnel), ou doivent remplir un ensemble de paramètres du nœud conditionnel. [Nouvelle règle]

Remarque Dans la règle précédente, par "remplir un ensemble de paramètres" on entend placer un jeton dans chaque pin de sortie qui appartient à un même ensemble de

[20] verif dynamique
[21] verif dynamique

paramètres.

• Règle 375 : Les sorties des corps des clauses doivent correspondre aux pins de sortie du nœud conditionnel auxquelles elles sont connectées. [Nouvelle règle]

• Règle 376 : Les sorties des corps des clauses ne peuvent être reliées qu'à des pins de sortie du nœud conditionnel. [Nouvelle règle]

• Règle lien 36 : Un nœud conditionnel est un nœud structuré d'activité et doit donc répondre aux conditions énoncées en 14.22.

Guides
[22]

14.24 Clause

Contexte Une clause (*clause* en anglais) est un élément qui est utilisé dans les nœuds conditionnels. Une clause représente une branche simple d'une construction conditionnelle. Elle inclut une section de test et une section de corps. Le corps est exécuté seulement si (mais pas nécessairement si) la section de test de la clause est évaluée à vraie. Il est possible de spécifier des conditions de précédence entre des clauses.

Règles de cohérence

• Règle 377 : La sortie des clauses doit être de type booléen. [Nouvelle règle]

• Règle 378 : Des clauses qui ont des relations de précédence doivent appartenir au même nœud conditionnel. [Nouvelle règle]

• Règle 379 : Aucun cycle entre clauses ne doit apparaître. [Nouvelle règle]

Remarque Un cycle entre clauses apparaît lorsqu'on arrive à la déduction qu'une clause doit se précéder elle-même.

14.25 Loop Node

Contexte Un nœud de boucle (*loop node* en anglais) est un nœud structuré d'activité qui représente une boucle avec ses sections d'initialisation, de test et de corps de boucle.

Le méta-attribut `isTestedFirst` indique s'il est vrai que le test de la condition de boucle est fait avant la première exécution de la boucle, et s'il est faux que la boucle est exécutée une fois avant le test de la condition.

Lorsque l'ensemble des jetons attendus pour le nœud de boucle est disponible, la première section exécutée est la section d'initialisation. Lorsque l'ensemble des nœuds appartenant à cette section est terminé, la section de test ou de corps de boucle est exécutée selon la valeur du méta-attribut `isTestedFirst`.

L'exécution de la section de test aboutit à la production d'un jeton dans un pin de sortie appelé `decider` qui est de type booléen. Dans le cas où ce booléen est vrai la section de corps de boucle est exécutée sinon l'exécution du nœud de boucle est terminée.

[22] guide prévention ? Il est conseillé de spécifier une clause `else` dans un nœud conditionnel.//justif :Ceci permet de s'assurer qu'au moins la section de corps associée à la clause else sera exécutée, et ne bloquera pas le flot si des valeurs sont attendues en sortie du nœud conditionnel.

L'exécution de la section de corps est terminée lorsque tous les nœuds qui constituent le corps de boucle ont terminé leur exécution.

Règles de cohérence

• Règle 380 : Chaque nœud exécutable qui appartient à un nœud de boucle appartient à une et une seule section (initialisation, test ou corps) du nœud de boucle. [Nouvelle règle]

Remarque Une action est un nœud exécutable et n'est donc ni un nœud de contrôle ni un nœud objet.

• Règle 381 : La section de test doit aboutir à la création d'un pin de sortie (référencée `decider` dans le méta-modèle) qui doit être de type booléen. [Nouvelle règle]

• Règle 382 : Au moins une des variables qui servent d'entrée à la section de test doit être produite par la section de corps. [Nouvelle règle]

• Règle lien 37 : Un nœud de boucle est un nœud structuré d'activité et doit donc répondre aux conditions énoncées en 14.22.

14.26 Expansion Region

Contexte Une région d'expansion (*expansion region* en anglais) est une région strictement emboitée dans une activité avec des entrées et des sorties explicites sous forme de nœuds d'expansion (cf. section 14.27) qui représentent une collection d'éléments.

La région d'expansion est exécutée pour chaque élément de la collection d'entrée. S'il y a plusieurs nœuds d'expansion en entrée d'une région, l'exécution de la région ne débute que quand toutes les collections d'entrée sont disponibles, et ces collections doivent avoir le même nombre d'éléments. La région d'expansion s'exécutera alors une fois pour chaque position d'élément.

Les sorties des régions d'expansion sont également modélisées par des nœuds d'expansion.

On peut placer un mot clé en haut à gauche, à l'intérieur de la région d'expansion pour spécifier la façon avec laquelle les exécutions interagissent. Le mot clé peut prendre les valeurs suivantes qui signifient :

- `parallel` : les multiples exécutions de la région d'expansion sont indépendantes et peuvent être effectuées en parallèle ;
- `iterative` : les exécutions sont dépendantes et doivent être exécutées une par une, dans l'ordre des éléments de la collection ;
- `stream` : un flux (séquence continue de données) des éléments de la collection est transmis dans une seule exécution, dans l'ordre des éléments de la collection.

Remarque Si la région d'expansion est structurée (avec le mot clé « structured » à l'intérieur de la région), l'activité ne prend qu'un seul jeton à la fois, attendant que l'activité soit finie pour en prendre une autre.

La figure 14.6 montre un exemple de région d'expansion qui a deux nœuds d'expansion en paramètre d'entrée et un en paramètre de sortie.

Règles de cohérence

• Règle 383 : Si une région d'expansion possède des sorties, elles doivent être des collections de même type et doivent contenir des éléments du même type que les entrées

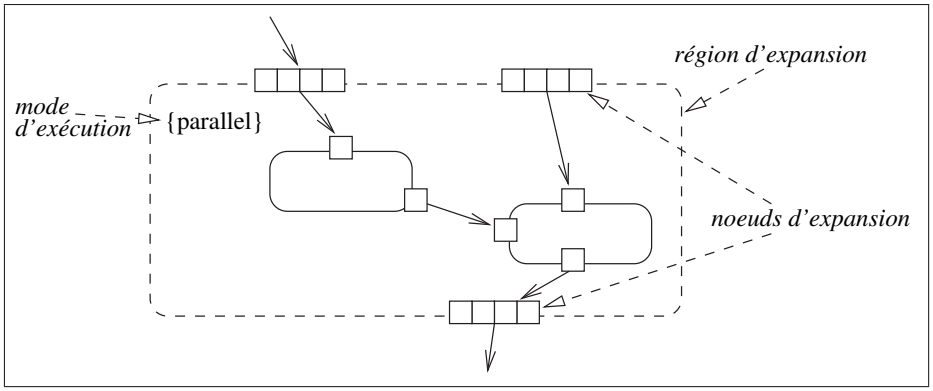

FIG. 14.6 – Exemple de région d'expansion

correspondantes. [tirée de [7] p.325] [23]

• RÈGLE 384 : Si une région d'expansion a deux collections d'entrée, il faut que ces collections aient le même nombre d'éléments. [Nouvelle règle][24]

• RÈGLE 385 : Le type des pins qui appartiennent à la région et qui sont connectés à des nœuds d'expansion (d'entrée ou de sortie) doit correspondre au type d'un élément de la collection représentée par le nœud d'expansion. [Nouvelle règle]

Remarque Cette règle ne peut être appliquée que si l'on connaît le type du pin et d'un élément de la collection (cf. guide 15).

• RÈGLE 386 : Le mode d'exécution d'une région (parallèle, itératif ou flot) doit toujours être précisé. [Nouvelle règle]

• RÈGLE LIEN 38 : Une région d'expansion est un nœud structuré d'activité et doit donc répondre aux conditions énoncées en 14.22.

14.27 Expansion Node

Contexte Un nœud d'expansion (*expansion node* en anglais) est un nœud d'objet utilisé pour indiquer un flot à travers les limites d'une région d'expansion. Le flot contient une collection de valeurs qui sont utilisées par la région (collection entrante) ou fournies à l'extérieur de la région (collection sortante).

La figure 14.6 présente un exemple de nœud d'expansion.

Règles de cohérence

• RÈGLE 387 : Un nœud d'expansion ne peut pas avoir un arc entrant et un arc sortant qui appartiennent tous les deux à la région d'expansion. [Nouvelle règle]

• RÈGLE 388 : Il doit y avoir un arc sortant d'un nœud d'expansion d'entrée vers au moins une action ou un pin de la région d'expansion. [Nouvelle règle]

• RÈGLE 389 : Si un nœud d'expansion de sortie est défini, il doit y avoir un arc sortant d'une action ou d'un pin de la région d'expansion et allant vers le nœud d'expansion de

[23] regarder avec nouvelle version spec

[24] Regle dynamique et Pas toujours détectable au niveau modèle

sortie. [Nouvelle règle]

• Règle 390 : Tout nœud d'expansion d'entrée doit posséder au moins un arc entrant dont la source est un élément externe à la région d'expansion.

• Règle 391 : Tout nœud d'expansion de sortie doit posséder au moins un arc sortant dont la cible est un élément externe à la région d'expansion.

• Règle lien 39 : Un nœud d'expansion est un nœud objet et doit donc respecter les règles énoncées en 14.4.

14.28 Variable (Activities Diagram)

Contexte Une variable (*variable* en anglais) spécifie le stockage de données partagées par des actions à l'intérieur d'un groupe. Il y des actions de lecture et d'écriture des variables. Il n'y a pas de contraintes de séquencement parmi les actions qui accèdent à la même variable.

Règles de cohérence

• Règle 392 : Toute valeur contenue par un variable doit être conforme au type de la variable et posséder des cardinalités permises par la multiplicité de la variable.[tirée de [7] p.362]

Chapitre 15

Relations

15.1 ActivityEdge

Contexte Un arc d'activité (*activity edge* en anglais) est une classe abstraite pour les connexions dirigées entre deux nœuds d'activités. Nous détaillons deux types particuliers d'arcs d'activités dans les sections 15.2 et 15.3.

Dans les diagrammes d'activités complets, on peut spécifier un poids (*weight* en anglais) qui donne le nombre minimum de jetons qui doivent traverser l'arc en même temps. Un poids spécifié par le littéral `null` veut dire que tous les jetons du nœud d'activité source sont offerts au nœud d'activité cible.

On peut utiliser une autre représentation des arcs d'activités à l'aide de connecteurs labellisés comme à la figure 15.1, où le cercle contient le nom de l'arc.

Optionnellement on peut faire figurer sur l'arc des conditions de garde, avec les mêmes conventions que pour les diagrammes de machines à états, une clause d'envoi et un champ `action-expression`.

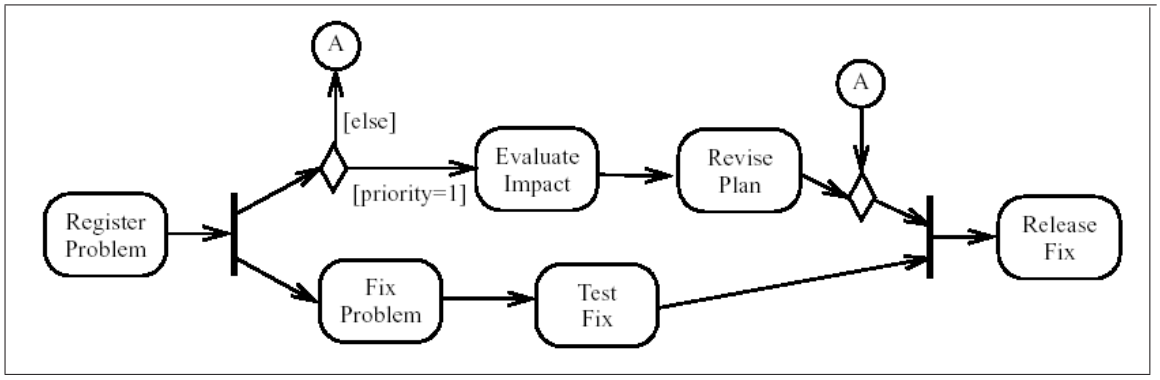

FIG. 15.1 – Notation des arcs d'activités avec connecteur

Règles de cohérence

• Règle 393 : Si un arc relie deux pins, alors ces pins doivent être de même type (comme 2 pins d'entrée, 2 nœuds de buffer central,etc.). [Nouvelle règle]

• Règle 394 : Un arc d'activités possède exactement un nœud d'activité source et un nœud d'activité cible. [Règle dérivée du méta-modèle]

• Règle 395 : Les nœuds d'activité source et cible d'un arc d'activité doivent être dans la même activité. [[7] p.293]

• Règle 396 : Les arcs d'activités ne peuvent être contenus que par des activités ou des groupes d'activités. [[7] p.293]

• Règle 397 : Si un poids est spécifié pour un arc d'activité, ce doit être une constante de type entier strictement positif. [Nouvelle règle]

• Règle 398 : Chaque connecteur d'arc d'activité avec un label donné doit être apparié avec exactement un autre connecteur possédant le même label dans le même diagramme d'activité. [Règle utilisateur][tirée de [7] p.294]

• Règle 399 : Dans la paire de connecteurs d'activités avec label, un des connecteurs doit avoir exactement un arc entrant et l'autre exactement un arc sortant, chacun avec le même type de flot (d'objet ou de contôle). [tirée de [7] p.294] [Règle utilisateur]

• Règle 400 : Le poids de l'arc d'activité doit être inférieur au poids des nœuds objet entrant et sortant lui étant reliés. [Nouvelle règle][1]

• Règle 401 : Seuls les arcs sortant d'un nœud de décision ou d'un nœud de bifurcation (*fork*) peuvent avoir des gardes différentes de `true`. [Nouvelle règle]

Guides

[2]

15.2 Control Flow

Contexte Un arc de flot de contrôle (*control flow* en anglais) est un arc qui permet de décrire le séquencement de deux nœuds objets. Ce type d'arc transmet uniquement des jetons de contrôle (par opposition aux jetons de données). Les jetons offerts en entrée par le nœud source sont tous transmis au nœud de contrôle cible.

Règles de cohérence

• Règle 402 : Un arc de flot de contrôle ne peut pas être relié à un nœud objet. [[7] p.315]

• Règle lien 40 : Les arcs de flot de contrôle étant un type d'arc d'activité, ceux-ci doivent respecter les contraintes de la section 15.1.

[1] origine de la règle ? ? ?

[2] guide prévention : Si les arcs sont nommés, nous conseillons de leur attribuer un nom unique dans l'espace de nommage que constitue l'activité.
justif : Meme si d'apres le méta-modèle, cela n'est pas necessaire (ownedElement), cela permet de distinguer des arcs différents

15.3 Object Flow

Contexte Un arc de flot d'objets (*object flow* en anglais) est un arc qui permet de décrire le séquencement de deux nœuds objets et transmet un jeton objet du nœud source au nœud cible.

Ceci permet de transmettre des données d'une action à une autre.

Il est possible de décrire l'effet qu'a une action sur un flot d'objets. Ceci se fait en indiquant une valeur de la méta-classe `ObjectFlowEffectKind` entre accolades. Les valeurs possibles sont donc `create, read, update, delete`.

Il est possible de décrire des envois d'objets en multicast grâce au mot clé `multicast`. Ceci décrit le fait que les mêmes données sont envoyées à plusieurs récepteurs qui ne sont représentés dans le diagramme d'activité que par un seul nœud. Il est possible de décrire une réception qui découle d'un envoi par multicast par le mot clé `multireceive`.

UML offre enfin la possibilité de spécifier des comportements de transformation des objets qui passent dans le flot. Chaque jeton offert à l'arc est passé par ce comportement, c'est la sortie du comportement de transformation qui est transmise au nœud cible de l'arc.

Dans les diagrammes d'activités complets, on peut spécifier l'effet que le comportement des actions a sur les objets traversant les arcs en représentant entre accolades l'effet à coté de l'arc ou dans l'autre représentation avec des broches, près de la broche. La figure 15.2 montre les deux représentations.

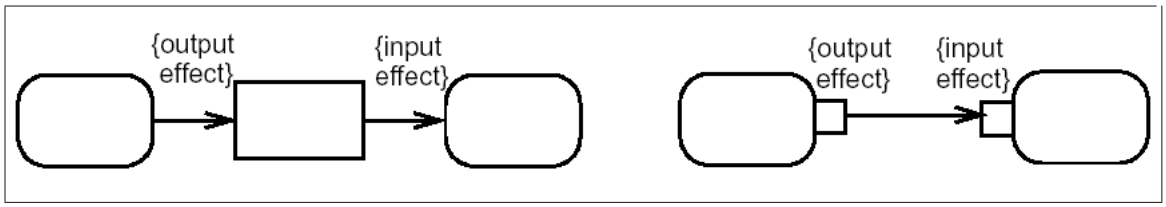

FIG. 15.2 – Effet spécifié sur les flots d'objets

Règles de cohérence

• RÈGLE LIEN 41 : Un flot d'objet est hérité d'un arc d'activité et doit respecter les règles de la section 15.1.

• RÈGLE 403 : Un flot d'objet doit avoir une action à au plus une de ses fins. [[7] p.345]

• RÈGLE 404 : Les nœuds d'objet connectés par un flot d'objet, avec l'intervention optionnelle de nœuds de contrôle, doivent avoir des types compatibles. [[7] p.345]

Remarque Ceci veut dire qu'en particulier le type du nœud d'objet découlant du flux doit être le même ou un supertype (type dont on a hérité) du type du nœud d'objet situé en amont du flux.

• RÈGLE 405 : Les nœuds d'objet connectés par un flot d'objet, avec l'intervention optionnelle de nœuds de contrôle, doivent avoir la même valeur limite haute. [[7] p.345]
Remarque La valeur limite haute correspond au nombre maximum de jetons autorisés

dans le nœud.

• Règle 406 : Un arc avec un poids constant ne doit pas avoir pour cible un nœud d'objet ou aboutir dans le flux aval à un nœud d'objet qui possède une valeur limite haute plus petite que ce poids. [[7] p.345]

• Règle 407 : Le mot clé « multireceive » ne peut être appliqué à un flot d'objets que si un flot d'objets en amont est associé au mot clé « multicast ». [Nouvelle règle]

• Règle 408 : Les seuls effets qui peuvent être décrits, d'une action sur un flot d'objets ne peuvent être que `create, read, update, delete`. [Règle dérivée du méta-modèle]

• Règle 409 : Un comportement de transformation a un paramètre d'entrée et un paramètre de sortie. Le type du paramètre d'entrée doit être de même type ou être d'un type plus général que le type des jetons objets entrants. Le paramètre de sortie doit être de même type ou une spécialisation du type de jetons objets attendu en aval. Le comportement ne doit pas avoir d'effet de bord. [[7] p.345]

• Règle 410 : Un flot d'objets peut avoir un comportement de sélection seulement s'il a un nœud objet comme source. [[7] p.345]

• Règle 411 : Un comportement de sélection a un paramètre d'entrée et un paramètre de sortie. Le paramètre d'entrée doit être un "bag" d'éléments de même type ou d'un type plus général que le type du nœud objet source. Le paramètre de sortie doit être le même ou un sous-type du type que le type du nœud objet source. Le comportement de sélection ne peut pas avoir d'effet de bord. [[7] p.345]

Remarque Un "bag" est une collection non ordonnée qui peut contenir deux fois le même élément.

• Règle 412 : Les méta-attributs `isMulticast` et `isMultireceive` ne peuvent pas être vrai simultanément. [[7] p.345]

Remarque La règle 412 se traduit au niveau modèle par le fait qu'on ne peut pas associer simultanément les mots clés « multicast » et « multireceive » au même flot d'objets.

• Règle 413 : Seuls les flots d'objets connectés à une action peuvent avoir des effets. Seuls les flots d'objets qui ont une action pour cible peuvent avoir pour effets `delete`. Seuls les flots d'objets qui ont des actions pour source peuvent avoir pour effets `create`. [[7] p.345]

Remarque Dans l'application de cette règle, avoir une action pour cible (respectivement pour source) peut également signifier avoir un pin d'entrée pour cible (resp. un pin de sortie pour source).

15.4 Exception Handler

[3]

Contexte Un handler d'exception (*exception handler* en anglais) est un élément qui spécifie un corps à exécuter dans le cas où une exception serait levée au cours de l'exécution d'un nœud protégé selon la notation de la figure 15.3.

[3] section a revoir avec la nouvelle version de la spec

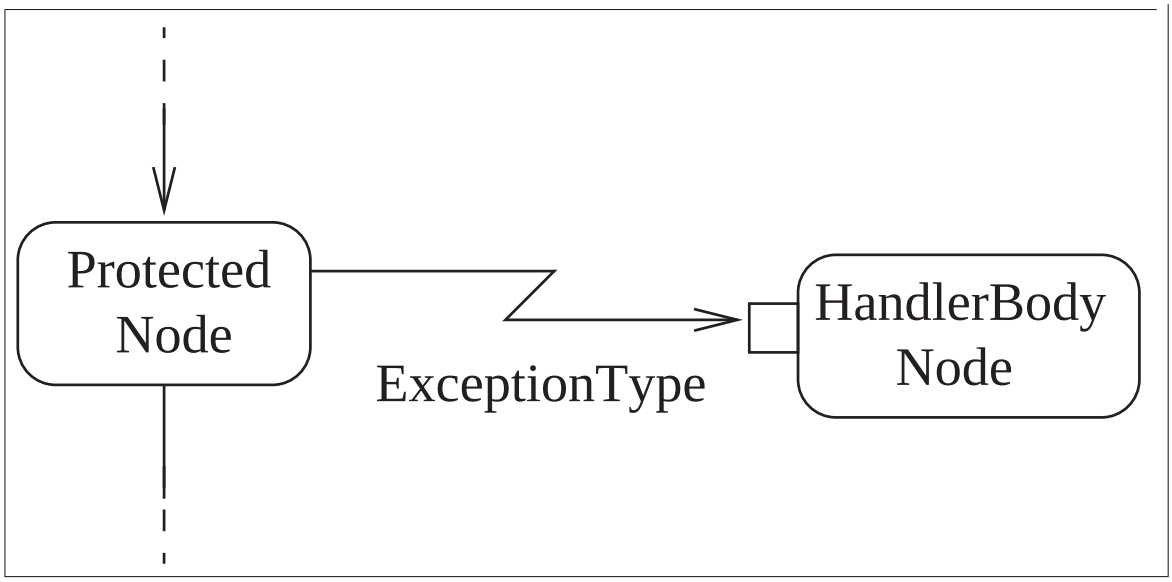

FIG. 15.3 – Notation d'un handler d'exception

Règles de cohérence

• RÈGLE 414 : Le corps de l'exception ne doit avoir aucun arc entrant ou sortant explicite. [[7] p.322]

• RÈGLE 415 : Les pins du corps du handler d'exception doivent correspondre en nombre et en type aux pins de sortie du nœud protégé. [[7] p.322]

• RÈGLE 416 : Seule une activité structurée peut avoir des handlers d'exception. [Nouvelle règle][4]

• RÈGLE 417 : Le nœud protégé et le handler d'exception doivent se trouver au même niveau d'emboitement. [tirée de [7] p.323]

• RÈGLE 418 : Tout corps d'exception a un nœud objet d'entrée. [Règle dérivée du méta-modèle]

Guides

[5]

[4]pas sûr

[5]guide prévention ?(inspiré de [1]p212) : Essayer d'isoler ou d'encapsuler des points de panne/faute connus plutot que lever des exceptions générales dans une region qui impliquerait l'arret total de l'activité.

Chapitre 16

Diagramme

16.1 Activity Diagram

Contexte Les diagrammes d'activités d'UML 2.0 permettent de représenter des flots dynamiques d'un système. Ils peuvent être utilisés pour décrire les activités effectués dans un processus général de *workflow*, aussi bien que pour décrire d'autres flots d'activités, comme un cas d'utilisation, ou un flot de contrôle détaillé.

Par opposition aux versions précédentes d'UML, les diagrammes d'activité peuvent désormais prendre en compte des flots complexes et beaucoup de concepts importés des réseaux de Petri ont été intégrés à UML 2.0.

On distingue plusieurs niveaux de diagrammes d'activités, selon que l'on utilise les concepts de base ou tous les concepts. On parlera alors :

- d'activités basiques ;
- d'activités intermédiaires ;
- d'activités structurées ;
- d'activités complètes ;
- d'activités complètes structurées ;
- et d'activités structurées additionnelles (*extra structured activities*).

Les règles suivantes portent sur les éléments qui peuvent être contenus dans un diagramme d'activités.

Règles de cohérence

• Règle 419 : Les nœuds graphiques contenus dans un diagramme d'activités ne peuvent être que :

1. un nœud d'activité (*activity node*) de type :
   (a) nœud objet (*object node*) de type :
   - broche (*pin*) de type entrée (*input pin*) ou sortie (*output pin*) ;
   - nœud paramètre d'activité (*activity parameter node*) ;
   - nœud buffer central (*central buffer node*) ou nœud central de mémoire tampon (incluant les nœuds de stockage de données (*data store node*)) ;
   - nœud d'expansion (*expansion node*).

   (b) nœud de contôle (*control node*) de type :
   - nœud initial (*initial node*) ;

  - nœud final (*final node*) (incluant le nœud final d'activité (*activity final node*) et le nœud final de flot (*flow final node*)) ;
  - nœud de décision (*decision node*) ;
  - nœud de bifurcation (*fork node*) ;
  - nœud d'union (*join node*) ;
  - nœud de fusion ou interclassement (*merge node*).

(c) un nœud exécutable (*executable node*) dont l'action (*action*) ou le nœud d'activité structuré (*structured activity node*) de type :
  - nœud conditionnel (*conditional node*) ;
  - nœud de boucle (*loop node*) ;

2. une partition d'activité (*activity partition*) ;
3. une région d'activité interruptible (*interruptible activity region*) ;
4. une région d'expansion (*expansion region*) ;
5. une pré-condition ou post-condition locale ;
6. un ensemble de paramètres (*parameter set*). [[7] p.365-368]

• Règle 420 : Les chemins graphiques pouvant être contenus dans un diagramme d'activités peuvent être : [[7] p.366]
- un arc d'activité (*activity edge*) de type
  - flot de contrôle (*control flow*) ;
  - flot d'objet (*object flow*) ;
- un handler d'exception (*exception handler*).

• Règle 421 : Les événements dans un diagramme d'activité peuvent seulement être attachés à la transition qui va du point de départ (*start point*) [1] à la première action. [tirée de [1] p.164]

• Règle 422 : Une activité a forcément un point de départ pour le flot de jetons. Ce point de départ peut être un nœud initial, une action d'acceptation d'événement (*accept event action* en anglais) ou un nœud de paramètre d'activité en mode `in`. [Nouvelle règle][2]

• Règle 423 : Toute activité décrite par un diagramme doit correspondre à un réseau de Petri quasi-vivant. [Nouvelle règle]

Remarque Un réseau de Petri est quasi-vivant si pour toute transition, il existe un marquage accessible qui active cette transition.

La figure 16.1 présente une activité qui ne respecte pas la règle 423. En effet, pour que le nœud d'union soit traversé et l'arc 4 tiré, il faut que les arcs 1, 2 et 3 offrent un jeton simultanément. Or, les arcs 1 et 2 ne pourront jamais offrir un jeton simultanément car un nœud de décision les précède.

• Règle 424 : Un arc composé ne doit pas contenir de cycle. [Nouvelle règle]

Remarque Nous appelons arc composé un chemin constitué d'un ensemble d'arcs, de nœuds de contrôle, et de nœuds objets. Un arc composé a comme source et comme cible un ensemble d'actions.

La figure 16.2 montre un exemple d'arc composé avec circuit.

[1] noeud initial ?, verifier avec la nouvelle version
[2] ou in/out ?

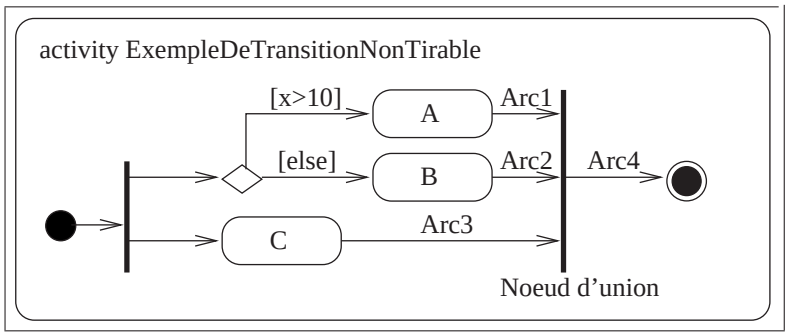

FIG. 16.1 – Exemple d'activité qui ne respecte pas la règle 423

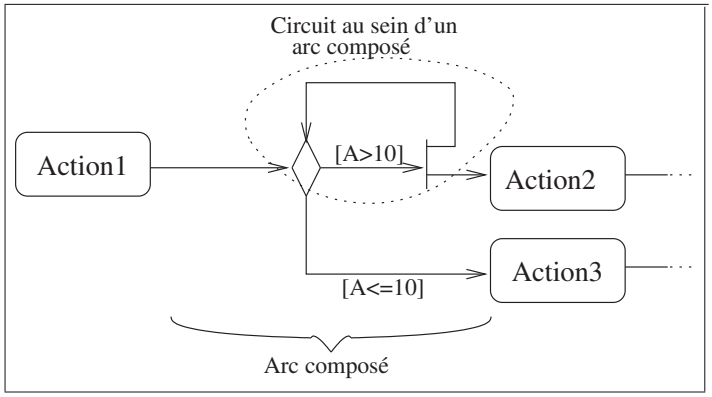

FIG. 16.2 – Arc composé avec circuit

Sixième partie

Diagramme d'interaction

Chapitre 17

Éléments

17.1 Combined Fragment

Contexte général Un fragment combiné (*combined fragment* en anglais) définit une expression de fragments d'interaction. Un fragment combiné est défini par un opérateur d'interaction et les opérandes d'interactions correspondants (cf. section 17.4). L'utilisation des fragments combinés permet de décrire de manière concise plusieurs traces d'exécution.

Les différents opérateurs d'interaction sont `seq`, `alt`, `opt`, `break`, `par`, `strict`, `loop`, `region`, `neg`, `assert`, `ignore` et `consider`.

Remarque La figure 17.1 montre un exemple d'utilisation d'un fragment combiné de type `alt` (*alternative*), de plusieurs opérandes d'interaction et d'une contrainte d'interaction ([x>0]).

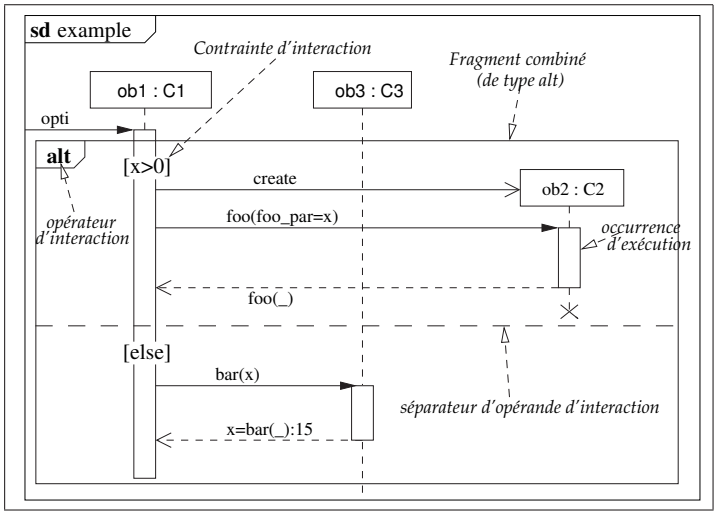

FIG. 17.1 – Exemple de fragments combinés de type `alt`

17.1.1 Règles générales

Règles de cohérence

• Règle 425 : Tout fragment combiné contient un opérateur d'interaction. [Règle dérivée du méta-modèle]

• Règle 426 : Les fragments combinés d'opérateur d'interaction `opt`, `loop` ou `neg` doivent contenir exactement un opérande d'interaction. [[7] p.409]

Guides

[1]

17.1.2 Fragment combiné d'opérateur d'interaction `alt`

Contexte Un fragment combiné dont l'opérateur d'interaction est `alt` exprime la possibilité de choisir différents comportements. Ceci est réalisé par le choix d'un unique opérande d'interaction en fonction des contraintes d'interaction.

Règles de cohérence

• Règle 427 : Dans le cas d'un fragment combiné d'opérateur d'interaction `alt`, il ne doit pas y avoir un seul opérande d'interaction. [Nouvelle règle]

• Règle 428 : Dans le cas d'un fragment combiné d'opérateur d'interaction `alt`, un opérande d'interaction de contrainte `else` doit se trouver après un autre opérande d'interaction et en dernier opérande d'interaction du fragment combiné. [Nouvelle règle]

• Règle 429 : Si un fragment combiné est d'opérateur d'interaction `alt`, tous les opérandes d'interaction qui le composent ou toute continuation (cf. section 17.7) doit possèder exactement une contrainte d'interaction avant la première occurrence d'événement (i.e. en début de l'opérande). [Nouvelle règle]

• Règle 430 : Si un fragment combiné est d'opérateur d'interaction `alt`, aucune des contraintes d'interaction comprises dans les différents opérandes d'interaction ne doivent toujours être vraies (excepté pour le dernier opérande) ou toujours être fausses. [Nouvelle règle]

Guides

[2]

17.1.3 Fragment combiné d'opérateur d'interaction `opt`

Contexte Un fragment combiné d'opérateur d'interaction `opt` représente un choix dans le comportement où soit seul l'opérande d'interaction contenu dans le fragment combiné s'exécute, soit rien ne se produit.

[1] guide de prévention : Nous conseillons de toujours faire figurer au moins une occurrence d'événement à l'intérieur d'un opérande d'interaction d'un fragment combiné. (Un opérande vide d'un fragment combiné ne sert à rien.) Ce guide a été généralisé suite à un guide initialement pensé pour un fragment combiné d'opérateur `opt`.

[2] guide de prévention : Nous conseillons d'utiliser en dernière opérande une opérande de contrainte d'interaction "else" plutôt qu'une contrainte implicitement vraie.

Règles de cohérence

• Règle lien 42 : Un fragment combiné d'opérateur d'interaction `opt`, doit contenir exactement une opérande d'interaction (cf. règle 426).

• Règle 431 : La contrainte d'interaction d'un opérateur d'interaction dont l'opérateur est `opt` ne doit pas être toujours vraie ou toujours fausse. [Nouvelle règle]

Remarque Une contrainte est toujours vraie ou toujours fausse si elle ne fait intervenir aucun terme variable (exemple : $2 > 1$).

Guides

[3]

17.1.4 Fragment combiné d'opérateur d'interaction `break`

Contexte Un fragment combiné d'opérateur d'interaction `break` représente un scénario d'arrêt qui est exécuté à la place du reste du fragment d'interaction englobant.

Règles de cohérence

• Règle 432 : Un fragment combiné d'opérateur d'interaction `break` doit être global (i.e. doit couvrir toutes les lignes de vie du fragment d'interaction englobant. [tirée de [7] p.410]

17.1.5 Fragment combiné d'opérateur d'interaction `par`

Contexte Un fragment combiné d'opérateur d'interaction `par` désigne un interclassement (ou entrelacement) entre les comportements des opérandes. Les occurrences des événements des divers opérandes d'interaction peuvent être entrelacées de toutes les façons tant que l'ordre imposé par chaque opérande est préservé.

Règles de cohérence

[4]

17.1.6 Fragment combiné d'opérateur d'interaction `seq`

Contexte Un fragment combiné d'opérateur d'interaction `seq` désigne un entrelacement faible entre les comportements des opérandes. L'entrelacement faible est défini par [7] à la page 410 par trois propriétés :

- L'ordre des occurrences d'événements à l'intérieur de chacun des opérandes est maintenu dans le résultat.
- Les occurrences d'événements de différentes lignes de vie venant de différents opérandes peuvent apparaître dans n'importe quel ordre.

[3] guide de prévention : nous déconseillons d'utiliser l'opérateur d'interaction `opt` vide mais plutôt `alt` avec un opérande d'interaction de contrainte `else` à l'intérieur duquel on spécifie un "non-événement". Ce dernier doit être préalablement défini.

[4] regle sur les deadlocks ? : il doit toujours exister un entrelacement possible ne conduisant pas à une situation de deadlock.

– Les occurrences d'événements d'une même ligne de vie venant de différents opérandes sont ordonnés de telle sorte que l'occurrence d'événement du premier opérande apparait avant celle du deuxième.

Règles de cohérence
Pas de règles trouvées.

17.1.7 Fragment combiné d'opérateur d'interaction `strict`

Contexte Un fragment combiné d'opérateur d'interaction `strict` définit un séquencement strict entre les comportements des opérandes.

Règles de cohérence
Pas de règles trouvées.

17.1.8 Fragment combiné d'opérateur d'interaction `neg`

Contexte Un fragment combiné d'opérateur d'interaction `neg` définit des traces invalides.

Règles de cohérence
• Règle lien 43 : Un fragment combiné d'opérateur d'interaction `neg`, doit contenir exactement une opérande d'interaction (cf. règle 426).

17.1.9 Fragment combiné d'opérateur d'interaction `critical`

Contexte Un fragment combiné d'opérateur d'interaction `critical` représente une région critique, ce qui signifie que des occurrences d'événement ne peuvent pas être entrelacées avec les traces de la région critique (même si le fragment combiné critique appartient à un autre fragment combiné d'opérateur d'interaction parallèle par exemple). Cela implique que la région doit être traitée de manière atomique.

La figure 17.2 (provenant de [7]) donne un exemple d'utilisation d'un fragment combiné d'opérateur d'interaction `critical` à l'intérieur d'un fragment combiné d'opérateur d'interaction `par`.

Règles de cohérence
Pas de règles trouvées

17.1.10 Fragment combiné d'opérateur d'interaction `ignore` et `consider`

Contexte Un fragment combiné d'opérateur d'interaction `ignore` indique que les types de certains messages sont ignorés dans le fragment combiné. Cela implique aussi que les messages ignorés peuvent apparaître à n'importe quel endroit de la trace.

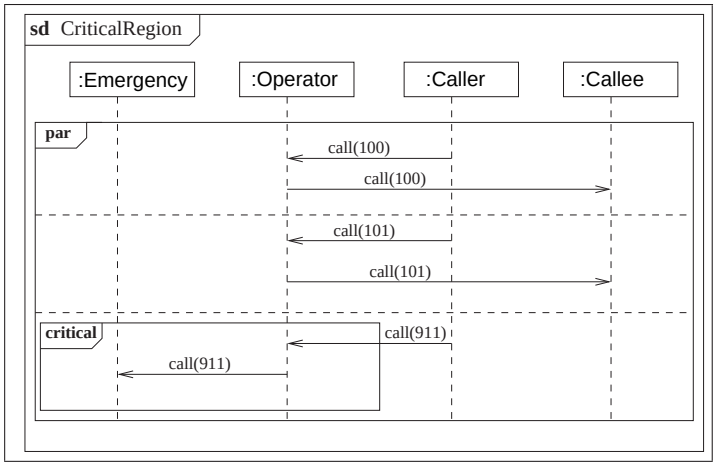

FIG. 17.2 – Exemple de fragments combinés d'opérateur d'interaction `par` et `critical`

À l'inverse un fragment combiné d'opérateur d'interaction `consider` signifie que seuls certains messages vont être considérés à l'intérieur du fragment combiné. C'est équivalent à définir tous les autres messages comme "ignorés".

Règles de cohérence

• RÈGLE 433 : La syntaxe des blocs `ignore/consider` doit suivre : [[7] p.413]

```
(ignore | consider ){ <message name>{,<message name>}* }
```

• RÈGLE 434 : Dans un fragment combiné d'opérateur d'interaction `consider`, seuls les messages marqués comme « considérés » peuvent apparaître. [Nouvelle règle]

• RÈGLE 435 : Dans un fragment combiné d'opérateur d'interaction `ignore`, les messages marqués comme « ignorés » ne doivent pas apparaître. [Nouvelle règle]

• RÈGLE 436 : Les messages ignorés ou considérés doivent correspondre à des noms de messages connus. [Nouvelle règle][5]

Remarque L'ensemble des noms de messages connus correspond à l'ensemble des opérations des différentes classes représentées par les lignes de vie de l'interaction et l'ensemble des signaux visibles par ces classes.

17.1.11 Fragment combiné d'opérateur d'interaction `assert`

Contexte Un fragment combiné d'opérateur d'interaction `assert` représente une assertion. Ainsi, la séquence de trace de l'opérande décrit par l'assertion est la seule trace valide.

Règles de cohérence

• RÈGLE LIEN 44 : Toute trace définie par un fragment d'interaction qui commence par les messages qui aboutissent à la trace définie par le bloc `assert` et qui continue par un échange de messages ne respectant pas le bloc `assert` doit être définie comme une trace invalide de l'interaction (voir règle 441).

[5] Inter-diag sequence - classes ?

17.1.12 Fragment combiné d'opérateur d'interaction `loop`

Contexte Un fragment combiné d'opérateur d'interaction `loop` représente une boucle. L'opérande d'interaction sera répété un certain nombre de fois.

Règles de cohérence

• Règle 437 : Un fragment combiné d'opérateur d'interaction `loop` doit suivre la syntaxe suivante : [[7] p.413]

```
loop ['(' <minint> [, <maxint> ] ')' ]
```

Remarque Le fragment combiné d'opérateur d'interaction `loop` est le seul cas où les contraintes d'interactions portant sur les champs `minint` et `maxint` s'appliquent [[7] p.409].

• Règle 438 : Pour un fragment combiné d'opérateur d'interaction `loop`, le champ `minint` est un entier positif ou nul et le champ `maxint` est un entier strictement positif. [[7] p.422]

• Règle 439 : Pour un fragment combiné d'opérateur d'interaction `loop`, le champ `minint` doit etre $\leq$ à `maxint`. [[7] p.422]

• Règle 440 : Un fragment combiné d'opérateur d'interaction `loop`, doit contenir exactement une contrainte d'interaction. [[7] p.409]

17.2 Interaction

Contexte Une interaction (*interaction* en anglais) est la description d'un comportement qui se focalise sur l'échange observable d'informations entre éléments connectables (*connectableElements* en anglais).

Une interaction est caractérisée par une paire de traces, certaines valides et d'autres invalides. Les traces peuvent cependant se trouver ni dans l'une ni dans l'autre catégorie et ne peuvent être jugées ni valides ni invalides. L'ensemble des traces invalides peut être caractérisé par des fragments combinés d'opérateur d'interaction `neg` (cf. partie 17.1.8).

Règles de cohérence

• Règle 441 : L'ensemble des traces valides et l'ensemble des traces invalides d'une même interaction sont disjoints. [Nouvelle règle]

• Règle 442 : Une interaction ne peut contenir que des diagrammes de séquence, de communication, de vue d'ensemble d'interaction et de *timing*. [[7] p.420]

• Règle 443 : Le classificateur qui contient l'interaction doit être un `BehavioredClassifier`. Se référer à l'annexe A.1 pour en avoir la liste. [Règle dérivée du méta-modèle]

17.3 Interaction Constraint

Contexte Une contrainte d'interaction (*interaction constraint* en anglais) est une expression booléenne qui conditionne l'exécution d'un opérande dans un fragment combiné.

Règles de cohérence

• Règle 444 : La spécification d'une contrainte d'interaction doit suivre la syntaxe : [[7] p.422]

```
interactionconstraint ::= [ [  Boolean Expression | else ] ]
```

Remarque Cette règle ne s'applique pas pour un fragment combiné d'opérateur d'interaction `loop`.

• Règle lien 45 : Une contrainte d'interaction est une contrainte et doit donc respecter les règles décrites en 2.7.

• Règle 445 : Les variables dynamiques qui prennent part à l'expression de la contrainte doivent appartenir à l'élément correspondant à la ligne de vie. [[7] p.422]

• Règle 446 : Les références aux entiers de boucle ne peuvent être présents que si la contrainte d'interaction appartient à un fragment combiné d'opérateur d'interaction `loop`.[[7] p.422]

Remarque Dans la cas où ces entiers sont spécifiés, il faut qu'ils respectent la règle 437.

• Règle 447 : Une contrainte d'interaction se trouve toujours en tout début d'opérande d'interaction. [[7] p.422]

• Règle 448 : Une contrainte d'interaction qui fait référence à des entiers ne peut s'appliquer que dans un opérande d'interaction dont le fragment combiné est d'opérateur d'interaction `loop`. [[7] p.409]

17.4 Interaction Operand

Contexte Un opérande d'interaction (*interaction operand* en anglais) est contenu dans un fragment combiné et représente un opérande de l'expression donnée par le fragment combiné englobant. Il peut être conditionné par une contrainte d'interaction qui sert de condition de garde.

Règles de cohérence

• Règle 449 : La condition de garde d'un opérande d'interaction doit être placée avant (i.e. au-dessus de) la première occurrence d'événement à l'intérieur de l'opérande d'interaction. [[7] p.426]

• Règle 450 : La condition de garde d'un opérande d'interaction doit référencer uniquement des valeurs locales à la ligne de vie sur laquelle elle se trouve ou des valeurs globales à l'ensemble de l'interaction. [[7] p.426]

17.5 Interaction Occurrence

Contexte Une occurrence d'interaction (*interaction occurrence* en anglais) se réfère à une interaction. C'est un moyen de copier le contenu de l'interaction référencée à l'endroit de l'occurrence d'interaction.

Règles de cohérence

• Règle 451 : Une occurrence d'interaction doit être décrite avec la syntaxe suivante : [[7] p.424]

```
name ::=[ attribute-name = ][collaborationoccurrence.]
            interactionname [ ( arguments ) ] [: return-value]

argument ::= in-argument [ out out-argument]
```

où `attribute-name` se réfère à un attribut d'une des lignes de vie de l'interaction, et `collaborationoccurence` est une identification de l'occurrence de collaboration qui fait le lien entre les lignes de vie d'une collaboration.

• Règle 452 : Les noms des portes (voir section 17.8) des occurrences d'interaction doivent correspondre aux noms des portes de l'interaction référencée. [[7] p.423]

• Règle 453 : L'occurrence d'interaction doit couvrir toutes les lignes de vie qui apparaissent dans l'interaction référencée. [[7] p.423]

• Règle 454 : Les arguments de l'occurrence d'interaction doivent correspondre aux paramètres de l'interaction référencée. [[7] p.423]

• Règle 455 : Les arguments doivent seulement être des constantes, des paramètres de l'interaction englobante ou des attributs du classificateur qui contient l'interaction englobante. [[7] p.423]

• Règle 456 : Toute occurrence d'interaction doit référencer exactement une interaction. [Règle dérivée du méta-modèle]

17.6 Part Decomposition

Contexte Une décomposition de partie (*part decomposition* en anglais) est la description d'interactions internes d'une ligne de vie.

En fait, cette construction permet de décrire les échanges de messages qui ont lieu au sein de la structure interne de la classe associée à la ligne de vie.

Une décomposition de partie est une spécialisation d'une occurrence d'interaction.

Règles de cohérence

• Règle 457 : La décomposition de parties s'applique uniquement sur des parties de structures internes et pas sur des parties de collaboration. [[7] p.431]

• Règle 458 : Soit dans une interaction X, une ligne de vie L de classe C et qui se décompose en D. À l'intérieur de X, il y a une séquence de construction le long de L (comme des fragments combinés, des occurrences d'interactions, et des occurrences d'événement). Aussi, une séquence correspondante de ces constructions doit apparaître dans D, qui se correspondent une à une et sont dans le même ordre.
i) Les fragments combinés qui couvrent L correspondent à des fragments combinés « extra-global » dans D.
ii) Toute occurrence d'interaction qui couvre L correspond à une occurrence d'interaction globale (qui couvre toutes les lignes de vie) dans D.
iii) Une occurrence d'événement sur L est considérée comme étant une porte qui doit correspondre à une porte de D. [[7] p.431]

Remarque Un fragment combiné « extra-global » a la caractéristique d'avoir ses frontières qui dépassent de l'interaction de décomposition.

• Règle 459 : Toute interaction qui contient un fragment combiné « extra-global » doit être référencée comme étant une décomposition de parties. [Nouvelle règle]

• Règle 460 : Soit dans une interaction X, une ligne de vie L de classe C qui se décompose en D. Supposons également qu'à l'intérieur de X une occurrence d'interaction U couvre L. En accord avec la contrainte sur U, il existe une occurrence correspondante CU dans D. À l'intérieur de l'interaction référencée par U, L devrait aussi être décomposée et la décomposition doit référencer CU. [[7] p.431]

17.7 Continuation

Contexte Une continuation (*continuation* en anglais) est un moyen syntaxique de définir des séquences de différentes branches de fragment combiné alternatif (cf. section 17.1.2). Les continuations sont intuitivement similaires à des labels représentant des points intermédiaires dans un flot de contrôle.

La figure 17.3 (inspirée de [7]) montre un exemple d'utilisation d'une occurrence d'interaction et de continuations.

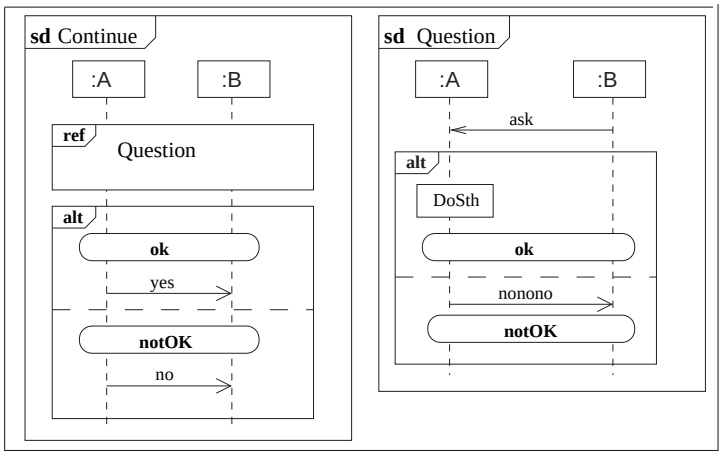

Fig. 17.3 – Exemple d'utilisation d'une occurrence d'interaction et de continuations

Règles de cohérence

• Règle 461 : Les continuations avec le même nom et qui appartiennent au même classificateur doivent couvrir le même ensemble de lignes de vie. [[7] p.415]

• Règle 462 : Les continuations sont toujours globales dans le fragment d'interaction, c'est-à-dire qu'elles couvrent toutes les lignes de vie couvertes par le fragment d'interaction englobant. [[7] p.415]

• Règle 463 : Les continuations apparaissent toujours en tout début ou en toute fin du fragment d'interaction englobant. [[7] p.415]

• Règle 464 : Les continuations peuvent être contenues uniquement par des fragments d'interaction qui sont des fragments combinés d'opérateur d'interaction `alt` ou `opt`. [Nouvelle règle]

[6]

• Règle 465 : Une continuation a toujours un nom. [Nouvelle règle]

• Règle 466 : Toute continuation en début d'opérande de fragment composé doit correspondre (i.e. porter le même nom) d'au moins une autre continuation :

– qui appartient au même classificateur ;
– qui appartient à une opérande de fragment composé antérieur (i.e. qui est exécutée avant) ;
– et qui se trouve en fin d'opérande. [Nouvelle règle]

• Règle 467 : Toute continuation en fin d'opérande de fragment composé doit correspondre (i.e. porter le même nom) d'au moins une autre continuation :

– qui appartient au même classificateur ;
– qui appartient à une opérande de fragment composé postérieur (i.e. qui est exécutée après) ;
– et qui se trouve en début d'opérande. [Nouvelle règle]

17.8 Gate

Contexte Une porte (*gate* en anglais) est un point de connexion entre un message extérieur à un fragment d'interaction et un message appartenant à ce fragment d'interaction.

Règles de cohérence

• Règle 468 : Le message sortant d'une (resp. entrant dans une) porte d'une occurrence d'interaction doit correspondre au message venant vers (resp. de) la porte de même nom et contenue dans l'interaction référencée par l'occurrence d'interaction. [[7] p.418]

• Règle 469 : Le message venant d'une (resp. vers une) porte contenue dans un fragment combiné doit correspondre au message venant vers (resp. de) cette porte à l'extérieur du fragment combiné. [[7] p.418]

Guides

[7]

17.9 Lifeline

Contexte Une ligne de vie (*lifeline* en anglais) représente un participant individuel d'une interaction.

Règles de cohérence

• Règle 470 : La définition d'une ligne de vie doit respecter la syntaxe suivante : [[7] p.427]

[6] réfléchir à la possibilité de contenir des contraintes d'interaction ou continuations, en début ou fin ?

[7] guide de prévention : Les portes peuvent être identifiées par un nom s'il a été spécifié ou [7] conseille de nommer les portes avec un identificateur construit en concaténant la direction du message avec le nom du message (par exemple, out_CardOut).–> guide de style

```
lifelineident::=[connectable_element_name[[selector]]][:class_name]
                [decomposition] | self
selector ::= expression
decomposition ::= ref interactionident
```

où `lifelineident` ne peut pas être vide.

Remarque Si le nom est le mot clé `self`, la ligne de vie représente l'objet du classificateur qui contient l'interaction possédant la ligne de vie.

• Règle 471 : Le champ `selector` ne doit apparaître que si l'élément connecté est multi-valué. [[7] p.427]

• Règle 472 : Le classificateur qui contient l'élément connectable (*ConnectableElement* en anglais, voir section A.3) référencé doit être le même classificateur, ou un ancêtre, du classificateur qui contient l'interaction englobant la ligne de vie. [[7] p.427] [8]

• Règle 473 : Le champ `class_name` doit correspondre à une classe qui peut jouer un rôle au sein du classificateur contenant l'interaction. [Nouvelle règle]

• Règle 474 : Si une ligne de vie est créée (resp. détruite) lors d'un diagramme de séquence, et que la classe de l'objet créé (resp. détruit) est reliée avec une relation de composition en tant que composant, alors la classe de l'objet créateur (resp. destructeur) doit être ou appartenir à la classe composite. [Nouvelle règle]

Guides

• Guide 16 : Nous conseillons d'exprimer explicitement le nom de la classe à laquelle fait référence la ligne de vie.

• Justification 16 : Nous conseillons d'exprimer explicitement le nom de la classe à laquelle fait référence la ligne de vie. Ceci permet de vérifier les règles 473, 488, 490, 492 et 474.

17.10 Event Occurrence

Contexte Une occurrence d'événement (*event occurrence* en anglais) représente l'action d'envoi ou de réception d'un message par une classe à un moment donnné du temps.

Dans les diagrammes d'interactions, des lignes de vie représentent chacune des classes.

Règles de cohérence

• Règle 475 : Tout occurrence d'événement se trouve sur une ligne de vie. [Règle dérivée du méta-modèle]

[8] inter diagramme sequence - classe ou component

17.11 Stop

Contexte Un symbole de stop (*stop* en anglais) est une occurrence d'événement qui définit la destruction de l'instance représentée par la ligne de vie sur laquelle le "stop" se produit.

Règles de cohérence

• Règle 476 : Sur une ligne de vie donnée, aucune occurrence d'événement ne peut se produire en-dessous de la marque de stop au sein d'un opérande d'interaction. [[7] p.434]

17.12 Execution Occurrence

Contexte Une occurrence d'exécution (*execution occurrence* en anglais) est l'instanciation d'une unité comportementale sur une ligne de vie. Puisque l'exécution du comportement a une certaine durée, elle est représentée par une occurrence d'événement de début et une occurrence d'événement de fin.

Règles de cohérence

• Règle 477 : Les occurrences d'événements de début et de fin d'une occurrence d'exécution doivent être sur la même ligne de vie. [[7] p.417]

• Règle 478 : Toute occurrence d'exécution commence par la réception d'un message synchrone. [Nouvelle règle]

• Règle 479 : Toute occurrence d'exécution termine par l'envoi d'un message de retour. [Nouvelle règle]

Remarque Le section 19.1.1 contient des règles complémentaires sur ce type de construction.

17.13 State Invariant

Contexte Un invariant d'état (*state invariant* en anglais) est une contrainte sur l'état d'une ligne de vie. Dans ce cas, le mot état peut également désigner des valeurs éventuelles d'attributs de la ligne de vie.

Règles de cohérence

• Règle 480 : Un invariant d'état doit :
- soit respecter les règles énoncées sur les contraintes (cf. section 2.7) ;
- soit correspondre au nom d'un état d'un diagramme d'état décrivant le comportement de la classe de la ligne de vie en question. [Nouvelle règle][9]

[9] seuls cas possibles ?

Chapitre 18

Relations

18.1 General Ordering

Contexte Dans les diagrammes de séquence, il est possible de spécifier des relations d'ordre général (*general ordering* en anglais) entre les différents messages. Il est à noter que certaines relations d'ordre général n'ont pas besoin d'être marquées explicitement puisque la précédence des messages est donnée de manière graphique avec l'ordre d'apparition vertical des messages dans les diagrammes de séquence.

La figure 18.1 donne la notation d'une relation de précédence.

FIG. 18.1 – Notation d'une relation d'ordre général

Règles de cohérence

• RÈGLE 481 : Une relation d'ordre général a exactement une occurrence d'événement source et une occurrence d'événement cible.[Règle dérivée du méta-modèle]

• RÈGLE 482 : Une relation d'ordre général doit être cohérente avec toute autre relation de précédence, dans le sens où deux relations de précédence ne doivent pas s'opposer. [Nouvelle règle]

18.2 Messages

Un message définit plusieurs formes de communication entre instances d'un modèle, elle peut prendre ces 3 formes :

– la levée d'un signal ;
– l'invocation d'une opération ;
– la création ou la destruction d'une instance.

Les règles de cette section vont être étudiées en trois grandes parties, l'une concernant les éléments connectés par un message, l'autre concernant les formes de messages, et la dernière concernant plus particulièrement les diagrammes de communication.

Remarque Dans les diagrammes de communication, les messages doivent respecter les règles de nommage et de numérotation énoncées en 18.2.3.

18.2.1 Éléments connectés

Contexte En général un message est envoyé d'une instance à une autre instance. Dans ce cas le message est de type complet (*complete* en anglais). Cependant, il existe des messages perdus (*lost* en anglais), trouvés (*found* en anglais), ou inconnus (*unknown*) pour lesquels ni l'expéditeur ni le récepteur ne sont présents.

Une occurrence d'événement est la fin d'un message connecté soit à une porte soit à une ligne de vie.

Règles de cohérence

• Règle 483 : Un message doit relier soit :
– deux occurrences d'événement (message de type `complete`) ;
– une occurrence d'événement comme source du message et un élément inconnu (message de type `lost`) ;
– un élément inconnu comme source et une occurrence d'événement comme cible (message de type `found`) ;
– deux éléments inconnus (message de type `unknow`). [Nouvelle règle]

Remarque La spécification d'UML préconise de ne pas utiliser les messages `unknown`.

18.2.2 Différentes formes de messages

Contexte Un message peut représenter :
– l'appel synchrone d'une opération ;
– l'appel asynchrone d'une opération ;
– la levée d'un signal synchrone ;
– la levée d'un signal asynchrone.

La figure 18.2 montre les différents types de messages et leurs représentations.

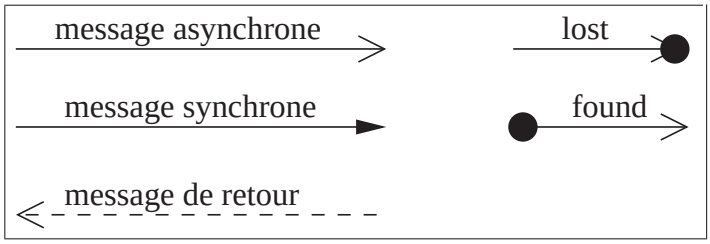

Fig. 18.2 – Les types de messages et leur représentation

Règles de cohérence

• Règle 484 : La signature d'un message doit suivre la syntaxe : [[7] p.430]

```
messageident::=[attribute=]signal-or-operation-name
                            [(arguments)][:return-value]|*
arguments ::= argument [ , arguments]
argument ::= [parameter-name=]argument-value |
           attribute= out-parameter-name [:argument-value]| -
```

où le caractère "-" représente une valeur d'un argument inconnue.

• Règle 485 : Si les événements d'émission et de réception d'un même message sont sur la même ligne de vie, l'émission du message doit se trouver avant (au-dessus pour les diagrammes de séquence) la réception de ce message. [[7] p.429]

• Règle 486 : La signature d'un message doit soit référencer une opération (auquel cas la sorte du message est soit `synchCall` ou `asynchCall`) soit un signal (auquel cas la forme du message est de type `synchSignal` ou `asynchSignal`).[[7] p.429]

• Règle 487 : La signature d'un message doit être la même que celle de l'élément référencé (opération ou signal). [[7] p.429]

• Règle 488 : Dans le cas où la signature du message est une opération, les arguments du message doivent correspondre aux paramètres de l'opération. Un argument correspond à un paramètre si l'argument est de la même classe ou une spécialisation de la classe de ce paramètre. [[7] p.429]

Remarque Pour vérifier la correspondance entre les arguments du message et les paramètres de l'opération, il faut prendre en compte le nombre et l'ordre des arguments.

• Règle 489 : Dans le cas où la signature du message est un signal, les arguments du message doivent correspondre aux attributs du signal. L'argument d'un message correspond à l'attribut d'un signal si l'argument est de la même classe que l'attribut ou une spécialisation de celle-ci. [[7] p.429]

• Règle 490 : Si le message référence une opération, cette opération doit appartenir à la classe référencée par la ligne de vie réceptrice et doit être visible par la classe source du message.[Nouvelle règle][1]

• Règle 491 : Dans le cas d'un appel d'opération, la classe émettrice doit être reliée par une association navigable vers la classe réceptrice ou avoir un attribut dont le type est la classe réceptrice. Cette association navigable ou cet attribut peuvent être des membres hérités d'une relation de généralisation par la classe émettrice ou réceptrice. [Nouvelle règle]

• Règle 492 : Si le message référence un signal, ce signal doit être visible par les classes émettrice et réceptrice du message. [Nouvelle règle][2]

• Règle 493 : Les événements d'envoi et de réception d'un message doivent être distincts (les relations du méta-modèle `SendEvent` et `ReceiveEvent` sont mutuellement exclusives).[[7] p.429]

• Règle 494 : Les arguments d'un message peuvent uniquement être :

– des attributs de la ligne de vie émettrice ;
– des constantes ;
– des valeurs symboliques ;
– des paramètres explicites de l'interaction englobante ;
– des attributs de la classe qui contient l'interaction. [[7] p.429]

• Règle 495 : Un message ne peut pas traverser les frontières des fragments combinés ou de leurs opérandes. [[7] p.429]

Remarque Pour cela un message doit passer par une porte, ce qui revient de façon conceptuelle à décrire deux messages.

[1] inter-diag interactions - classes
[2] Inter-diagramme interactions - classes

• Règle 496 : Si les deux fins d'un message sont des occurrences d'événement alors le connecteur doit relier les deux lignes de vie des deux fins de message. [[7] p.429]

• Règle 497 : Dans le cas d'un appel synchrone d'opération, l'exécution de l'objet source est suspendue pendant toute la durée de l'exécution de l'objet cible. L'objet source ne peut donc ni envoyer de message ni traiter de message reçu. [Nouvelle règle]

• Règle 498 : Un message d'appel asynchrone d'opération doit référencer une opération active. [Nouvelle règle][3]

• Règle 499 : Dans le cas d'un signal asynchrone, les deux objets communicants doivent appartenir à des threads de contrôle différents.[Nouvelle règle]

Remarque Les objets actifs sont des instances de classes actives. Les cas possibles concernant la règle 85 sont les suivants :

- Les deux objets sont deux objets actifs.
- Un des objets est actif, l'autre doit être un objet qui appartient à un objet actif différent du premier.
- Les 2 objets appartiennent à des objets actifs différents.

[4]

• Règle 500 : Dans un diagramme de communication, le sens de tout message doit être spécifié de façon non ambiguë par une petite flèche. [Nouvelle règle][Règle utilisateur]

18.2.3 Messages et stimuli dans les diagrammes de communication

Dans un diagramme de communication (anciennement diagramme de collaboration dans la version 1.5 de la spécification UML), les messages et stimuli sont numérotés afin de pouvoir en reconstituer l'ordre. On peut identifier trois familles de règles de cohérence :

- une portant sur l'ensemble du label ;
- une portant sur le champ `predecessor` du label ;
- et une portant sur le champ `sequence-expression`.

18.2.3.1 Label

Contexte Le label spécifie le message en cours d'émission.

Règles de cohérence

• Règle 501 : Tout label de message ou de stimulus doit suivre la syntaxe suivante : [[1] p.183] et [tirée de [8] p.3-131]

```
predecessor sequence-expression return-value
                              := message-name argument-list
```

[3] Inter-diag interactions - objets

[4] inter-diag interactions- objets

où les champs `predecessor`, `return-value`, `argument-list` sont optionnels.

• Règle 502 : Si le champ `return-value` est présent, l'élement référencé par le champ `message-name` doit retourner une valeur. [Nouvelle règle]

• Règle 503 : Si le champ `argument-list` est présent, celui-ci doit faire figurer tous les arguments relatifs au message. Une liste vide entre parenthèses suppose que le message ne contient aucun argument. [Nouvelle règle]

Guides

• Guide 17 : Il est vivement conseillé de faire figurer la liste des arguments.

• Guide 18 : Si la message retourne une valeur, il est conseillé de faire figurer le champ `return-value`.

Justification

• Justification 17 : Nous conseillons d'exposer la liste des arguments car ceci permet de contrôler la validité de la signature du message en créant de la redondance (cf. règle 487).

• Justification 18 : Ceci permet de vérifier que ce message doit bien retourner une valeur.

18.2.3.2 Predecessor

Contexte Le champ `predecessor` du label est une liste de numéros de séquence séparée par des virgules suivie d'un slash ('/').

Règles de cohérence

• Règle 504 : Le champ `predecessor` suit la syntaxe suivante : [tirée de [8] p.3-131]

```
message-sequence-number-1 , ... , message-sequence-number-N /
```

Ceci exprime que le message ne peut pas être activé avant que tous les messages référencés par les `message-sequence-number-i` ne se soient produits.

• Règle 505 : Aucune cycle ne doit apparaître.[Nouvelle règle]

Remarque Un cycle apparaît si le message u doit précéder le message v et que u≥v. Un cycle peut mettre en jeu plus de deux messages. C'est le cas par exemple si le message 1.a.3 précède 1.b.2 et que 1.b.2 précède 1.a.2. Vu que 1.a.2 précède forcément 1.a.3, on arrive à une situation d'incohérence de précédence.

• Règle 506 : Pour tout i, `message-sequence-number-i` doit correspondre à un et exactement un message connu, dont le champ `sequence-expression` verifie `sequence-expression=message-sequence-number-i`.[Nouvelle règle]

Remarque Il est inutile de faire figurer dans le champ `predecessor` les messages correspondants à une précédence qui peut être déduite du champ `sequence-expression`. Par exemple, les messages 1.2 et 1.3.5 sont forcément des messages dont l'occurrence arrive avant l'occurrence du message 1.3.6, par contre le message 1.a.1.1 ne précède pas forcément 1.b.1.2 (cf. 18.2.3.3).

18.2.3.3 Sequence-expression

Contexte Le champ `sequence-expression` est une liste de `sequence-term` séparés par des points.

Règles de cohérence

• Règle 507 : Le champ `sequence-expression` doit suivre la syntaxe suivante : [tirée de [8] p.3-131]

`sequence-term-1.sequence-term-2. ... .sequence-term-N :`

Chaque `sequence-term` doit suivre la syntaxe : `[Entier|Nom][Récurrence]`.

Où :

- `Entier` représente l'ordre séquentiel des messages (ex : le message 3.1.3 suit le message 3.1.2, le message 3.2.1 étant le premier message envoyé lors de l'activation entraînée par le message 3.2) ;
- `Nom` représente un thread de contrôle concurrent ; par exemple 1.a et 1.b sont deux messages différents qui induisent des activations concurrentes, aucune relation de précédence ne peut donc être posée sur les messages 1.a.1.3 et 1.b.1 exceptée en précisant explicitement cette relation de précédence au moyen du champ `predecessor` (cf. 18.2.3.2) ;
- `Récurrence` représente une exécution conditionnelle ou itérative ; une itération est représentée par `*[iteration-clause]` et une condition est représentée par `[condition-clause]`.

• Règle 508 : Deux objets qui reçoivent des messages concurrents (1.a et 1.b par exemple) doivent être des objets actifs. [Nouvelle règle]

Remarque En effet ils doivent faire partie de threads différents.

• Règle 509 : Tout champ `sequence-expression` doit être unique au sein d'un même diagramme. [Nouvelle règle]

Remarque préliminaire Deux messages exécutés en séquence seront différenciés par une incrémentation de 1 de leur `sequence-term` final (ex : 2.1.3 et 2.1.4). Cette numérotation permet de déduire tous les messages qui sont séquentiellement antérieurs au message. Pour le message 2.1.4, c'est l'ensemble des messages `{2.1.3, 2.1.2, 2.1.1, 2.1, 2, 1}`.

• Règle 510 : Pour tout message, la collaboration doit contenir l'ensemble des messages implicitement antérieurs au message ; dans le cas contraire la collaboration est erronée. [Nouvelle règle]

• Règle 511 : Aucun message ne doit contenir dans son champ `predecessor` le champ `sequence-expression` d'un message qui lui est implicitement postérieur. [Nouvelle règle]

Remarque Par exemple le message `1.3.4/1.2 : message-illicite()` est invalide par nature. Ceci revient à créer un cycle entre messages et donc une situation d'incohérence de précédence.

Chapitre 19

Diagrammes

19.1 Diagramme de séquence

Contexte général Les règles de cohérence se rapportant aux diagrammes de séquence peuvent être classées en plusieurs familles : l'une concerne les messages d'appel synchrone d'une opération, une autre les messages asynchrones, une autre les traces d'une interaction, et enfin les éléments contenus dans un diagramme de séquence.

19.1.1 Messages d'appel synchrone d'une opération

Contexte Dans cette partie, nous présentons les règles de cohérence qui mettent en jeu plusieurs messages, plusieurs lignes de vie et au moins un appel d'opération synchrone.

Un appel d'opération synchrone se fait via l'envoi d'un message (cf. section 18.2) synchrone étiqueté par le nom d'une opération.

Règles de cohérence

• Règle 512 : Dans le cas d'un appel synchrone d'une opération, un message déclenche une occurrence d'exécution sur l'objet cible. [Nouvelle règle]

• Règle 513 : La fin d'une occurrence d'exécution doit engendrer un message de retour. [Nouvelle règle]

• Règle 514 : Le message de retour à la fin d'une occurrence d'exécution doit avoir pour source la ligne de vie qui supporte l'occurrence d'exécution et pour cible la ligne de vie qui est responsable de l'envoi du message déclencheur de l'occurrence d'exécution. [Nouvelle règle]

• Règle 515 : Un message de retour est toujours le résultat de la fin d'une occurrence d'exécution. [Nouvelle règle]

• Règle 516 : Un appel synchrone d'opération se fait entre objets qui appartiennent au même thread d'exécution et qui ne doivent donc pas être des objets actifs. [Nouvelle règle]

19.1.2 Messages asynchrones

Contexte Dans cette partie sont enumérées les règles en lien avec les messages asynchrones.

Règles de cohérence

• Règle 517 : Aucun message de retour ne doit être la conséquence d'un message asynchrone. [Nouvelle règle]

• Règle 518 : Un message d'envoi de signal asynchrone ne doit déclencher aucune occurrence d'exécution. [Nouvelle règle]

19.1.3 Traces d'une interaction

Contexte Les règles qui suivent portent sur les restrictions des constructions lorsque des messages sont entrelacés dans un diagramme de séquence.

Remarque Une trace d'exécution est une séquence d'occurrences d'événement.

Règles de cohérence

• Règle 519 : Aucun circuit dont les arcs sont formés par des messages et par la ligne de vie en descendant ne doit apparaître. [Nouvelle règle]

Exemple La figure 19.1 montre un circuit formé par un entrelacement de messages. Cette construction est inacceptable. En effet, la position verticale des occurrences d'événement induit des précédences, on a donc `msgA.receiveEvent->msgB.sendEvent` et `msgB.receiveEvent -> msgA.sendEvent`. D'autre part, l'occurrence d'événement d'envoi d'un message précède toujours l'occurrence d'événement de réception de ce message, d'où `msgA.sendEvent -> msgA.receiveEvent` et `msgB.sendEvent -> msgB.receiveEvent`. On a donc `msgA.receiveEvent -> msgB.sendEvent -> msgB.receiveEvent -> msgA.sendEvent`. Ce qui rentre en contradiction avec `msgA.sendEvent -> msgA.receiveEvent`.

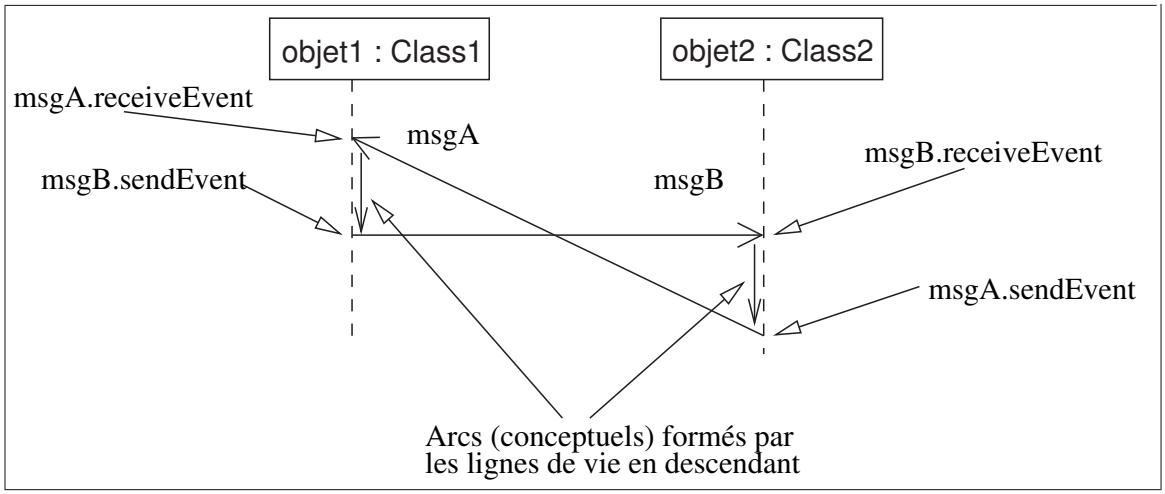

Fig. 19.1 – Circuit formé par des messages et les lignes de vie

19.1.4 Éléments contenus

Contexte Dans cette partie nous énumérons tous les éléments qui peuvent être contenus dans un diagramme de séquence.

Règles de cohérence

• Règle 520 : Les nœuds contenus dans un diagramme de séquence doivent être :
- un cadre (*frame* en anglais) ;
- des lignes de vie ;
- des occurrences d'exécution ;
- des occurrences d'interaction ;
- des fragments combinés ;
- des invariants d'état ;
- des corégions ;
- des signes stop qui signalent la destruction d'instances ;
- des contraintes de durée et des actions d'observation de durée ;
- des contraintes de temps et des actions d'observation temporelle. [[7] p.436]

• Règle 521 : Les chemins graphiques contenus dans des diagrammes de séquence doivent être :
- des messages (de type asynchrone, synchrone, trouvés, perdus) ;
- des relations d'ordre général. [[7] p.438]

Guides

• Guide 19 : Nous conseillons de montrer dans les classes, les diagrammes de séquences qui lui sont associés et les différentes parties qui la composent.

Justification

• Justification 19 : Le guide 19 permet d'identifier clairement quel est le classificateur qui contient le diagramme de séquence et donc de vérifier l'ensemble des règles qui se réfèrent au contexte de ce diagramme.

19.2 Diagramme de communication

Contexte Les diagrammes de communication se focalisent sur les interactions entre les lignes de vie, où ce qui est primordial est l'architecture de la structure interne et comment elle correspond avec le passage de message. La séquence des messages est donnée par un numéro de séquence associé aux messages.

Les règles suivantes portent sur les éléments qui peuvent être contenus dans un diagramme de communication.

Règles de cohérence

• Règle 522 : Les nœuds contenus dans un diagramme de communication peuvent être :
- des cadres comme dans les diagrammes de séquence ;
- des lignes de vies. [[7] p.445]

• Règle 523 : Les seuls chemins graphiques pouvant être contenus dans un diagramme de communication sont les messages. [[7] p.445]

19.3 Interaction Overview Diagram

Contexte Le diagramme de vue d'ensemble d'interaction définit des interactions au travers d'une variante des diagrammes d'activités afin de donner une vue d'ensemble du flot de contrôle. On retrouve dans ce type de diagramme les différentes constructions présentes dans les diagrammes d'activités en remplaçant les nœuds par des interactions ou des occurrences d'interaction.

Règles de cohérence

• Règle 524 : Les nœuds présents dans un diagramme de vue d'ensemble d'interaction doivent être :
- des cadres ;
- des diagrammes de séquence et de communication contenus dans des cadres et par conséquement tous les nœuds et chemins graphiques qu'ils peuvent contenir ;
- des occurrences d'interaction ;
- des nœuds finaux d'activité ;
- des nœuds de décision ;
- des nœuds initiaux ;
- des nœuds de fusion ;
- des nœuds d'union ;
- des nœuds de bifurcation. [[7] p.447]

• Règle 525 : Les différents chemins graphiques présents dans un diagramme de vue d'ensemble d'interaction doivent être :
- des chemins de contrôle de flots ;
- des chemins présents dans les diagrammes de séquence et de communication. [[7] p.447]

19.4 Timing Diagram

Contexte Le diagramme de timing sert à montrer des interactions avec comme souci principal celui de raisonner sur les aspects temporels. Un diagramme de timing se focalise sur les changements de conditions à l'intérieur et parmi des lignes de vie, selon un axe temporel linéaire.

Règles de cohérence

• Règle 526 : Les nœuds présents dans un diagramme de timing doivent être :
- des cadres ;
- des labels de message ;
- des états ou conditions de ligne de temps (*state or condition timeline*) ;
- des lignes de vie ;
- des signes stop ; [[7] p.450]

• Règle 527 : Les différents chemins graphiques présents dans un diagramme de timing doivent être :
- des messages ;
- des relation d'ordre général. [[7] p.450]

Septième partie

Diagramme de machines à états

Chapitre 20

Éléments

20.1 State

Contexte Les automates à états finis permettent de modéliser le comportement d'un objet individuel. Les diagrammes de machines à état reposent sur ce formalisme.

Un état (*state* en anglais) est une condition ou une situation qui survient au cours de la vie d'un objet pendant laquelle cet objet satisfait à certaines conditions, exécute une activité ou attend un événement.

Il existe trois grandes sortes d'états, les états simples (cf section 20.1.4), les états composites (cf section 20.1.5), et les états sous-machines (cf section 20.1.6).

Dans les diagrammes d'états-transitions, les états sont décomposés en trois compartiments :

- un compartiment optionnel contenant le nom ;
- un compartiment optionnel contenant les activités internes (cf. section 20.1.2) ;
- un compartiment optionnel contenant les transitions internes à l'objet (cf. section 20.1.3) .

Les états ne sont pas forcément nommés et s'ils le sont ne doivent pas obligatoirement avoir des noms différents même s'ils appartiennent à la même région.

Les règles de cohérence qui ont trait aux états sont réparties en six parties. Les règles générales sont présentées dans la section 20.1.1, les règles concernant les activités internes sont présentées en 20.1.2, les règles sur les transition internes en 20.1.3, les règles relatives aux états simples en 20.1.4, les règles traitant les états composites en 20.1.5, et enfin les règles sur les états sous-machines se trouvent dans la section 20.1.6.

20.1.1 Règles générales

Règles de cohérence

• Règle 528 : Un état ne peut pas à la fois être un état sous-machine et un état composite. [[7] p.479]

• Règle 529 : Un état ne peut être redéfini que si la valeur de son méta-attribut `isFinal` est `false`. [[7] p.481] [1]

• Règle 530 : Le contexte de redéfinition d'un état est sa machine à état contenante. [[7] p.480] [Règle sur le méta-modèle]

Guides

• Guide 20 : Nous conseillons de nommer les états de façon distincte.

Justifications

• Justification 20 : Ceci permet de référencer les états par leur nom dans les diagrammes de séquence lorsque l'on spécifie des invariants d'états d'une ligne de vie (cf. règle 480).

20.1.2 Activités internes

Contexte Le compartiment contenant les activités liste le comportement interne à un état en réponse à des événements. Ces événements sont soit les événements prédéfinis `entry`, `exit` ou `do` soit des événements définis par l'utilisateur.

Règles de cohérence

• Règle 531 : La spécification d'une activité interne doit suivre la syntaxe suivante : [tirée de [1] p.149]

```
event-name argument-list / action-expression
```

• Règle 532 : Le champ `event-name` doit correspondre :
– soit à un événement prédéfini du langage (`entry`, `exit`, `do`) ;
– soit à événement défini par l'utilisateur comme opération ou réception (cf. section 2.23) du classificateur contexte de la machine à état. [Nouvelle règle]

• Règle 533 : Quand le champ `event-name` correspond à un événement prédéfini du langage, il n'est pas possible de définir des arguments pour l'événement. [tirée de [1] p.150]

• Règle 534 : Dans le cas où `event-name` correspond à un signal, le champ `argument-list` doit correspondre (en tenant compte de l'ordre, du type et du nombre) aux attributs du signal. [Nouvelle règle]

• Règle 535 : Le champ `action-expression` doit correspondre soit :
– à une opération de l'entité décrite par la machine à états ;
– à une manipulation des attributs de l'entité décrite par la machine à états ou d'attributs d'autres entités qui lui sont accessibles via des fins d'association navigables ;
– soit à une expression opaque. [Nouvelle règle]

[1] On pense qu'il y a une erreur dans la norme et que c'est isLeaf

Guides

• Guide 21 : Nous conseillons de ne pas utiliser les expressions opaques lorsque l'on décrit le champ `action-expression`.

• Justification 21 : Le guide 21 permet de s'assurer que l'action est bien décrite pour l'entité modélisée par la machine à états par la vérification de la règle 535.

20.1.3 Transitions internes

Contexte Le troisième compartiment contient les transitions internes. Celles-ci sont activées par les événements auxquels est sensible l'état et des actions sont exécutées en conséquence.

Une transition interne a la même sémantique que les autres transitions excepté qu'aucun état n'est quitté ni activé. C'est pourquoi de nombreuses règles présentes dans cette section renvoient aux règles sur les transitions présentées en section 21.1.

Règles de cohérence

• Règle lien 46 : La spécification d'une transition interne suit le même syntaxe que les transitions, cette syntaxe est présentée par la règle 591.

• Règle 536 : Le champ `event-name` peut apparaître plusieurs fois seulement si le champ `[guard-condition]` qui le suit est différent. [Nouvelle règle]
[2]

• Règle lien 47 : Le champ `event-signature` doit respecter la règle 593.

• Règle lien 48 : Le champ `[guard-condition]` est une contrainte et doit donc respecter les règles de la section 2.7.

• Règle lien 49 : Le champ `action-expression` doit respecter la règle 535.

• Règle lien 50 : Le champ `destination-expression` doit respecter la règle 594.

• Règle lien 51 : Le champ `destination-event-name` doit respecter la règle 595.

20.1.4 Simple State

Contexte Un état simple (*simple state* en anglais) est un état qui ne contient pas de région et qui n'est pas un état sous-machine. Un état simple peut contenir des activités et des transitions internes (les états composites également).

Règles de cohérence

• Règle 537 : Un état simple ne doit pas contenir de région ni d'état sous-machine.[[7] p.479]

20.1.5 Composite State

Contexte UML offre la possibilité de décrire des états au fonctionnement complexe par le mécanisme d'états composites (*composite states* en anglais).

[2] Je pense qu'il faut enlever cette règle, comment peut ne pas la satisfaire ? ? : RÈGLE : Le champ `action-expression` ne doit pas mettre en jeu des expressions récursives. [tirée de [1] p.153]

Un état composite contient, en plus des trois compartiments d'un état simple, un compartiment de décomposition.

Un état composite a la possibilité d'être orthogonal, c'est-à-dire qu'il peut contenir plusieurs régions orthogonales.

Règles de cohérence

• Règle 538 : Un état composite contient au moins une région. [[7] p.479] [Règle sur le méta-modèle]

• Règle 539 : Un état composite orthogonal doit contenir au moins deux régions. [[7] p.479] [Règle sur le méta-modèle]

• Règle 540 : Un état orthogonal doit être un état composite avec au moins deux régions. [[7] p.480] [3]

• Règle 541 : Lorsqu'un état composite est actif et orthogonal, toutes ses régions sont actives, à raison d'un sous-état par région. Il ne doit donc pas exister de chemins qui sort d'une région et en active une autre alors qu'une région de l'état composite est active. [Nouvelle règle][4]

Remarque La figure 20.1 montre un exemple où la règle 541 n'est pas respectée.

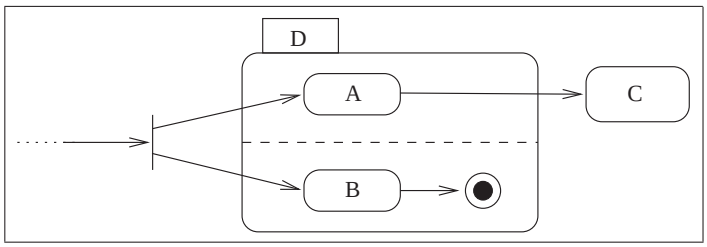

Fig. 20.1 – Exemple de non respect de la règle 541

20.1.6 État sous-machine

Contexte Un état sous-machine spécifie l'insertion de la spécification d'une sous-machine à état. Un état sous-machine est un mécanisme de décomposition qui permet de factoriser des comportements communs et de les réutiliser. Un état sous-machine est sémantiquement équivalent à un état composite.

Règles de cohérence

• Règle 542 : La chaîne de caractères qui décrit une état sous-machine doit suivre la syntaxe suivante : [[7] p.486]

```
<state name>  :  <name of referenced state machine>
```

[3] Cette regle pose pb : redite de la règle précédente et formalisation en OCL qui ne correspond pas au texte

[4] Les deux règles qui précédent ne sont pas vérifiables au niveau modèle

• Règle 543 : Le champ `<name of referenced state machine>` doit correspondre à exactement une machine à état. [Nouvelle règle]

• Règle 544 : Dans le cas où un état sous-machine est contenu dans la machine à état qu'il référence, la machine à état référencée doit contenir un chemin tel que ce chemin ne fait pas appel à l'état sous-machine en question. [Nouvelle règle]

• Règle 545 : Seul un état sous-machine peut posséder des références de points de connection (*connection point reference* en anglais). [[7] p.479]

• Règle 546 : Les références aux points de connection utilisés comme des cibles (resp. sources) de transitions associées avec un état sous-machine doivent être définies comme des points d'entrée (resp. de sortie) dans la machine à état référencée. [[7] p.479]

• Règle 547 : Seuls les états sous-machine ont une référence à une machine à état. [[7] p.480] [Règle sur le méta-modèle]

• Règle 548 : Un état sous-machine ne peut pas posséder d'activité interne. [Nouvelle règle]

• Règle 549 : Un état sous-machine ne peut pas posséder de transition interne. [Nouvelle règle]

20.2 Region

Contexte Une région est une partie orthogonale (i.e. dont l'exécution est autonome) d'un état composé ou d'une machine à états. Elle contient des états et des transitions.

Règles de cohérence

• Règle 550 : Une région contient au maximum un pseudo-état initial. [[7] p.476]

• Règle 551 : Une région doit avoir au plus un sommet historique profond (*deep history vertex* en anglais). [[7] p.476]

• Règle 552 : Une région doit avoir au plus un sommet historique superficiel (*shallow history vertex* en anglais). [[7] p.476]

• Règle 553 : Le contexte de redéfinition d'une région est la première machine à état englobante. [[7] p.476] [Règle sur le méta-modèle]

• Règle 554 : Une région contient au minimum un sous-état qui n'est ni un pseudo-état, ni un état final. [Nouvelle règle]

• Règle 555 : Toute région doit soit contenir un état initial, soit un point d'entrée, soit un état ciblé par une transition qui n'a pas sa source dans dans l'état englobant. [Nouvelle règle]

Remarque Dans la règle précédente les "soit" ne sont pas exclusifs.

• Règle 556 : Toute région contenue dans un état composite qui lui-même est la cible d'une transition doit contenir un état initial. [Nouvelle règle]

• Règle 557 : Tout état contenu dans une région doit être atteignable.[Nouvelle règle]

Remarque Un état A est atteignable si il existe une suite d'événements qui à partir d'un point d'entrée, de l'état initial ou d'un état ciblé par une transition qui n'a pas sa source dans l'état englobant implique l'activation de l'état A.

20.3 Pseudo-state

Contexte Un pseudo-état (*Pseudo-state* en anglais) est une abstraction qui regroupe différents types de sommets transitoires.

Il existe 10 pseudo-états :

- pseudo-état initial (pareil en anglais) ;
- pseudo-état historique profond (*deep history* en anglais) ;
- pseudo-état historique superficiel (*shallow history* en anglais) ;
- pseudo-état d'union (*join* en anglais) ;
- pseudo-état de bifurcation (*fork* en anglais) ;
- pseudo-état de jonction (*junction* en anglais) ;
- pseudo-état de choix (*choice* en anglais) ;
- pseudo-état point d'entrée (*entry point* en anglais) ;
- pseudo-état point de sortie (*exit point* en anglais) ;
- pseudo-état de terminaison (*terminate* en anglais) ;

Remarque La figure 20.2 donne la représentation graphique des pseudo-états.

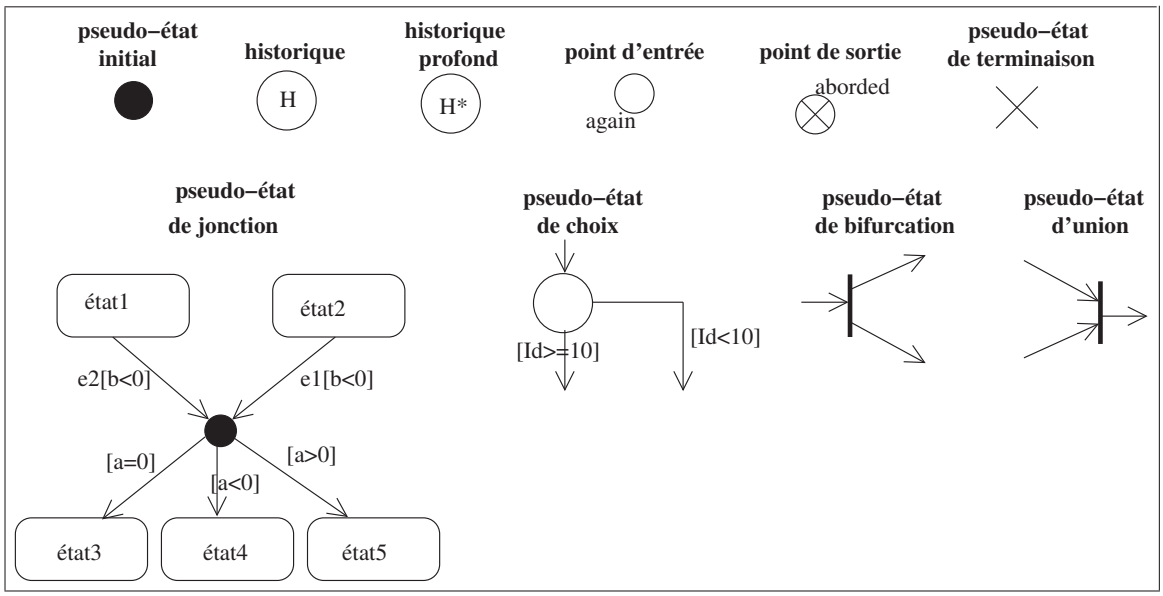

FIG. 20.2 – Les pseudo-états et leurs représentations

Règles de cohérence

• RÈGLE 558 : Un pseudo-état doit être forcément l'un des 10 pseudo-états définis par [7] : `initial, deepHistory, shallowHistory, join, fork, junction, choice, entryPoint, exitPoint, terminate`. [Règle dérivée du méta-modèle]

• RÈGLE 559 : Un pseudo-état initial (*initial* en anglais) ne peut avoir qu'une seule transition de sortie. [[7] p.469]

• RÈGLE 560 : Un pseudo-état historique superficiel ou profond (resp. *shallowHistory* ou *deepHistory* en anglais) ne peut avoir qu'une seule transition de sortie. [[7] p.470]

• RÈGLE 561 : Dans une machine à états complète, un sommet d'union (*join* en anglais) doit posséder au moins deux transitions entrantes et exactement une transition

sortante. [[7] p.470]

Remarque Dans la règle 561, nous le mot « complète » vient de la traduction du mot « complete » qui se trouve dans l'énoncé de la règle de la norme. Cette précision pourrait s'appliquer à toutes les règles car on ne fait de l'analyse statique que sur des modèles stables. [5]

• Règle 562 : Toutes les transitions entrantes d'un sommet d'union doivent provenir de différentes régions d'un état orthogonal. [[7] p.470]

• Règle 563 : Lorsqu'une région d'un état othogonal contient un état qui est la source d'une transition vers un sommet d'union, alors toutes les région doivent contenir un état qui est la source d'une transition vers ce sommet d'union. [Nouvelle règle]

• Règle 564 : Dans une machine à états complète, un sommet de bifurcation (*fork* en anglais) doit posséder au moins 2 transitions sortantes et exactement une transition entrante. [[7] p.470]

Remarque Les règles 561 et 564 ne sont pas toujours respectées graphiquement. En effet, comme le montre la sous-figure i) de la figure 20.3, il est graphiquement possible de représenter un sommet d'union et de bifurcation de manière fusionnée. Cette représentation est totalement équivalente à la représentation de la sous-figure ii).

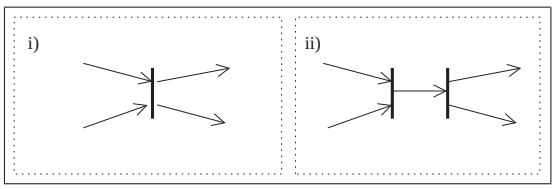

Fig. 20.3 – Représentation des sommets de bifurcation et d'union

• Règle 565 : Toutes les transitions sortantes d'un sommet de bifurcation doivent se diriger vers des états de différentes régions d'un état orthogonal. [[7] p.470]

• Règle 566 : Les transitions sortantes d'un sommet d'union doivent cibler toutes les régions orthogonales d'un état composite. [Nouvelle règle]

• Règle 567 : Dans une machine à états complète, un sommet de jonction (*junction* en anglais) doit posséder au moins une transition entrante et au moins une transition sortante. [[7] p.470]

• Règle 568 : Dans une machine à états complète, un sommet de choix (*choice* en anglais) doit posséder au moins une transition entrante et au moins une transition sortante. [[7] p.470]

• Règle 569 : Les pseudo-états points d'entrée ne peuvent être définis que dans les régions de plus haut niveau de la machine à états. [[7] p.470]

• Règle 570 : Les pseudo-états points de sortie ne peuvent être définis que dans les régions de plus haut niveau de la machine à états. [[7] p.470]

• Règle 571 : Il doit toujours exister une transition sortante provenant d'un pseudo-état de choix qui est éligible. Il faut donc que les conditions de ces transitions couvrent l'ensemble des valeurs possibles du ou des paramètres du test. [Nouvelle règle]

[5] peut être reprendre cette remarque

Remarque Il est possible d'utiliser la condition de branche `else` pour parvenir à ce résultat. [6]
[7]

• Règle 572 : Les conditions de garde des transitions sortantes d'un pseudo-état de choix doivent être mutuellement exclusives afin qu'une seule transition ne puisse être éligible en même temps. [Nouvelle règle]

• Règle 573 : Les conditions de branche des transitions sortantes d'un pseudo-état de jonction doivent être mutuellement exclusives afin qu'une seule transition ne puisse être éligible en même temps. [Nouvelle règle]

Remarque Les règles 572 et 573 sont valides uniquement lorsque l'on veut interdire les comportements non-déterministes.

• Règle 574 : Un pseudo-état initial ne doit pas être la cible d'une transition. [Nouvelle règle]

20.4 Final State

Contexte Un état final (*final state* en anglais) est une sorte d'état qui signifie que la région qui le contient est accomplie, c'est-à-dire que la condition de conclusion pour cet état est satisfaite.

Règles de cohérence

• Règle 575 : Un état final n'a pas de transition sortante. [[7] p.462] [Règle sur le méta-modèle]

• Règle 576 : Un état final ne peut pas avoir de région. [[7] p.462] [Règle sur le méta-modèle]

• Règle 577 : Un état final ne peut pas référencer d'état sous-machine. [[7] p.462] [Règle sur le méta-modèle]

• Règle 578 : Un état final n'a pas d'activité d'entrée. [[7] p.462] [Règle sur le méta-modèle]

• Règle 579 : Un état final n'a pas d'activité de sortie. [[7] p.462] [Règle sur le méta-modèle]

• Règle 580 : Un état final n'a pas de `doActivity`. [[7] p.462] [Règle sur le méta-modèle]

• Règle lien 52 : Un état final est un état et doit donc respecter les règles énoncées en 20.1 (avec les restrictions importantes exprimées ci-dessus).

20.5 Connection Point Reference

Contexte Une référence de point de connexion (*connexion point reference* en anglais) représente l'utilisation d'un point d'entrée ou de sortie définie dans une machine à état référencée par un état sous-machine.

[6] Guide de prévention : préférer couvrir l'ensemble des choix possibles sans le 'else' et mettre le else qd même qui lance une exception

[7] guide de prévention : tout état historique doit être le source d'un transition : transition qui est tirée par défaut (cf. 481 de la spec)

Règles de cohérence

• Règle 581 : Le rôle de la méta-association `entry` doit être un pseudo-état point d'entrée. [[7] p.460] [Règle sur le méta-modèle]

• Règle 582 : Le rôle de la méta-association `exit` doit être un pseudo-état point de sortie. [[7] p.460] [Règle sur le méta-modèle]

• Règle 583 : Une référence à un point de connexion « d'entrée » (i.e. qui correspond à un point d'entrée) ne doit pas être la source de transition. [Nouvelle règle]

• Règle 584 : Une référence à un point de connexion « de sortie » (i.e. qui correspond à un point de sortie) ne doit pas être la cible de transition. [Nouvelle règle]

Chapitre 21

Relations

21.1 Transition

Contexte UML permet de représenter des machines à états afin de décrire le comportement d'une partie d'un système. La description du changement d'un état à un autre se fait par l'intermédiaire de transitions.

Remarque Nous ne traitons ici que les transitions simples. Le cas des transitions composées est traité en section 22.1.

Remarque Les transitions internes sont abordées en section 20.1.3.

21.1.1 Éléments connectés

Contexte Une transition est toujours un lien entre deux sommets. Ces sommets sont soit un état, soit un pseudo-état, soit une référence à un point de connexion.

Remarque La liste des pseudo-états se trouve en section 20.3.

Il existe trois sortes de transitions, les transitions internes qui se produisent sans sortir ni entrer dans un état, les transitions locales qui ne sortent pas de l'état composite source et les transition externes qui sortent de l'état composite source.

Règles de cohérence

• Règle 585 : Toute transition est connectée à exactement deux sommets, un sommet source et un sommet cible. [Règle dérivée du méta-modèle][1]

• Règle 586 : Les transitions de sortie d'un sommet de bifurcation (`fork`) doivent avoir pour cible un état (et non un pseudo-état). [[7] p.499]

• Règle 587 : Les transitions entrantes d'un sommet d'union doivent avoir pour source un état (et non un pseudo-état). [[7] p.499]

• Règle 588 : Dans le cas d'un état orthogonal, une transition simple ne peut pas connecter deux états qui appartiennent à des régions orthogonales différentes de l'état

[1] Cette règle est "câblée" dans le méta-modèle, cependant elle ne prend pas en compte les transitions internes => problème du méta-modèle

composite. [Nouvelle règle]

• Règle 589 : L'état source de toute transition de type locale doit être un état composite. [[7] p.506]

• Règle 590 : L'état source de toute transition de type externe doit être un état composite. [[7] p.506]

21.1.2 Label des transitions

Contexte Le label des transitions est une chaîne de caractères qui peut être utilisée pour décrire l'événement d'activation de la transition, la condition de garde et l'action qui résulte de l'activation de la transition. [2]

Une transition d'achèvement (*completion transition* en anglais) est une transition dont la source est un état composite, un état sous-machine ou un point de sortie et qui ne contient pas d'événement explicite. Une transition d'achèvement est tirée à la réception d'un événement d'achèvement levé lorsque toutes les activités de la source de la transition sont terminées par exemple lorsque tous les états finaux de régions orthogonales sont atteints.

Règles de cohérence

• Règle 591 : Tout label de transition doit suivre la syntaxe : [[1] p.150]

```
event-signature '['guard-condition']' '/' action-expression
          '^'send-clause
```

où aucun champ n'est obligatoire, où le champ `'/'action-expression` peut apparaître plusieurs fois et où :

– le champ `event-signature` doit suivre la syntaxe :
```
event-name(comma-separated-parameter-list)
```
– le champ `send-clause` doit suivre la syntaxe :
```
destination-expression '.' destination-event-name
                                '(' argument ',' ... ')'
```

• Règle 592 : Le champ `action-expression` doit suivre la règle 535. [Nouvelle règle]

• Règle 593 : Le champ `event-signature` doit correspondre soit :

– à une opération de l'entité modélisée par le diagramme d'état ;

– à un signal accepté par l'entité modélisée ;

– à un événement temporel (exprimé à l'aide du mot clé `after()`). [Nouvelle règle]

• Règle lien 53 : Le champ `[guard-condition]` est une contrainte et doit donc respecter les règles énoncées en 2.7.

• Règle 594 : Le champ `destination-expression` doit correspondre à une machine à état connue de la machine à état. [Nouvelle règle]

• Règle 595 : Le champ `destination-event-name` doit pouvoir être accepté par la machine à état référencée par le champ `destination-expression`. La machine à état réceptrice du message doit donc posséder au moins une transition qui est déclenchée par le signal reçu. [Nouvelle règle]

[2] guide de prévention : toujours mettre un label. Les événements de complétion seront spécifiés par un événement explicite

Remarque Cette transition peut appartenir à un sous-état de la machine à état.

• Règle 596 : Les transitions sortantes d'un sommet de bifurcation (fork) ne doivent ni contenir le champ `event-signature` ni le champ `[guard-condition]`. [[7] p.498]

• Règle 597 : Les transitions sortantes du sommet d'union ne doivent ni contenir le champ `event-signature` ni le champ `[guard-condition]`. [[7] p.499]

• Règle 598 : Les transitions sortantes d'un pseudo-état ne doivent pas contenir le champ `event-signature`. [[7] p.499]

• Règle 599 : Une transition sortante d'un pseudo-état initial au plus haut niveau (région d'une machine à état) ne doit pas contenir le champ `event-signature` sauf si cette transition est de stéréotype « create ». [[7] p.499]

• Règle 600 : Dans le cas où il y a plus d'un déclencheur, leurs signatures doivent être compatibles dans le cas où les paramètres du signal sont assignés à des attributs ou des variables locales. [[7] p.499] [Règle sur le méta-modèle][3]

• Règle 601 : Le contexte de redéfinition d'une transition est la première machine à état contenante ou le classificateur contexte. [[7] p.498] [Règle sur le méta-modèle]

• Règle 602 : Si plusieurs transitions ont le même état source et le même événement d'activation alors ces transitions doivent avoir des conditions de gardes mutuellement exclusives. [Nouvelle règle]

Remarque Cette règle prend également en compte le cas où les transitions sont des transitions d'achèvement et qu'elles n'ont donc pas d'événement d'activation.

• Règle 603 : Si un événement active à la fois une transition interne et une transition externe (transitions qui provoquent la sortie de l'état), les conditions de garde de l'événement de la transition interne et de la transition externe doivent être mutuellement exclusives. [Nouvelle règle]

Remarque Dans le cas contraire, le comportement spécifié n'est pas déterministe. Les transitions potentiellement non déterministes à différents niveaux de machine à états ne sont pas traités ici car des règles sémantiques ont été établies pour les résoudre (cf. p.493 de [7]). [4]

Remarque Une transition interne est une transition qui est exécutée sans sortir ni "re-rentrer" dans l'état dans lequel elle est définie.

• Règle 604 : Lorsqu'un état composé est la source d'une transition d'achèvement, chaque région directement incluse dans cet état doit contenir un état final. [Nouvelle règle]

Remarque D'après [1] p.159, il est possible d'envoyer des messages à d'autres machines à états. Ceci peut être fait par le champs `action-expression` (cf. la partie 21.1.2) ou par des relations de dépendances (cf. Figure 21.1 (tirées de [1]) et 21.2). 22.1.3.

21.2 Protocol Transition

Contexte Une transition de protocole (*protocol transition*) est une spécialisation des transitions présentes dans les machines à états.

Une transition de protocole peut être associée à des préconditions et postconditions.

[3] pas bien compris

[4] Ça peut cependant donner lieu à un guide de prévention : éviter les transitions activées sur le même déclencheur pour un état et un état qui lui appartient.

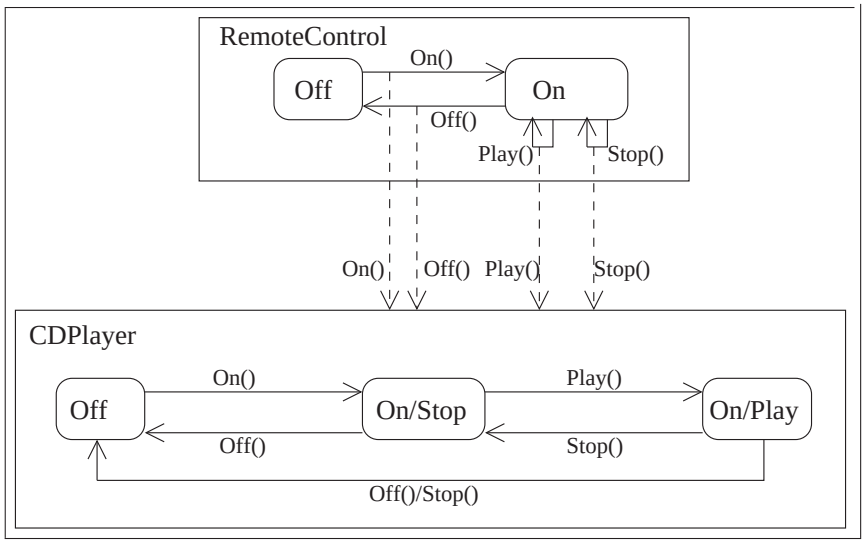

FIG. 21.1 – Première notation pour envoyer des messages entre machines à états

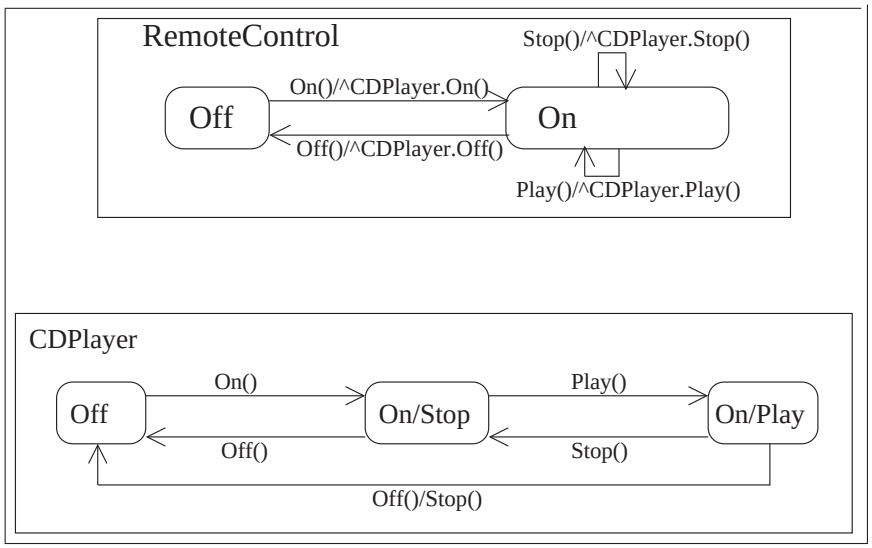

FIG. 21.2 – Deuxième notation pour envoyer des messages entre machines à états

Règles de cohérence

• RÈGLE LIEN 54 : Une transition de protocole est une transition et doit donc répondre aux règles exprimées en 21.1.

• RÈGLE 605 : Une transition de protocole appartient toujours à une machine à états de description de protocole (cf. section 22.2). [[7] p.467]

• RÈGLE 606 : Une transition de protocole n'a jamais d'action associée. [[7] p.467]

• RÈGLE 607 : Si une transition de protocole se réfère à une opération (i.e. si le déclencheur correspond à une opération), alors cette opération doit appartenir au classificateur contexte de la machine à états de description de protocole. [[7] p.467]

• RÈGLE LIEN 55 : Les préconditions et postconditions éventuellement associées aux transitions de protocole sont des contraintes et doivent répondre aux règles exprimées en 2.7. [Règle dérivée du méta-modèle]

Chapitre 22

Diagrammes

22.1 State Machine Diagram

Contexte Les machines à état (*states machines* en anglais) sont utilisées pour décrire le comportement dynamique d'une partie d'un système.

Les règles concernant les machines à état sont décomposées en cinq sections, les règles générales qui sont présentées dans la section 22.1.1, les règles sur les extensions de machines à état présentées en 22.1.2, les règles concernant la communication entre machines à état présentée en 22.1.3, les règles sur l'utilisation des points d'entrée et de sortie sont présentées en 22.1.4, enfin les règles prenant en compte les événements différés sont présentés en 22.1.5.

22.1.1 Règles générales

Règles de cohérence

• Règle 608 : Le classificateur contexte de la machine à état ne peut pas être une interface. [[7] p.490]

• Règle 609 : Le classificateur contexte d'une machine à état qui représente une méthode d'une caractéristique comportementale doit être le classificateur qui contient la caractéristique comportementale. [[7] p.490] [Règle sur le méta-modèle]

• Règle 610 : Les points de connection d'une machine à état sont des pseudo-états de type point d'entrée ou point de sortie. [[7] p.490] [Règle sur le méta-modèle]

• Règle 611 : Une machine à état qui représente une méthode d'une caractéristique comportementale ne peut pas avoir de point d'entrée ni de point de sortie. [[7] p.490]

• Règle 612 : Le contexte de redéfinition d'un état est la machine à état contenante ou le classificateur contexte. [[7] p.490] [Règle sur le méta-modèle][1]

• Règle 613 : Une transition composée (*compound transition* en anglais) ne doit pas contenir de circuit. [Nouvelle règle]

Remarque Une transition composée est un chemin de une ou plusieurs transitions qui a comme source et comme cible un ensemble d'états (par opposition aux pseudo-états). Elle est composée uniquement de pseudo-états de jonction (*junction pseudo-states* en anglais), de choix, de bifurcation (fork) et d'union (join) qui sont reliés par des

[1] contredit en partie la règle 530 avec laquelle elle est redondante. La formulation ci-dessus est en accord avec la règle OCL, pas la règle 530

transitions.

La figure 22.1 montre un exemple de transition composée avec circuit.

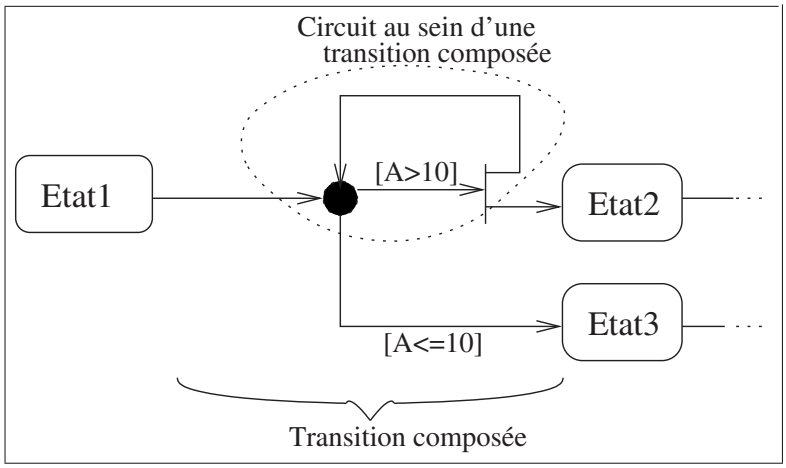

FIG. 22.1 – Transition composée avec circuit

22.1.2 Extension de machines à état

Contexte Une machine à état est généralisable. Une machine à état spécialisée est ne extension d'une machine à état plus générale dans laquelle des régions, des sommets et des transitions peuvent être ajoutés et des états, des régions, des transitions et des états sous-machines peuvent être redéfinis.

L'extension de machines à état permet de redéfinir le comportement d'un classificateur spécialisé comme une extension du comportement d'un classificateur général.

Une entité redéfinie est marquée avec la mot clé `{extended}`.

Il est possible d'empêcher la redéfinition d'une entité en lui associant le mot clé `{final}`.

Des exemples de redéfinition de machines à état sont disponibles en pages 496 et 497 de [7].

Règles de cohérence

• RÈGLE 614 : Dans le cas de classificateurs généraux multiples, l'extension implique que la machine à état est composée d'une région orthogonale pour chacune des machines à état des classificateurs généraux. [tirée de [7] p.494]

• RÈGLE LIEN 56 : Le classificateur contexte d'une machine à état spécialisée doit spécialiser le classificateur contexte de la machine à état redéfinie. Cette règle est héritée de la règle 48.

• RÈGLE 615 : Si une machine à état est marquée comme étant une redéfinition d'une machine à état plus générale, il doit exister une machine à état plus générale qui porte le même nom que la machine à état spécialisée. [Nouvelle règle]

• RÈGLE 616 : Tout état, région ou transition qui redéfinit respectivement un état, une région ou une transition doit appartenir à une machine à état qui elle-même redéfinit

une autre machine à état. [Nouvelle règle]

• Règle 617 : Aucun état, région, machine à état, état sous-machine et transition marqué comme final ne doit être redéfini. [Nouvelle règle]

• Règle 618 : La redéfinition d'une transition ne peut porter ni sur l'état source, ni sur le déclencheur de la transition (spécifié par le champs `event-signature` cf. section 21.1.2). [Nouvelle règle]

• Règle 619 : Seules les transitions qui peuvent être identifiées de façon unique par leur couple (état source, déclencheur) peuvent être redéfinies. [Nouvelle règle]

22.1.3 Envoi de messages entre machines à état

Contexte Il est possible de faire communiquer plusieurs machines à état entre elles. Ceci se fait en spécifiant lors de l'envoi de messages dans une machine à état, la machine à état réceptrice du message.

Règles de cohérence

• Règle 620 : Le classificateur ou l'instance qui est le contexte de la machine à état réceptrice du message doit être connu par le classificateur ou l'instance contexte de la machine à état émettrice. [Nouvelle règle]

Remarque Un classificateur en "connaît" un autre lorsque tous deux font partie des membres d'un même espace de nommage (en incluant les membres importés par l'espace de nommage).

22.1.4 Utilisation des points d'entrée et de sortie pour un machine à état

Contexte UML offre la possibilité d'utiliser des points d'entrée et de sortie pour les machines à état.

Règles de cohérence

• Règle 621 : Une transition dont la source est un pseudo-état point d'entrée doit avoir pour cible un sommet qui appartient à la machine à état qui contient le point d'entrée. [Nouvelle règle]

• Règle 622 : Une transition dont la cible est un pseudo-état point d'entrée doit avoir pour source un sommet qui n'appartient pas à la machine à état qui contient le point d'entrée. [Nouvelle règle]

• Règle 623 : Une transition dont la cible est un pseudo-état point de sortie doit avoir pour source un sommet qui appartient à la machine à état qui contient le point de sortie. [Nouvelle règle]

• Règle 624 : Une transition dont la source est un pseudo-état point de sortie doit avoir pour cible un sommet qui n'appartient pas à la machine à état qui contient le point de sortie. [Nouvelle règle]

22.1.5 Deferred events

Contexte UML offre la possibilité de différer la prise en compte des événements dans certains états. Ces événements sont destinés à être consommés par une transition ou perdus par un état qui ne diffère pas l'événement.

Remarques La possibilité de faire des états composites peut engendrer des conflits sur les événements différés ou qui activent une transition. Ces conflits sont résolus par les règles p.482 de [7].

Règles de cohérence

• Règle 625 : Quand un état diffère un événement, il doit exister un chemin à partir de cet état tel que l'événement est consommé par une transition. [Nouvelle règle]

Remarques Le figure 22.2 montre un exemple où le règle 625 est respectée alors que la figure 22.3 ne la respecte pas.

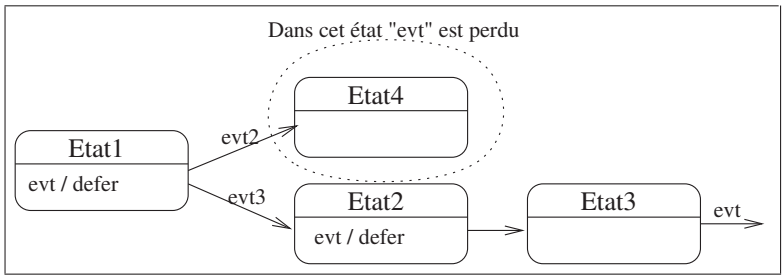

Fig. 22.2 – Événement différé correctement

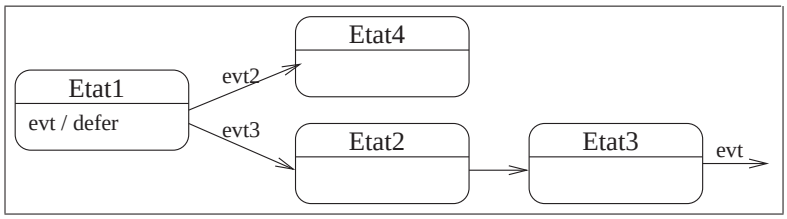

Fig. 22.3 – Événement différé inutilement

22.2 Protocol State Machine

Contexte Il est possible de spécifier des machines à état de description de protocole (*Protocol State Machine* en anglais) qui est une spécialisation des machines à état classiques. Cette spécialisation implique quelques contraintes supplémentaires.

Remarque Une machine à état de description de protocole se différencie des machines à état en associant le nom de la machine à état au mot clé `{protocol}`.

Règles de cohérence

• Règle 626 : Le contexte d'une machine à état de description de protocole est forcément un classificateur (donc pas un opération). [[7] p.464]

• Règle 627 : Toutes les transitions d'une machine à état de description de protocole doivent être des transitions de protocole (cf. section 21.2). [[7] p.464]

• Règle 628 : Si deux ports sont connectés, alors la machine à état de description de protocole de l'interface requise (si définie) doit être conforme à la machine à état de description de protocole de l'interface fournie (si définie). [[7] p.464] [2]

• Règle 629 : Les états de protocoles (c'est-à-dire les états qui appartiennent à une machine à état de description de protocole) n'ont pas d'activité d'entrée, de sortie et d'activité qui s'exécutent tant que l'état est actif. [[7] p.480]

• Règle 630 : Les machines à état de protocoles ne peuvent pas contenir de pseudo-états historiques. [[7] p.480]

• Règle lien 57 : Une machine à état de description de protocole est une machine à état et doit donc satisfaire les contraintes énoncées en 22.1.

[2] faire un guide qui dit d'associer une interface à un port

Huitième partie

Diagramme des cas d'utilisation

Chapitre 23

Éléments

23.1 Actor

Contexte Un acteur (*actor* en anglais) représente un ensemble cohérent de rôles que peuvent jouer des utilisateurs d'un système lorsqu'ils interagissent avec les cas d'utilisation.

Règles de cohérence

• Règle 631 : Un acteur ne peut avoir des associations que avec des cas d'utilisation, des sous-systèmes, des composants et des classes. [[7] p.513]

• Règle 632 : Les associations mettant en jeu un acteur doivent être binaires. [[7] p.513]

• Règle 633 : Un acteur doit avoir un nom. [[7] p.513]

• Règle 634 : Un acteur ne doit pas être contenu par un autre classificateur. [Règle dérivée du méta-modèle]

23.2 Extension Point

Contexte Un point d'extension identifie un point dans le comportement d'un cas d'utilisation où le comportement peut être étendu par le comportement d'un autre cas d'utilisation, comme spécifié par une relation d'extension (cf. section 24.1).

Règles de cohérence

• Règle 635 : L'expression d'un point d'extension doit suivre la syntaxe suivante : [[7] p.517]

```
<extension point> ::= <name> [: <explanation>]
```

Remarque Le champ `<explanation>` peut être exprimé en langage naturel ou de façon plus précise et indique à quel moment dans le comportement du cas d'utilisation

le point d'extension se produit.

• Règle 636 : Un point d'extension doit avoir un nom. [[7] p.517]

• Règle 637 : Pour tout point d'extension, il doit exister une relation d'extension dans le modèle qui y fait référence. [Nouvelle règle]

• Règle 638 : Un point d'extension est toujours contenu dans un cas d'utilisation. [Règle dérivée du méta-modèle]

23.3 Use Case

Contexte Un cas d'utilisation (*use case* en anglais) est un classificateur qui représente une unité cohérente de fonctionnalités fournies par un système, un sous-système ou une classe.

Règles de cohérence

• Règle 639 : Un cas d'utilisation doit avoir un nom. [[7] p.520]

• Règle 640 : Les cas d'utilisation ne doivent participer qu'à des associations binaires. [[7] p.520]

• Règle 641 : Un cas d'utilisation ne doit pas avoir d'associations avec d'autres cas d'utilisation qui spécifient le même système. [[7] p.520]

Chapitre 24

Relations

24.1 Extend

Contexte Une relation d'extension (*extend relationship* en anglais) met en relation un cas d'utilisation « étendant » et un cas d'utilisation étendu et spécifie comment et quand le comportement du cas d'utilisation « étendant » peut être inséré dans le comportement du cas d'utilisation étendu.

Il est possible de préciser via une note, la condition qui déterminera si le cas d'utilisation est étendu ou non par le cas d'utilisation cible. La note peut également faire apparaître le nom du point d'extension à laquelle se réfère la condition (cf. figure 24.1 (tirée de [7] p.516)).

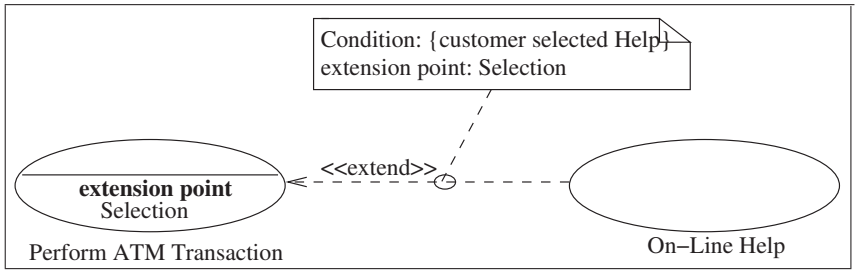

FIG. 24.1 – Exemple de relation d'extension avec spécification du nom du point d'extension et de la condition

Règles de cohérence

• RÈGLE 642 : Une relation d'extension a comme source un (et un seul) cas d'utilisation. [Règle dérivée du méta-modèle]

• RÈGLE 643 : Une relation d'extension a comme cible un (et un seul) cas d'utilisation. [Règle dérivée du méta-modèle]

• RÈGLE 644 : Le point d'extension qui est référencé par la relation d'extension doit appartenir au cas d'utilisation étendu. [[7] p.515]

• RÈGLE LIEN 58 : La condition (si présente dans la note) est une contrainte et doit donc répondre aux règles de la section 2.7. [Règle dérivée du méta-modèle]

Remarque Les cas d'utilisation étant généralement utilisés pendant la phase d'analyse, il est souvent difficile d'exprimer la contrainte autrement qu'en langage naturel, ce qui limite la capacité d'analyse statique.

24.2 Include

Contexte Une relation d'inclusion (*include relationship* en anglais) entre deux cas d'utilisation implique que le comportement du cas d'utilisation qui est inclus est inséré dans le comportement du cas d'utilisation « incluant ».

Le cas d'utilisation cible de la relation est inclus dans la cas d'utilisation source de la relation.

Règles de cohérence

• Règle 645 : Une relation d'inclusion a comme source un (et un seul) cas d'utilisation. [Règle dérivée du méta-modèle]

• Règle 646 : Une relation d'inclusion a comme cible un (et un seul) cas d'utilisation. [Règle dérivée du méta-modèle]

• Règle 647 : Les relations d'inclusion ne doivent pas former de circuit. [Nouvelle règle]

Chapitre 25

Diagrammes

25.1 Use Case Diagram

Contexte Les cas d'utilisation sont un moyen de spécifier les usages attendus d'un système. Typiquement ils sont utilisés pour cerner les exigences fonctionnelles d'un système, c'est-à-dire les fonctionnalités que le système doit remplir. Les concepts clés d'un tel diagramme sont les cas d'utilisation, les acteurs et les sujets (*subject* en anglais). Le sujet est le système considéré sur lequel les cas d'utilisation s'appliquent. Les utilisateurs et autres systèmes qui interagissent avec le sujet sont représentés comme des acteurs.

Les diagrammes des cas d'utilisation sont une spécialisation des diagrammes de classes dans lesquels les seuls classificateurs qui apparaissent sont soit des acteurs soit des cas d'utilisation.

Règles de cohérence

• Règle 648 : Les nœuds graphiques qui peuvent apparaître dans un diagramme des cas d'utilisation sont : [[7] p.523]

- des acteurs ;
- des points d'extension ;
- des cas d'utilisation.

Remarque Il est également possible de représenter le sujet des cas d'utilisation comme un rectangle qui entoure tous les cas d'utilisation.

Remarque Les cas d'utilisation peuvent apparaître dans d'autres diagrammes afin de montrer à quel classificateur un cas d'utilisation appartient.

• Règle 649 : Les chemins graphiques qui peuvent apparaître dans un diagramme des cas d'utilisation sont : [[7] p.523]

- des relation d'extension ;
- des relations d'inclusion ;
- des associations binaires ;
- des relations de généralisation.

[1]

[1] Les règles sur les diagrammes sont elle bien formulées ? plutot dire que les outils de modélisation doivent être capables de représenter les éléments et relations : en effet on peut par exemple trouver des use case dans des paquetages...

Neuvième partie

Cohérence inter-diagrammes

Chapitre 26

Cohérence inter-diagrammes

Nous reportons dans ce chapitre les règles déjà trouvées dans les parties consacrées aux diagrammes et concernant des problèmes de cohérence inter-diagrammes. A cette fin, nous faisons une section par couple de diagrammes considérés. Seuls les couples de diagrammes pour lesquels des règles existent font l'objet d'une section.

26.1 Classes - Objets

• Règle lien 59 : (cf. règle 204) Aucun circuit faisant intervenir des liens qui correspondent à des associations dont le type d'agrégation est `share` ou `composite` ne doit apparaître au niveau objet. [Nouvelle règle]

• Règle lien 60 : (cf. règle 207) Toute instance de la classe composante doit appartenir à une et une seule instance de la classe composée. [Nouvelle règle]

• Règle lien 61 : (cf. règle 208) Lorsque plusieurs classes composent une même classe et que ces classes sont mises en relation via une association, les objets (instances des classes composantes) qui sont mis en relation par un lien (instance de l'association) doivent tous appartenir au même objet composite (qui est l'instance de la classe composée). [1] En d'autres termes, dans ce cas, le lien doit mettre en relation des objets (instances des classes composantes) qui appartiennent au même objet (instance de la classe composée). [Nouvelle règle]

26.2 Classes - Structures composites

• Règle lien 62 : (cf. règle 251) Dans un diagramme de structures composites, si une possession A apparaît comme composite (A est une partie (*part* en anglais) de B), alors le méta-attribut `isComposite` de la possession A doit être vrai. Ceci implique dans le diagramme de classes, que si la classe A est reliée par une association à la classe B alors cette association doit être composite (la classe englobante étant la classe B). [Nouvelle règle]

• Règle lien 63 : (cf. règle 252) Une possession A qui n'est pas une partie de la classe B dans un diagramme de structures composites, doit être une possession dont le

[1] !!! à approfondir : Est-ce que qu'il est possible qu'une classe composante soit en relation avec une classe qui ne fait pas partie de la classe composée ?

méta-attribut `isComposite` est faux. Ceci implique dans le diagramme de classes, que si la classe A est reliée par une association à la classe B (la classe englobante) alors cette association doit être une association navigable. [Nouvelle règle]

• Règle lien 64 : (cf. règle 268) Le champ `classname` d'un connecteur nommé doit être le nom d'exactement une association. [Nouvelle règle]

• Règle lien 65 : (cf. règle 273) Dans le cas où un connecteur est typé par une association, la multiplicité de chaque fin de connecteur doit être un sous-ensemble de la multiplicité de la fin d'association correspondante. [Nouvelle règle]

26.3 Classes - Interactions

26.3.1 Règles générales

• Règle lien 66 : (cf. règle 163) Une dépendance de stéréotype « `call` » implique que la source de la relation fait appel au moins une fois à la cible de la relation. Cet appel doit être visible dans un des diagrammes d'interaction (diagrammes de séquence, de communication). [Nouvelle règle]

Remarque Dans le cas contraire, le modèle n'est pas incohérent mais la relation « d'appel » est aberrante.

• Règle lien 67 : (cf. règle 200) Seuls des échanges de messages dans les sens navigables sont possibles. [Nouvelle règle]

• Règle lien 68 : (cf. règle 209) Lorsque deux classes sont reliées par une relation de composition, seul l'objet composé ou un des objets qu'il englobe (par héritage) peuvent créer ou détruire les objets composants. [Nouvelle règle]

• Règle lien 69 : (cf. règle 490) Si le message référence une opération, cette opération doit appartenir à la classe référencée par la ligne de vie réceptrice et doit être visible par la classe source du message.[Nouvelle règle]

• Règle lien 70 : (cf. règle 492) Si le message référence un signal, ce signal doit être visible par les classes émettrice et réceptrice du message. [Nouvelle règle]

26.3.2 Classes - Séquence

• Règle lien 71 : (cf. règle 167) Si l'objet créé (instance d'une classe B) appartient à une classe A (spécifié par une relation de composition entre les classes A et B), alors l'objet créateur doit être une instance de A ou doit pouvoir appartenir à la classe A. [Nouvelle règle]

• Règle lien 72 : (cf. règle 72) Les messages ignorés ou considérés doivent correspondre à des noms de messages connus. [Nouvelle règle][2]

Remarque L'ensemble des noms de messages connus correspond à l'ensemble des opérations des différentes classes représentées par les lignes de vie de l'interaction et l'ensemble des signaux visibles par ces classes.

• Règle lien 73 : (cf. règle 472) Le classificateur qui contient l'élément connectable (*ConnectableElement* en anglais, voir section A.3) référencé doit être le même classificateur, ou un ancêtre, du classificateur qui contient l'interaction englobant la ligne de vie. [[7] p.427]

[2]Inter-diag sequence - classes uniquement ?

26.4 Classes - Machines à états

• Règle lien 74 : (cf. règle 63) Dans un diagramme de machines à états, une opération sans effet de bord ne doit pas être le déclencheur d'une transition qui relie deux états différents du classificateur contenant l'opération. [Nouvelle règle]

• Règle lien 75 : (cf. règle 64) La méthode associée à une opération ne doit pas modifier les paramètres dont le mode de passage est `in`. [Nouvelle règle]

• Règle lien 76 : (cf. règle 80) Les méthodes associées à des opérations ne doivent pas modifier d'attributs qui ont la propriété `readOnly`. [Nouvelle règle]

• Règle lien 77 : (cf. règle 81) Si un attribut est marqué avec la propriété `{readOnly}`, aucune action d'une transition (champ `action-expression` du label) du diagramme d'état du classificateur ne doit modifier cet attribut. [Nouvelle règle]

26.5 Classes - Activités

• Règle lien 78 : (cf. règle 64) La méthode associée à une opération ne doit pas modifier les paramètres dont le mode de passage est `in`. [Nouvelle règle]

• Règle lien 79 : (cf. règle 80) Les méthodes associées à des opérations ne doivent pas modifier d'attributs qui ont la propriété `readOnly`. [Nouvelle règle]

• Règle lien 80 : (cf. règle 305) Si l'état de l'objet est indiqué, cet état doit correspondre à un état connu de l'objet. [Nouvelle règle]

Remarque Cette règle ne peut s'appliquer que si le type du nœud objet est spécifié. De plus, on appelle état connu d'un objet, un attribut suivi de sa valeur ou un état du diagramme d'état du classificateur qui type le nœud objet.

• Règle lien 81 : (cf. règle 350) Si une partition d'activité représente une partie, alors toutes les partitions non externes qui appartiennent à la même dimension doivent représenter des parties directement contenues par la structure interne du même classificateur. [[7] p.307]

• Règle lien 82 : (cf. règle 351) Si une partition d'activité non-externe représente un classificateur et est contenue par une autre partition, alors la partition contenante doit représenter un classificateur. De plus, le classificateur de la sous-partition doit être contenu par le classificateur représenté par la partition contenante, ou être associé à ce classificateur par une association de forte composition (et être du coté qui est contenu). [[7] p.307]

• Règle lien 83 : (cf. règle 355) Dans le cas où une partition représente une classe, chaque action qui se trouve dans la partition (et qui n'est pas marquée comme « externe ») doit correspondre à une opération de la classe ou à une réception de la classe. [Nouvelle règle]

26.6 Objets - Interactions

• Règle lien 84 : (cf. règle 498) Un message d'appel asynchrone d'opération doit référencer une opération active. [Nouvelle règle]

• Règle lien 85 : (cf. règle 499) Dans le cas d'un signal asynchrone, les deux objets communicants doivent appartenir à des threads de contrôle différents.[Nouvelle règle]

Remarque Les objets actifs sont des instances de classes actives. Les cas possibles concernant la règle 85 sont les suivants :

- Les deux objets sont deux objets actifs.
- Un des objets est actif, l'autre doit être un objet qui appartient à un objet actif différent du premier.
- Les 2 objets appartiennent à des objets actifs différents.

26.7 Composants - Machines à états

• Règle lien 86 : (cf. règle 232) Dans le cas où un comportement tel une machine à état de protocole est attaché à une interface, un port ou au composant lui-même pour définir la vue externe du composant plus précisément en rendant explicites les contraintes dynamiques dans la séquence des appels d'opérations, les événements correspondants à une opération ou à un signal doivent pouvoir être réalisés par le composant ou délégués via une interface requise. [Nouvelle règle]

26.8 Composants - Activités

• Règle lien 87 : (cf. règle 350) Si une partition d'activité représente une partie, alors toutes les partitions non externes qui appartiennent à la même dimension doivent représenter des parties directement contenues par la structure interne du même classificateur. [[7] p.307]

26.9 Composants - Séquence

• Règle lien 88 : (cf. règle 472) Le classificateur qui contient l'élément connectable (*ConnectableElement* en anglais, voir section A.3) référencé doit être le même classificateur, ou un ancêtre, du classificateur qui contient l'interaction englobant la ligne de vie. [[7] p.427]

Dixième partie

Conclusion et annexes

Chapitre 27

Conclusion

Nombre de règle total : 649
Nombre de nouvelles règles : 291
Nombre de règles directement tirées de la norme : 296
Nombre de règles directement tirées de la bibliographie : 42
Nombre de règles dérivées du méta-modèle : 57

Nombre de règles sur le méta-modèle : 47
Nombre de règles utilisateur : 26

Annexe A

Éléments du méta-modèle

A.1 Classificateur

Le document fait référence à la classe du méta-modèle `Classifier` à divers endroits. Afin de comprendre ce qui est représenté par cette classe nous avons déterminé l'ensemble des classes qui hérite de cette classe et qui sont donc des classificateurs (cf. FIG A.1).

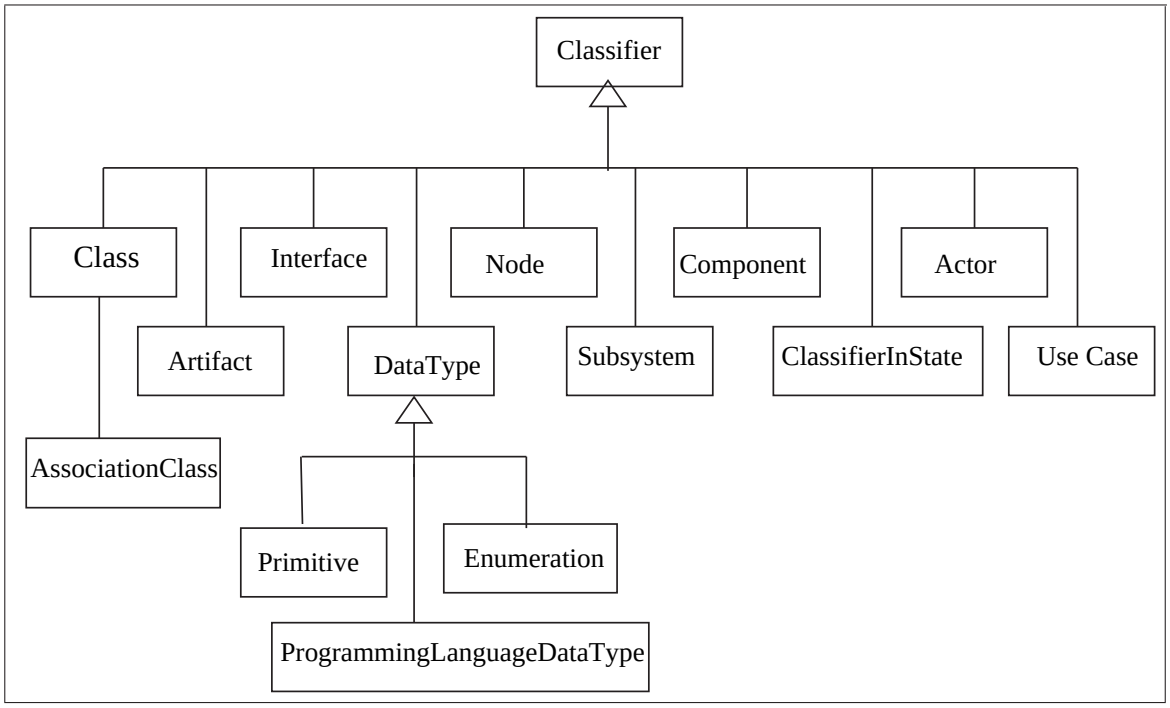

FIG. A.1 – Arbre de spécialisation de la classe `Classifier`

Ainsi, lorsque le mot classificateur est employé dans le document ceci peut être :
- une classe ;
- une classe association ;
- une interface ;
- un nœud ;
- un type de donnée ;

- une énumération ;
- une primitive [1] ;
- un type de donnée issue d'un langage de programmation [2] ;
- un sous-système ;
- un acteur ;
- un cas d'utilisation ;
- un composant ;
- un classifier in state [3] ;
- un artifact ;

A.2 Dépendances

La figure A.2 montre l'arbre de spécialisation des dépendances.

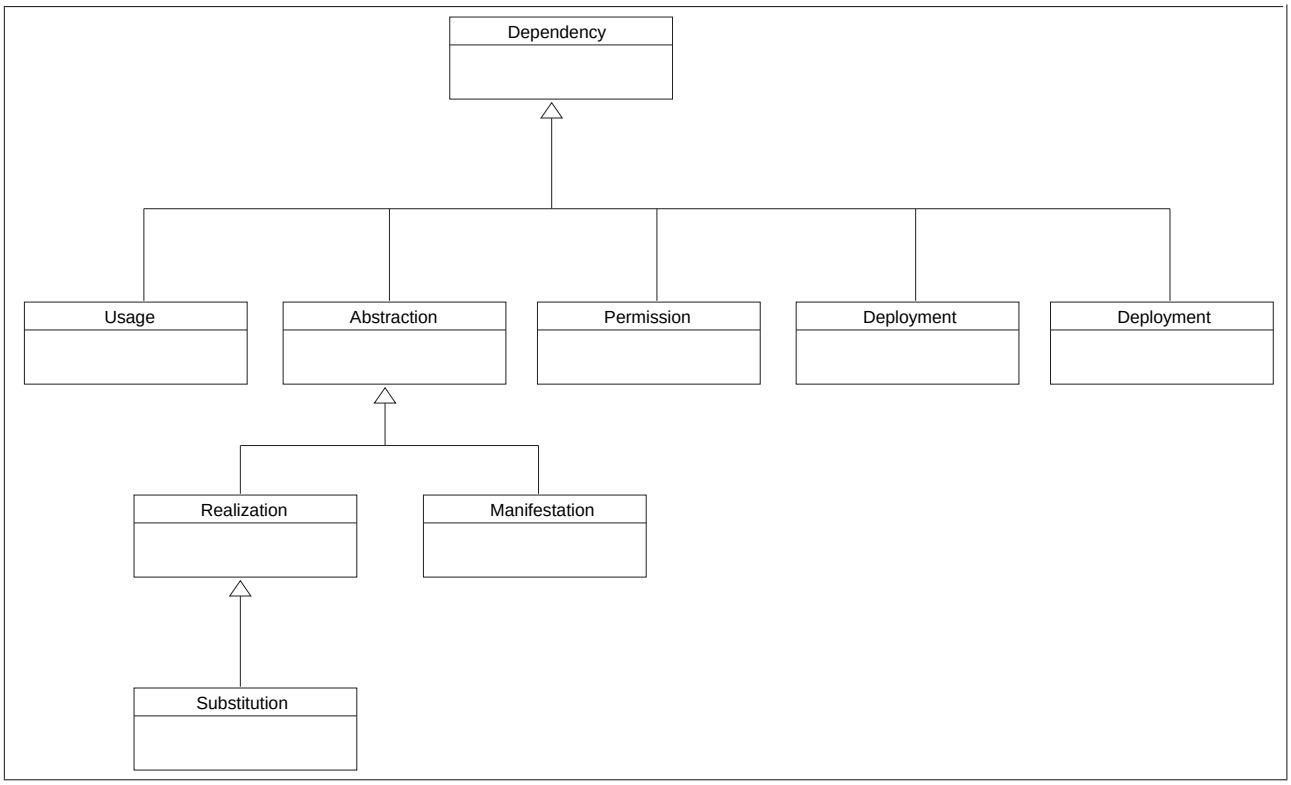

Fig. A.2 – Arbre de spécialisation des dépendances

A.3 ConnectableElement

La figure A.3 montre l'arbre de spécialisation de la classe `ConnectableElement`.
Remarque La classe `{Property}` représente les attributs ou les fins d'associations.

[1] à préciser
[2] à préciser
[3] à préciser

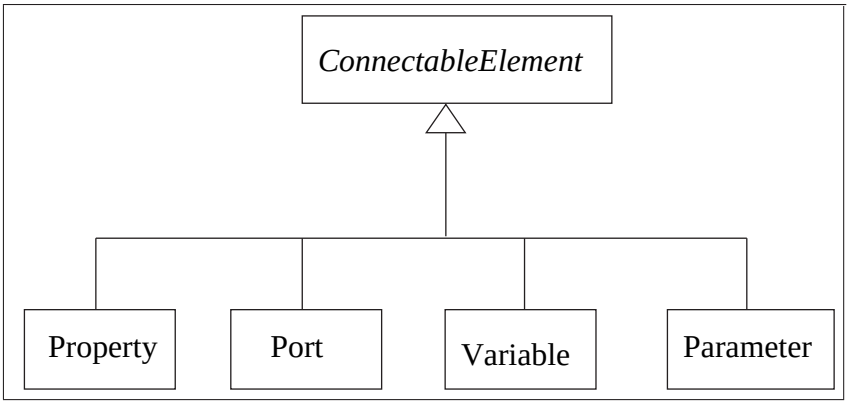

FIG. A.3 – Arbre de spécialisation de la classe abstraite `ConnectableElement`

A.4 Namespace

La figure A.4 montre l'arbre de spécialisation de la classe `Namespace`.

L'arbre de généralisation de `Classifier` n'est pas montré mais est présent en section A.1.

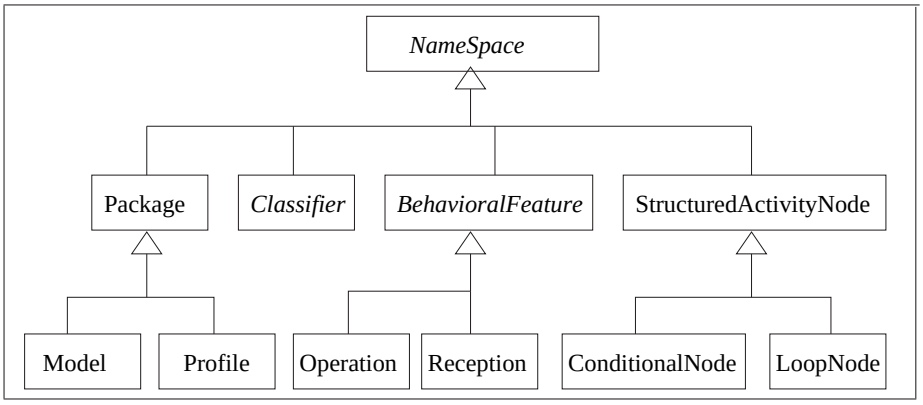

FIG. A.4 – Arbre de spécialisation de la classe abstraite `Namespace`

A.5 RedefinableElement

La figure A.5 montre l'arbre de spécialisation de la classe `RedefinableElement`. Attention les classes `ActivityNode` et `ActivityEdge` ne sont pas développée.

A.6 PackageableElement

La figure A.6 montre l'arbre de spécialisation de la classe `PackageableElement`.

Attention les arbres de spécialisation de `Classifier` et de `Dependency` ne sont pas développés (se référer respectivement aux sections A.1 et A.2).

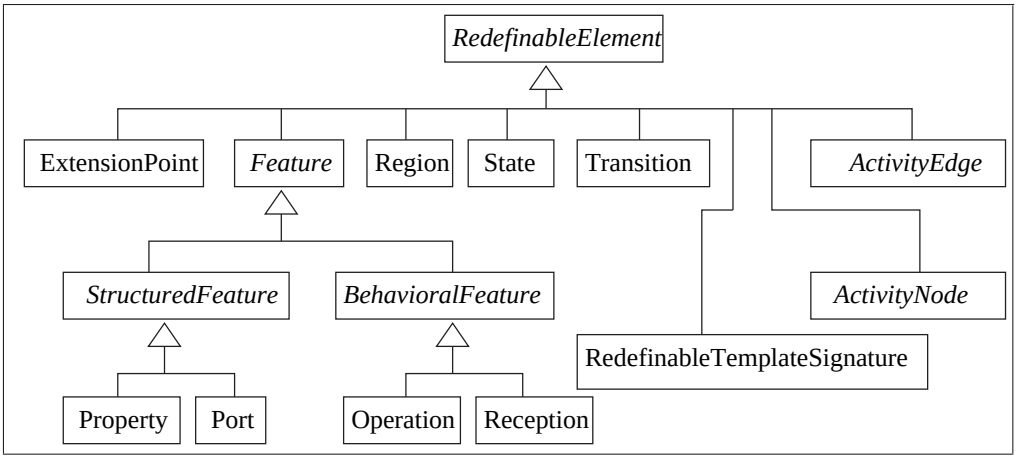

FIG. A.5 – Arbre de spécialisation de la classe `RedefinableElement`

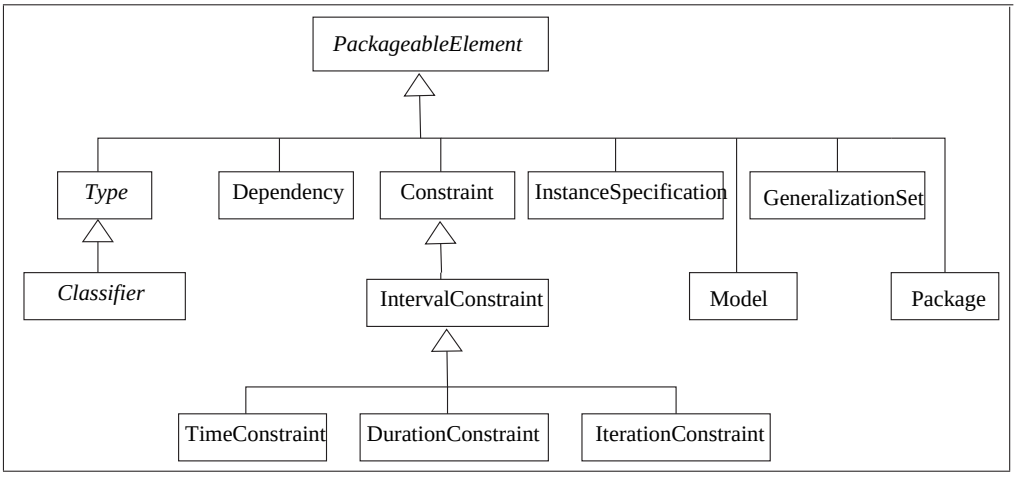

FIG. A.6 – Arbre de spécialisation de la classe `PackageableElement`

A.7 ParameterableElement

[4]

Contexte La figure A.7 montre l'arbre de spécialisation de la classe `ParameterableElement`. Cette méta-classe abstraite représente l'ensemble des éléments d'UML qui peuvent être passés comme paramètre générique d'un éléments.

Notons que dans ce contexte, la classe `Property` représente des attributs (et pas des fins d'associations).

Ajoutons que l'arbre de spécialisation de la classe `classifier` n'est pas développé ici mais qu'il est disponible en annexe A.1.

[4] la traduction "élement paramétrique" pour ParameterableElement n'est pas bonne, car ParameterableElement signifie que l'élément peut être un paramètre d'un signature de généricité

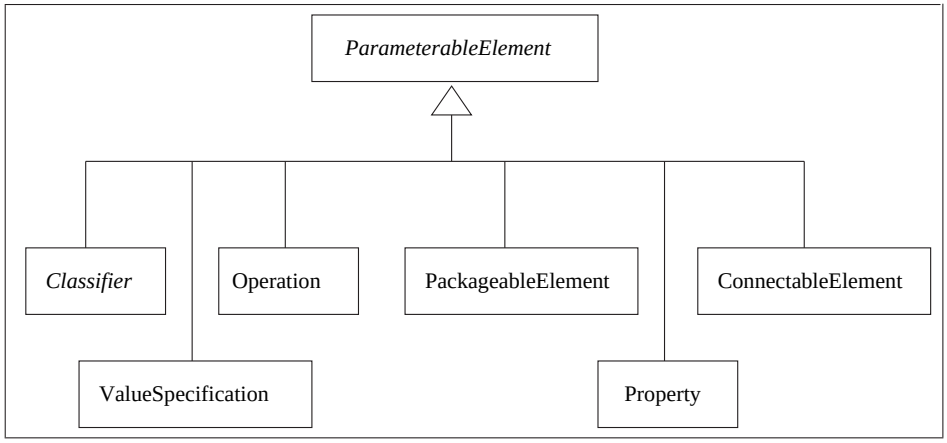

FIG. A.7 – Arbre de spécialisation de la classe `ParameterableElement`

A.8 TemplateableElement

La figure A.8 montre l'arbre de spécialisation de la classe `TemplateableElement`. Cette méta-classe abstraite représente l'ensemble des éléments d'UML qui supporte la généricité.

Notons que l'arbre de spécialisation de la classe `classifier` mais est disponible en annexe A.1.

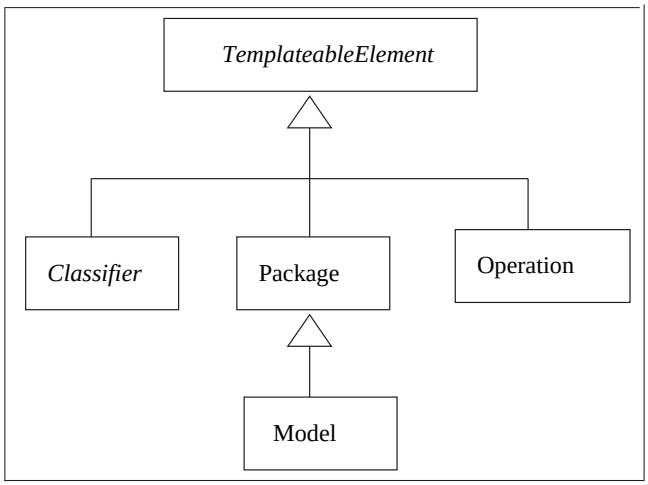

FIG. A.8 – Arbre de spécialisation de la classe `TemplateableElement`

A.9 EncapsulatedClassifier

La figure A.8 montre l'arbre de spécialisation de la classe `EncapsulatedClassifier`. Cette méta-classe abstraite représente l'ensemble des éléments d'UML qui peuvent avoir des ports de communication.

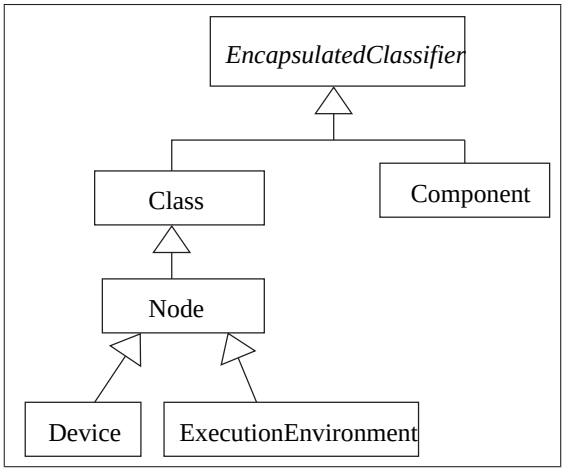

FIG. A.9 – Arbre de spécialisation de la classe `EncapsulatedClassifier`

A.10 Object Node

La figure A.10 montre l'arbre de spécialisation de la classe `ObjectNode`. Cette méta-classe abstraite est utilisée pour définir les flots d'objets dans les diagramme d'activité.

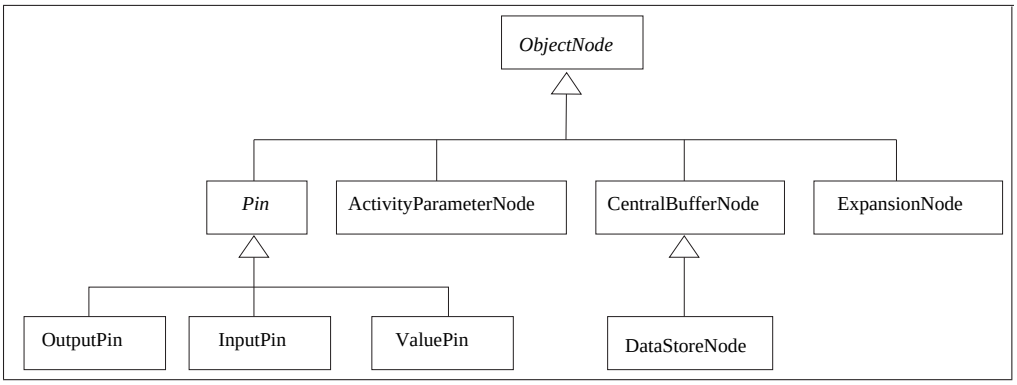

FIG. A.10 – Arbre de spécialisation de la classe `ObjectNode`

A.11 Control Node

La figure A.11 montre l'arbre de spécialisation de la classe `ControlNode`. Cette méta-classe abstraite est utilisée pour définir le flot de contrôle dans les diagramme d'activité.

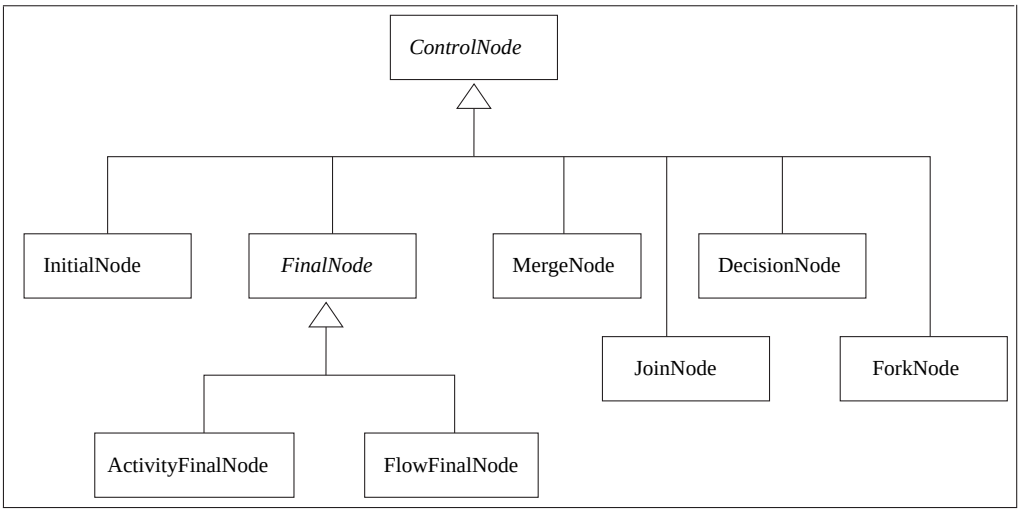

FIG. A.11 – Arbre de spécialisation de la classe `ControlNode`

Annexe B

Mots prédéfinis du langage UML

Dans cette annexe nous présentons l'ensemble des propriétés, des mots clés et des contraintes prédéfinis du langage UML (première colonnes des différents tableaux). La deuxième colonne des tableaux spécifie l'élément sur lequel doit s'appliquer la propriété, le mot clé ou la contrainte.

B.1 Propriétés

La présente section a pour but de lister l'ensembles des propriétés qui sont prédéfinies en UML.
Le tableau B.1 donne en première colonne la propriété et en deuxième colonne l'élément UML sur lequel la propriété s'applique.

{abstract}	Classifier
{readOnly}	Association
{union}	Association
{subsets <prop-name>}	Association
{redefines <prop-name>}	Association
{unrestricted}	StructuralFeature (et donc attribut)
{readOnly}	StructuralFeature (et donc attribut)
{union}	attribute
{subsets <property-name>}	attribute
{redefines <property-name>}	attribute
{ordered}	attribute
{bag}	attribute
{seq} ou {sequence}	attribute
{composite}	attribute
{query}[1]	operation
{subsets <property-name>}	fin d'association
{redefined <end-name>}	fin d'association

[1] pas sûr

{union}	fin d'association
{ordered}	fin d'association
{bag}	fin d'association
{sequence} ou {seq}	fin d'association
{readOnly}	fin d'association
Si la fin d'association est navigable, toute propriété qui s'applique aux attributs	fin d'association
{complete, disjoint}	ensemble de généralisation
{incomplete, disjoint}	ensemble de généralisation
{complete, overlapping}	ensemble de généralisation
{incomplete, overlapping}	ensemble de généralisation
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	
{}	

Tab. B.1: Liste des propriétés définies dans UML

B.2 Mots Clés

« import »	relation de dépendance
« access »	relation de dépendance
« data type »	symbole général des classificateurs (un rectangle)

« enumeration »	symbole général des classificateurs (un rectangle)
« primitive »	symbole général des classificateurs (un rectangle)
« interface »	symbole général des classificateurs (un rectangle)
« merge »	relation de dépendance
« permit »	relation de dépendance
« realize »	relation de dépendance
« substitute »	relation de dépendance
« use »	relation de dépendance
« import »	relation de dépendance
« access »	relation de dépendance
« data type »	class symbol
« enumeration »	class symbol
« primitive »	class symbol
« merge »	dependency symbol
« permit »	dependency symbol
« realize »	dependency symbol
« substitute »	dependency symbol
« use »	dependency symbol
« interface »	class symbol
« public »	liste d'attributs ou d'opérations
« protected »	liste d'attributs ou d'opérations
« private »	liste d'attributs ou d'opérations
« component »	class symbol
« occurrence »	dependency (collaboration occurrence)
« activity »	class symbol
« selection »	note attachée à un noeud objet
« centralBuffer »	nœud objet
« precondition »	activité, ...
« postcondition »	activité, ...
« singleExecution »	activité
« decisionInput »	dans une note attachée à un noeud de décision

Tab. B.2: Liste des mots clés définies dans UML

B.3 Contraintes Prédéfinies

{Xor}	Entre deux associations au minimum

Tab. B.3 – Liste des mots clés définies dans UML

Lexique Anglais Francais

abstract abstrait.

abstraction abstraction.

action action.

activity activité.

activity diagram diagramme d'activités.

activity edge arc d'activité.

activity group groupe d'activité.

activity parameter node nœud paramètre d'activité.

activity partition partition d'activité.

actor acteur.

aggregation agrégation.

artifact artefact.

assembly connector connecteur d'assemblage.

association class classe d'association.

attribute attribut.

behavior comportement.

behaviored classifier classificateur comportemental.

binary association association binaire.

central buffer node nœud buffer central.

choice pseudo-state pseudo-état de choix.

class classe.

class diagram diagramme de classes.

classifier classificateur.

clause clause.

collaboration collaboration.

collaboration occurrence occurrence de collaboration.

combined fragment fragment combiné.

comment commentaire.

communication diagram diagramme de communication.

communication path chemin de communication.

component composant.

component diagram diagramme de composants.

composite state état composite.

composite structure diagram diagramme de structure composite.

composition composition.

conditional node nœud conditionnel.

connection point reference référence de point de connexion.

connector connecteur.

constraint contrainte.

continuation continuation.

control flow flot d'objet.

control flow flot de contrôle.

control node nœud de contrôle.

data store node nœud de stockage de données.

data type type de données.

decision node nœud de décision.

deep history pseudo-state pseudo-état historique profond.

delegation connector connecteur de délégation.

dependency dépendance.

deployed artifact artefact déployé.

deployment déploiement.

deployment diagram diagramme de déploiement.

deployment specification spécification de déploiement.

deployment target cible de déploiement.

derive dérive.

device support d'exécution.

element élément.

element import importation d'éléments.

element properties propriétés des éléments.

encapsulated classifier classificateur encapsulé.

entry point pseudo-state pseudo-état point d'entrée .

enumeration énumération.

event occurrence occurrence d'événement.

exception handler handler d'exception.

executable node nœud exécutable.

execution environment environement d'exécution.

execution occurrence occurence d'exécution.

exit point pseudo-state pseudo-état point de sortie.

expansion node nœud d'expansion.

expansion region région d'expansion.

expression expression.

extend relationship relation d'extension.

extension point point d'extension.

final node nœud final d'activité.

final node nœud final.

final state état final.

flow final node nœud final de flot.

fork node nœud de bifurcation.

fork pseudo-state pseudo-état de bifurcation.

gate porte.

general ordering relation d'ordre général.

generalization héritage.

generalization set ensemble d'héritage.

include relationship relation d'inclusion.

initial node nœud initial.

initial pseudo-state pseudo-état initial.

input pin pin d'entrée.

instance specification spécification d'instance.

instantiate instancie.

interaction interaction.

interaction constraint contrainte d'interaction.

interaction occurrence occurrence d'interaction.

interaction overview diagram diagramme de vue d'ensemble d'interaction.

interface interface.

interruptible activity region région d'activité interruptible.

join node nœud d'union.

join pseudo-state pseudo-état d'union.

junction pseudo-state pseudo-état de jonction.

keyword mot clé.

label label.

lifeline ligne de vie.

loop node nœud de boucle.

manifestation manifestation.

merge node nœud d'interclassement.

message message.

model modèle.

multiplicity multiplicité.

named element élément nommé.

namespace espace de nommage.

namespace espace de nommage.

namespace espace de nommage.

node nœud.

object diagram diagramme d'objets.

object node nœud objet.

opérande d'interaction interaction operand.

operation opération.

output pin pin de sortie.

package paquetage.

package diagram diagramme de paquetages.

package import importation de paquetage.

package merge interclassement de paquetages.

packageable element élément empaquetable.

parameter paramètre.

parameter paramètre.

parameter paramètre.

parameter set ensemble de paramètres.

parameterable element élément pouvant jouer le rôle de paramètre générique.

parameterable element élement paramétrique.

part decomposition décomposition de partie.

permission permission.

pin pin ou broche.

port port.

power-type méta-type.

predecessor prédécesseur.

primitive type type primitif.

property possession.

property possession.

protocol state machine machine à état de description de protocole.

protocol transition transition de protocole.

pseudo-state pseudo-état.

qualified name nom qualifié.

qualifier qualificateur.

realization réalisation.

reception réception.

redefinable element élément redéfinissable.

redefinable element élément redéfinissable.

region région.

role binding affectation de rôle.

role binding affectation de rôle.

sequence diagram diagramme de séquence.

sequence-expression séquence d'expressions ?.

shallow history pseudo-state pseudo-état historique superficiel.

signal signal.

simple state état simple.

state état.

state invariant invariant d'état.

state machine diagram diagramme de machine à état.

stop stop.

structured activity node nœud d'activité structuré.

structured classifier classificateur structuré.

submachine state état sous-machine.

substitution substitution.

template binding délimitation d'éléments génériques.

template parameter paramètre générique.

template signature signature de généricité.

templateable element élément supportant la généricité.

templateable element éléments supportant la généricité.

terminate pseudo-state pseudo-état de terminaison.

timing diagram diagramme de timing.

transition transition.

usage utilisation.

use cas diagram diagramme des cas d'utilisation.

use case cas d'utilisation.

value pin pin de valeur.

value specification spécification de valeur.

variable variable.

Bibliographie

[1] Brian LYONS Hans-Erik ERIKSON, Magnus PENKER and David FADO. *UML 2 toolkit*. Whiley Publishing, 2004. OMG press.

[2] Bogumila Hnatkowska, Zbigniew Huzar, Ludwik Kuzniarz, and Lech Tuzinkiewicz. *A systematic approach to consistency within UML based software development process*. In *Blekinge Institute of Technology, Research Report*, pages 16–29 in [3].

[3] Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, and Zbigniew Huzar, editors. *Blekinge Institute of Technology, Research Report 2002 :06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*.

[4] Jean-Claude Royer Pascal André, Annya Romanczuk and Aline Vasconcelos. An algebraic view of UML class diagrams. pages 261–276, January 2000. In H. Sahraoui C. Dony, editor, Acte de la conférence LMO'2000 ISBN 2-6462-0093-7.

[5] Jean Louis Sourrouille and Guy Caplat. Checking UML model consistency. In Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, and Zbigniew Huzar, editors, *Blekinge Institute of Technology, Research Report 2002 :06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, pages 1–15. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.

[6] Grady BOOCH, James RUMBAUGH, and Ivar JACOBSON. *Le guide de l'utilisateur UML*. Collection Technologies objet/Référence. Eyrolles, février 2000. ISBN 2-212-09103-6.

[7] OMG. *UML 2.0 Superstructure Specification*, Septembre 2003.

[8] OMG. *Unified Modeling Language Specification, version 1.5*, mars 2003. [Consulté le 14/04/2003]. Disponible sur Internet. URL : <http ://www.omg.org/uml/>.

Index