

N° d'ordre : 833

THÈSE

présentée à

L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE TOULOUSE

pour l'obtention du titre de

DOCTEUR

de l'Institut National des Sciences Appliquées de Toulouse

Spécialité : Informatique

par

Jean-Pierre SEUMA VIDAL

LABORATOIRE D'ÉTUDE DES SYSTÈMES INFORMATIQUES ET AUTOMATIQUES
École Doctorale Systèmes

Titre de la thèse :

MAÎTRISE DE LA COHÉRENCE DES MODÈLES UML D'APPLICATIONS CRITIQUES

**APPROCHE PAR L'ANALYSE DES RISQUES
LIÉS AU LANGAGE UML**

Soutenue le 19 Juin 2006, devant le jury :

Président :	M. Christian PERCEBOIS	Professeur, UPS Toulouse
Rapporteurs :	M. Franck BARBIER	Professeur, Université de Pau
	M. Jean-Louis SOURROUILLE	Professeur, INSA de Lyon
Examineurs :	<u>M. Gilles MOTET</u>	Professeur, INSA Toulouse
	M. Pierre DE SAQUI-SANNES	Professeur, ENSICA
	M. Christian PERCEBOIS	Professeur, UPS Toulouse
Membre invité :	M. Thierry LE SAUX	Thales Avionics

N° d'ordre : 833

THÈSE

présentée à

L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE TOULOUSE

pour l'obtention du titre de

DOCTEUR

de l'Institut National des Sciences Appliquées de Toulouse

Spécialité : Informatique

par

Jean-Pierre SEUMA VIDAL

LABORATOIRE D'ÉTUDE DES SYSTÈMES INFORMATIQUES ET AUTOMATIQUES
École Doctorale Systèmes

Titre de la thèse :

MAÎTRISE DE LA COHÉRENCE DES MODÈLES UML D'APPLICATIONS CRITIQUES APPROCHE PAR L'ANALYSE DES RISQUES LIÉS AU LANGAGE UML

Soutenue le 19 Juin 2006, devant le jury :

Président :	M. Christian PERCEBOIS	Professeur, UPS Toulouse
Rapporteurs :	M. Franck BARBIER	Professeur, Université de Pau
	M. Jean-Louis SOURROUILLE	Professeur, INSA de Lyon
Examineurs :	<u>M. Gilles MOTET</u>	Professeur, INSA Toulouse
	M. Pierre DE SAQUI-SANNES	Professeur, ENSICA
	M. Christian PERCEBOIS	Professeur, UPS Toulouse
Membre invité :	M. Thierry LE SAUX	Thales Avionics

« Il y a une montagne unique que nous gravissons les uns et les autres par des sentiers différents, avec l'espoir de nous retrouver les uns et les autres au sommet, dans la lumière, au-dessus des nuages »
Professeur Théodore MONOD

« Il est vrai que la misère détruit, mais si nous partageons et si nous voulons vivre en démocratie, aussi bien politiquement que philosophiquement, la pauvreté seule nous sauvera, et l'esprit de pauvreté est peut être aujourd'hui le fondement de la morale »
Michel SERRES

Remerciements

Le travail présenté dans ce manuscrit résulte d'échanges et de collaborations avec des personnes qui ont su apporter leur éclairage sur ce chemin. Sans leur aide, ce travail n'aurait tout simplement pu être réalisé.

Tout d'abord, je remercie M. Franck Barbier et M. Jean-Louis Sourrouille pour l'intérêt qu'ils ont porté à ma thèse en acceptant d'être rapporteurs. C'est à double titre que j'adresse mes remerciements à M. Franck Barbier, d'une part pour ses remarques qui ont permis de préciser certains aspects techniques et de donner plus de cohérence à l'ensemble de ce mémoire et d'autre part pour avoir participé à nos interviews. Je remercie également M. Jean-Louis Sourrouille pour avoir accepté d'être rapporteur de ma thèse et pour m'avoir donné l'occasion de présenter nos travaux dans le cadre de journées de travail.

Je tiens à témoigner ma sincère reconnaissance à M. Gilles Motet, mon directeur de recherche, pour m'avoir fait confiance dès le début de la thèse, pour sa disponibilité, le sérieux de ses relectures, et son indéfectible soutien lors du "travail de bénédictin" dans lequel nous nous étions lancés. J'ai beaucoup apprécié la grande liberté qui m'a été donnée notamment lors de l'exploration de la thématique des problèmes de cohérence, dans un domaine qui n'était pas initialement notre domaine de prédilection.

Je tiens à remercier M. Christian Percebois pour avoir accepté de participer et de présider au jury de thèse. J'adresse aussi mes remerciements à M. Pierre de Saqui-Sannes pour avoir accepté d'être membre de mon jury. Leurs remarques furent grandement appréciées.

Je remercie M. Thierry Le Saux, industriel à Thalès Avionics pour nous avoir accueilli à maintes reprises pour participer à nos interviews et pour nous avoir permis d'accéder à des modèles UML industriels.

Je voudrais remercier M. Jérémie Guiochet dont les précédents travaux de recherche ont été une base solide sur laquelle repose le travail présenté dans ce manuscrit.

J'adresse un remerciement spécial à mes collègues directs de travail Hugues et Roberto, avec lesquels nous avons partagé de nombreuses heures de travail et transpiré ensemble sur le métamodèle d'UML et pour les corrections qu'ils ont apporté suite aux relectures de certaines parties de ce manuscrit. Que les stagiaires qui ont aussi apporté leur pierre à l'édifice soient ici remerciés.

Je remercie M^{me} Danièle Fournier-Prunaret, directrice du LESIA pour son accueil

au sein du laboratoire. Je remercie également tous les doctorants du laboratoire pour faire de ce lieu un lieu accueillant, notamment Samuel et Stéphanie et tous les amis mexicains et leurs initiatives culinaires. Comment omettre de remercier M. Pascal Acco pour ses précieux conseils de grand frère des thésards du LESIA et l'aide sur de nombreux soucis "latex-iques". Je garderai en mémoire ses entrées dans le bureau pour le moins communicatives.

Je remercie également les collègues du DGEI pour la qualité des échanges professionnels ou personnels, et notamment le travail de secrétariat et de maintenance informatique toujours accompli avec bienveillance et professionnalisme. J'attends cette rando avec mes aînés du DGEI, Christophe, Fofu et Terlys pour faire vivre ces lieux dont nous avons tant parlé. Tant pis pour les genoux.

Je demande pardon aux personnes que j'aurais oublié et dont la gentillesse discrète m'aurait échappé. Qu'elles aient au moins l'honnêteté de se reconnaître maintenant et de recevoir mes discrets remerciements, l'essentiel restant invisible pour les yeux.

Enfin, je voudrais traduire ici avec émotion la reconnaissance à celles et ceux qui m'ont accompagné durant tout ce temps, qui m'ont montré combien chaque personne a de la valeur, et qui me font grandir jour après jour : ma famille, mes amis, les personnes de la rue et Alice.

Je dédie ce travail à mes parents qui m'ont donné ce cadeau d'être en vie.

Table des matières

1	Risques d'incohérences UML	17
1.1	Cadre de l'étude	17
1.1.1	Concept de risque	17
1.1.2	Risques logiciels	18
1.1.3	Langage UML	19
1.2	Types de risques des modèles UML	21
1.2.1	Événements dommageables et cohérence	21
1.2.2	Incohérence et UML	22
1.3	Vue d'ensemble des travaux	24
2	État de l'art relatif aux incohérences	27
2.1	Détection d'incohérences des modèles UML	27
2.2	Études actuelles	28
2.2.1	Type de modèle étudié	28
2.2.2	Éléments du modèle considérés	29
2.2.3	Type de propriété utilisée	30
2.2.4	Technique de détection	33
2.3	Nécessité d'une étude systématique	36
3	Identification des règles de cohérence UML	37
3.1	Méthode d'identification	37
3.1.1	Objectifs	37
3.1.2	Informations associées à chaque règle	38
3.1.3	Structure du document	40
3.2	Résultats et analyse	42
3.2.1	Illustration de chaque type de règle	42
3.2.1.1	Élément	42
3.2.1.2	Relation	45
3.2.1.3	Diagramme	47
3.2.1.4	Inter-diagramme	48
3.2.2	Commentaires sur les résultats	50
3.2.2.1	Organisation du document	50

3.2.2.2	Bilan quantitatif	51
3.2.2.3	Liens des règles entre elles	51
3.2.2.4	Commentaires concernant la capacité de détection	52
3.2.3	Limites de l'étude	52
3.2.4	Conclusion	54
4	Estimation des risques d'incohérences UML	57
4.1	Besoins, intérêts et moyens	57
4.1.1	Présence d'incohérences	58
4.1.2	Insuffisance des outils de détection	58
4.1.3	Moyens d'estimation	60
4.1.3.1	Deux paramètres	60
4.1.3.2	Critères et métriques	60
4.1.3.3	Méthodes d'estimation	61
4.1.4	Application à l'estimation des risques de chaque construction	62
4.2	Estimation des risques par interview d'experts	63
4.2.1	Principes	63
4.2.2	Critères et métriques	65
4.2.2.1	Gravité	65
4.2.2.2	Degré de vraisemblance	66
4.2.3	Processus d'estimation	67
4.2.3.1	Rôles	67
4.2.3.2	Portée de l'interview	67
4.2.4	Résultats	68
4.2.4.1	Résultats spécifiques à la métarelacion Association	68
4.2.4.2	Résultats spécifiques au métaélément Property	69
4.2.4.3	Estimation globale pour les diagrammes de classes	70
4.2.5	Commentaires	71
4.3	Estimation quantitative par analyse de modèles	72
4.3.1	Critères et métriques	73
4.3.2	Processus d'estimation	74
4.3.3	Résultats	75
4.4	Conclusion	76
5	Traitement des risques d'incohérences UML	77
5.1	Traitement du risque	77
5.1.1	Besoins	77
5.1.2	Classes de moyens	79
5.1.2.1	Quatre grandes approches de traitement	79
5.1.2.2	L'évitement des risques	80
5.1.2.3	L'optimisation des risques	81

5.2	Application à UML	82
5.2.1	Approches de traitement des incohérences UML par optimisation	82
5.2.1.1	Des guides de modélisation	83
5.2.1.2	Des guides liés au processus de développement	84
5.2.1.3	Des patrons de conception	85
5.2.1.4	Un cadre de gestion des incohérences	86
5.2.2	Approche proposée	87
5.3	Prévention	88
5.3.1	Principes	88
5.3.2	Ne pas utiliser une notation	89
5.3.2.1	Exemple des <i>pins</i>	89
5.3.2.2	Exemple de la fusion des nœuds de décision et de fusion	89
5.3.3	Forcer l'expression d'une information supplémentaire	90
5.3.4	Adopter des conventions de nommage	95
5.3.4.1	Exemple de l'alias optionnel de la relation d'importation d'élément	96
5.3.4.2	Exemple des portes	96
5.3.5	Veiller à l'agencement graphique des éléments dans un diagramme	97
5.3.6	Appliquer des guides de modélisation	98
5.3.6.1	Nœud final d'activité ou nœud final de flot	98
5.3.6.2	De l'usage des conditions de garde dans les diagrammes d'activités	99
5.4	Protection	100
5.4.1	Principes	100
5.4.2	Rajouter une information	101
5.4.3	Appliquer des guides de modélisation	103
5.5	Conclusion	105
6	Synthèse, contributions majeures et perspectives	107
6.1	Synthèse	107
6.2	Contributions majeures et perspectives	108
A	Interviews	111
A.1	Feuille-type d'estimation d'une règle par interview	111
A.2	Règles utilisées pour l'interview	113
A.2.1	Liste des concepts de niveau méta-modèle considérés	113
A.2.2	Enoncé textuel des règles	113
A.2.2.1	Éléments	113
A.2.2.2	Relations	115
	Lexique Anglais Français	118

Index	120
Bibliographie	122

Table des figures

1.1	Architecture en quatre couches	20
1.2	Exemple de la hiérarchie de méta-modélisation en 4 couches	20
2.1	Modèle de faute et incohérence	32
2.2	Approche transformationnelle	34
3.1	Tableau synthétique de l'origine des règles	39
3.2	Relation entre les métaclasses <code>Package</code> and <code>PackageableElement</code>	43
3.3	Arbre de spécialisation de la métaclasse <code>PackageableElement</code>	43
3.4	Composant candidat pour en substituer un autre	44
3.5	Incohérence dans la définition des multiplicités dans un diagramme de classes	45
3.6	Exemple de manifestation	45
3.7	Métamodèle impliquant la relation de manifestation	46
3.8	Incohérence dans les arcs d'activités avec des connecteurs labellisés	46
3.9	Exemple d'une importation d'élément avec alias incohérente	47
3.10	Diagramme d'activité avec événement activateur sur le premier arc	48
3.11	Incohérence dans un diagramme d'activités	48
3.12	Incohérence entre diagramme d'activités et diagramme de classes	49
3.13	Cohérence entre diagramme de classes et diagramme de structures composites	50
3.14	Tableau de répartition des règles par niveau d'abstraction	51
3.15	Types de dommages	55
4.1	Résultats synthétiques des taux de détection d'erreur	59
4.2	Pourcentage de détection par moyen de détection	60
4.3	Estimation du risque de la présence d'incohérences associées à une construction UML	62
4.4	Acceptabilité du risque	62
4.5	Estimation du risque pour la construction <code>Association</code> - Un point de vue industriel	68
4.6	Estimation du risque pour la construction <code>Association</code> - Un point de vue universitaire	68

4.7	Estimation du risque pour la construction Property - Un point de vue industriel	69
4.8	Estimation du risque pour la construction Property - Un point de vue universitaire	69
4.9	Estimation globale du risque pour les diagrammes de classes selon le critère de la difficulté de détection - Un point de vue industriel	70
4.10	Estimation globale du risque pour les diagrammes de classes selon le critère de la difficulté de détection - Un point de vue universitaire	70
4.11	Estimation globale du risque pour les diagrammes de classes selon le critère de l'impact sur le code - Un point de vue industriel	71
4.12	Estimation globale du risque pour les diagrammes de classes selon le critère de l'impact sur le code - Un point de vue universitaire	71
4.13	Critères et métriques pour la gravité et le degré de vraisemblance	73
4.14	Informations quantitatives relatives aux modèles considérés	75
4.15	Graphe d'acceptabilité du risque pour les contraintes d'interaction	75
5.1	Les deux façons de réduire le niveau de risque	78
5.2	Risque et niveaux de criticité	78
5.3	Message dans le diagramme de séquence	82
5.4	Opération dans le diagramme de classes	82
5.5	Association représentée plutôt que le type d'un attribut	83
5.6	Attribut de classe typé par une classe en association	83
5.7	Représentations équivalentes de flux d'objets	89
5.8	Noeud de fusion, de décision, et les 2 combinés	90
5.9	Un composant avec une interface requise, spécifiée par dépendance d'usage	91
5.10	Un composant avec ses interfaces spécifiées par l'intermédiaire de ports .	91
5.11	La collaboration Sale	92
5.12	La collaboration BrokeredSale	92
5.13	La collaboration Sale utilisée par le classificateur BrokeredSale	93
5.14	Une utilisation de collaboration reliée à un classificateur avec le mot clé « represents »	93
5.15	Éléments empaquetable sans visibilité	94
5.16	Éléments empaquetable avec visibilité	94
5.17	Classe abstraite non explicite	94
5.18	Classe abstraite explicite	94
5.19	Fragment optionnel vide	95
5.20	Fragment optionnel non vide	95
5.21	Importation d'élément sans alias	96
5.22	Importation d'élément avec alias	96
5.23	Porte nommée d'une spécification d'occurrence	97
5.24	Arbre de généralisation non agencé	98

5.25	Arbre de généralisation réagencé par ensemble de généralisation	98
5.26	Diagramme d'activités avec deux nœuds finaux de flots	99
5.27	Arc d'activité avec condition de garde sortant d'un nœud de bifurcation (<i>fork node</i>), non protégé	99
5.28	Arc d'activité avec condition de garde sortant d'un nœud de bifurcation (<i>fork node</i>), protégé par un nœud de décision et un nœud de fusion . . .	100
5.29	Classe Alert et sa machine à états	102
5.30	Dépendance d'abstraction de stéréotype « derive »	103
5.31	Représentation d'un port comportemental	104

Avant-propos

Il est des activités humaines qui procurent à la personne une certaine conscience de sa finitude, de sa petitesse, mais en même temps de faire partie d'un tout, cohérent, qui nous dépasse. La randonnée en montagne en est une. Si les équipements de protection contre les intempéries et de guidage tels que GPS ne cessent d'évoluer, si les bulletins de prévision météorologiques ne cessent de s'affiner, la pratique de la randonnée n'en reste pas moins une activité à risque dans un milieu, qui peut se montrer aussi bien accueillant qu'hostile. Bien souvent, certaines habitudes comme celles de rester chez soi un jour prévu d'orage, ou de se fier plutôt qu'à un GPS mal calibré à une vieille carte résultant de l'ajout continu d'informations au gré des randonnées passées, permettent d'éviter bien des désagréments et la prise de risques inutiles.

Certains modèles humains permettent de préparer avec un certain degré de confiance une sortie. Les cartes, qui mettent sur papier une représentation de la topographie, ainsi que les modèles d'évolution du temps fournis par les météorologues donnent à tout candidat à une petite échappée montagnarde des données pour prendre la décision de sortir ou de rester chez soi. La décision étant prise de rejoindre le pied de montagne pour la gravir, un certain nombre de mesures de prévention doivent être suivies avant d'entamer la marche, comme celle d'éviter de partir seul, ou celle de prévenir une personne restant en bas de l'itinéraire prévu, etc. Mais, une fois le sac sur le dos, qui n'est pas resté perplexe au moins une fois devant une carte avec des indications manquantes ou erronées par rapport au terrain ? La carte n'est pas le territoire, répondront certains. Cependant la carte est indispensable pour qui veut s'aventurer hors des sentiers battus et qui veut revenir au point de départ, par un chemin qui "marche".

Tout(e) montagnard(e) a au moins temporairement remis(e) en cause la pratique de son activité à l'occasion d'une mauvaise décision, qu'elle soit au moment de partir, au cours de la marche, ou au moment de revenir. La balance entre les conditions météorologiques, sa condition physique, son désir de dépassement et la prise de risque, n'est pas chose aisée. Elle l'est encore moins lorsqu'il s'agit d'une activité menée en groupe où chaque participant aspire à différents degrés de prise de risque. Cependant le risque de la perte d'une vie humaine restant inacceptable, il est de la responsabilité de chacun d'être capable d'identifier et d'estimer les risques impliqués par un parcours et de prévoir des mesures de repli et d'itinéraires bis pour pallier à un événement imprévu comme une cheville foulée.

La montagne possède différents aspects selon la saison. Nous commencerons par l'appréhender par la représentation qui en est faite sur papier. La question qui se pose alors est : comment rendre la carte aussi précise que possible et gérer les incomplétudes et les erreurs qu'elle comporte ? Mais avant tout, quel est le propos de cet avant-propos ?

L'évocation de cette activité à risque n'a pour autre but que d'introduire par analogie le sujet de l'étude présentée dans ce manuscrit. Nous avons emprunté le long sentier qui permet à un informaticien d'avoir une vue d'ensemble de cette montagne qui est le langage UML, dans la variété de ses diagrammes et de ses vues. Nous voulons juste en avant-propos à ce manuscrit faire un parallèle entre cette activité et l'activité de développement de logiciels, qui *a priori*, n'auraient que peu de similitudes.

Nous allons dans cette thèse nous situer dans le cadre d'applications logicielles qui sont développées à l'aide du langage de modélisation informatique UML. Ce langage offre des constructions graphiques pouvant être assemblées afin de constituer des diagrammes, dont l'assemblage constitue un modèle complet. Le langage UML permet la description de systèmes informatiques complexes dont la sécurité est parfois essentielle et dont l'assurance d'un bon fonctionnement est une priorité vitale. Afin de maîtriser la complexité des choses à représenter, et d'arriver à trouver une solution informatique qui en découle et qui soit correcte, la conception de telles applications nécessite l'emploi de modèles tout le long d'un processus de développement.

Mais les risques encourus dans l'activité de développement de modèles de logiciels écrits avec UML ne sont pas nuls. En particulier nous nous sommes intéressés au risque de la présence d'incohérences dans un modèle. En effet les diagrammes peuvent comporter des fautes pouvant mener à des incohérences entre diagrammes ou au sein même d'un seul diagramme. Un des principaux problèmes qui se pose alors est la gestion ou la maîtrise de la cohérence des modèles. Nous traitons la question de la gestion des risques liés à l'utilisation d'UML, en ayant comme objectif et comme "boussole" la maîtrise de cette cohérence.

Ce manuscrit va présenter la démarche générale suivie de management des risques en abordant d'abord les concepts introductifs aux notions de risque et d'incohérence. L'identification, puis l'estimation des risques d'incohérences UML seront présentées ainsi qu'un aperçu des techniques de prévention et de protection à mettre en œuvre pour réduire les risques.

Un lexique Anglais/Français regroupe en fin de document l'ensemble des termes UML employés. Les notes de bas de page présentes dans le texte correspondent à des termes anglais ou à des phrases en anglais tirées de la spécification d'UML.

Chapitre 1

Risques d'incohérences UML

1.1 Cadre de l'étude

1.1.1 Concept de risque

Les systèmes naturels ou technologiques avec lesquels nous interagissons sont à l'origine de nombreux risques : risques vis-à-vis de l'équilibre environnemental, risques vis-à-vis de l'homme, risques vis-à-vis de l'économie, etc. Concernant les systèmes technologiques, la « prise de risque » est souvent volontaire car en général associée à l'espérance d'un bénéfice suffisamment important pour la justifier : c'est dans l'objectif de tirer profit d'un système particulier que l'on *choisit* de prendre certains risques et que l'on doit donc les gérer afin de conserver les bénéfices escomptés tout en réduisant au maximum les occurrences des dommages potentiels. La gestion des risques est donc une activité inhérente et commune à un grand nombre de domaines d'activités générales (par exemple le domaine industriel) ou spécifiques (aéronautique) ou de domaines technologiques (chimie, informatique, etc.).

Ainsi, dans le domaine informatique en particulier, les technologies logicielles comportent chacune des caractéristiques conduisant à des gains et à des pertes potentielles (voir section 1.1.2).

La sécurité ¹ d'un système peut se définir par l'absence de dommages inacceptables. Un dommage peut être qualifié par sa nature (par exemple brûlure, écrasement, coupure, etc. s'il affecte l'être humain) et mesuré par sa gravité et sa probabilité d'occurrence. Afin de rendre compte de l'interaction entre la gravité et l'occurrence d'un dommage, la notion de risque a été introduite. La définition donnée par l'ISO/CEI Guide 51 (1999)[1] du risque est la suivante :

$$\text{Risque} = \text{Combinaison de la probabilité d'un dommage et de sa gravité}$$

¹au sens *safety*; il s'agit d'un des attributs de la sûreté de fonctionnement. A ne pas confondre avec la sécurité-confidentialité (*security* en anglais)

1.1.2 Risques logiciels

Le risque dans les applications logicielles peut tout d'abord être vu en considérant les événements dommageables que sont les défaillances à l'exécution. Les travaux de recherche actuels, tout comme les activités de développement, relatives aux risques inhérents aux applications logicielles, traitent essentiellement des événements redoutés que sont les défaillances potentielles de ces applications. Ces défaillances sont considérées comme issues de la présence de fautes de conception dans les programmes. Ces risques sont traités en agissant sur le processus de développement ou sur les résultats de chaque étape de ce processus (documents, modèles ou programmes). Selon le niveau de criticité de l'application développée, c'est-à-dire selon le niveau acceptable de gravité des conséquences induites par l'occurrence d'une défaillance de cette application, des exigences plus ou moins importantes sont imposées aux concepteurs. Les étapes du processus de développement peuvent ainsi être contraintes par l'application de guides par exemple. Ces exigences peuvent aussi contraindre les actions qui doivent être menées sur les résultats des étapes : évaluation des risques de défaillance, techniques de vérification pour signaler la présence de fautes, mise en œuvre de redondance pour inhiber ou réduire les effets des défaillances de composants, etc.

Si les fautes présentes dans les logiciels peuvent certainement être induites par des erreurs survenant lors du processus de conception, elles peuvent être également attachées à la technologie logicielle employée. Or, cet aspect est assez peu étudié alors que les autres technologies sont fréquemment l'objet d'études : composants chimiques, matériaux de construction, etc. [56]. Les risques liés aux technologies logicielles qui nous intéressent sont les risques dont les dommages sont les fautes dans les programmes et dans les modèles.

La notion de risque a été précisée plus récemment en la sortant du seul contexte de la sécurité exprimée dans le guide 51 [1] de l'ISO. Ainsi le guide 73 de l'ISO [2] définit le risque comme comportant deux éléments :

- un événement qui identifie l'occurrence potentielle d'un ensemble particulier de circonstances,
- une conséquence qui qualifie son impact sur une cible.

La cible peut être une personne, un bien, l'environnement, mais aussi un modèle informatique, etc. Il est important de noter que la conséquence (qui généralise la notion de dommages causés) sur la cible n'est pas toujours grave et que l'occurrence de l'événement n'est que potentielle et non induite assurément par des causes (source du risque). Les événements dits "dommageables" sont alors particulièrement intéressants à étudier lorsqu'on est concerné par le domaine de la sécurité.

Pour gérer les risques, il convient en premier lieu de chercher à énumérer et à décrire les risques possibles ainsi que leurs sources.

1.1.3 Langage UML

Les langages graphiques fournissent un moyen prometteur pour maîtriser la complexité des logiciels. De ce fait, leur utilisation réduit la présence de fautes dans les applications modélisées et constitue donc un moyen appréciable pour augmenter la sécurité des logiciels développés. Le langage UML 2.0 [60] lui-même repose sur un formalisme bien défini. Nous allons rappeler ici brièvement l'architecture du langage et le contexte dans lequel nous plaçons nos études.

L'architecture du langage Le langage UML s'inscrit dans une architecture à quatre couches ou niveaux de méta-modélisation qui sont : objets utilisateurs (ou instances exécutables, *run-time instances* en anglais, niveau dit M0), modèle (niveau M1), métamodèle (niveau M2) et méta-métamodèle (niveau M3). Nous allons détailler ce découpage avec des notions volontairement simplifiées reposant sur les différents niveaux.

- ▷ La couche **méta-métamodèle** a pour rôle de définir le langage pour spécifier un métamodèle. Pour le langage UML (langage de niveau M2), son métamodèle est le langage MOF [57](niveau M3). Il fournit par exemple le concept de classe. Il permet de généraliser les constructions d'UML que sont les attributs, classes et instances grâce à la notion de classe du niveau M3.
- ▷ Un **métamodèle** est une instance d'un méta-métamodèle. Le rôle de la couche "métamodèle" est de définir le langage pour spécifier un modèle et c'est à ce niveau là que les notions comme celles de classe, attribut, opération, sont présentées dans la superstructure de la spécification d'UML [59]. Ces notions sont des instances du concept de classe de niveau M3, comme présenté à la figure 1.2 tirée de l'infrastructure de la version 2.0 de la spécification d'UML [58]².
- ▷ La couche **modèle** est une instance du métamodèle. Le principal rôle du niveau "modèle" est de définir un langage qui décrit un domaine d'information. C'est à ce niveau que seront définis les noms des classes ou d'instances de classes, attributs et opérations propres au système modélisé. Des exemples d'objets du niveau modèle sont : Fenêtre, couleurFenêtre (ou bleu pour une instance de l'attribut couleur de la classe Fenêtre) et afficherFenêtre().
- ▷ Les **objets utilisateurs** sont des instances exécutables d'un modèle. Le rôle du niveau "données utilisateurs" est de décrire un domaine d'information spécifique. Des exemples d'objets du niveau objets utilisateurs sont : <Fenêtre3> (instance de Fenêtre), un appel de l'opération instance d'afficherFenêtre().

²Signalons que la figure comporte une erreur, car le métatype **Instance** n'existe plus et a été remplacé par le métatype **InstanceSpecification**; Cependant la dernière version disponible de l'infrastructure [63] présente la même erreur, ce qui illustre les erreurs qui peuvent se propager d'une version à une autre.

et concrètes ont une correspondance sous forme d'une primitive abstraite, comme par exemple la classe (qui est représentée par la boîte), et la relation d'association (qui est représentée par le trait plein). Nous définirons le terme "construction de base" pour désigner cet ensemble des éléments et des relations, ces primitives abstraites, qui constituent la notion d'élément UML au niveau métamodèle.

1.2 Types de risques des modèles UML

Quels types de risques, et tout d'abord d'événements dommageables sont rencontrés et probables dans les modèles UML ? Pour répondre à cette question, il faut d'abord présenter quelques notions concernant la modélisation cognitive en nous référant à [68] et [24].

Un modèle est une représentation d'un système dans un formalisme et toute représentation licite dans ce formalisme possède une interprétation basée sur la sémantique de ce formalisme. Le formalisme considéré ici est le langage UML, dont la sémantique n'est pas entièrement traduite en des contraintes formelles et il n'existe pas de vérificateur automatique qui détecterait si toutes les "briques de base" d'un modèle sont conformes à la sémantique du langage. De plus, certaines contraintes sont induites par la sémantique d'autres domaines (contraintes du processus de modélisation, contraintes du domaine applicatif, contraintes spécifiques à la plate-forme ou à l'implantation).

Un modèle n'est cohérent que lorsqu'il est conforme à la sémantique de tous les domaines impliqués dans le processus de développement. Une affirmation communément admise est que, sauf à la fin du processus, les modèles sont incomplets et de ce fait, souvent incohérents.

1.2.1 Événements dommageables et cohérence

Dans notre cadre de gestion des risques liés à la technologie UML, on peut alors distinguer deux classes principales d'événements dommageables :

- l'*incomplétude* qui est l'absence de l'information attendue, et
- l'*incohérence* qui concerne les conflits entre des informations.

[45] signale que l'affectation d'un de ces qualificatifs est souvent due à une interprétation de l'origine de la faute. Par exemple, si une classe A appelle une méthode M d'une classe B, et si B ne contient pas de déclaration de cette méthode, la cause peut être une omission de cette déclaration (incomplétude) ou une faute à l'appel de la méthode (incohérence). En pratique, selon [68], la nature de l'événement dommageable (incomplétude ou incohérence) est une décision prise par l'utilisateur.

L'incomplétude dans un modèle est révélée lorsqu'une information est manquante dans le modèle, et des données, par exemple des multiplicités sur des fins d'association, devraient être rajoutées dans le modèle.

On peut mentionner une taxonomie plus générale, donnée par [8] qui distingue 5 types de fautes dans les produits logiciels détectés lors d'inspections de documents de spécification et de définition d'exigences. Dans [75], ces mêmes fautes sont présentées dans le cadre plus précis de la conception orientée objet et elles peuvent être révélées par un exercice de détection manuelle de fautes que l'on qualifie alors par l'une de ces catégories précises :

- *l'omission* : un ou plusieurs diagrammes de conception qui devraient contenir un concept issu des exigences générales ou du cahier des charges ne contiennent pas la représentation de ce concept.
- *un fait incorrect* : un diagramme de conception contient une mauvaise représentation d'un concept décrit dans les exigences générales ou dans le cahier des charges.
- *une incohérence* : la représentation d'un concept dans un diagramme de conception est en désaccord avec la représentation du même concept dans le même ou un autre diagramme.
- *une ambiguïté* : la représentation d'un concept dans une conception n'est pas claire, et pourrait causer une mauvaise interprétation ou compréhension du sens de ce concept par un utilisateur du document (développeur, concepteur, etc.).
- *une information étrangère* : la conception inclut une information qui, bien que peut-être vraie, ne s'applique pas à ce domaine et ne devrait pas être incluse dans la conception.

[65] reprend cette classification pour regrouper les fautes de type “omission” et “information étrangère” dans la catégorie des fautes syntaxiques, puis les fautes de type “faits incorrects” et “ambiguïté” dans les fautes sémantiques.

La notion d'incohérence à laquelle nous nous intéressons plus particulièrement correspond toujours à un désaccord entre une représentation X d'un concept et une représentation Y du même concept. On parle alors d'incohérence entre 2 représentations.

Le concept affecté par l'incohérence peut appartenir à l'application. Par exemple, l'état d'une alimentation électrique d'un moteur est effective (bouton *ON*) et l'état du fonctionnement du moteur est arrêt. La détection d'une telle incohérence suppose alors une connaissance de la sémantique de l'application. Nous n'aborderons pas ce point de vue. Nous ne considérons uniquement que les concepts offerts par le langage UML (classe, attribut, diagramme, etc.) en cherchant à révéler les incohérences dans leurs usages séparés ou conjoints dans les applications.

1.2.2 Incohérence et UML

Avant de préciser le concept d'incohérence dans le cadre d'UML, il est bon de partir de celui de faute et d'erreur.

Le constat qu'un artefact ou modèle UML est incohérent peut être issu de l'observation :

- d'informations structurelles de l'artefact (par exemple, une classe possède deux

- opérations de même signature) identifiant alors une faute, ou
- d’informations comportementales de l’artefact (par exemple, un diagramme de machine à états qui n’est pas vivant) exprimant dans ce cas une erreur [46].

Exprimer une incohérence associée à UML revient donc à définir des “modèles de fautes” ou des “modèles d’erreurs” (au niveau M2) dont une incohérence associée à un modèle UML est une instance (une faute ou une erreur) au niveau M1 [34].

Un modèle de fautes définit un ensemble de fautes caractérisées par des propriétés structurelles du modèle (on parle aussi de propriétés syntaxiques). Ces propriétés sont nommées *well-formedness rules*.

Le concept de modèle d’erreurs a d’abord été introduit en mécanique puis en électronique. Un modèle d’erreur décrit un modèle comportemental de défaillance qui s’applique à tous les composants d’une certaine nature comme par exemple le blocage d’une vanne ou le collage à 0 d’une sortie d’une porte logique. La cause de l’erreur n’est pas précisée, l’objectif étant de fournir une caractérisation du constat de la défaillance. On parle parfois de “mode de défaillance”.

Par exemple, pour un programme écrit en langage Ada, la propriété « la valeur d’une expression affectée à une variable doit appartenir à l’ensemble des valeurs défini par le type de la variable » exprime une propriété comportementale car liée à l’exécution des programmes. C’est une propriété générique attendue et applicable à toutes les variables d’un programme. Il s’agit donc bien d’un modèle d’erreur associé à une variable qui est une construction (*feature* en anglais) du langage Ada. Une erreur particulière est levée lorsqu’une variable donnée d’un programme particulier est affectée par une valeur qui n’appartient pas à l’intervalle défini par son type.

Pour notre étude, définir un modèle de fautes ou d’erreurs nécessite donc d’exprimer une propriété exigée d’une construction ou d’un ensemble de constructions d’UML. La violation de cette propriété dans un artefact particulier utilisant cette (ou ces) construction(s) étant inacceptable.

La propriété est syntaxique ou structurelle dans le cas des modèles de fautes. Elle est sémantique ou comportementale dans le cas des modèles d’erreurs.

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.

Voici un exemple qui illustre la différence de ces sémantiques en reprenant l’exemple 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.

Nous nous intéressons donc aux règles de cohérence de l'emploi d'une construction du langage UML et non à la fonction qu'elle remplit dans une modélisation.

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

<i>Incohérence</i>	<i>Une incohérence est la violation d'une propriété associée au langage UML qui doit être respectée par tout modèle UML.</i>
--------------------	--

Une incohérence peut être aussi vue comme la conséquence de constructions redondantes ou complémentaires incompatibles entre elles. La compatibilité entre les différentes constructions peut être exprimée par des règles de cohérence.

1.3 Vue d'ensemble des travaux

La présence d'incohérences est intrinsèque à la complexité des modèles, à la diversité des personnes intervenant pour produire ces modèles, et à leurs modifications en phase de maintenance. Il est donc indispensable de mettre en place une étude permettant de maîtriser la présence de ces incohérences. Notre approche s'est fondée sur un processus de management du risque dont les principales étapes sont :

- **l'identification** qui a pour but d'énumérer et de décrire les risques possibles ainsi que leurs sources,
- **l'estimation** des risques qui quantifie les risques identifiés,
- **l'évaluation** des risques conduisant à la hiérarchisation des risques à partir de leurs estimations mais également d'autres critères,
- **l'acceptation** des risques qui porte un jugement sur les risques hiérarchisés en fixant un seuil d'acceptabilité, et
- **le traitement** des risques fournissant des moyens permettant, par exemple, de réduire les valeurs des estimations des risques à un niveau acceptable.

Notre travail a tout d'abord consisté à identifier les types d'événements dommageables étudiés, c'est-à-dire les incohérences. Pour cela, nous avons exprimé la sémantique de vérification de chaque élément de base du langage UML, de chaque relation entre ces éléments, pour chaque diagramme décrit par le langage UML, en considérant les règles à respecter entre les diagrammes. Cette étude est fournie au chapitre 3. Elle est précédée par un état de l'art sur les travaux relatifs aux incohérences (chapitre 2).

L'approche fondée sur le risque suppose que la présence des événements dommageables que sont les incohérences n'est pas inéluctable (attribut de vraisemblance du risque) et que leurs conséquences sont plus ou moins graves (attribut d'importance ou gravité du risque). Pour ces raisons, nous étudions au chapitre 4 comment les risques

d'incohérence peuvent être estimés.

Cette estimation permet en particulier d'affecter les moyens nécessaires aux traitements de ces risques :

- le refus du risque d'incohérence en interdisant l'usage des constructions UML concernées, approche assez peu satisfaisante, et
- la réduction du risque en fournissant des moyens, par exemple des guides de modélisation.

Ces traitements du risque seront étudiés au chapitre 5.

Chapitre 2

État de l'art relatif aux incohérences

Notre objectif est d'identifier au chapitre 3, les événements dommageables que sont les incohérences dans les modèles UML. Au préalable, ce chapitre fournit un état de l'art sur les moyens d'expression et de détection de ces incohérences.

2.1 Détection d'incohérences des modèles UML

Le langage UML permet de décrire plusieurs “vues” d'un même système grâce à la notion de diagramme. Cependant ces différentes vues ne sont pas indépendantes : elles fournissent des informations généralement complémentaires mais aussi fréquemment redondantes. De ce fait, ces informations peuvent être en conflit, en étant incohérentes [27]. De plus, l'expérience montre que de très nombreuses fautes de modélisation sont perceptibles à travers la détection d'incohérences [45].

Les activités de vérification sont actuellement surtout effectuées sur le programme source (revue, etc.) et exécutable (test, etc.). Les entreprises désirent transférer une partie de cette vérification vers les modèles de spécification et de conception exprimés en UML. Les fautes sont ainsi détectées et corrigées au plus tôt. Ceci permet aussi d'augmenter l'assurance de l'absence de fautes dont la garantie est très difficile à obtenir sur des programmes sources complexes. Par ailleurs, on constate qu'un grand nombre de fautes sont introduites lors des phases initiales. En particulier, une grande part des fautes actuelles concerne des problèmes d'intégration de sous-systèmes qui relèvent des niveaux d'abstraction élevés et non du niveau de la programmation. C'est donc lors de ces phases initiales que ces fautes doivent être détectées en analysant les modèles produits.

Deux approches de la vérification sont possibles :

- vérifier des assertions formulant des propriétés propres aux fonctionnalités du système modélisé (niveau M1, cf. section 1.1.3), ou
- vérifier des propriétés intrinsèques au moyen de modélisation, à savoir UML 2.0 dans notre cas (niveau M2).

La mise en oeuvre de la première approche est rendue difficile par le fait que son

efficacité à détecter des fautes dépend de la pertinence des propriétés exprimées. En conséquence, elle dépend des compétences des ingénieurs qui les formulent. La seconde approche, qui consiste à vérifier des propriétés associées au moyen de modélisation qu'est UML, ne nécessite pas de reformuler ces propriétés pour chaque application modélisée. Elle est complémentaire à la première approche en détectant des incohérences des modèles par la violation de propriétés exprimées sur les éléments du moyen de modélisation à savoir les diagrammes d'UML 2.0 dans notre cas. Ces propriétés sont exprimées sous forme de règles de cohérence.

La détection d'incohérences entre les divers constituants des modèles est d'autant plus nécessaire que l'expression de vues multiples que permet UML conduit à augmenter le nombre de diagrammes et donc le nombre d'incohérences [27].

Dans cette partie, nous présentons tout d'abord à la section 2.2 les études existantes sur l'expression et la détection d'incohérences spécifiques au langage UML. Nous y signalons également les limites des résultats de ces études. Nous concluons à la section 2.3 sur la nécessité d'obtenir une liste exhaustive des règles de cohérence.

2.2 Études actuelles

De nombreux résultats ont déjà été proposés sur la cohérence des modèles UML. Mais ces études sont partielles. Elles sont souvent limitées à la vérification de quelques propriétés entre certains diagrammes. Cette limitation est souvent due au choix fait *a priori* d'une technique de détection.

Dans cette section nous positionnons notre étude dans le cadre des travaux de recherche déjà publiés. Pour cela nous considérons 4 critères qui sont abordés dans les 4 sous-sections suivantes : le type du modèle sur lequel les études de cohérence sont menées (section 2.2.1) ; les éléments du modèle considérés pour détecter des incohérences (section 2.2.2) ; le type de propriété utilisée pour exprimer les règles de cohérence (section 2.2.3) ; les techniques mises en oeuvre pour détecter les incohérences (section 2.2.4).

2.2.1 Type de modèle étudié

Permettant d'exprimer de multiples points de vue sur un système, UML favorise l'expression d'informations redondantes.

Cette redondance peut tout d'abord être totale : une information d'un modèle peut être déduite d'autres informations de ce modèle. Par exemple, les diagrammes de communication peuvent être déduits des diagrammes de séquence.

La redondance totale décrite précédemment est cependant très limitée. En effet, chaque vue est une spécification partielle dédiée à un aspect technique ou destinée à un type d'utilisateur. Ainsi, il n'existe pas un unique diagramme donnant une vue complète du système. Les vues partielles doivent cependant être concordantes. On ne cherche donc pas à montrer l'équivalence de divers points de vue mais la cohérence de l'ensemble. Cette

recherche de cohérence doit s'effectuer entre un grand nombre d'éléments constituant les modèles. Par exemple, les diagrammes de machines à états des classes permettent de déduire les interactions acceptables d'une classe prise individuellement. Les scénarios décrits par les diagrammes de séquence expriment les interactions entre différents objets et doivent être conformes avec les interactions acceptables en accord avec le diagramme de classes.

Dans ces études, la cohérence concerne un seul modèle (composé d'un ensemble de diagrammes) d'un système. Ce modèle est associé à une phase du développement. On parle de cohérence *intra-modèle*.

L'approche *inter-modèle* cherche à détecter des incohérences entre plusieurs modèles d'un même système. Ces modèles peuvent concerner des phases successives du développement. Dans cette approche, il s'agit de déterminer par exemple qu'un modèle d'analyse est cohérent avec un modèle de conception. [39] étudie la cohérence entre des exigences fonctionnelles écrites en langage naturel et des spécifications de conception exprimées avec UML. Ces modèles peuvent être également des versions différentes d'un modèle d'un système, spécifié comme une collection de diagrammes de classes, de diagrammes de séquence et de diagrammes d'états [71].

Nos travaux se limitent à l'approche *intra-modèle*.

2.2.2 Éléments du modèle considérés

Pour un modèle donné (approche *intra-modèle* considérée), les incohérences sont généralement détectées sur les utilisations conjointes d'éléments d'un unique diagramme ou de plusieurs diagrammes.

La littérature est riche en études diverses, la plupart consistant en la définition et la détection de la cohérence au sein d'un seul diagramme ou entre quelques diagrammes spécifiques. Par exemple, [30] évalue les diagrammes d'activités au moyen d'exigences exprimées en logique temporelle linéaire (*linear temporal logic* en anglais). Les diagrammes de machines à états sont évalués après avoir été transformés en réseaux de Petri [77] par transformation de graphes, ou exprimés en Promela [50], langage d'entrée du model-checker SPIN.

Concernant la détection d'incohérences entre plusieurs diagrammes, d'autres travaux traitent des aspects dynamiques afin de détecter des incohérences comportementales. [51] décrit une approche algorithmique pour la vérification de la cohérence entre diagrammes de séquence et diagrammes de machines à états. [49] s'intéresse à la cohérence entre le diagramme de séquence et d'autres diagrammes, par exemple en vérifiant la cohérence entre l'invocation d'une méthode et l'état du système par l'étude conjointe des diagrammes d'objet et des diagrammes de machines à états. On peut aussi citer [40] qui traduit les diagrammes comportementaux d'UML en terme de prédicats d'états et qui utilise le *theorem prover* PVS (*Prototype Verification System*). Mais les diagrammes de composants et de déploiement ne sont pas considérés.

Nous constatons donc que ces propositions concernent des éléments particuliers du langage UML sans chercher une couverture exhaustive. Elles se limitent à un ou plusieurs diagrammes mais sans tenter de considérer l'ensemble des diagrammes UML dans sa totalité. Nous étendons ces travaux en étudiant de façon systématique les cohérences qui doivent exister :

- tout d'abord sur l'utilisation des éléments d'UML, à l'exception des relations, pris séparément ;
- ensuite sur les relations qui relient ces éléments dans la description d'un diagramme ;
- puis sur l'expression d'un diagramme intégrant éléments et relations en considérant chacun des types de diagrammes offerts par UML 2.0 ;
- enfin entre diagrammes.

2.2.3 Type de propriété utilisée

Comme décrit dans [70], UML est un langage semi-formel dont certaines constructions peuvent être interprétées de façon sensiblement différente. Notre but n'est pas de figer une seule interprétation mais de définir précisément les interprétations licites ; l'utilisation du langage conduisant à un modèle contenant des incohérences est considérée comme illicite.

La définition des incohérences nécessite la définition des propriétés qui doivent être respectées (c'est-à-dire ce qui est cohérent).

On distingue généralement [29] :

- La cohérence syntaxique, mettant en jeu la “bonne formation” structurelle de la syntaxe abstraite spécifiée dans le métamodèle (*well-formedness rule* en anglais). Un exemple typique de règle de “bonne formation” est : “*Une généralisation consiste en une relation entre un classificateur général et un classificateur spécifique.*” Une incohérence associée à cette règle serait l'utilisation d'une relation de généralisation entre un classificateur et autre chose qu'un classificateur, par exemple un commentaire.
- La cohérence sémantique est induite par le sens donné aux constructions du langage. Elle est souvent exprimée en langage naturel et est fréquemment définie par un ensemble de règles qui sont difficilement exprimables dans un langage formel [28]. Un exemple de règle de cohérence sémantique est : “*Toute hiérarchie de généralisation doit être directe et acyclique.*” [59]. Une incohérence associée à cette règle serait la présence d'un cycle dans une hiérarchie de généralisation.

Cet exemple illustre que la détection de cette incohérence peut se faire par la syntaxe, mais que l'incohérence porte bien sur le sens donné à la relation de généralisation qui est une relation d'ordre total. Notons que cette règle de cohérence sémantique est ici exprimée dans [59] sous la forme d'une contrainte OCL.

Nos travaux concernent ces deux aspects.

On distingue également la cohérence statique ou dynamique [28] :

- La cohérence statique d’UML représente les propriétés que doit respecter un modèle UML sans “l’exécuter”. Elle est décrite formellement par le métamodèle et des contraintes OCL, ainsi que de manière textuelle. Elle peut être vérifiée entre autres par une inspection statique du modèle. Par exemple, cette contrainte sémantique est d’ordre statique : *“Une contrainte attachée à un stéréotype ne doit pas rentrer en conflit avec aucune des contraintes d’aucun stéréotype hérité, ou avec aucune contrainte de la classe de base.”*
- La cohérence dynamique représente les propriétés à respecter qui seront vérifiées à l’exécution. UML n’étant pas exécutable, ces contraintes dynamiques doivent être embarquées dans un formalisme exécutable (comme le code source). En conséquence elle ne peut pas toujours être complètement vérifiée avant l’exécution du programme dérivé du modèle. Par exemple, il n’est pas possible de vérifier statiquement qu’une précondition d’une opération est satisfaite avant que l’opération n’ait été appelée dans un diagramme d’interaction.

On peut noter que la sémantique d’UML restant ambiguë sur certains points, elle offre la possibilité de plusieurs interprétations différentes. Un exemple est l’absence de spécification de la stratégie pour défiler les événements produits par une machine à états [70]. Cet état de fait aura pour conséquence de rendre la vérification de la cohérence dépendante de choix faits par les concepteurs sur ces ambiguïtés rémanentes.

Nous nous intéressons aussi bien aux cohérences statiques que dynamiques. Notre étude cherche donc à identifier toutes les propriétés sur les modèles qui sont indépendantes des fonctionnalités du système modélisé et du processus de modélisation, que ces incohérences soient syntaxiques ou sémantiques, statiques ou dynamiques. Elles concernent uniquement les contraintes venant du formalisme UML. Certaines de ces propriétés sont exprimées dans la norme soit en OCL, soit d’une façon textuelle. Notre étude étend l’expression des *well-formedness rules* d’UML et reprend les règles textuelles qui n’ont pas (encore) été traduites en OCL et en définit de nouvelles. Enfin nous pouvons mentionner que certaines règles spécifiques à la plate-forme et à l’implantation, dernier niveau de la classification des contraintes définie dans [69], ont aussi été définies lors de nos études.

D’autres études proposent d’autres propriétés. Par exemple, [26] vérifie la plupart des critères généraux de sûreté définis par N. Leveson [48] (*“Toutes les variables doivent être initialisées”*, *“Tous les états doivent être atteignables”*, etc.) concernant une forme réduite des *statecharts*. [77] vérifie les propriétés comportementales que sont les interblocages (*deadlocks* en anglais) ou les problèmes d’atteignabilité (*reachability*), le caractère borné

(*boundedness*), la vivacité (*liveness*), etc., sur les réseaux de Petri obtenus par transformation de modèles UML. Mais l'approche proposée n'a été validée qu'en partant de diagrammes de machines à états. L'intention des auteurs est de vérifier les corrélations qui existent entre diagrammes (règles de cohérence inter-diagrammes) en les prenant en compte lors des transformations de diagrammes UML en réseaux de Petri.

Notre but est de fournir toutes les propriétés exprimant la cohérence des modèles UML afin d'élargir le nombre de fautes de modélisation détectées. D'autres travaux cherchent également à établir des modèles de fautes possibles. Comme exemple, on peut citer des ports de composants non connectés dus à un connecteur manquant, ce qui a pour effet d'empêcher la réception de messages. Un autre exemple est un état manquant dans une machine à états [12]. Cette étude parle de "modèles de fautes" que l'on peut voir comme des fautes génériques exprimant chacune une possible incohérence. Le schéma 2.1 illustre comment un ensemble de fautes de même type au niveau modèle peut être vu au niveau métamodèle UML comme un modèle de faute (générique) et que l'on pourra retrouver par instanciation dans n'importe quel modèle. Ce modèle de faute est donc associé au langage UML et les règles de cohérence que nous fournissons sont définies à ce niveau-là.

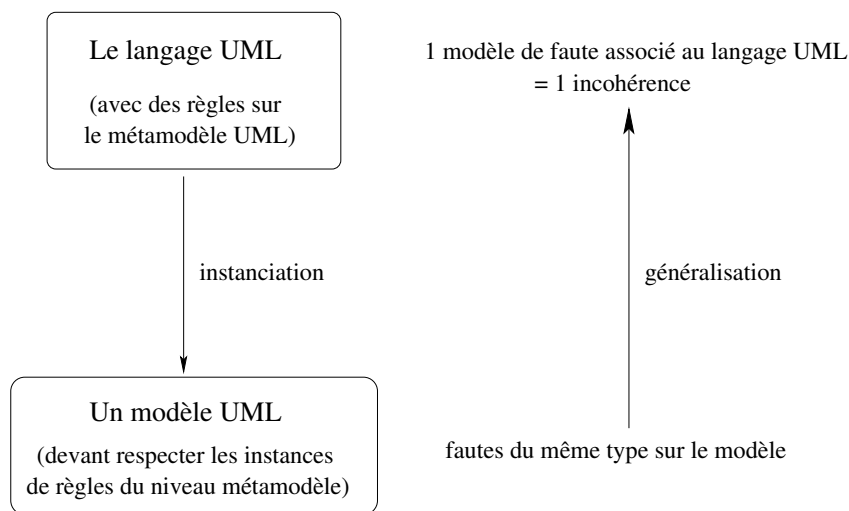


FIG. 2.1: Modèle de faute et incohérence

Mentionnons également que si les incohérences signalent la présence certaine de fautes dans les modèles, toute faute ne génère pas des incohérences et ne peut pas être détectée par l'approche proposée. En effet, une faute spécifique, ou propre au problème traité, ne pourra pas être détectée. Si l'expression de la propriété est générale, sa détection, n'est possible qu'en disposant d'informations supplémentaires issues par exemple d'un modèle de départ ou de documentations. En d'autres termes, un modèle UML cohérent peut cependant posséder une ou plusieurs fautes de conception. La vérification de la cohérence permet cependant de détecter un certain nombre de ces fautes.

2.2.4 Technique de détection

Les moyens utilisés pour exprimer et détecter des incohérences ont une grande influence sur les travaux effectués. Les types d'incohérences considérés par les études sont souvent très réduits du fait de ces choix. C'est pour cette raison que nous avons établi des règles de cohérence sans chercher à prendre en compte le moyen de leur expression ni les techniques ou outils nécessaires à la détection du non-respect des règles. Cependant, dans cette sous-section, nous synthétisons les approches utilisées pour mettre en oeuvre expression et détection. Nous les présentons en les regroupant en deux grandes familles : l'approche transformationnelle et l'approche qui tend à enlever l'ambiguïté d'UML, soit en offrant de meilleures contraintes à UML, soit en tentant de définir une sémantique formelle pour le métamodèle d'UML. Nous mettrons l'accent sur la première famille d'approches qui est la plus classique.

La première famille dite approche transformationnelle consiste à traduire les éléments d'un ou de plusieurs diagrammes UML dans un langage formel puis d'analyser des propriétés associées au modèle formel obtenu. Par exemple, [64] a défini pour des types particuliers (*statecharts*) puis pour des types plus larges de machines à états, une sémantique CSP (*Communicating Sequential Processes*). [29] transforme les modèles UML vers CSP pour vérifier des propriétés dynamiques concernant les diagrammes de classes et les machines à états associées. Afin de vérifier des propriétés dynamiques exprimées en CTL, [30] considère le diagramme d'activités et le transforme en système de transitions (*transition system* en anglais). On pourrait citer d'autres exemples basés sur divers formalismes : une algèbre des processus exprimée en RT-LOTOS pour le profil d'UML TURTLE [14], l'algèbre logique [15] ou encore B [55].

L'approche transformationnelle est intéressante car elle permet d'utiliser les capacités d'analyse des langages formels cibles et permet ainsi de détecter un certain nombre d'incohérences. Elle est schématisée à la figure 2.2.

En revanche, la traduction d'un langage "ambigu" (UML) vers une notation formelle requiert de prendre des décisions pour réduire l'ambiguïté, choix qui ont pour conséquence de figer la sémantique du langage "ambigu" selon [28].

Cette approche présente plusieurs autres inconvénients :

- Premièrement, ces études ne concernent que certaines caractéristiques de certains diagrammes d'UML dont la traduction peut être faite dans le langage formel choisi. Par exemple, certains aspects dynamiques des modèles UML vers les réseaux de Petri.
- Deuxièmement, les propriétés vérifiées sont restreintes à celles analysables par le langage formel (par exemple, la vivacité pour les réseaux de Petri).
- Troisièmement, une fois une propriété du modèle formel niée, les outils signalent très rarement les éléments du modèle UML qui sont à l'origine de l'incohérence révélée [27].

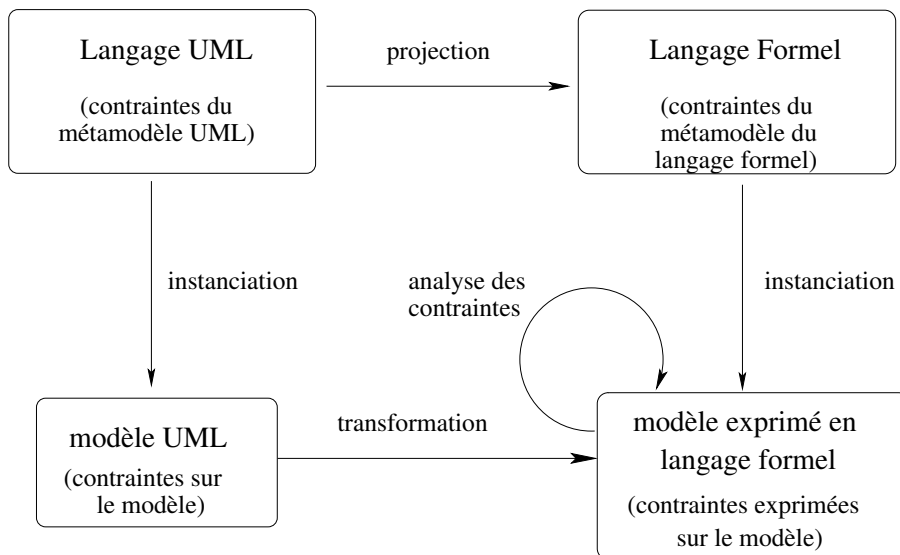


FIG. 2.2: Approche transformationnelle

- Quatrièmement, ceci induit le fait que ces études énumèrent très rarement les incohérences détectées sur les modèles UML car elles ne détectent pas de telles incohérences mais leurs effets sur le modèle formel dérivé.

Il est à noter que toute l'information contenue dans le modèle UML de départ n'est pas forcément conservée après la transformation. Pour obtenir un grand nombre de détections d'incohérences par des approches transformationnelles, il est donc nécessaire d'utiliser un grand nombre d'outils basés sur des modèles formels variés : CSP, B, Z, algèbre logique, etc. Cependant, quel que soit le nombre de ces outils, il est difficile de connaître effectivement le taux de couverture obtenu, c'est-à-dire le nombre d'incohérences étudiées sur le nombre d'incohérences possibles, puisque ces deux données ne sont pas définies.

Une autre approche qui étend l'approche transformationnelle cherche à préciser ce qui est transformé et la façon dont la transformation doit avoir lieu. Les notions de correspondance et de raffinement entre points de vue sont prises dans [27] comme base de la vérification de la cohérence. D'une part, les correspondances décrivent les aspects communs de différents points de vues. D'autre part, des spécifications partielles peuvent être considérées comme cohérentes entre elles dès lors que l'on peut trouver un raffinement commun de ces spécifications, c'est-à-dire une spécification plus détaillée commune. Cela suppose que l'on se mette d'accord au préalable sur la notion de raffinement utilisée. Ainsi [27] définit la cohérence de la manière suivante : un ensemble de points de vue de spécification sont cohérents entre eux s'il existe une spécification qui est le raffinement de chacun des points de vue et qui soit en accord avec les relations de raffinement choisies et avec les correspondances entre les points de vues. Ce développement est appelé une unification. Cette approche nous semble très intéressante mais nécessite au préalable de définir les incohérences traitées.

Enfin [68] propose l'utilisation d'un système expert [23] comme base pour la gestion de la cohérence et l'assistance à la conception en UML. Un système expert est particulièrement adapté pour traduire des morceaux de connaissance exprimés en langage naturel en des règles formelles. Certaines contributions préconisent l'utilisation du formalisme de la logique d'ordre n comme domaine sémantique pour formuler les contraintes du métamodèle [72, 71]. De manière analogue, les systèmes experts utilisent un langage de logique d'ordre n pour décrire l'ensemble des faits et des règles dans leur base de connaissances. L'une des caractéristiques intéressantes des systèmes experts est la possibilité d'écrire des règles selon le format : si $\langle \text{expression} \rangle$, alors $\langle \text{action} \rangle$, où l'expression est un prédicat et l'action est une opération à exécuter. Ainsi, le système permet de donner des suggestions et d'aider au diagnostic. Une autre proposition d'utilisation des systèmes experts est celle de [52], où les incohérences sont reportées et stockées dans la base de connaissances. Ceci permet à l'utilisateur de remettre à plus tard la résolution des incohérences, grâce à plusieurs catégories de règles "experts".

La deuxième famille d'approche tente d'enlever l'ambiguïté inhérente à UML. Elle peut se subdiviser en deux sous-familles, celle qui tente d'offrir de meilleures contraintes à UML, et celle qui formalise le métamodèle.

La première sous-famille cherche à étendre OCL, le langage de description des contraintes intégré à UML, par exemple pour pouvoir vérifier des propriétés dynamiques [20]. Une autre façon de chercher à obtenir de meilleures contraintes est d'utiliser d'autres langages de description des contraintes comme les conditions de cohérence graphiques (*Graphical Consistency Conditions*) [35, 76].

La deuxième sous-famille tente d'enlever cette même ambiguïté en formalisant le métamodèle. Les propositions allant dans ce sens tendent à affiner le métamodèle d'UML en précisant la sémantique des caractéristiques d'UML ou en l'étendant. La première contribution amenée par le *precise UML (pUML) group* a pour objectif d'augmenter le pouvoir expressif du méta-langage d'UML, pour réduire le nombre de contraintes sémantiques qui ne sont pas formellement définies dans le standard. Ainsi, le groupe a identifié des ambiguïtés dans la notation UML et a proposé une sémantique plus précise [31]. La deuxième contribution citée suggère l'extension du métamodèle UML pour mieux supporter le *model checking* pour le raffinement logiciel. Cette contribution se focalise sur la vérification de la cohérence entre modèles à différents niveaux d'abstraction [67]. Dans cette deuxième sous-famille, on peut citer des approches qui formalisent le métamodèle d'UML à l'aide d'un langage formel. De cette façon, les modèles UML sont traduits comme des instances du métamodèle formalisé. [19] formalise le métamodèle d'UML par un système de transitions. [38] prend le langage Object-Z comme méta-langage. On peut alors écrire les contraintes de cohérence UML de niveau métamodèle comme des invariants Object-Z. Toute instance (c'est-à-dire tout modèle UML à vérifier) d'une métaclasse décrite en Object-Z doit respecter les invariants.

2.3 Nécessité d'une étude systématique

Les études sur la cohérence des modèles UML présentées à la section précédente montrent que la prise en compte des moyens d'expression et de manipulation des éléments des modèles UML restreignent très fortement les incohérences effectivement traitées. Notre contribution concerne, en premier lieu, la proposition d'une étude systématique des propriétés devant contraindre l'usage des constructions offertes par la spécification d'UML 2.0 [60]. Ce besoin est par exemple souligné en conclusion de [38] : "Alors il est nécessaire de définir ce qui devrait être vérifié avant de discuter de la façon d'effectuer cette vérification."¹

Signalons dès à présent que notre étude ne concerne pas initialement le développement d'outils de vérification de ces propriétés. Cependant, les résultats produits peuvent dès maintenant être utilisés comme check-list lors d'une revue classique de modèles UML, ou pour établir des scénarii de lecture pour une revue "basée-perspective" (*perspective-based reading*) [44].

En conclusion, nous avons montré que le choix *a priori* d'une approche et de techniques ou d'outils d'implantation réduisait fortement l'établissement d'une liste exhaustive de règles de cohérence. C'est pour cette raison que nos études ont été effectuées à partir de la norme UML 2.0 et en particulier de son métamodèle. Ce dernier nous permettra d'établir un nombre important de règles de cohérence même si le métamodèle omet un nombre non-négligeable de contraintes de cohérence [38].

¹"Thus, it is necessary to define what should be checked prior to discussing how to perform this actual verification".

Chapitre 3

Identification des règles de cohérence UML

L'identification des risques est la première étape du management des risques. Elle consiste à énumérer ceux-ci. Les incohérences que nous cherchons à identifier sont exprimées par la négation de propriétés attendues des modèles pour qu'ils soient cohérents. Ces propriétés sont formulées par des règles de cohérence. Dans ce chapitre, nous abordons donc l'identification des règles de cohérence UML en précisant dans la section 3.1 la méthode adoptée, et la manière d'organiser les résultats obtenus. La section 3.2 présente un choix illustré des résultats et les limites que l'on doit signaler.

3.1 Méthode d'identification

Notre étude avait pour but de fournir un document aussi exhaustif que possible comprenant les propriétés auxquelles doivent satisfaire les modèles UML afin d'être cohérents.

Nous proposons dans cette section d'introduire la démarche qui a été mise en œuvre (cf. section 3.1.1) et les informations identifiées (cf. section 3.1.2) pour aboutir à la structure du rapport produit (cf. section 3.1.3), disponible sur internet [54] et présenté en [66].

3.1.1 Objectifs

Le chapitre 2 a démontré la nécessité de mener une étude systématique afin de définir l'ensemble des règles que doit respecter tout modèle UML. Rappelons ici que le but principal de cette étude est de définir aussi précisément que possible l'ensemble des interprétations licites d'UML. Afin de réaliser une étude aussi complète que possible, deux personnes (Hugues Malgouyres et moi-même) ont tout d'abord effectué une étude systématique de toutes les contraintes implicites ou explicites formulées dans la définition du langage grâce à une double lecture en parallèle de la spécification d'UML 2.0 [60]

dès sa sortie. Les règles dites “possibles” ayant été identifiées séparément ont été mises en commun. Après l’étude séparée de chaque diagramme, de nombreuses discussions ont donné lieu à une synthèse des règles de cohérence trouvées.

En parallèle, une étude bibliographique a été réalisée. Les articles parus dans les groupes de travail sur “Consistency Problems in UML-based Software Development” [43, 42] ont été particulièrement analysés.

Chaque fois qu’une règle était trouvée, elle était exprimée en langage naturel (français) pour unifier l’expression de l’ensemble des règles et faciliter l’expression de règles difficilement exprimables dans des langages formels. En effet, comme vu à la section 2.2.4, l’utilisation d’une notation formelle réduit souvent les contraintes exprimées. Nous considérons que la formulation des règles en langage naturel est la première étape avant toute tentative pour les formuler en d’autres langages.

3.1.2 Informations associées à chaque règle

Durant cette phase d’identification, nous nous sommes rendus compte que des informations précieuses pouvaient être rajoutées à chaque règle. Il s’agit de l’origine de la règle et du niveau d’abstraction sur lequel s’applique la règle.

L’origine de chaque règle a été précisée et nous avons identifié quatre sources différentes :

1. Les règles qui sont explicitement données dans la spécification du langage UML, directement exprimées comme des contraintes dans la section “Constraints” du métaélément (les *well-formedness rules*, WFR).
2. Les règles déduites du métamodèle. Le métamodèle exprime chaque “caractéristique” d’UML en utilisant UML [59]. Ces règles expriment de façon textuelle des contraintes incluses dans le métamodèle comme des contraintes de multiplicité, des contraintes sur les types des relations entre divers éléments, etc.
3. Les règles venant de la littérature. Ces règles sont, soit trouvées textuellement dans la spécification, lesquelles sont plutôt des règles sémantiques, mais non exprimées explicitement comme des contraintes, soit des règles trouvées dans des articles ou des livres.
4. Et finalement, les règles qui sont le résultat d’une analyse personnelle de la spécification d’UML. Nous avons déduit ces règles en imaginant des fautes qui pourraient être commises dans des modèles UML 2.0. Ces règles sont nouvelles et représentent la principale contribution à cette partie de notre travail.

Pour illustrer chaque type de règle, nous présentons ici des règles simples qui s’appliquent à une relation d’arc d’activité dans un diagramme d’activités.

1. Contrainte de la spécification UML 2.0 : “*Les arcs d’activité ne doivent être possédés uniquement que par des activités ou des groupes d’activités.*”¹

¹“*Activity edges may be owned only by activities or groups.*”

2. Règle déduite du métamodèle d'UML 2.0 : “*Un arc d'activité possède exactement un nœud d'activité source et un nœud d'activité cible.*”
3. Règle venant de la littérature (exprimée textuellement dans la spécification d'UML 2.0 ici) : “*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'activités.*”²
4. Nouvelle règle : “*Seuls les arcs sortants d'un nœud de décision ou d'un nœud de bifurcation peuvent avoir des gardes différentes de **true**.*”

Afin de quantifier notre travail, le tableau 3.1 synthétise la répartition de l'origine des 650 règles exprimées dans notre document [54].

Origine	Nombre de règles	Pourcentage
Contraintes de la spécification UML 2.0 (WFR)	290	44.62
Déduites du métamodèle UML 2.0	56	8.61
Provenant de la littérature	44	6.77
Nouvelles règles	260	40
TOTAL	650	100

FIG. 3.1: Tableau synthétique de l'origine des règles

D'autre part, nous différencions les règles selon leur niveau d'écriture et d'application. C'est le niveau d'abstraction sur lequel la règle s'applique. Cette information est marquée dans le document à la suite de chaque règle entre crochets. Le premier niveau est constitué des règles pouvant être écrites au niveau “méta” et qui contraignent le métamodèle. Ces règles sont utiles aux concepteurs d'outils. On parle de règle sur le métamodèle. Le deuxième niveau correspond aux règles pouvant être écrites au niveau “méta” et qui contraignent les modèles. Ces règles sont les plus intéressantes et les plus nombreuses. Le troisième niveau correspond aux règles ne pouvant pas être écrites au niveau “méta”. Ceci s'explique par le fait qu'aucune métaclasse ne correspond à l'élément graphique sur lequel s'applique la règle. On parle de règles de niveau “utilisateur”. Il est à noter que lorsque la règle ne s'applique pas spécifiquement à un niveau d'abstraction (cas du deuxième niveau), l'information n'est pas marquée pour cette règle dans le document, signifiant qu'elle est à la fois de niveau “méta” et “utilisateur”.

Finalement, quelques règles supplémentaires complètent notre document. Nous les appelons “règles lien”. Elles redéfinissent simplement une règle existante dans un autre contexte. Par exemple, comme un flot de contrôle au sens UML 2.0 (“*control flow*” des diagrammes d'activités) est une sorte d'arc d'activités, il doit respecter toutes les règles

²“*Every connector with a given label must be paired with exactly one other with the same label on the same activity diagram.*”

définies pour un arc d'activités. Notre document fournit 59 règles liens qui s'ajoutent aux 650 règles précédentes.

3.1.3 Structure du document

Il est à noter que notre démarche s'inscrit dans la classification proposée par [36] dans les niveaux d'étude conceptuels modèle et diagramme :

- l'étude de la sémantique de vérification de chaque élément de base du langage UML et de chaque relation entre ces éléments constitue le niveau conceptuel diagramme ;
- l'étude de la cohérence des diagrammes du langage UML et des relations existant entre les diagrammes d'un même modèle correspond au niveau conceptuel modèle.

Le document final contient autant de parties que de diagrammes UML 2.0 afin de faciliter son utilisation lors d'analyse de modèles particuliers (par exemple par des revues). Les différentes parties abordent successivement les diagrammes suivants :

1. diagramme de classes (incluant ses variations en diagramme de paquetages et d'objets),
2. diagramme de composants,
3. diagramme de structures composites,
4. diagramme de déploiement,
5. diagramme d'activités,
6. diagramme d'interactions (incluant diagrammes de séquence, communication, vue d'ensemble d'interaction et timing),
7. diagramme de machines à états (incluant machines à états et machines à états de description de protocole),
8. diagramme des cas d'utilisation.

Toutes les règles concernant un diagramme UML spécifique sont incluses dans une partie spécifique. Chaque partie est constituée de trois sections distinctes. Les règles concernant les éléments du métamodèle et étudiés séparément sont présentées dans une première section. Il s'agit de tout élément impliqué dans le diagramme à l'exception des relations définies entre éléments. Notons que le terme « relation » n'est pas employé ici dans le sens propre à UML (association, généralisation, dépendance et ses descendants, importation d'élément et de paquetage). Le terme relation est en fait un sur-ensemble des relations d'UML qui inclut les liens graphiques (*graphical path*) (énumérés dans la spécification en fin de chaque section correspondant à un diagramme), les liens entre objets, les transitions entre états, etc. Les règles concernant ces relations entre les éléments du métamodèle sont détaillées dans une seconde section. Et enfin la troisième section contient les règles de cohérence concernant la bonne formation du diagramme

considéré.

La dernière partie du document rassemble les règles de cohérence concernant l'étude des relations inter-diagrammes. Cette partie n'est que la reprise des règles déjà mentionnées par ailleurs dans les autres parties au fil du document, mais elle les organise par couple de diagrammes. Il s'agit donc de règles liens.

Comme mentionné au chapitre 1, la définition des modèles de fautes ou d'erreurs par les règles de cohérence constitue l'*identification des événements dommageables* exigée par l'*analyse des risques* associés à la technologie UML. Simultanément, nous avons mis en valeur des guides de style de modélisation devant participer au traitement (à la réduction) de ces risques. Ces guides concernent :

- la *prévention* afin de diminuer la probabilité de présence de fautes ou d'occurrence d'erreurs, et
- la *protection* afin de diminuer leur gravité, c'est-à-dire de faciliter leur détection.

Une vingtaine de guides de protection et plus d'une quarantaine de guides de prévention ont été actuellement formulés. Ils seront abordés au chapitre 5.

En conclusion, pour chaque construction du langage UML, nous avons adopté le même canevas d'étude :

- en premier lieu, nous présentons le « **contexte** », c'est-à-dire les circonstances dans lesquelles une erreur conduisant à une incohérence peut être commise lors de la modélisation UML et, si besoin est, nous nous appuyons sur les définitions de la spécification 2.0 d'UML [60] ;
- ensuite, nous définissons les « **règles de cohérence** » (*consistency rules* en anglais) qui expriment par intention la sémantique de vérification que l'élément considéré doit respecter ; chaque fois que la règle nous semble difficile à comprendre, nous donnons un exemple de « **modèle de faute** » servant à clarifier la négation de la règle de cohérence ;
- lorsque cela est possible, nous donnons des « **guides** » de style pour prévenir ou faciliter la détection des fautes ou des erreurs ;
- une rubrique « **justification** » apporte enfin une justification pour chaque guide.

Rappelons que notre but est d'énumérer les incohérences et non de les détecter. Précisons en particulier que certains des modèles d'erreurs correspondent à des erreurs non-détectables de manière statique. Ce type d'erreur qui ne sera détecté que durant la phase d'exécution du code généré, demande à instrumenter le code lors de la génération du code. Nous pouvons faire le parallèle avec un langage de programmation comme Ada : certaines erreurs sont détectées à la compilation (comme par exemple la non-concordance des types des termes d'une affectation) alors que d'autres erreurs seront

détectées à l'exécution du programme grâce à l'instrumentation du code exécutable généré (vérification de l'appartenance d'une valeur d'une expression au sous-type de la variable affectée).

3.2 Résultats et analyse

Afin de matérialiser le résultat de notre travail, nous commencerons par donner à la section 3.2.1 quelques exemples significatifs de règles en reprenant la structuration du document produit. Nous incluerons dans chaque exemple des commentaires critiques sur le travail effectué. Ensuite nous ferons quelques commentaires globaux sur les résultats de l'étude en section 3.2.2 et en aborderons ses limites en section 3.2.3.

3.2.1 Illustration de chaque type de règle

Dans cette section, nous décrivons des exemples de règles de cohérence ayant trait à chaque chapitre de notre document. En section 3.2.1.1 nous donnons des exemples de règles de cohérence ayant trait aux éléments, en section 3.2.1.2 aux relations, en section 3.2.1.3 aux diagrammes pris dans leur totalité, en section 3.2.1.4 à des incohérences mettant en jeu plusieurs diagrammes. Nous tenterons donc d'illustrer ces sections par des exemples représentatifs du document final, sans toutefois chercher l'exhaustivité dont le document d'où sont issues ces règles tente de faire preuve. Nous ne donnerons pas ici d'exemple des règles OCL issues de la spécification d'UML.

3.2.1.1 Élément

Dans cette section, nous illustrons des règles de cohérence concernant des éléments d'UML 2.0. Le premier exemple est une règle déduite du métamodèle. Le second est une contrainte définie textuellement dans le document de spécification d'UML et enfin, nous exposons une nouvelle règle de cohérence.

Règle dérivée du métamodèle Le métamodèle précise que tout élément ne peut pas être inclus dans un paquetage (voir figure 3.2 prise de [59]). Mais on peut se demander légitimement quelle est la liste des éléments empaquetable (*packageable element*) car la définition donnée dans la spécification précise seulement qu'un élément empaquetable "indique un élément nommé qui peut être directement possédé par un paquetage."³

La seule chose que nous pouvons déduire comme règle de cohérence est que : "*Tous les éléments inclus dans un paquetage doivent être des éléments empaquetable.*"⁴

Cette règle est dérivée du métamodèle car elle est directement issue du diagramme de la section "Kernel-The Packages Diagram" de [59]. Comme la liste complète de tous les

³"A packageable element indicates a named element that may be owned directly by a package."

⁴"All elements included in a package must be packageable elements."

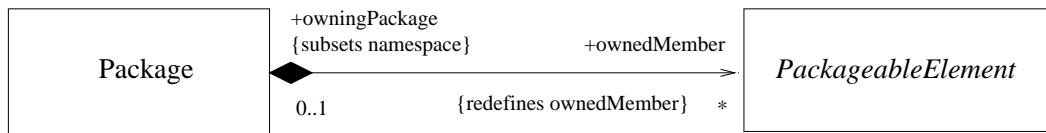


FIG. 3.2: Relation entre les métaclasses Package and PackageableElement

éléments empaquatables n'est pas fournie de manière textuelle, nous l'avons reconstituée sous forme d'arbre de spécialisation de la métaclasse `PackageableElement` (voir figure 3.3). Pour ce faire, nous avons extrait de chaque diagramme présentant des "bouts" du métamodèle UML de [61], toutes les parties incluant la métaclasse `PackageableElement` ou un de ses sous-types.

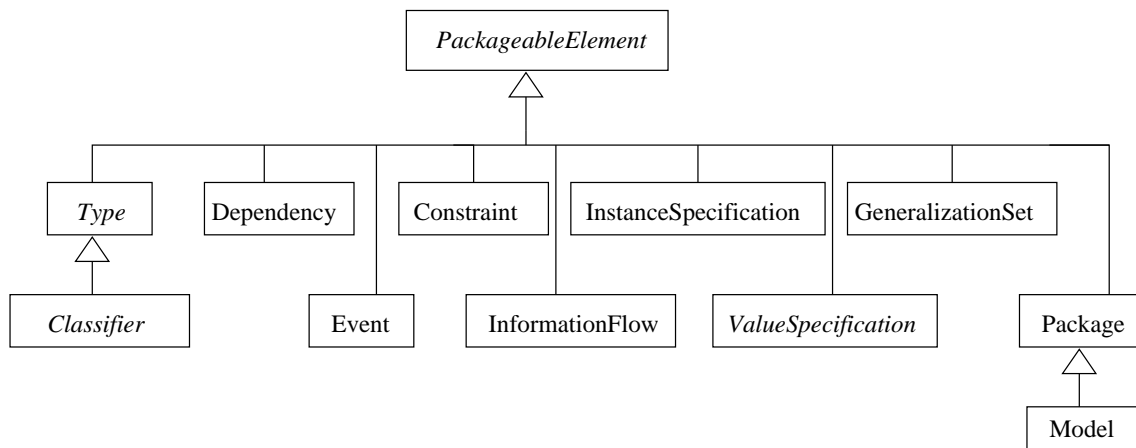


FIG. 3.3: Arbre de spécialisation de la métaclasse PackageableElement

Notons que dans notre document, les hiérarchies de spécialisation des métaclasses `Classifier` et `Dependency` sont aussi détaillées. Ce genre d'information est utile aux utilisateurs d'UML pour concevoir des modèles UML bien formés.

Une autre réalité est que la plupart des outils de type AGL (Atelier de Génie Logiciel) ne respectent pas certaines contraintes définies dans la dernière version du métamodèle UML. En conséquence, le recensement des contraintes déduites du métamodèle est également utile pour les concepteurs de ces outils.

Règle provenant de la littérature Voici l'exemple d'une règle exprimée textuellement dans la spécification mais qui n'est pas incluse sous forme de contrainte OCL. Elle concerne les composants et elle énonce qu'“*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.*”

En considérant que les interfaces `I_NewOrderEntry` et `I_CommonOrderEntry` fournies

par les composants sont différentes, la figure 3.4 montre un exemple d'incohérence. En effet, il manque l'information qui attesterait que le composant `NewOrder` réalise aussi l'interface `I_CommonOrderEntry`. Il manque ici donc une flèche de réalisation qui irait du composant `NewOrder` vers l'interface `I_CommonOrderEntry`. Si elle est "sous-entendue", le modèle est alors incomplet et gagnerait à être complété.

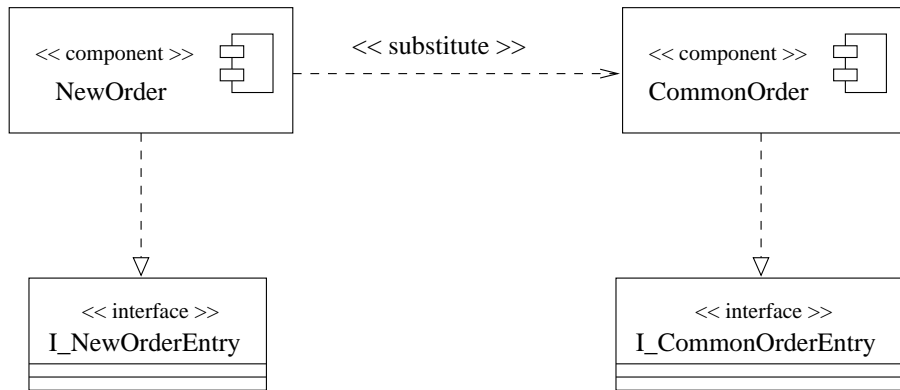


FIG. 3.4: Composant candidat pour en substituer un autre

Notons que cette règle est en réalité une instantiation pour un composant, d'une de nos nouvelles règles exprimées pour la relation de dépendance « substitute » qui établit que *“les interfaces implantées par le classificateur cible d'une relation de substitution doivent être implantées par le classificateur source (le substituant).”*

Nouvelle règle de cohérence Dans cette section, nous donnons un exemple sur l'élément du métamodèle `Property`. Il est important de différencier l'élément du métamodèle `Property` de la notion générale de propriété dans un modèle UML. L'élément du métamodèle `Property` représente soit un attribut d'une classe, soit une fin d'association. Nous désignons cet élément par une propriété au sens “possession”. Dans cet exemple, nous exprimons une nouvelle règle sur une propriété représentant une fin d'association navigable.

Comme toute relation d'héritage doit mettre en jeu exactement un parent et un enfant (règle déduite du métamodèle UML 2.0 pour la relation de généralisation), le modèle de la figure 3.5 présente une incohérence qui aurait pu être levée en respectant la règle suivante : *“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.”*

Ici, le problème de la modélisation provient du couplage entre les multiplicités de fin d'association jouant le rôle b ou étant un sous-ensemble de ce rôle. En effet, il y aura au maximum 5 objets instances de la classe FB reliés à une instance de la classe FA, et les multiplicités indiquent qu'il faudrait au moins 3 objets instances de la classe SB1 et 3 objets instances de la classe SB2, soit au total 6 enfants de la classe FB.

Il est à noter que cette règle est un exemple de règle qui s'applique aussi bien au niveau utilisateur qu'au niveau métamodèle. Une faute de modélisation violant cette règle est difficile à détecter car elle demande de coupler des informations dispersées dans le modèle. Et ce, d'autant plus que prises séparément, les multiplicités 3..4 des fins d'associations de asso2 et asso3 respectent bien la propriété {subsets b} comme l'exprime la contrainte de la spécification d'UML : "Une propriété sous-ensemble doit renforcer le type d'une propriété sur-ensemble, et la borne maximale de sa multiplicité doit être inférieure à celle de la propriété sur-ensemble."⁵

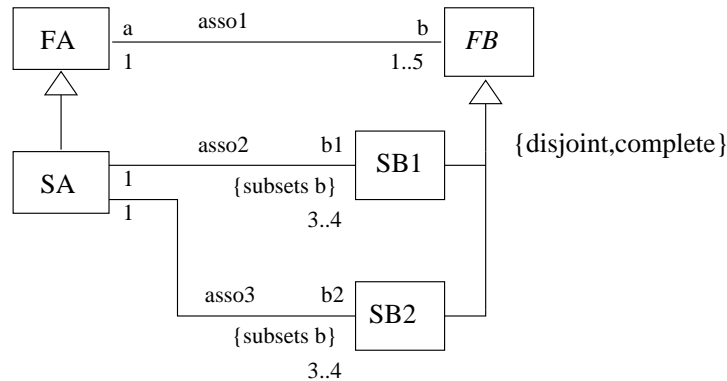


FIG. 3.5: Incohérence dans la définition des multiplicités dans un diagramme de classes

3.2.1.2 Relation

Afin d'illustrer quelques règles mettant en jeu des relations entre éléments, nous allons suivre le même plan que dans la section précédente en abordant trois exemples.

Règle dérivée du métamodèle La figure 3.6 donne un exemple de manifestation entre un artefact et un composant (qui est un élément empaquetable), dans un diagramme de déploiement. Cet exemple ne comporte pas d'incohérence.

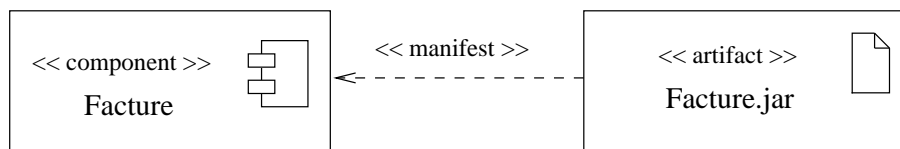


FIG. 3.6: Exemple de manifestation

Le métamodèle lie un élément empaquetable à l'artefact dans lequel il est incorporé (voir figure 3.7).

⁵"A *subsetting property* may strengthen the type of the subsetted property, and its upper bound may be less."



FIG. 3.7: Métamodèle impliquant la relation de manifestation

Une règle dérivant de l’analyse du métamodèle a été ainsi définie : “*Une dépendance de manifestation doit avoir comme source un artefact et un élément empaquetable comme cible.*”

Ainsi toute autre combinaison de couples d’éléments reliés par une dépendance de manifestation est incorrecte.

Règle provenant de la littérature Pour illustrer une règle concernant la cohérence d’une relation d’arc d’activité (*activity edge* en anglais) dans un diagramme d’activités, nous allons reprendre l’exemple cité dans la section 3.1.2. “*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’activités.*”

Une autre règle exprime textuellement dans la spécification d’UML qu’ “*un connecteur doit avoir exactement un arc entrant et l’autre connecteur doit avoir exactement un arc sortant, chacun avec le même type de flot, soit d’objet soit de contrôle.*”⁶

La figure 3.8 inspirée de [59] présente une incohérence qui peut être difficile à détecter manuellement dans le cas où on utiliserait dans un diagramme d’activités beaucoup de connecteurs labellisés pour l’alléger. Dans cet exemple, un premier connecteur est labellisé “1” avec un arc (edge) d’entrée et un deuxième connecteur labellisé “1” est sans arc sortant. La faute peut être détectée avec le sens des flèches assez aisément car le diagramme est simple et ne fait pas usage de beaucoup de connecteurs labellisés.

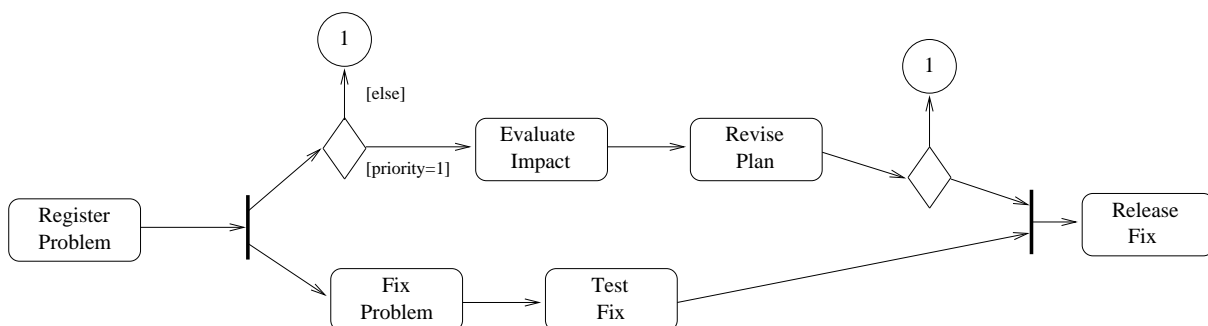


FIG. 3.8: Incohérence dans les arcs d’activités avec des connecteurs labellisés

⁶ “*One connector must have exactly one incoming edge and the other exactly one outgoing edge, each with the same type of flow, object or control.*”

Notons que cette règle n'est pas exprimable au niveau métamodèle car un label est une notation purement graphique qui n'a aucune correspondance dans le métamodèle.

Nouvelle règle de cohérence Pour clore les exemples illustrant des règles sur les relations, nous pouvons prendre celui qui concerne la relation d'importation d'élément. 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.

Une nouvelle règle cependant est à respecter : *"Si un alias est spécifié, aucun nom d'élément appartenant à l'espace de nommage importateur ne doit être identique à l'alias spécifié."*

La figure 3.9 montre un exemple d'importation d'un élément avec un alias erroné. En effet, l'alias choisi (Radio) correspond déjà à une classe de l'espace de nommage importateur, à savoir le paquetage DataSys.

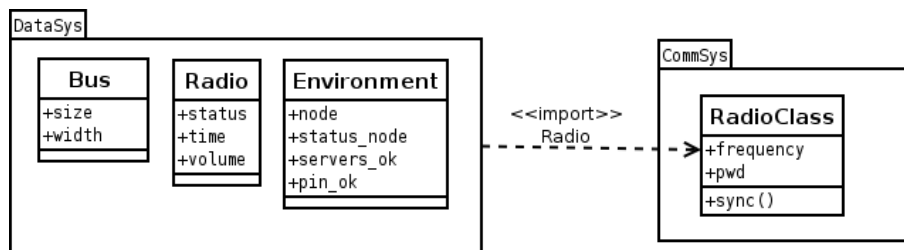


FIG. 3.9: Exemple d'une importation d'élément avec alias incohérente

3.2.1.3 Diagramme

Les règles concernant les diagrammes sont peu importantes, elles sont le fruit de l'assemblage des éléments et des relations. Mais afin d'en illustrer quelques unes, nous prendrons deux exemples.

Règle provenant de la littérature Voici l'énoncé d'une règle définie par [6] : *"Les événements dans un diagramme d'activité peuvent seulement être attachés à l'arc qui va du point de départ (ou nœud initial) à la première action."*

Les arcs qui représentent les transitions entre deux actions ont la même syntaxe que dans les machines à états, à l'exception des événements. Les arcs peuvent comporter des conditions de garde, une clause d'envoi (*send-clause* en anglais), et une *action-expression*.

La figure 3.10 tirée de [6] représente un diagramme d'activité simple sur lequel l'arc entre la deuxième et la troisième action possède une clause d'envoi qui envoie le message `Print(file)` à l'objet `Printer`.

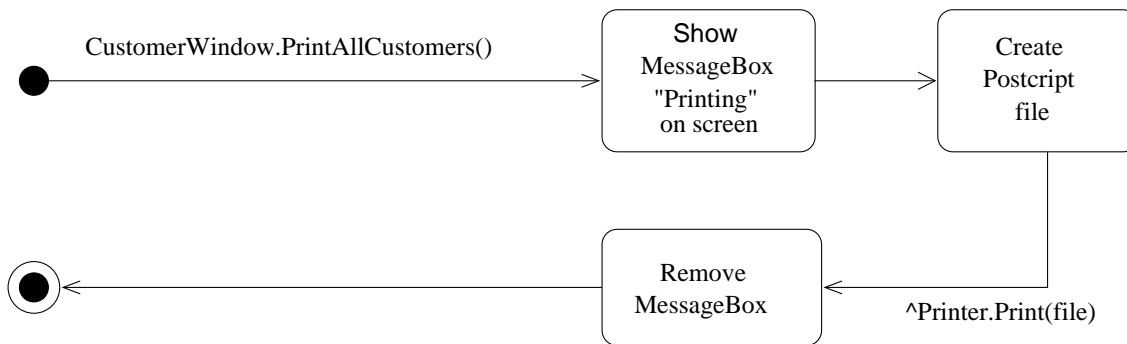


FIG. 3.10: Diagramme d'activité avec événement activateur sur le premier arc

Nouvelle règle de cohérence Les diagrammes d'activités UML 2.0 ont été largement inspirés par les réseaux de Petri. On peut définir cette nouvelle règle : *“Tout arc d'activité doit être tirable.”*

Cette règle pourrait aussi s'exprimer différemment, en disant que *“Tout diagramme d'activités doit correspondre à un réseau de Petri quasi-vivant.”* Un réseau de Petri est quasi-vivant si pour toute transition, il existe un marquage accessible qui active cette transition.

La figure 3.11 présente une activité qui ne respecte pas cette règle. 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.

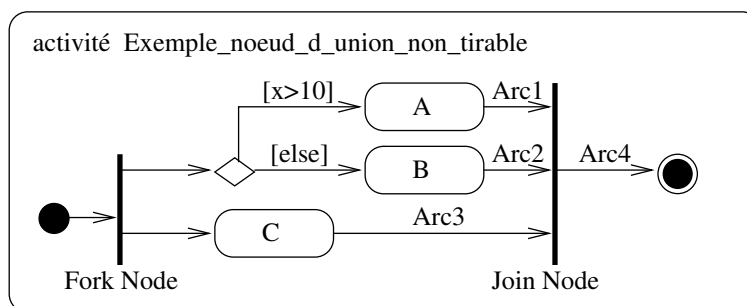


FIG. 3.11: Incohérence dans un diagramme d'activités

3.2.1.4 Inter-diagramme

Dans cette section nous présentons deux exemples de règles, l'une tirée de [59], et une nouvelle règle.

Règle de cohérence issue du standard On donne l'exemple cité pour la métaclasse `Behaviour` mais qui met en jeu la cohérence entre un diagramme d'activités et un diagramme de classes. Cette règle provient de la section "Constraints" de la métaclasse `Behaviour`, mais est simplement exprimée textuellement, car elle doit être instanciée pour chaque comportement. Ici nous avons pris l'exemple d'une activité qui représente le comportement d'une caractéristique comportementale (une opération) d'une classe.

La figure 3.12 montre que le diagramme d'activités est censé représenter le comportement de l'opération `logger(user, key)` de la classe `NetSys`, (c'est-à-dire l'opération qu'elle hérite de la classe `EntrySys`) comme cela est indiqué dans le coin supérieur gauche du diagramme d'activités. Pourtant la règle tirée du standard que nous voulons introduire ici précise ceci : " 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 redéfinissante." ⁷

Or ici, la dernière caractéristique comportementale (ici opération) redéfinissante est l'opération `NetSys.loggerNet(user_net, net_pass)`, et c'est son comportement que l'activité devrait représenter et non celui de `NetSys.logger(user, key)`. En d'autres termes, comme `NetSys.loggerNet(user_net, net_pass)` redéfinit l'opération `logger` de la classe `EntrySys`, c'est donc l'opération `NetSys.LoggerNet` qui devrait être représentée par le diagramme d'activités.

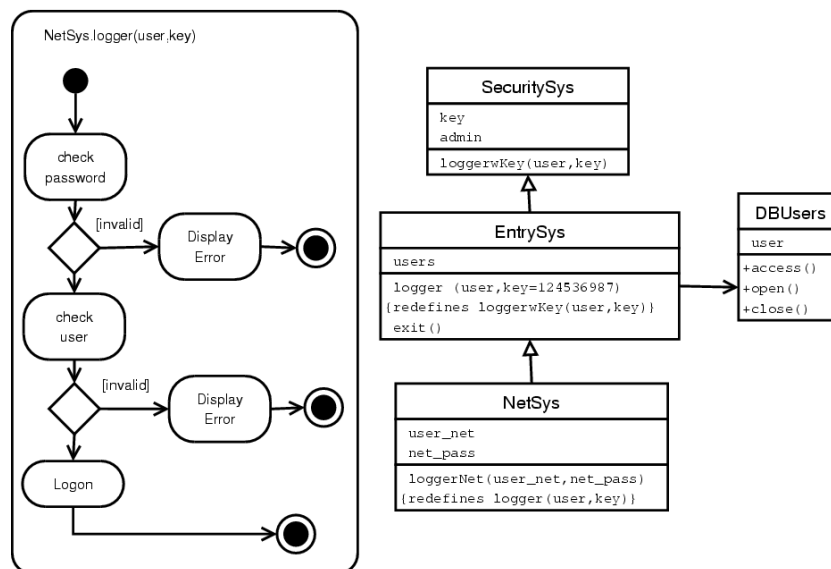


FIG. 3.12: Incohérence entre diagramme d'activités et diagramme de classes

Cet exemple de règle est l'illustration de règles génériques formulées par la norme mais qui demandent à être spécialisées ou du moins explicitées sur des cas concrets comme celui-là.

⁷ "If the implemented behavioral feature has been redefined in the ancestors of the owner of the behavior, then the behavior must realize the latest redefining behavioral feature."

Nouvelle règle La règle présentée dans cette dernière section concerne les possessions (*properties*) dans les diagrammes de structure composite. Les possessions représentent 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 3.13 tirée de [59] montre un exemple de possession. La sous-figure (i) définit une 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, elle 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**.

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

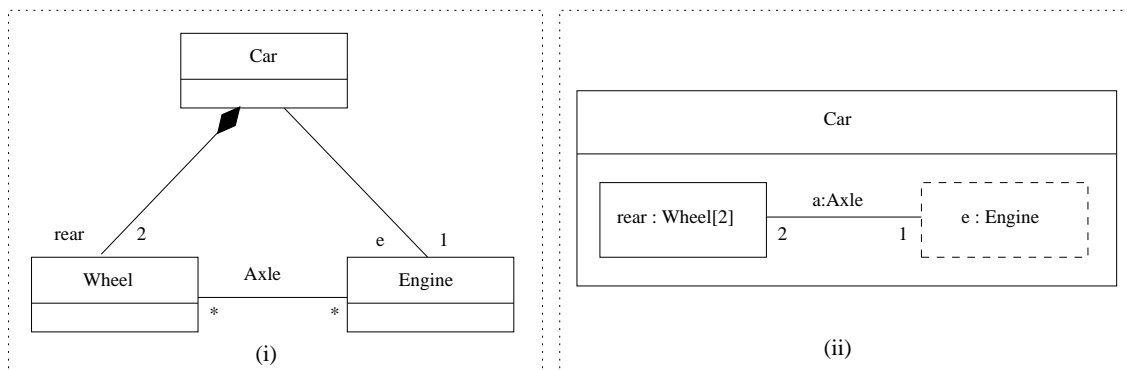


FIG. 3.13: Cohérence entre diagramme de classes et diagramme de structures composites

Nous énonçons une nouvelle règle vérifiée par l'exemple, et dont le respect est une des conditions pour assurer la cohérence entre un diagramme de structure composite et le diagramme de classes correspondant : “ Une possession *A* qui n'est pas une partie de la classe *B* dans un diagramme de structure composite, doit être une possession dont le métaattribut `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 non-composite.”

3.2.2 Commentaires sur les résultats

3.2.2.1 Organisation du document

On constate que les règles classées pour un élément particulier du métamodèle, mettent en jeu parfois plusieurs autres éléments, et par exemple que certaines règles

classées dans la partie “Éléments” d’un diagramme pourraient aussi trouver leur place dans la partie “Relations”. Nous avons essayé de respecter toutefois le découpage de la spécification. En particulier, toutes les règles qui se trouvaient déjà dans la section de ce métaélément particulier décrit dans la spécification sont présentées dans la partie “Éléments” du document [54].

3.2.2.2 Bilan quantitatif

Comme cela a déjà été présenté au tableau 3.1 décrivant l’origine des règles énoncées, un bon nombre d’entre elles (environ 40 % des règles) sont nouvelles. Les règles directement extraites de la norme (les contraintes de la spécification) sont les plus nombreuses (44.62 %), puis viennent ensuite les règles directement tirées de la bibliographie ou exprimées textuellement dans la norme (6.67 %) et les règles dérivées du métamodèle (8.61 %).

Concernant la deuxième information associée à chaque règle (niveau d’abstraction et d’application de la règle), le tableau 3.14 reprend les chiffres synthétisant la répartition de cette information.

Les règles spécifiques au métamodèle (7.23 %) et les règles utilisateur (4 %) forment un petit ensemble dont le reste (88.77 %) correspond aux règles de niveau “méta” et utilisateur qui sont des règles s’appliquant aussi bien au niveau modèle que métamodèle.

Niveau d’abstraction	Nombre de règles	Pourcentage
Règles sur le métamodèle	47	7.23
Règles utilisateur	26	4
Règles à la fois “méta” et “utilisateur”	577	88.77
TOTAL	650	100

FIG. 3.14: Tableau de répartition des règles par niveau d’abstraction

3.2.2.3 Liens des règles entre elles

Règles dites “liens” Les règles ne sont pas forcément complètement décorrélées les unes des autres. On peut voir que d’après les relations de généralisations du métamodèle, certaines propriétés (règles) s’appliquant sur un métaélément s’appliquent aussi sur ses métaéléments “fils”. Ainsi un élément dépend de plusieurs règles et doit les respecter, quand il hérite de toutes les propriétés du métaélément dont il est “une sorte de”.

Comme cela a déjà été présenté en fin de section 3.1.2, on peut dire que les règles “liens” redéfinissent simplement une règle existante dans un autre contexte. Le fait que le flot de contrôle au sens UML 2.0 (“*control flow*” des diagrammes d’activités) qui est une

sorte d'arc d'activités, doit respecter toutes les règles définies pour un arc d'activités, est un exemple parmi d'autres.

Règles nouvelles Illustrons le processus de mise au point d'une version de la norme UML à l'OMG sur le cas du document de la Superstructure d'UML 2.0. La version adoptée d'UML 2.0 sortie en Août 2003 (OMG Adopted Specification ptc/03-08-02) [59] sur laquelle nous avons mené notre étude est restée en vigueur pendant un an, temps au cours duquel les utilisateurs de ce standard pouvaient envoyer des remarques, sous forme de propositions de corrections (*issues* en anglais) à l'OMG. La version comprenant ces correctifs ajoutés en plus du texte initial (Document ptc/04-10-02, convenience document) [61] a finalement été remplacée par la version définitive, dite disponible (*Available* en anglais) [62] en Août 2005.

Voici un exemple de règle ancienne (exprimée textuellement dans [59]) qui a été prise en compte comme contrainte de la version *Available* : “*Un nœud d'interclassement possède au moins deux arcs entrants.*”

Il serait difficile à l'heure actuelle de chiffrer le nombre de règles marquées comme “nouvelles” dans notre document et qui ont été intégrées dans [62], sans se replonger dans une étude complète à partir de la version *Available*.

3.2.2.4 Commentaires concernant la capacité de détection

Certaines règles dans notre document ont été annotées par des remarques concernant l'incapacité de détecter une incohérence de manière statique. Ce type de règle est vérifiable sur le code généré seulement. Voici l'exemple simple d'une règle dynamique qui n'est pas détectable : “*Si le métaattribut `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'accès au même objet par une action extérieure au nœud.*”

Un conflit est défini comme une tentative d'écriture de l'objet par une action ou par les deux actions simultanément. La spécification précise que si un tel conflit existe potentiellement alors, un tel accès par une action extérieure au nœud ne doit s'intercaler dans l'exécution d'aucune action à l'intérieur du nœud. La spécification ne contraint pas les manières de faire en sorte que la règle soit assurée. Mais la spécification du langage UML [59] dit que si c'est impossible d'exécuter un modèle qui respecte ces règles, alors le modèle est mal-formé.

Cet exemple illustre cette catégorie de règles contraignant l'implantation mais qui ne sont pas détectables au niveau modèle.

3.2.3 Limites de l'étude

Moyen d'expression La question du moyen d'expression des règles de cohérence (langage naturel français, ou anglais) s'était naturellement posée dès le début de ce

processus et le choix du français s'est imposé pour augmenter l'expressivité des règles pour notre phase d'identification. Nous sommes conscients qu'un important travail de traduction de ces règles en anglais serait désormais nécessaire pour que celui-ci constitue un apport valorisable pour toute la communauté UML.

Une autre limite liée au moyen d'expression et très difficilement quantifiable, est que la lecture des règles en langage naturel dépend de nombreux facteurs (linguistiques, culturels, etc.) difficiles à maîtriser. La formalisation des règles en CLP (Constraint Logic Programming) est étudiée dans la thèse en cours de M. Hugues Malgouyres [53].

Complétude du document L'ensemble des règles obtenues est conséquent (650 règles). Quelques règles exprimées en langage naturel sont très complexes. Pour une meilleure compréhension, nous les illustrons par des schémas montrant un modèle erroné et le modèle associé respectant la règle. Cependant cette "explication par l'illustration" des règles mérite d'être étendue à toutes les règles définies afin de rendre le document plus explicite encore. Un des travaux déjà entrepris pour les diagrammes de structure (un tiers des règles du document) est l'illustration systématique de chacune des règles par un modèle cohérent et un modèle incohérent associé.

Exhaustivité des Règles D'une part, lors des relectures, travail de fusion, d'errata des versions, et lors de l'exploitation du document sous forme d'interviews d'experts comme nous le verrons au chapitre suivant, certaines règles sont apparues comme redondantes, d'autres ont dû être reformulées pour être plus complètes ou plus compréhensibles. Après ce travail de suppression des redondances, on peut dire qu'il reste encore certains liens entre règles à formaliser (les règles héritées d'autres règles entre autres).

D'autre part, on peut toujours se demander si l'ensemble des règles défini est complet. Il resterait à faire à ce jour une relecture avec la version *Available* pour mettre à jour nos règles nouvelles qui ne le sont plus, et intégrer les modifications qui ont été apportées dans cette dernière version. Sur le plan de l'adéquation du document avec la superstructure en terme de complétude, un travail de fusion des règles sur les profils est encore en cours.

Maintenance et évolution du standard UML Nous pouvons partir du constat que des parties entières de la première version de la spécification [59] étaient incomplètes pour les diagrammes de structure composite et déploiement. Nous avons pu nous rendre compte pour certains passages de la spécification de la présence d'une imprécision résiduelle dans la version intégrant les remarques [61]. De plus, et comme le montre [17], beaucoup d'expressions OCL de [59] comportaient des erreurs, non pas simplement dues à un mauvais usage d'OCL, mais à des déficiences dans les diagrammes de classes sous-jacents.

Il est regrettable de constater que la dernière spécification d'UML [62] comporte encore de nombreuses incohérences et organise ou conserve certains métaéléments de manière difficilement compréhensible. Par exemple, et en accord avec la remarque donnée dans [16], la distinction entre **Signal** et **Event** n'a pas lieu d'être. De plus, le passage des versions 1.x à 2.0 n'apporte pas d'avancée significative dans la cohérence entre les diagrammes de modélisation de structure (*Structure Diagrams*) et de modélisation du comportement (*Behavior Diagrams*). Le manque à combler réside dans la formalisation des liens entre les 2 grandes familles de diagrammes. L'idée proposée dans [16] est de référencer dans les modèles de comportement les propriétés des modèles de structure et vice-versa.

Un point supplémentaire avec lequel il faut compter est le temps. En effet, le processus d'évolution des versions du langage est somme toute plutôt long, et on peut aussi constater un temps d'inertie entre la sortie du standard et la "tentative" d'intégration du nouveau métamodèle dans les outils. Ce laps de temps est nécessaire, afin d'effectuer des retours sur le standard en cours de finalisation. Ceci étant dit, nous constatons que l'évolution naturelle du standard tend à préciser les notions encore floues.

Pour conclure, pour que ce travail soit poursuivi, il nous semble nécessaire de mettre à jour le document recensant les règles de cohérence d'UML 2.0 avec la version *Available* de la norme UML 2.0 [62]. De plus, il est souhaitable que la communauté UML fasse des retours sur notre document afin de compléter, à défaut d'achever l'exhaustivité de l'énumération des incohérences.

3.2.4 Conclusion

L'ensemble des règles de cohérence identifiées est conséquent. Nous avons proposé une classification des règles par diagrammes et précisé pour chacune leur origine et le niveau d'abstraction sur lequel elle s'applique. Les règles "liens" explicitent les dépendances qu'il existe entre des règles existantes. Cependant, bien que cette phase d'identification ait été menée de manière aussi exhaustive que possible à partir de la version disponible de la spécification d'UML, elle n'est pas à considérer comme un travail figé. Bien que la plupart des règles resteront exprimées sous leur forme actuelle, le document résultat du travail d'identification mené est voué à évoluer avec les versions futures du standard UML.

Pour aborder la suite de ce travail, il convient de rappeler que le risque est défini par la combinaison d'un événement dommageable (introduction d'une incohérence dans notre cas) et d'un dommage (qui qualifie l'impact de l'incohérence sur un modèle ou le code généré). L'identification présentée dans ce chapitre a permis d'énumérer les événements dommageables que sont les types d'incohérences.

Les dommages, eux, peuvent être classés de trois sortes (voir figure 3.15) :

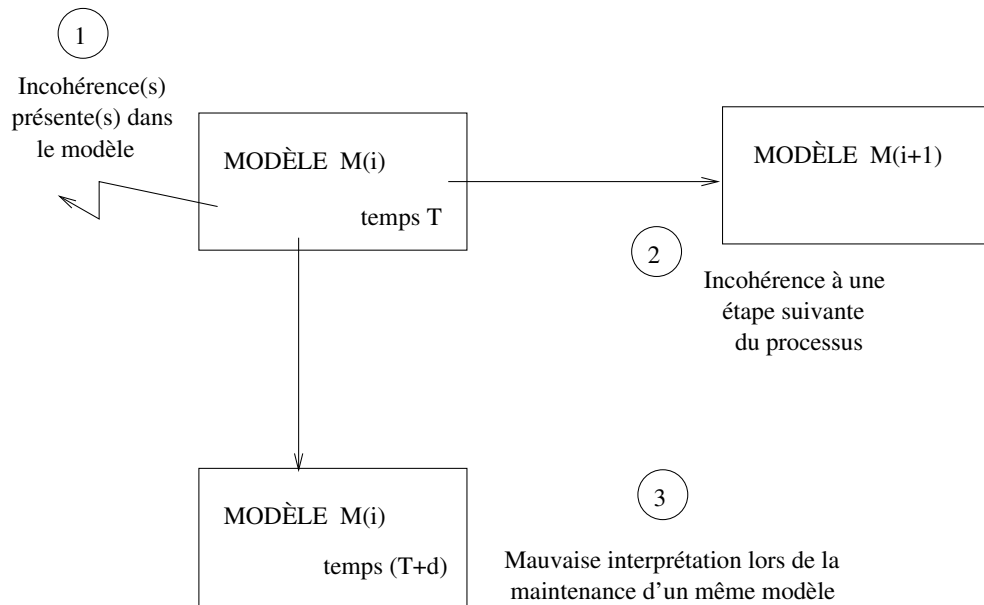


FIG. 3.15: Types de dommages

1. Modèle rendu incohérent par la présence d'une ou plusieurs incohérences (1)
2. Création de fautes dans les étapes suivantes du processus dans les modèles raffinés et en particulier le code généré (2)
3. Mauvaise interprétation lors de la maintenance future d'un même modèle (3).

Nous allons nous intéresser par la suite à l'estimation des incohérences se rapportant aux deux premiers cas.

Chapitre 4

Estimation des risques d'incohérences UML

Dans le cadre du management des risques, après avoir identifié les événements dommageables que sont les fautes révélées par les incohérences, ainsi que les possibles dommages (modèle contenant une faute, transformation erronée aux étapes de raffinement d'un modèle, ou en maintenance d'un même modèle), il convient d'estimer le risque de ces fautes liées de l'emploi de la technologie UML. Comme les types d'incohérences sont associés à des constructions du langage de modélisation UML, l'estimation des risques d'incohérences permettra d'en dériver une estimation du risque associé à l'emploi d'une construction particulière. Ceci permettra d'aider les entreprises ou les organismes de certification à définir les constructions UML qui sont utilisables pour modéliser leurs applications en fonction des niveaux d'acceptabilité du risque qui leur sont associés, ou à définir des contraintes particulières de l'utilisation de ces constructions (cf. chapitre 5 sur le traitement des risques).

Pour ce faire, nous avons abordé cette question en étudiant les besoins d'une telle estimation et les classes de moyens nécessaires à sa mise en œuvre (section 4.1). Ensuite, cette estimation a été tout d'abord conduite d'un point de vue qualitatif par l'interview d'ingénieurs et de chercheurs (section 4.2) et d'un autre point de vue quantitatif par l'analyse de modèles UML (section 4.3). Ce travail a été présenté en [74] et [73].

4.1 Besoins, intérêts et moyens

A partir d'un premier constat sur l'insuffisance actuelle des outils AGL à détecter des incohérences, nous allons définir le risque associé à l'emploi d'une construction, et le besoin de l'estimer pour chaque construction dans le cadre des applications critiques. Enfin nous présenterons les classes de moyens, à savoir les paramètres d'une estimation du risque et leurs attributs ainsi que les types de procédés d'estimation existants.

Pourquoi estimer les risques d'incohérences ? Si elles sont pratiquement absentes des

modèles ou si leur détection est triviale, l'estimation de ces risques n'a en effet pas de sens. Dans le cas contraire, il est indispensable de mesurer leur importance afin de les traiter utilement.

4.1.1 Présence d'incohérences

En premier lieu, la présence d'incohérences dans les modèles n'est pas une vue de l'esprit mais un constat réel. Ceci est tout d'abord mentionné dans la littérature [45]. Cette étude propose une comparaison de 4 cas d'études (3 industriels et 1 fait par un groupe d'étudiants diplômés) en prenant en compte les trois types de fautes de conception de leur classification, à savoir, des fautes de bonne-formation, des fautes d'incohérences, et des fautes d'incomplétude. Cette communication précise le nombre de cas qui violent les règles de cohérence énoncées. Par exemple, pour la règle de cohérence qui dit que *“des messages ne peuvent être échangés qu'entre des classes reliées par association ou dépendance”*, 3 cas d'études sur 4 relevaient autour de 75 % de violations de cette règle. Les études restent cependant limitées à la vérification de quelques règles seulement (5 règles pour les fautes d'incomplétude, 3 pour des incohérences concernant les messages) sur des diagrammes de classes comportant 34 à 168 classes selon le cas d'étude.

D'autres études comparent différentes techniques d'inspection de documents de conception UML [44, 65, 25] selon leur efficacité à détecter des défauts de modélisation parmi lesquels on retrouve l'omission, une information ambiguë, une incohérence, un fait incorrect, une information non utile ou non utilisée, selon la classification tirée des travaux de [8]. Les modèles supports des expérimentations de [44] présentent par exemple un total de 20 défauts. [25] souligne l'importance de ces techniques d'inspection menées sur un projet logiciel de grande échelle chez Ericsson en Norvège, qui permettent de relever un nombre important de fautes de tous types dont les incohérences.

Nous pouvons dire que la présence réelle d'incohérences est un état de fait connu mais qui est difficile à quantifier au cours de la durée de vie d'un logiciel.

4.1.2 Insuffisance des outils de détection

En second lieu, la détection de la présence d'incohérences dans les modèles n'est pas évidente, même si elle est supportée par des outils. Nous l'avons montré en étudiant les capacités de détection des incohérences de 3 outils AGL durant l'été 2004 : Visual Paradigm (VP, Professional Edition version 3.1, Mars 2004), Rational Rose Enterprise Edition (IBM, Juillet 2004), Ameos (Aonix, Juillet 2004). Ce travail s'est limité à l'étude des incohérences concernant les diagrammes de classes, séquence, machines à états et cas d'utilisation.

La capacité de détection a été examinée à trois niveaux :

- durant la phase de saisie des modèles avec l'outil, pour montrer si l'outil était capable de prévenir l'introduction d'incohérences,

	<i>Visual Paradigm</i>	<i>Rational Rose</i>	<i>Ameos</i>
ELEMENTS			
Détectée	25.00 %	30.83 %	25.42 %
Non détectée	48.33 %	35.00 %	44.17 %
Non testable	26.67 %	34.17 %	30.42 %
RELATIONS			
Détectée	12.17 %	13.04 %	20.87 %
Non détectée	73.91 %	64.35 %	52.17 %
Non testable	13.91 %	22.61 %	26.96 %
TOTAL			
Détectée	20.85 %	25.07 %	23.94 %
Non détectée	56.62 %	44.51 %	46.76 %
Non testable	22.54 %	30.42 %	29.30 %

FIG. 4.1: Résultats synthétiques des taux de détection d'erreur

- en utilisant le *checker* intégré à l'outil sur un modèle incohérent déjà saisi,
- en important et en chargeant un fichier XMI d'un modèle incohérent.

Pour chaque incohérence, une des 3 valeurs suivantes a été donnée :

- *Détectée* si l'outil détecte une occurrence de l'incohérence.
- *Non détectée* si l'outil ne la détecte pas.
- *Ne peut pas être testée* si la vérification est impossible du fait que l'outil ne supporte pas la construction graphique de l'élément UML concerné par l'incohérence.

Les résultats généraux donnés au tableau 4.1 révèlent de manière chiffrée un constat auquel nous nous attendions. On peut constater que les taux de détection sont bas : 25% à 30% pour les éléments, 12% à 21% pour les relations. Cependant ces résultats sont à tempérer aussi par le fait que 30% des règles considérées dans cette étude n'étaient pas testables car les outils ne proposaient pas la construction graphique définie par la spécification d'UML 2.0.

Ces résultats extrêmement bas viennent notamment du fait que les outils ne prennent pas en compte certains mots clés comme « *import* », « *export* ». Nous pensons que c'est une grosse lacune car les mots clés sont la description graphique de méta-classes différentes et donc de concepts différents.

Pour chaque incohérence, notre analyse a précisé aussi à quel(s) niveau(x) sa détection était possible : durant la saisie des modèles, en activant le checker ou à l'importation/exportation XMI. A titre d'exemple, sur les 255 règles considérées, en écartant les 32 règles liens et 23 non représentées (règles sur le métamodèle), c'est-à-dire sur 200 règles réellement observées, seule une trentaine d'incohérences (en faisant la moyenne des résultats des 3 outils) était détectée durant la saisie du modèle soit 15 %. Pour ce

A la saisie du modèle	15 %
Par le checker	3.5 à 14 %
A l'importation/exportation	10%

FIG. 4.2: Pourcentage de détection par moyen de détection

qui concerne la détection lors de l'échange de fichiers XMI erronés, seulement une vingtaine d'incohérences a été relevée, soit 10 %. Enfin 7 à 28 incohérences l'étaient grâce au *checker*, sachant qu'un des outils n'avait pas de *checker* intégré, soit entre 3,5 et 14 %. Il est à noter que ces pourcentages ne doivent pas être additionnés pour connaître le nombre d'incohérences total détectées (donné au tableau 4.1). En effet, ils donnent une vue détaillée du moyen par lequel l'incohérence est détectée. Cependant une même incohérence peut être détectée, par juste un, une combinaison de deux ou les trois moyens à la fois.

Le tableau 4.2 résume ces chiffres.

Nous supposons que la vérification concernant la cohérence intra-diagramme et inter-diagrammes est probablement moins efficace encore, mais cela n'a pas encore été prouvé par une étude.

Pour conclure sur cette étude, on peut confirmer que le nombre d'incohérences non détectées et donc potentiellement présentes dans les modèles n'est pas négligeable et qu'il convient donc de mener des études approfondies pour les estimer, ce qui est l'objet de ce chapitre. Cette estimation est en particulier utile pour mettre en valeur les constructions d'UML pouvant conduire à un taux important d'incohérences et donc pour porter ensuite l'effort de réduction des risques sur celles-ci.

4.1.3 Moyens d'estimation

4.1.3.1 Deux paramètres

Le risque est estimé au moyen de 2 paramètres :

- *le degré de vraisemblance*, qui mesure la probabilité de l'occurrence de l'événement dommageable, et
- *la gravité*, qui évalue les conséquences de son occurrence.

L'estimation du risque corrèle ensuite ces deux paramètres. Pour estimer ces paramètres, nous devons proposer des critères et des métriques (cf. section 4.1.3.2). Ces critères et métriques étant choisis, l'estimation consiste à assigner une valeur de métrique à chaque critère au moyen d'une méthode d'estimation (cf. section 4.1.3.3).

4.1.3.2 Critères et métriques

Pour chacun des deux paramètres d'une estimation du risque (degré de vraisemblance et gravité), deux attributs doivent être définis :

- *un critère* qui exprime le point de vue employé pour estimer ce paramètre ; par exemple, si le paramètre est la gravité d'une incohérence liée à une construction d'UML, elle peut être évaluée par la difficulté de sa détection dans les modèles d'UML mais aussi par l'impact de sa présence sur le code généré, etc. ;
- *les métriques* qui énumèrent les valeurs possibles du critère.

La métrique peut être quantitative ou qualitative. Par exemple, la difficulté de détection d'incohérences (critère) peut être évaluée par :

- un entier définissant le nombre d'incohérences restantes dans un modèle ou un ensemble de modèles donné (métrique quantitative), ou
- par trois valeurs qualitatives (facile, assez difficile, extrêmement difficile).

Les métriques qualitatives conduisent à une certaine subjectivité dans l'attribution d'une valeur de ces métriques. Par exemple, il est souvent difficile de stipuler si une détection d'incohérence est "facile" ou "assez difficile". Cependant, ces métriques fournissent des estimations synthétiques qui sont souvent plus réalistes qu'une estimation quantitative pour laquelle on ne dispose pas d'outils d'estimation adéquats. Par exemple, estimer une durée de détection par le nombre de minutes nécessaires n'a pas de sens s'il s'agit d'une interview basée sur l'expérience passée car les ingénieurs n'ont pas mesuré précisément ces durées. Dans ce cas, la distinction entre 15 minutes et 21 minutes semblerait donner une information plus précise, mais cette distinction n'est pas faite en pratique. C'est pour cela qu'une métrique qualitative nous semble plus appropriée.

De plus, pour chaque critère, la valeur assignée doit être marquée par une information additionnelle évaluant la confiance placée en la valeur estimée. Ceci permet à la personne interviewée de nuancer sa réponse en donnant son *degré de certitude*. Cela renforce la crédibilité lorsque la réponse est sûre, ou dans le cas contraire, indique une réponse incertaine. Lors du traitement des réponses, ces indications aident à l'identification des données intéressantes et fiables, et par là même à l'identification de constructions maîtrisées, ou au contraire celles dont l'usage est moins mûr.

4.1.3.3 Méthodes d'estimation

Quand des critères et des métriques ont été définis pour les deux paramètres du risque (degré de vraisemblance et gravité), une méthode doit être choisie pour obtenir les valeurs estimées. Trois genres de méthodes d'estimation sont fréquemment employés.

- *Les interviews d'experts* dérivent des estimations d'avis d'experts du domaine. Elles conviennent aux nouvelles technologies car aucune ou très peu d'informations provenant de l'utilisation précédente de ces technologies n'est disponible. Cette méthode a été employée et est présentée à la section 4.2.
- *Le retour d'expérience* utilise les résultats des pratiques antérieures. Comme Thales Avionics développe des systèmes avioniques complexes en utilisant UML, des modèles d'UML ont été examinés pour déduire des valeurs d'estimation du risque d'incohérences, étude présentée à la section 4.3.

- *Des mesures* qui sont appliquées à l'étude considérée. Pour les modèles UML, cette notion se rapproche des mesures de complexité comme celles de Mc Cabe. Par exemple, le nombre de niveaux d'héritage dans un diagramme de classes peut être utilisé pour estimer la vraisemblance de l'incohérence énoncée comme suit : "tentative de redéfinition d'une méthode qui viole de principe de Liskov". En effet, plus l'arbre d'héritage sera important, plus l'amplitude du phénomène de propagation implicite de l'information par héritage sera important et la probabilité d'occurrence de cette incohérence élevée. Ces études sont actuellement en cours mais aucun résultat significatif ne peut être maintenant présenté.

4.1.4 Application à l'estimation des risques de chaque construction

Un niveau de criticité est associé à chaque fonctionnalité d'une application. Il dépend des conséquences de l'occurrence d'une défaillance de cette fonctionnalité. Par exemple, l'arrêt des moteurs d'un avion est très critique puisqu'il peut conduire à un crash alors que la défaillance du système vidéo des passagers ne provoque que des désagréments. Ce niveau de criticité induit des contraintes sur le développement de cette fonctionnalité. C'est le cas en particulier du choix ou de l'utilisation des moyens dont UML fait partie. En effet, comme nous allons le montrer, le choix des constructions d'UML peut impacter la présence d'incohérences dans les modèles.

Avant d'appliquer des méthodes pour estimer les risques d'incohérences, cette sous-section présente les intérêts des résultats qui motivent les études en cours. Chaque règle de cohérence identifiée concerne une construction d'UML 2.0 ou plusieurs d'entre elles (par exemple, cohérence inter-diagramme). Ainsi, pour une construction donnée d'UML 2.0, une liste de règles de cohérence, c'est-à-dire des types d'incohérences, est associée à cette construction. Par conséquent, après l'estimation du risque de chaque incohérence, une liste de couples des valeurs (degré de vraisemblance, gravité) est associée à chaque construction. La figure 4.3 illustre un exemple d'un tel résultat pour une construction donnée. Elle suppose que cette construction est concernée par les règles numéros 37, 275,

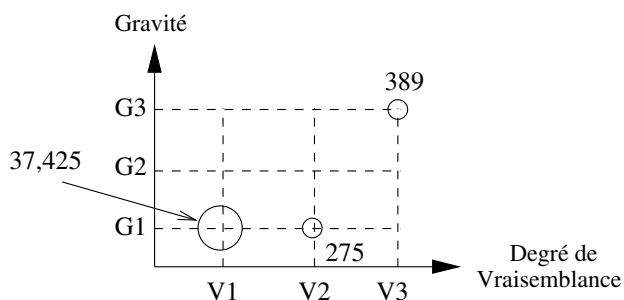


FIG. 4.3: Estimation du risque de la présence d'incohérences associées à une construction UML

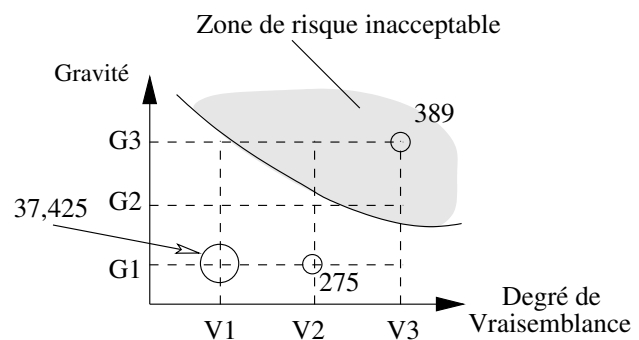


FIG. 4.4: Acceptabilité du risque

389 et 425. Par exemple, l'évaluation de la règle numéro 275 est (V2, G1), c'est-à-dire, un degré de vraisemblance intermédiaire (V2) et une gravité basse (G1).

Comme mentionné précédemment, chaque application critique possède un niveau de criticité dont la valeur dépend de la fonction conçue. Par exemple, les normes de l'avionique indiquent 5 niveaux, du niveau A (le plus haut) au niveau E (le plus bas) [3]. Pour un niveau indiqué de criticité, un niveau d'acceptabilité du risque est défini. La courbe présentée sur la figure 4.4 illustre cette notion. Les risques qui sont sur la partie supérieure droite de cette courbe sont inacceptables.

En conséquence, le risque du type d'incohérence numéro 389 ne peut pas être accepté. Ce risque doit être traité. Plusieurs types de traitements sont disponibles qui seront présentés au chapitre suivant.

4.2 Estimation des risques par interview d'experts

Dans cette section, les risques d'incohérences sont estimés en utilisant des interviews d'experts UML.

4.2.1 Principes

Cette première technique d'estimation consiste à demander à des experts de fournir des valeurs de métriques choisies pour les critères considérés. La crédibilité de ces valeurs est basée sur l'expertise humaine. Cette technique est souvent utilisée pour les technologies pour lesquelles il n'existe pas encore d'utilisations passées suffisamment nombreuses et dont on ne dispose donc pas de *retour d'expérience*.

La campagne d'interviews vise à prendre un échantillon représentatif des personnes impliquées dans la recherche et le développement d'applications modélisées avec UML aussi bien dans le domaine industriel qu'universitaire. Les personnes interviewées et dont les interviews servent de support à cette section représentent chacun un point de vue particulier que l'on peut porter sur les problèmes de cohérence UML. Le premier point de vue est celui d'un expert avionique qui gère le développement de grandes applications embarquées. Le deuxième point de vue est celui d'un universitaire ayant été conseiller scientifique dans une société vendant des services et des outils et actif à l'OMG. Ces deux personnes sont de réels experts d'UML ayant eu un apprentissage et une utilisation d'UML différents.

Le principe général de la récolte d'informations par interview consiste en la séparation des rôles entre l'interviewé et l'intervieweur. Une valeur estimée peut être biaisée si l'intervieweur a tendance à influencer la personne interviewée, ou si elle lui propose des réponses trop particulières. Il est aussi important que chaque interviewé reçoive les mêmes stimuli, autant en termes de questions que de réponses possibles [37]. Cette étude insiste sur l'importance de réduire la charge cognitive de l'intervieweur, en rendant

les questions aussi explicites que possible, afin de diminuer les interactions dues à une mauvaise compréhension de la question ou à des mauvais exemples.

Nous devons prendre en compte dès le début, les limitations classiques liées aux études d'estimation des risques faites depuis longtemps dans d'autres domaines tels que l'industrie chimique. Nous ne citerons que les cinq limitations principales données par [5] :

- la complétude : on ne peut jamais avoir la garantie que toutes les situations accidentelles, leurs causes et leurs effets ont été considérées ;
- la reproductibilité : de nombreux aspects des études sont sensibles aux hypothèses faites par les personnes interviewées au cours du procédé d'estimation et différents experts, en utilisant la même information, peuvent générer différents résultats en analysant le même problème ;
- l'impénétrabilité (*inscrutability* en anglais) : la nature inhérente de certaines techniques rendent les résultats difficiles à comprendre ou analyser (si les études génèrent des centaines de pages de tables, ou de modèles, par exemple) ;
- la pertinence de l'expérience : les personnes interviewées peuvent ne pas avoir l'expérience suffisante et appropriée pour répondre ;
- la subjectivité : les personnes interviewées doivent utiliser leur jugement pour extrapoler leur expérience et déterminer si un problème est important.

Dans le cadre de notre interview, nous devons répondre à chacune de ces limitations possibles :

- Il nous faut être aussi représentatif et complet que possible des diagrammes et donc des constructions et des règles évaluées ;
- Il est nécessaire de conserver exactement la même interview pour une campagne donnée ainsi que la trace des hypothèses faites ou des commentaires affinant les réponses ;
- Les résultats doivent pouvoir rester facilement accessibles et exploitables ;
- Le choix des personnes interviewées est important et l'expertise est le premier critère ;
- Nous devons être conscients que certaines incohérences présentées demanderont à la personne interviewée un effort supplémentaire car elles ne lui seront pas familières. Nous devons avoir un certain degré de confiance dans la subjectivité de la réponse.

En prenant en compte tous ces éléments, nous avons cherché à produire un modèle d'interview en proposant des questions à choix multiples.

Dans la section 4.2.2, les critères et les métriques choisis sont présentés. La section 4.2.3, décrit ensuite le procédé d'estimation. Enfin la section 4.2.4 fournit les résultats obtenus.

4.2.2 Critères et métriques

Les interviews visent à présenter à un expert les types d'incohérences et à poser des questions pour déterminer la gravité et le degré de vraisemblance de leur présence dans des modèles UML. Pour une telle approche d'estimation, seules des métriques qualitatives sont employées.

4.2.2.1 Gravité

Comme mentionné à la section 3.2.4, les dommages causés par les incohérences d'UML peuvent être de trois types. Ici on ne s'intéressera qu'aux deux premiers qui sont :

- la présence d'une ou plusieurs fautes de modélisation dans un modèle (Modèle erroné)
- la création de fautes dans les étapes suivantes d'analyse dans les modèles raffinés et en particulier le code généré.

Deux critères sont alors considérés pour estimer la gravité d'un type d'incohérence.

1. *La difficulté de détection.* Ce critère et les valeurs de la métrique associée sont définis par la question posée : Dans un modèle, la détection de cette incohérence vous semble :
 - (a) être évidente, ou
 - (b) demander un effort mais être faisable, ou
 - (c) nécessiter une investigation trop importante pour y arriver.
2. *L'impact de la présence de cette incohérence sur le code produit.* Ce critère et les valeurs de la métrique associée sont définis par la question posée : Si cette incohérence apparaît dans un modèle, l'impact sur le code dérivé est :
 - (a) très important, ou
 - (b) moyen, ou
 - (c) nul (sans impact).

Ces deux critères mesurent deux aspects complémentaires de la gravité des dommages d'une incohérence.

- Le premier critère (difficulté de la détection) affectera certainement la future maintenance du modèle délivré. En supposant qu'une incohérence était présente, nous avons voulu évaluer si cette incohérence avait tout de même des chances d'être détectée de manière manuelle. Si cela nécessite trop d'effort d'investigation en terme de temps, on considère que la présence d'une incohérence non facilement détectable est très pénalisante car elle ne sera certainement détectée que plus tard, ou dans le pire des cas, elle ne sera pas détectée.

- Le deuxième critère (impact sur le code) évalue la génération d'un programme incorrect. Il concerne les effets sur les étapes suivantes du développement. Pour ce critère, il s'agit d'évaluer l'impact d'une incohérence sur le code considéré avant ou après compilation, c'est-à-dire, est-ce que l'incohérence rend possible la compilation et si oui, est-ce que le code généré a du sens. Nous cherchons à évaluer surtout le deuxième cas. Il est important de l'évaluer car un modèle a pour vocation d'être utilisé pour générer du code. Si l'introduction de l'incohérence rend le code inexploitable, ce critère mesure un dommage de niveau élevé.

La confiance portée à ces évaluations par l'expert, c'est-à-dire dans les réponses aux questions, est donnée par une valeur choisie parmi ces trois : "très bonne ou bonne", "moyenne" ou "faible".

4.2.2.2 Degré de vraisemblance

Deux critères sont considérés pour estimer le degré de vraisemblance d'un type d'incohérence.

1. *L'occurrence d'incohérence.* Ce critère et les valeurs de la métrique associée sont définis par la question posée : À supposer que cette construction d'UML soit employée, l'incohérence associée présentée peut se produire :

- (a) très fréquemment, ou
- (b) assez fréquemment, ou
- (c) parfois, ou
- (d) pratiquement jamais.

Remarque Les constructions présentées étant des constructions UML 2.0, les experts du domaine industriel ne sont pas forcément familiers avec les dernières d'entre elles, donc il a été demandé de se projeter dans une utilisation future.

2. *La difficulté de compréhension de la règle de cohérence.* Elle évalue la connaissance des ingénieurs sur l'incohérence étudiée. Plus la signification de l'incohérence est peu claire, plus cette incohérence peut se produire dans les modèles. Ce critère et les valeurs de la métrique associée sont définis par la question posée : Cette règle de cohérence vous semble :

- (a) claire, ou
- (b) compréhensible, ou
- (c) complexe.

La confiance dans l'évaluation de la probabilité d'occurrence est corrélée avec la fréquence de l'utilisation de la construction. Ainsi, la construction support de l'incohérence est utilisée : "de très nombreuses fois" ou "assez souvent" ou "de façon ponctuelle" ou "jamais". En effet, nous considérons qu'une utilisation plus expérimentée d'une construction est un gage de jugement plus avisé.

La feuille type utilisée pour l'estimation du risque associé à chaque règle se trouve en Annexe A.1. Elle reprend les critères et métriques présentés précédemment.

4.2.3 Processus d'estimation

Nous définissons ici le rôle et les personnes intervenant dans les interviews ainsi que la portée de la première campagne d'interviews.

4.2.3.1 Rôles

Trois personnes participent à chaque interview : l'expert, l'intervieweur et le secrétaire. L'expert doit employer l'expérience acquise par sa participation à différents développements de systèmes et donner des réponses qualitatives (une valeur de métrique) pour la question formulée (un critère choisi). Lors de la phase de présentation de l'interview, l'intervieweur indique que l'interview n'est pas une évaluation de la personne en termes de ses compétences ou de sa maîtrise d'UML mais une évaluation de l'utilisation qu'il a faite du langage UML, de son point de vue d'expert UML. Les réponses sont censées considérer son expérience entière et non un projet de développement en particulier.

L'intervieweur présente la description textuelle de la règle de cohérence de la construction considérée d'UML. En outre, chaque planche projetée montre un modèle incohérent ou contredisant la règle et un modèle cohérent comme illustrations de la règle. Ces modèles sont éventuellement commentés tant que la règle n'est pas comprise. L'intervieweur pose alors les questions (critères) donnant comme choix de réponses les valeurs prédéfinies (métriques présentées en section 4.2.2). L'intervieweur fait attention à ce que l'interview ne diverge pas dans des discussions hors du cadre de la question et ne suscite pas d'autres commentaires que les réponses aux questions.

La collecte des données (des réponses, des valeurs et des commentaires) est faite par une troisième personne (le secrétaire) pour éviter d'omettre d'enregistrer des réponses et pour rendre l'interview plus fluide. Cette personne vérifie que toutes les questions ont été posées et répondues pertinemment.

4.2.3.2 Portée de l'interview

Comme il n'était pas possible d'évaluer chaque diagramme et à l'intérieur même d'un diagramme, chaque construction impliquée, nous avons fait un choix de quelques constructions et des règles représentatives. Les règles de cohérence choisies concernent les diagrammes de classes et ses variations (diagrammes de paquetages et diagrammes d'objets) d'UML 2.0. Les 47 règles de cohérence choisies (dont 25 nouvelles) manipulent 24 constructions du diagramme de structure comprenant l'opération, la propriété, l'interface, le comportement, la classe d'association, la généralisation, la substitution et l'association. La description textuelle des règles est fournie en annexe A.2. Ce choix a

été fait selon la fréquence de l'utilisation de ces constructions dans les développements basés sur UML en essayant de couvrir les constructions fréquemment utilisées.

L'interview comporte actuellement presque 300 questions (47 séries de 6 questions). Le temps moyen pour mener à bien l'interview est approximativement de 3 heures et demie. A l'heure actuelle, 2 interviews complètes ont été effectivement conduites.

4.2.4 Résultats

On présente ici le résultat d'une interview d'un ingénieur de Thalès Avionics et d'un universitaire, experts UML. D'autres sont en cours.

L'objectif de cette section n'est pas d'énumérer toutes les réponses aux questions mais de présenter seulement quatre résultats généraux : deux sont spécifiques à une construction et deux fournissent une vue globale sur les diagrammes de classes UML (incluant ses variations en diagrammes de paquetages et d'objets). Les deux constructions **Association** et **Property** ont été choisies car elles illustrent deux résultats typiques.

4.2.4.1 Résultats spécifiques à la métarelation Association

Les figures 4.5 et 4.6 présentent deux points de vue différents sur la métarelation **Association**, celui d'un industriel et celui d'un universitaire.

Le degré de vraisemblance est évalué par la *probabilité d'occurrence de l'incohérence* et la gravité par la *difficulté de détection*. La métrique associée au critère de probabilité d'occurrence de l'incohérence est constituée des valeurs comprises entre 1 (*ne se rencontre pratiquement jamais*) et 4 (*peut se rencontrer très fréquemment*). Les valeurs de la difficulté de détection vont de 1 (*vous semble être évidente*) à 3 (*vous semble nécessiter une investigation trop importante pour y arriver*). La taille des cercles-points est proportionnelle au nombre de règles ayant le même couple de valeurs (probabilité d'occurrence de l'incohérence, difficulté de détection).

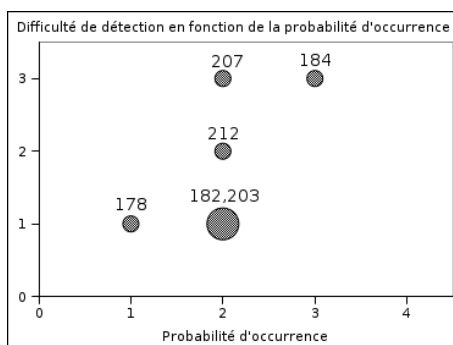


FIG. 4.5: Estimation du risque pour la construction **Association** - Un point de vue industriel

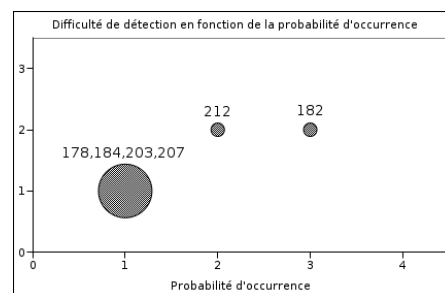


FIG. 4.6: Estimation du risque pour la construction **Association** - Un point de vue universitaire

Six règles de cohérence concernent la construction **Association** (numéros 178, 182, 184, 203, 207 et 212 dans [54]). La figure 4.5 montre l'estimation des risques

de cette construction, du point de vue de l'industriel interviewé. Les valeurs sont plutôt distribuées dans l'espace (nuages de points), ce qui exprime le fait que l'expert n'a pas d'opinion arrêtée sur les effets réellement dangereux de cette construction. Lorsqu'on retrouve ce genre de distribution pour une construction, celle-ci exigera une attention particulière pendant la phase de traitement de risque afin de pouvoir traiter les règles de niveau de risque élevé pour les amener vers une zone de risque acceptable.

Pour l'estimation du point de vue universitaire (figure 4.6), les valeurs du risque sont globalement plus faibles. Notre interprétation des résultats est que l'universitaire a une connaissance très approfondie de l'emploi de la métarelation **Association** et que pour cette construction, il en a une très bonne maîtrise.

4.2.4.2 Résultats spécifiques au métaélément Property

Six règles de cohérence concernent la construction **Property** (numéros 71,72,73,79,80,81). Les figures 4.7 et 4.8 en présentent l'estimation.

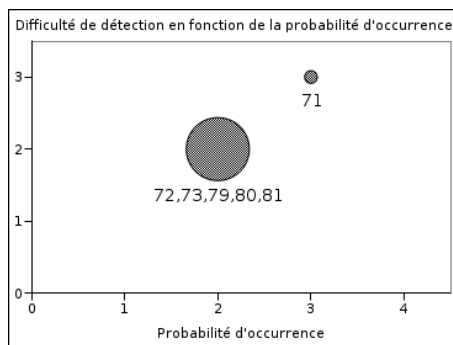


FIG. 4.7: Estimation du risque pour la construction **Property** - Un point de vue industriel

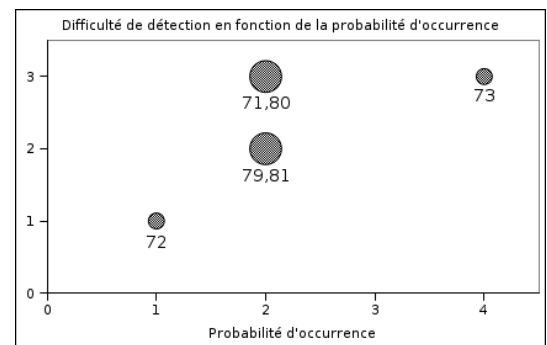


FIG. 4.8: Estimation du risque pour la construction **Property** - Un point de vue universitaire

La partie gauche représente l'estimation du point de vue industriel et la partie droite celle du point de vue universitaire. Les types de résultats sont inversés par rapport à ceux concernant la métarelation **Association**.

Pour l'industriel, les incohérences associées sont plutôt regroupées autour d'une même zone et d'un même niveau de risque que l'on pourrait qualifier d'intermédiaire. Seule la règle numéro 71¹ (couple (3,3)) exige un traitement spécifique pour réduire le risque et atteindre un niveau intermédiaire de risque.

Pour l'universitaire, l'estimation du risque, plus répartie dans l'espace, révèle pour cette construction une confiance moindre dans la maîtrise de son utilisation. Seule la

¹“La somme des bornes inférieures des multiplicités des propriétés qui sont le sous-ensemble d'une propriété appelée A doit être inférieure à la borne supérieure de la multiplicité de A.”(voir section 3.2.1.1, nouvelle règle)

règle 73 (couple 4,3) exige un traitement spécifique pour atteindre un niveau de risque plus bas.

Dans les deux cas, on remarque que les points se situent plutôt à des niveaux élevés de risque, ce qui signifie que cette construction est aussi "à risque". Si l'on compare maintenant les deux estimations pour en dégager des similitudes, on peut juste constater que 2 règles (79 et 81) se situent à un même niveau (couple 2,2). Avec d'autres données issues de nouvelles interviews, il deviendrait intéressant de chercher des similitudes entre estimations de manière statistique en regardant les écart-types entre les valeurs estimées par chaque personne pour une même règle.

4.2.4.3 Estimation globale pour les diagrammes de classes

Les figures 4.9 et 4.10 récapitulent tous les résultats, c'est-à-dire les estimations des 47 règles concernant toutes les constructions manipulées dans les diagrammes de classes, à savoir dans notre cas d'étude pour les 24 constructions considérées. La partie gauche est issue de l'expert industriel et la partie droite de l'expert universitaire. Le degré de vraisemblance est estimé par le critère de *la probabilité de l'occurrence de l'incohérence*. La gravité est évaluée par *la difficulté de détection de l'incohérence* (figures 4.9 et 4.10) ou par *l'impact sur le code généré* (figures 4.11 et 4.12).

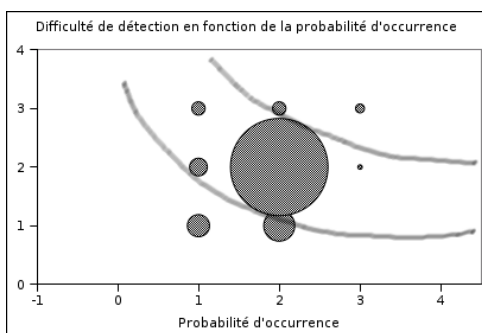


FIG. 4.9: Estimation globale du risque pour les diagrammes de classes selon le critère de la difficulté de détection - Un point de vue industriel

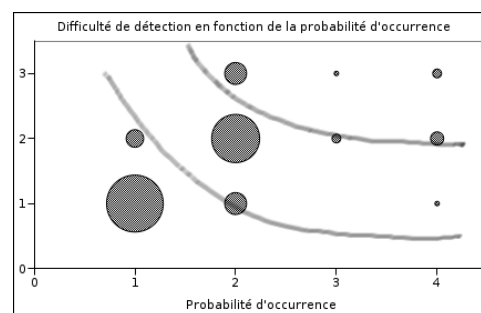


FIG. 4.10: Estimation globale du risque pour les diagrammes de classes selon le critère de la difficulté de détection - Un point de vue universitaire

Les deux courbes représentent deux niveaux de risque. La zone entre ces deux courbes s'appelle ALARP (*As Low As Reasonably Possible*) zone dite "de risque raisonnable". Elle exprime que d'autres critères peuvent être pris en compte pour décider si le risque est acceptable ou non. Parmi ceux-ci, le coût de la mise en œuvre des moyens de réduction du risque est essentiel.

D'un côté, les figures 4.9 et 4.10 prouvent que la plupart des valeurs d'estimation se trouvent dans la zone ALARP ou en-dessous. Ceci signifie que la présence de ces incohérences potentielles dans des modèles d'UML ne mène pas au rejet des constructions étudiées.

Pour l'estimation du point de vue industriel (figure 4.9), la moitié des règles se situent à un niveau de risque intermédiaire et dans la zone ALARP.

Pour l'estimation d'un point de vue universitaire (figure 4.10), plus de règles ont été évaluées à des niveaux de risque faible (couple (1,1)). Il semble qu'il existe pour la majorité des règles une corrélation entre l'occurrence de l'incohérence et la difficulté de sa détection car la plupart des couples (les plus gros cercles) se situent sur la diagonale. Dans les deux cas, il reste tout de même des règles évaluées à des niveaux élevés de risque.

D'un autre côté, les figures 4.11 et 4.12 représentent le critère de l'impact sur le code en fonction de la probabilité d'occurrence de l'incohérence. Elles mettent en exergue le fait que les incohérences rendent inacceptable la génération automatique de code sans aucun traitement du risque. En effet, on retrouve beaucoup de valeurs à un niveau de gravité fort (impact 3 = Très fortes chances que le code dérivé soit erroné). Il est aussi à noter que pour l'industriel, 13 règles (non représentées) sont restées pour lui sans réponse, c'est-à-dire qu'il ne savait pas si l'incohérence avait un impact ou non sur le code.

Une façon de réduire le risque est de diminuer l'occurrence de l'incohérence par des moyens de prévention ou de réduire sa gravité par des moyens de protection abordés au chapitre 5.

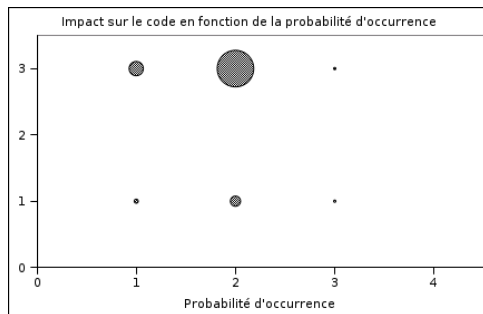


FIG. 4.11: Estimation globale du risque pour les diagrammes de classes selon le critère de l'impact sur le code - Un point de vue industriel

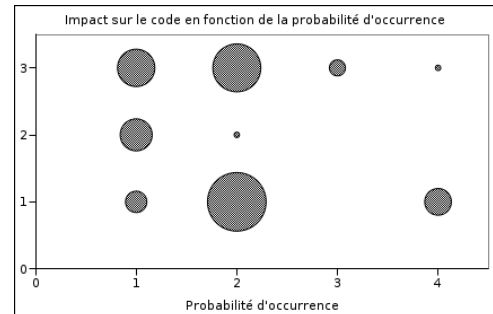


FIG. 4.12: Estimation globale du risque pour les diagrammes de classes selon le critère de l'impact sur le code - Un point de vue universitaire

4.2.5 Commentaires

Pendant l'interview, il est ressorti que la personne interviewée avait tendance à se référer à l'exemple illustratif pour comprendre la règle textuelle, car les deux étaient projetés en même temps. Afin d'avoir un retour plus significatif sur la qualité d'expression de nos règles en langage naturel, nous voudrions modifier l'enchaînement de l'interview, en présentant d'abord la règle sans illustration, puis poser la première question (difficulté

de compréhension de la règle), avant d'afficher les exemples illustratifs. Notre objectif est de forcer la personne interviewée à faire un effort d'abstraction. En effet, nous nous rendons compte que les réponses données aux questions sont très dépendantes de la qualité des exemples car l'emploi de ceux-ci semble limiter la réflexion à ce cas particulier. Or nos règles sont des règles génériques, qui nécessitent d'être instanciées pour pouvoir être représentées. On peut citer en exemple la première règle de l'interview qui dit que *“Un élément ne peut appartenir qu'à un seul espace de nommage en même temps”*. Pour concrétiser cette règle, il faudrait donner toutes ces instanciations possibles, les représenter et évaluer si elle est bien respectée dans tous les cas. Ce qui ne peut pas être fait au cours d'une interview en pratique si l'on veut évaluer plusieurs constructions et ne pas faire exploser le temps de l'interview. Ce sont pourtant ces règles génériques et les constructions dont elles codifient l'emploi qui sont l'objet de notre étude et pour lesquelles nous préférons représenter un exemple illustratif pour aider l'interviewé à se représenter le concept évalué. Dès lors que les exemples ne sont qu'un cas particulier, souvent simple, d'instanciation de la règle sur un modèle donné, il convient de choisir si l'on réfléchit sur un modèle abstrait, avec des classes non liées à un domaine applicatif particulier, c'est-à-dire nommées de manière “impersonnelle”, élément A, B ou C, etc. ou au contraire sur une modélisation “métier” qui a un sens dans un domaine applicatif particulier, avec des éléments type Voiture, Couleur, etc. Nous avons mélangé les deux formes de présentations, tout en privilégiant des exemples “métier”.

Se pose enfin le problème des règles sur le métamodèle qui ne sont pas représentables et qui n'ont pas encore été considérées dans l'interview.

Il serait bon de pouvoir mener ce type d'interviews sur un échantillon plus large d'ingénieurs, et aussi d'universitaires. A ce jour, nous considérons que nous n'avons pas effectué suffisamment d'interviews pour que nos résultats aient une réelle valeur ou justification d'ordre statistique, mais ils donnent de bonnes indications pour identifier d'ores et déjà des constructions “à risques” comme le méta-élément **Property**. Pour ce faire, il faudrait non pas choisir des règles représentatives d'une construction, mais les présenter toutes pour avoir une vue complète de cette construction. Le but de cette étude étant de donner des éléments permettant de juger des constructions particulières d'UML, le nombre de règles évaluées pour une construction est un indicateur important de la qualité des résultats. Les autres constructions de l'interview sont évaluées à partir de 2 ou 3 règles seulement et c'est pour cela que nous n'avons choisi de présenter ici que les constructions **Property** et **Association** car c'étaient celles qui mettaient en jeu le plus de règles dans notre interview.

4.3 Estimation quantitative par analyse de modèles

Une deuxième approche pour estimer le risque a été faite par *retours d'expérience*. Cette approche consiste à estimer le risque d'incohérence en examinant si des modèles

UML existants respectent les règles de cohérence ou non. Ceci a été fait par des revues manuelles de modèles en prenant notre document [54] comme support de la revue.

Nous allons présenter les critères et métriques (section 4.3.1), la démarche (section 4.3.2) et les résultats obtenus (section 4.3.3).

4.3.1 Critères et métriques

Comme présenté dans l'introduction du chapitre, le risque est estimé par deux paramètres génériques (degré de vraisemblance et gravité). Une estimation réelle exige la définition de critères et des métriques.

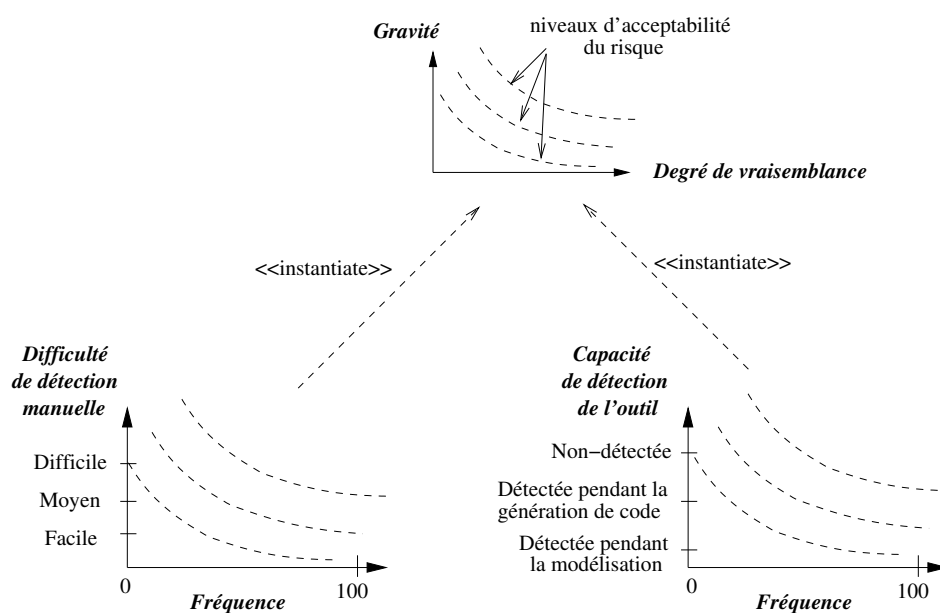


FIG. 4.13: Critères et métriques pour la gravité et le degré de vraisemblance

La mesure de la vraisemblance ne pose pas de problème : elle est établie en mesurant la fréquence d'occurrence des incohérences dans les modèles étudiés, c'est-à-dire, pour une règle donnée, le nombre de fois où elle est non respectée sur le nombre de fois où la construction en jeu est utilisée. Une incohérence peut apparaître quand le modèle emploie les constructions d'UML impliquées dans la règle de cohérence. Par exemple, la règle 486 (*"La signature d'un message dans un diagramme de séquence doit être assortie avec une opération ou une réception (qui traite un signal) contenue par la classe réceptrice"*) peut apparaître à chaque fois qu'un message est reçu dans un diagramme de séquence. Si pour 100 réceptions de message dans un diagramme de séquence, on constate 60 cas où aucune opération ou réception n'est définie dans la classe réceptrice, alors la fréquence d'occurrence de ce type d'incohérence sera évaluée à 60 %.

La figure 4.13 montre les deux critères employés pour évaluer la gravité : la difficulté de la détection manuelle et la capacité des outils UML à détecter des incohérences.

La difficulté de la détection manuelle est mesurée qualitativement (facile, moyenne, difficile). Cette difficulté est évaluée par la durée requise pour détecter les incohérences. Ce critère et les trois valeurs de sa métrique sont semblables à ceux utilisés pendant les interviews (la détection vous semble : être évidente, demander un effort mais être faisable, nécessiter une investigation trop importante pour y arriver). Cependant, le procédé d'estimation est différent : les résultats déduits des interviews viennent de la compétence d'une personne experte tandis que les résultats déduits des retours d'expérience viennent de l'analyse des modèles réels.

La capacité des outils à détecter les incohérences est mesurée selon l'étape de détection. La pire situation est quand l'incohérence n'est pas détectée, la situation intermédiaire correspond à une détection pendant la vérification du modèle par l'outil (*checker*) et la meilleure est la détection en ligne, à la saisie du modèle. En effet, il vaut mieux détecter l'incohérence aussitôt que possible.

4.3.2 Processus d'estimation

Comme cela a été précédemment expliqué, l'identification du risque a mené à la formulation de plus de 600 règles de cohérence [54]. Ces règles adressent tous les diagrammes d'UML 2.0. Pendant notre étude d'estimation par retour d'expérience, nous avons exploité seulement des règles concernant les diagrammes de classes, et de séquence et des règles de cohérence entre ces deux diagrammes. Ce choix a été fait en accord avec l'utilisation intensive de ces diagrammes dans les applications avioniques étudiées. Cette étude a pris en considération environ 350 de nos règles de cohérence.

Notre étude concerne deux des sous-systèmes d'un système de gestion du vol (*flight manager system*) d'un vrai avion :

- le "middleware" qui vise à fournir des services génériques comme la communication entre les composants physiques du système de gestion du vol ou la gestion de différentes versions d'une donnée.
- la base de données de navigation (*navigation data base, NAVDB*) qui vise à contrôler une base de données employée pour la planification de vol.

Le tableau 4.14 fournit les informations quantitatives au sujet des deux modèles considérés. Il prouve que les modèles manipulés sont tout à fait complexes. Par exemple, NAVDB était modélisé au moment de l'étude par le biais de 116 diagrammes de classes. Cette complexité était nécessaire pour fournir des résultats significatifs pour notre étude. Cependant, 3 mois à temps plein ont été nécessaires pour mener à bien cette étude. Pour conclure cette section, mentionnons que ces modèles représentent approximativement 15% des modèles finaux des sous-systèmes du système complet dont la conception s'étalera sur plusieurs années additionnelles avant d'être finie. Nos résultats permettent au procédé de développement d'être amélioré.

	Middleware	NAVDB
Diagrammes de classes	16	116
Classes	53	243
Attributs	101	824
Opérations	174	523
Dépendances	35	376
Diagrammes de séquence	72	86
Messages	491	1419

FIG. 4.14: Informations quantitatives relatives aux modèles considérés

4.3.3 Résultats

Prenons ici l'étude de l'estimation des risques d'incohérences liés à la construction d'UML appelée *Interaction Constraint* ou "contrainte d'interaction" en français afin d'illustrer un exemple extrême. En raison de l'utilisation actuelle de la version 1.4 d'UML pour modéliser les systèmes étudiés, la seule règle applicable pour cette construction est "les variables dynamiques qui participent à la contrainte doivent être possédées par l'élément connectable (*ConnectableElement*) correspondant à la ligne de vie couverte." (cf. [59] p. 529).

Les résultats suivants ont été obtenus : La *difficulté manuelle de vérification* a été évaluée à une *valeur moyenne* en tenant compte de la durée de la détection. La *capacité de détection de l'outil* a été évaluée à *non détecté*. Le résultat le plus intéressant concerne la fréquence obtenue passant en revue les deux modèles : les valeurs étaient 97% et 93% respectivement sur le middleware et les systèmes de NAVDB. Ceci signifie que la règle de cohérence n'a jamais été respectée à quelques exceptions près. Ces résultats mènent aux deux estimations de risque présentées à la figure 4.15. De nouveau, signalons que cet exemple a été choisi car il est extrême. Pour les autres constructions, le nombre d'incohérences est faible.

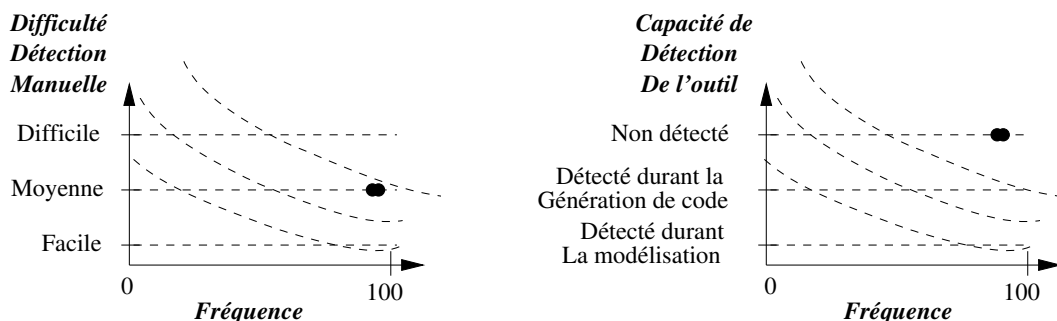


FIG. 4.15: Graphe d'acceptabilité du risque pour les contraintes d'interaction

La raison de la présence des incohérences a été étudiée. Les origines principales de

ces incohérences ont été identifiées :

- La formation des ingénieurs est une des causes. La notion de cohérence n'est pas bien connue ;
- Les documents mettant en exergue des incohérences ou permettant de les éviter tels que des guides de style ou des guides sur le processus doivent être complétés ;
- La vérification manuelle des modèles industriels est difficile parce que les modèles sont complexes mais également parce que beaucoup de règles de cohérence sont par nature complexes, comme souvent elles font intervenir diverses constructions du langage. De plus, les règles concernant le métamodèle doivent être instanciées sur les constructions concrètes et les ingénieurs ne possèdent pas une connaissance approfondie du métamodèle d'UML.

4.4 Conclusion

Nous avons présenté les besoins d'une estimation des risques d'incohérences liés à l'emploi des constructions d'UML et avons initié ce travail pour poser le cadre général des interviews et des retours d'expérience qui sont repris et approfondis dans la thèse de Roberto Lopez Toro. Celui-ci proposera également des moyens de mesures sur les modèles UML. Pour compléter ce travail, les prochaines interviews devraient porter l'effort d'estimation sur les diagrammes de machines à états qui concernent le cœur de la modélisation dynamique, puis sur les diagrammes de séquence et d'activités. Comme cela a déjà été mentionné dans le chapitre précédent, le manque de formalisation dans la spécification d'UML des liens entre les modèles comportementaux et de structure, a été relevé dans les commentaires lors des interviews. Ce manque semble être une des préoccupations de terrain les plus concrètes et les efforts de la communauté UML devraient aller dans ce sens.

La gestion des risques est une manière systématique de contrôler les risques des technologies. Notre étude se concentre sur les incohérences considérées comme des dommages affectant des modèles UML. Le travail présenté sur l'estimation des risques d'incohérences prouve que divers critères peuvent être choisis pour définir la gravité de la présence d'une incohérence. Nous avons examiné deux d'entre eux : la difficulté de détection de l'incohérence par revue manuelle ou en utilisant des outils et, leurs conséquences sur le code produit. Ces deux critères semblent être pertinents aux yeux des personnes interviewées.

Les valeurs obtenues montrent que certains des risques estimés doivent être traités pour atteindre un niveau de risque acceptable. C'est ce qui est présenté au chapitre 5.

Chapitre 5

Traitement des risques d'incohérences UML

Certains risques estimés de l'emploi de constructions d'UML nécessitent d'être traités afin d'en réduire la gravité et le degré de vraisemblance. Nous resituons dans ce chapitre l'étape de traitement dans le cadre général du management des risques en section 5.1 et nous précisons comment nous l'avons appliqué au langage UML et aux risques d'incohérences en particulier en section 5.2. Nous allons présenter deux classes de moyens permettant à la fois de réduire le degré de vraisemblance de ce risque par des techniques de prévention (section 5.3), et de réduire la gravité par des techniques de protection (section 5.4).

5.1 Traitement du risque

L'estimation du risque d'incohérence due à l'emploi d'une construction donnée montre que certaines incohérences ont été évaluées à des niveaux de risque élevé. Il devient alors nécessaire dans certains cas de les traiter. Dans cette section, nous formulons les besoins du traitement du risque dans la démarche de management des risques (cf. section 5.1.1) et nous présentons en section 5.1.2 les classes de techniques possibles dont nous développons ensuite en sections 5.3 et 5.4 des exemples d'application pour les techniques choisies dans le cadre de cette étude.

5.1.1 Besoins

En premier lieu, nous devons pouvoir identifier dans quels cas le risque peut être considéré comme élevé. Des règles de cohérence dont les niveaux de risque sont élevés vont conduire les concepteurs à se demander si les constructions mises en jeu peuvent être utilisées telles quelles ou avec certaines précautions concernant la ou les règle(s) "à risque", ou si à l'extrême, leur emploi doit être exclu. Cependant, cette exclusion conduit à perdre tous les bénéfices apportés par cette construction, par exemple en

terme de capacité de modélisation comme nous le développerons plus loin. Si on ne considère pas ce dernier cas de figure, le principal besoin est de réduire le risque de la présence d'incohérence pour rendre acceptable l'utilisation des constructions. Si le risque pour une règle est trop élevé (cf. figure 5.1), on cherchera alors à réduire le degré de vraisemblance (action Rv.) et/ou à réduire la gravité (action Rg.) pour cette règle. Mais pour pouvoir dire qu'un niveau de risque est trop élevé, il convient de définir un niveau "seuil" au-dessus duquel on considérera effectivement que le niveau de risque est devenu inacceptable pour la (ou les) règle(s) concernée(s). Le risque étant donné par la combinaison de deux paramètres, cette limite sera définie par une courbe d'acceptabilité du risque.

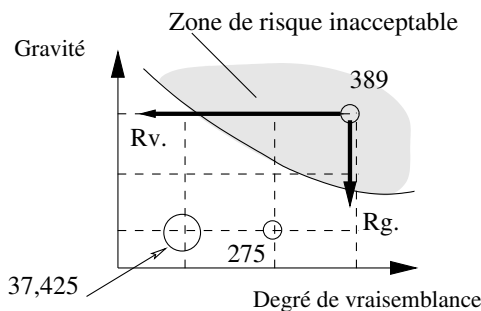


FIG. 5.1: Les deux façons de réduire le niveau de risque

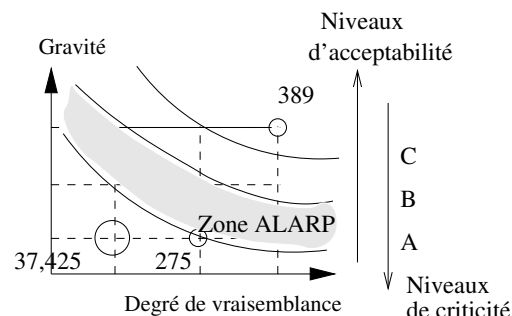


FIG. 5.2: Risque et niveaux de criticité

Dans le cadre qui nous intéresse des applications critiques, chaque application (ou chaque fonction d'une application) possède un niveau de criticité donné. Ce niveau de criticité est inversement proportionnel au niveau d'acceptabilité des défaillances de l'application ou de la fonction. Plus elle est critique, moins la défaillance est acceptable. Afin d'atteindre le niveau de criticité désiré, on contraint plus ou moins fortement le choix des technologies et leurs utilisations ainsi que le processus de conception. Les normes telles que la DO-178B [3] pour l'avionique, l'ECSS-80 [4] pour le spatial, décrivent ce type d'exigences. Elles sont souvent issues d'organismes de certification (processus) ou de qualification (produit) gouvernementaux ou de syndicats professionnels. Notre approche est similaire concernant la technologie de modélisation UML : les constructions dont les risques d'incohérence se situent au-dessus d'une courbe d'acceptabilité X donnée seront à traiter si l'application possède un niveau de criticité $1/X$.

Pour atteindre ce niveau pour toutes les constructions utilisées, des actions de traitement particulières (par exemple l'application d'un ensemble bien défini de guides) doivent être préconisées. On pourra alors déterminer pour chaque niveau de criticité, l'ensemble des guides à suivre, et ainsi adapter les actions de traitement selon le niveau de criticité. L'affectation d'un ensemble spécifique de guides à un niveau de criticité est du ressort des organismes de certification ou de qualification, mais il nous appartient de donner des éléments de choix par l'estimation des risques et la proposition de guides permettant de réduire les valeurs estimées. Les préconisations des organismes pourront

alors reposer sur des éléments argumentés issus de nos travaux . Il est à noter d'ores et déjà que l'application de ces moyens ne suffit pas pour attester que le niveau de criticité souhaité a bien été effectivement atteint. Une deuxième estimation doit idéalement avoir lieu afin de constater l'effet "réel" du traitement et d'évaluer ce que l'on appelle le risque résiduel. Il conviendrait également d'étudier les potentiels "risques induits" par nos guides. En effet, l'application de certains guides pour traiter une incohérence ne conduit-elle pas à augmenter le risque d'une autre ?

Prenons l'exemple de la figure 5.2. Considérons un risque de niveau de criticité supérieur à celui défini par la courbe "C". On le réduit en le ramenant dans une zone définie entre les courbes A et B par l'application des moyens de traitement. Si l'application ou la fonction est de niveau de criticité B, alors le risque résiduel est acceptable. Par contre, si le niveau est A, il ne l'est pas. Pour certaines applications, le seuil d'acceptabilité est plus flou et la décision d'accepter ou non le risque fait intervenir d'autres facteurs (tel que le coût de la réduction supplémentaire) si le risque résiduel n'est pas trop éloigné du seuil prévu. Dans notre exemple, cela peut être le cas entre les courbes A et B. Une zone de risque éventuellement acceptable peut être ainsi définie. On parle de zone ALARP, déjà abordée dans le chapitre 4. C'est la zone de risque "aussi faible" qu'il est raisonnablement possible d'atteindre suite à des actions de traitement. Cette zone est cependant sujette à polémique. Par ailleurs, l'étude des risques induits n'a pas été menée.

5.1.2 Classes de moyens

Nous allons présenter les quatre grandes approches de traitement du risque puis, de manière un peu plus détaillée, nous abordons les limites de l'évitement des risques en section 5.1.2.2 en illustrant la nécessité d'une analyse bénéfices/coûts pour chaque construction. Nous allons ensuite voir pourquoi nous avons choisi de nous concentrer sur le traitement par optimisation des risques dont la section 5.1.2.3 fournit une vue d'ensemble.

5.1.2.1 Quatre grandes approches de traitement

De manière générale, comme présenté dans [56], les traitements des risques sont répartis en quatre approches : le refus du risque, le transfert de risque, la prise de risque et l'optimisation du risque. Nous allons définir ces quatre approches brièvement avant de nous concentrer sur l'optimisation.

- *Le refus ou évitement du risque* (*risk avoidance* en anglais) est une décision visant à ne pas impliquer une cible (voir section 1.1.2, il s'agit dans notre cas d'un modèle) dans la source d'un risque. Il peut être obtenu en agissant sur l'exposition au risque, par exemple en interdisant l'usage de la construction.
- *Le transfert du risque* (*risk transfer* en anglais) consiste à partager avec une autre partie la charge de la perte ou du gain d'un risque. Il s'agit donc de partager les

effets des conséquences. Pour en comprendre le principe, on peut faire l'analogie avec les assurances qu'une personne prend sur un bien immobilier pour se protéger d'un risque naturel par exemple.

- *La prise de risque* (*risk retention* en anglais) consiste à accepter la charge de la perte ou le bénéfice du gain d'un risque. Ce traitement n'est pas acceptable dans le contexte des applications critiques qui nous intéressent.
- *L'optimisation du risque* (*risk optimization* en anglais) a pour but d'améliorer la valeur de l'estimation. Dans le domaine de la sécurité (traduction de *safety*), il s'agit essentiellement de réduction du risque. Celle-ci est obtenue par des techniques de prévention visant à réduire la vraisemblance de l'occurrence de l'événement dommageable ou par des techniques de protection minimisant la gravité du dommage.

Parmi ces techniques, nous avons choisi de considérer en priorité l'évitement et l'optimisation dans le cadre de notre étude.

5.1.2.2 L'évitement des risques

L'évitement du risque consiste, dans notre cas, en l'exclusion des constructions UML dont les estimations des risques associées sont trop élevées. Si cette approche était employée telle quelle, elle réduirait considérablement le nombre de constructions de base d'UML utilisables, ce qui n'est pas acceptable. De plus, ceci n'est pas satisfaisant dès lors que l'on ne fait que cacher le problème (on ne le révèle pas, car il n'est pas modélisé) et tous les bénéfices apportés par ces constructions pour la modélisation seront perdus.

L'emploi de cette technique de traitement n'est pas à conseiller mais elle a le mérite d'éveiller notre réflexion sur le fait que la présence d'incohérences ne suffit pas à conclure à l'exclusion de certaines constructions mais qu'il faut aussi considérer tous les aspects positifs apportés par ces constructions. L'estimation des bénéfices de l'emploi de chaque construction UML est importante, parce qu'un nombre important de valeurs du risque estimées se situe en réalité dans la zone ALARP. Ceci requiert de prendre en compte d'autres informations comme les bénéfices apportés par ces constructions pour montrer la correction des modèles ou les coûts de la réduction du risque d'incohérence, avant de conclure sur l'acceptabilité de leur emploi. Une balance bénéfice/coût est alors à faire. En effet, une construction n'est pas seulement dangereuse, elle peut être utilisée avec précaution et on peut faire valoir les bénéfices qu'elle apporte pour la modélisation. Un exemple de construction UML très employée et qui apporte des bénéfices indéniables pour exprimer des relations d'héritage est la relation de généralisation. Au niveau modélisation, on acceptera son emploi tout en faisant attention au risque d'introduire une dépendance cyclique entre les éléments connectés. Cette même construction peut être considérée aussi pour ses implications au niveau programmation, même si l'on sort du cadre précis de l'illustration de ce paragraphe. En effet, on peut aussi mentionner l'apport bénéfique de la possibilité de redéfinir une méthode de classe (*overriding* en anglais)

qui est à balancer avec le risque d'une possible violation du principe de substituabilité de Liskov [7] : le contrat d'une méthode héritée d'une classe mère peut ne pas être respecté par la méthode redéfinissante de la classe fille si le programmeur n'est pas attentif à la propagation implicite d'informations dont les contrats de méthodes (pré-conditions, invariants, post-conditions...) sont un exemple.

Cette analyse bénéfices/coûts est une étape à part entière du management du risque mais que nous ne détaillons pas plus ici. Cependant, elle pourrait être un sujet intéressant à traiter comme suite à cette thèse.

5.1.2.3 L'optimisation des risques

Nous considérons la maîtrise des constructions UML existantes par le traitement des risques identifiés et estimés. Il s'agit de fournir des moyens de réduire le risque existant. L'approche de traitement des risques par optimisation que nous limiterons à la "réduction des risques" est découpée en deux grandes sous-familles : la prévention et la protection du risque.

La prévention du risque vise à diminuer la vraisemblance de la présence d'incohérences dans les modèles (réduction de la probabilité). Plusieurs moyens peuvent être envisagés. Nous en citerons deux. En tout premier lieu, les notions de cohérence et d'incohérences UML doivent être enseignées afin que les concepteurs y soient vigilants. En second lieu, des guides de modélisation doivent être écrits et suivis pour empêcher l'introduction d'incohérences dans les modèles. Nous développerons ce second type de moyens.

La protection du risque vise à minimiser les effets des incohérences présentes dans les modèles (réduction de la gravité). Si comme dans le chapitre 4, la gravité est estimée par la difficulté de détection ou par l'impact de la présence de l'incohérence sur le code produit, les vérificateurs de cohérence doivent premièrement être développés pour détecter toutes les classes d'incohérences identifiées. En second lieu, nous avons noté que la détection est plus facile si des informations redondantes existent dans les modèles. Prenons par exemple, dans les diagrammes d'interaction, un des guides de protection que nous conseillons d'appliquer sur les messages et stimuli : *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 : La signature d'un message doit être la même que celle de l'élément référencé (opération ou signal)).

Comme dans les figures 5.3 et 5.4, le message entre deux lignes de vie, disons un appel d'opération qui sera représenté dans le diagramme de séquence par exemple, exposera la même information en terme de signature que celle présente dans la classe dont la ligne de vie réceptrice est une instance. Bien que l'information soit redondante, elle est utile à la détection d'une possible incohérence entre les deux diagrammes. Comme cet exemple illustratif le laisse entendre, des guides de modélisation doivent être créés et appliqués pour faciliter la détection des incohérences.

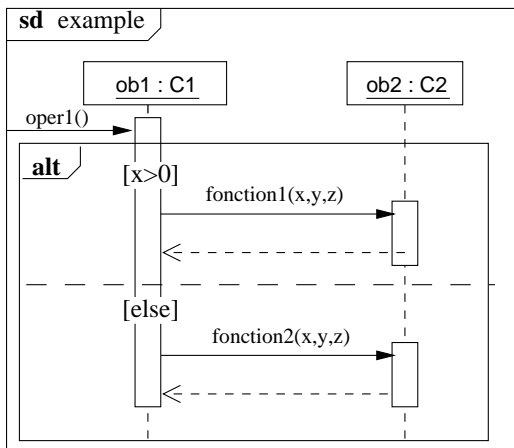


FIG. 5.3: Message dans le diagramme de séquence

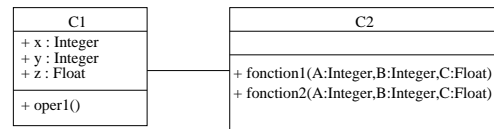


FIG. 5.4: Opération dans le diagramme de classes

Nous voulons insister sur le fait que l'effort de prévention est complémentaire à celui de protection. Même si la prévention est bien faite, c'est-à-dire si la probabilité d'occurrence de l'incohérence a été diminuée, certaines incohérences peuvent toujours être présentes. Les mesures de protection sont toujours nécessaires dans le cas d'une incohérence isolée si son effet est grave. Ces deux moyens de traitement des risques par réduction ont été considérés pour améliorer l'assurance de la sûreté des applications développées en utilisant UML et quelques exemples sont présentés en sections 5.3 et 5.4.

5.2 Application à UML

Avant d'aborder notre approche du traitement des risques d'incohérences, nous allons nous situer par rapport aux propositions existantes dans le cadre de la prévention et de la protection des risques d'incohérences (section 5.2.1) puis présenter notre façon d'aborder la question en section 5.2.2.

5.2.1 Approches de traitement des incohérences UML par optimisation

Dans les différentes approches proposées du traitement des risques d'incohérences, nous avons identifié quelques familles, parmi lesquelles nous allons présenter celles qui se rapprochent le plus de l'optimisation du risque de la présence d'incohérences et qui fournissent des moyens de prévention et/ou protection. Nous allons aborder différents guides de modélisation et guides de style généraux, guides liés au processus de développement, et les patrons de conception qui contribuent aussi à la formation de modèles cohérents.

5.2.1.1 Des guides de modélisation

Dans [11], un total de 300 guides de style de modélisation est fourni qui couvre l'ensemble des diagrammes d'UML 2.0. C'est un document assez précieux car il est un des rares livres consacrés aux guides de modélisation et mis à jour pour la version UML 2.0. Les guides présentés sont de différents types. Des conseils d'ordre graphique sur le placement des éléments dans un diagramme sont donnés pour faciliter sa lisibilité et sa compréhensibilité comme *éviter de faire se croiser des lignes de relations et éviter de représenter des lignes en diagonales, organiser spatialement les diagrammes pour une lecture de la gauche vers la droite et du haut vers le bas, placer l'acteur principal en haut à gauche d'un diagramme de cas d'utilisation, etc.* Outre des guides plus généraux (*faire un diagramme par page, montrer seulement ce qui doit être montré, etc.*) on y trouve des conseils de nommage. En voici un exemple appliqué sur la figure 3.8 de ce manuscrit : *préférer utiliser plutôt que des lettres, des numéros pour des connecteurs labellisés des diagrammes d'activités.* Il s'agit ici, d'éviter la confusion avec un nœud de flot final noté avec une croix à l'intérieur d'un cercle (X) ou avec l'état historique (H), même si dans ce cas là, cette notation appartient à un autre diagramme.

Enfin, certains guides présentés dans [11], sont mentionnés comme pouvant se contredire. Par exemple, dans les diagrammes de classes, il est conseillé d'un côté de *ne pas modéliser de code superflu, dit "de construction"*¹ (comme les opérations Get/Set, ou les attributs typés par une classe en association avec la classe considérée), qu'il ne faut pas forcément montrer avec l'outil pour ne pas surcharger le modèle, ou le rendre fastidieux à lire. La figure 5.5 représente ainsi un attribut de classe par le biais de la relation graphique d'association entre deux classes.

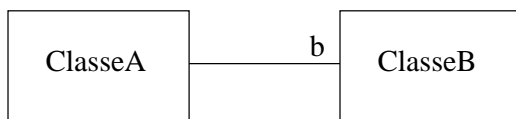


FIG. 5.5: Association représentée plutôt que le type d'un attribut

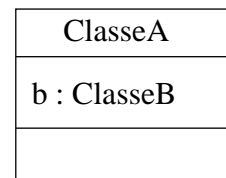


FIG. 5.6: Attribut de classe typé par une classe en association

D'un autre côté, il est conseillé de *remplacer les lignes de relation par des types d'attributs de classes*, ce qui a pour effet de simplifier le diagramme si on ne veut pas voir apparaître trop de relations. La figure 5.6 montre un attribut de classe typé par une classe qui possède une relation d'association avec la première.

Ces deux guides semblent se contredire et c'est au concepteur que revient le jugement d'appliquer l'un ou l'autre des deux guides, en prenant comme critère "lequel des deux guides améliorera plus le diagramme ?" Dans ce cas, une bonne méthode empirique à suivre, est de se demander si les personnes qui vont manipuler la classe sont fami-

¹Do not model scaffolding code

lières avec la classe ; alors, dans ce cas, mieux vaut la montrer comme un type (comme un attribut typé sans représenter la boîte de classe). Dans l'autre cas, c'est-à-dire si le concept est nouveau, mieux vaut représenter la classe nouvelle et la relation d'association entre les deux boîtes de classes ; ceci permet de mieux visualiser en captant le regard et en mettant l'accent sur la connexion entre deux classes. [47] apporte une autre politique pour choisir entre le texte ou les associations pour représenter les attributs. Sa proposition est d'utiliser la notation sous forme de texte si le type est un type de données (type qui désigne les objets dont l'identité unique n'est pas importante, comme les types de données courants suivants : Booléen, Caractère, Chaîne de caractères, Date, Heure, Adresse, Couleur, Numéro de téléphone, types énumérés, etc.). Dans ce cas, la notation textuelle (attribut typé mentionné dans la classe) est préconisée pour les objets correspondant à des types de données. Dans le cas contraire, il est conseillé d'utiliser les associations. Cet exemple montre combien une seule stratégie de modélisation ne peut pas forcément convenir dans tous les cas.

5.2.1.2 Des guides liés au processus de développement

Selon Grady Booch dans [21], les 4 rôles d'un processus de développement de logiciel sont :

- fournir un guide pour les activités d'une équipe,
- spécifier quels artefacts devraient être développés et quand ils devraient être développés,
- diriger les tâches individuelles des développeurs, ainsi que celle de l'équipe comme un ensemble,
- offrir des critères pour visualiser et mesurer les produits du projet et les activités.

Nous n'allons pas dans cette partie aborder l'aspect gestion d'une équipe de développement, bien qu'il soit un facteur certainement très important dans le maintien de la cohérence d'une modélisation UML impliquant plusieurs modeleurs. Devant l'importance de cette question et notre peu d'expertise, nous n'avons pas concentré nos efforts dans la tentative d'établissement d'une liste exhaustive de guides à suivre pour se rapprocher d'un processus de développement comme le RUP² [41]. Nous pouvons tout juste mentionner quelques exemples de guides issus de la littérature et relevant du deuxième rôle attribué par Booch à un processus de développement.

La première catégorie de guides concerne donc l'identification des constructions d'UML pour exprimer les artefacts à développer. Dans [11], une bonne partie des guides sont d'ordre méthodologique et sont définis pour conseiller de modéliser quelque chose (un comportement particulier par exemple) avec une construction ou un diagramme particulier (par exemple une machine à états) et de le faire que si cela est vraiment nécessaire. La forme de ces guides est : *Créer un diagramme de ... pour décrire ... avec* comme exemple *"Ne pas utiliser de diagramme de communication pour modéliser un*

²Rational Unified Process

flot de processus”, mais “*plutôt considérer un diagramme d’activités*”. Ces guides font référence à une bonne pratique de modélisation agile [10] qui préconise d’appliquer le(s) bon(s) artefact(s).

La deuxième catégorie concerne plutôt l’ordre dans lequel développer des artefacts. Si [18] présente aussi des guides correspondant à la première catégorie, il les inclut en plus dans une démarche complète appliquée à des modèles produits par Siemens. Les heuristiques définies concernent les modèles d’analyse en commençant par des conseils sur la manière d’organiser le modèle, comment définir les cas d’utilisation, modéliser les objets du domaine. Puis d’autres heuristiques se rapportent aux modèles de conception. Le couplage entre les deux types d’heuristiques est enfin mentionné et l’on peut citer deux exemples : *Une classe d’interface devrait dériver d’une classe d’analyse aux limites*³ (les classes d’analyse aux limites sont les interfaces entre acteurs et cas d’utilisation), ou bien : *chaque classe dans un modèle de conception devrait pouvoir être tracée à un cas d’utilisation dans le modèle d’analyse*.

Les guides de modélisation et de processus dits “utilisateurs” dans [22, 11] donnent des conseils d’ordre méthodologique, stylistique afin de produire des modèles UML à la fois bien formés et complets. Cependant, aucun de ces ouvrages n’aborde systématiquement la question des guides d’un point de vue basé explicitement sur les incohérences. C’est ce que nous présentons dans ce chapitre.

5.2.1.3 Des patrons de conception

Une des pratiques les plus répandues parmi les concepteurs est celle des patrons de conception [33, 47] plus connus sous leur terme anglais de *design patterns*. Ils trouvent leur origine dans les travaux de l’architecte Alexander [9]. Ils caractérisent de manière nommée des problèmes fréquents et connus de conception et spécifient des solutions claires et élégantes à ces problèmes. Ils sont accompagnés de conseils sur la manière de les appliquer. La finalité des patterns n’est pas d’exprimer de nouvelles notions en matière de conception. Au contraire, ils tentent de codifier des connaissances bien établies. Plus ils sont affinés et largement utilisés, mieux c’est. Un expert en objets les trouvera très élémentaires et très familiers. Les bénéfices apportés au développement sont de divers ordres : en premier lieu, les connaissances des informaticiens plus expérimentés sont rendues accessibles aux concepteurs novices. Grâce à ces connaissances, ceux-ci peuvent identifier une situation où l’utilisation suggérée par un patron de conception serait bénéfique. En deuxième lieu, ils guident les développeurs dans le choix de la mise en œuvre la plus adaptée à cette solution. Enfin, ils formalisent le savoir-faire d’une entreprise ou d’une équipe de développement.

Dans [47], les patterns GRASP⁴ sont présentés. Ils nomment et décrivent un certain nombre de principes de base pour affecter les responsabilités aux objets. On peut citer un

³analysis boundary class

⁴General Responsibility Assignment Software Patterns

exemple de pattern pour illustrer comment un pattern permet d'éviter une incohérence. Le pattern "Créateur" pose ce problème : "qui crée une instance d'une classe A ?" La solution donnée par ce pattern est la suivante : affecter à la classe B la responsabilité de créer une instance de la classe A si une ou plusieurs des conditions suivantes est vraie (plus il y en a, mieux c'est) :

- B contient ou agrège des objets A ;
- B enregistre des objets A ;
- B utilise étroitement des objets A ;
- B possède les données d'initialisation des objets A.

Ce pattern permet de donner une autre résonance, issue de la pratique, à la règle de cohérence définie dans [54] par ce qui suit : *"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."* L'incohérence qui consisterait à vouloir créer un objet instance d'une classe A par un objet instance d'une classe B, A étant la classe composante et B la classe composée, est ainsi prévenue par l'application du pattern "Créateur".

5.2.1.4 Un cadre de gestion des incohérences

Bien que l'application de guides permette de réduire le nombre d'incohérences effectives, nous partageons l'idée qu'un canevas d'application (ou cadre) de la gestion des incohérences⁵ beaucoup plus large devrait motiver d'autres pistes de recherche. Il pourrait prendre comme référence le cadre présenté dans [28] comme une personnalisation pour UML de la stratégie proposée par [32]. En opposition avec la manière traditionnelle de chercher à éliminer les incohérences, [28] expose un cadre de travail qui cherche plutôt à gérer les incohérences car dans beaucoup de cas, des modèles incohérents ne posent pas de problème jusqu'à une date bien ultérieure à la saisie du modèle. Par exemple, une incohérence peut être tolérée, voire même désirée quand le modeleur est encore en train d'évaluer des solutions alternatives à son problème sans se compromettre avec l'une d'entre elles. Il y a aussi des incohérences temporaires résultant d'un changement incrémental dans un modèle (et qui sont modifiées presque "dans la foulée"). Les auteurs exposent comme idées directrices de leur cadre que la cohérence doit être préservée quand cela est désiré, et les corrections des incohérences doivent être faites seulement quand cela est nécessaire (d'habitude avant de procéder à des actions qui doivent être basées sur une information cohérente). Ainsi, [76] présente une stratégie pour tolérer les incohérences, avec le prototype de la suite d'outils Fujaba, qui permet la détection et la résolution d'incohérences à différents moments.

Une fois l'incohérence détectée, se pose le problème de la visualisation de l'incohérence et de sa résolution. Les options pour la résolution d'une incohérence sont multiples : on peut ignorer, réduire sa gravité, reporter sa prise en charge, l'améliorer ou la résoudre

⁵CMF : Consistency Management Framework

complètement [18].

Enfin on peut désirer, pour compléter un cadre complet de gestion de la cohérence, avoir la capacité de diagnostiquer l'incohérence et de présenter à l'utilisateur un rapport détaillé de l'analyse de la cohérence comme l'outil DesignAdvisor le permet [18]. De plus, un bon outil devrait permettre d'aider à la modélisation en donnant des conseils au fil de l'eau pour savoir comment restaurer une incohérence ou comment améliorer la qualité du modèle.

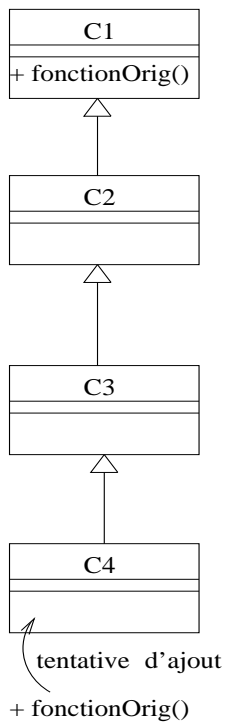
5.2.2 Approche proposée

A défaut de proposer un cadre complet de gestion des incohérences, notre contribution se focalise sur l'étape de traitement des risques par optimisation. Une fois que les types d'incohérences ont été identifiés et que l'on a donné des valeurs de probabilité et de gravité à chacun d'eux, on peut tenter de réduire le risque par un traitement spécifique. Pour un niveau de criticité donné, les valeurs maximales de degré de vraisemblance et de gravité sont spécifiées (notion de niveau d'acceptabilité du risque). Nous avons signalé que l'exclusion des constructions UML concernées faisait perdre leurs bénéfices à la modélisation. Pour cette raison, nous préférons réduire le risque en réduisant les valeurs estimées de vraisemblance (prévention) et de gravité (protection).

Les guides qui vont être présentés en sections 5.3 et 5.4 ont été élaborés pendant la phase d'identification des règles de cohérence et de nouveaux guides continuent d'être ajoutés au document [54]. Nous devons avoir à l'esprit qu'une construction de base d'UML 2.0 donnée met en jeu un ensemble de règles de cohérence. Cette correspondance a été faite par construction du document [54] puisqu'à chaque élément du métamodèle UML 2.0 est associé un ensemble de règles. Il faut aussi noter que la plupart des règles de cohérence identifiées peuvent concerner une ou plusieurs constructions de base d'UML 2.0 (cas des règles inter-diagrammes, entre autres exemples).

Pour réduire la gravité et le degré de vraisemblance d'une incohérence, nous pouvons appliquer des guides de prévention et/ou de protection. Afin de bien faire comprendre la différence entre guide de prévention et guide de protection, nous allons prendre un exemple introductif aux deux sections qui vont suivre, inspiré de [56]. L'incohérence illustrée relève de la règle qui dit que tous les membres d'un espace de nommage doivent pouvoir être distingués.

Dans le cas d'une hiérarchie de généralisation à plusieurs niveaux (voir figure page suivante), de nombreuses méthodes héritées ne sont pas toujours utilisées dans les sous-classes, ce qui peut être à l'origine, de la part du concepteur, d'une mauvaise connaissance de l'ensemble des méthodes héritées dans une classe donnée. En effet, lorsqu'il décide d'ajouter une méthode nouvelle dans une sous-classe, il peut redéfinir involontairement une méthode de même signature (même nom et paramètres) qu'une méthode déjà héritée.



Ici les classes C2, C3 et C4 héritent toutes de la méthode `fonctionOrig()` de C1, mais le concepteur, s'il n'a plus une mémoire précise des méthodes héritées, peut souhaiter définir une méthode `fonctionOrig()`, nouvelle pour lui (qui sera en fait une redéfinition de celle héritée de C1) et celle-ci possède une fonctionnalité différente de celle héritée. L'incohérence est bien que l'on se retrouve avec deux méthodes pensées pour être différentes, mais que l'on ne pourrait pas distinguer dans l'espace de nommage que constitue la classe C4.

Un guide de prévention de cette situation serait formulé comme suit : *Lorsqu'on veut rajouter une nouvelle opération, il est conseillé de parcourir l'arbre d'héritage et de vérifier que dans les classes ancêtres, il n'y a pas d'opération existante ou héritée qui ne possède pas déjà le même nom et plus généralement la même signature. Si ce n'est pas le cas, il faudra alors donner un autre nom à cette nouvelle opération.*

Considérons que la gravité est évaluée par la difficulté de détection de l'incohérence. Afin de faciliter la détection d'une incohérence de ce type, un guide de protection formulerait : *Toute redéfinition d'une opération doit utiliser explicitement la propriété `{redefines operation}`*. Ainsi lorsque deux opérations ayant la même signature et non munies de la propriété `{redefines operation}` apparaissent dans un niveau d'héritage, l'incohérence est détectée.

5.3 Prévention

5.3.1 Principes

La prévention du risque a pour but de prévenir la présence d'une incohérence. Cela implique l'apprentissage de la notion d'incohérence et l'écriture de guides de modélisation clairs et aussi faciles que possible à appliquer. Dans [54], nous proposons des guides de prévention et nous allons en extraire ici quelques exemples.

D'après les résultats de notre étude, nous avons distingué plusieurs catégories de guides de prévention que l'on peut classer selon le type d'action que le guide propose de faire sur la construction UML considérée. Les guides ont pour but d'éviter les problèmes d'incomplétude ou d'incohérence (notions présentées en section 1.2.1), en proscrivant l'emploi de certaines notations ambiguës (section 5.3.2), en incitant les utilisateurs à spécifier des valeurs optionnelles ou implicites (section 5.3.3), en adoptant des conventions de nommage (section 5.3.4) et d'agencement graphique (section 5.3.5), et en appliquant des guides de modélisation (section 5.3.6).

5.3.2 Ne pas utiliser une notation

La première catégorie de guides tient plutôt au principe de refus du risque, en conseillant de ne pas utiliser une notation pourtant permise par la spécification d'UML, car elle est source probable d'incohérence.

5.3.2.1 Exemple des *pins*

La figure 5.7 représente deux notations équivalentes de flux d'objets entre deux actions dans les diagrammes d'activités. La sous-figure i) représente un flux d'objets en utilisant la notation des *pins* (broches en français) que nous désignerons par la terminologie anglaise *pin*. La sous-figure ii) utilise la notation d'un nœud objet.

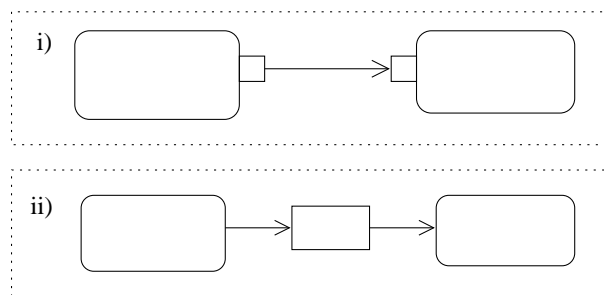


FIG. 5.7: Représentations équivalentes de flux d'objets

La spécification d'UML [59] précise que la situation dans laquelle le pin de sortie d'une action est connecté au pin d'entrée de même nom d'une autre action peut être montrée par la deuxième notation. Le pin autonome (*standalone pin*) ainsi formé correspond dans le modèle sous-jacent à un pin de sortie et un pin d'entrée. Mais cette représentation sous-entend que les pins d'entrée et de sortie sont de même type. Une incohérence se produirait si l'on relie par la représentation d'un pin autonome des pins qui ne sont pas de même type. De cette possible incohérence découle notre guide : *ne pas utiliser la notation pin autonome (standalone pin) entre deux actions si les pins de sortie et d'entrée ne sont pas du même type.*

5.3.2.2 Exemple de la fusion des nœuds de décision et de fusion

La spécification d'UML autorise que graphiquement, l'on puisse combiner un nœud de décision et un nœud d'interclassement (ou de fusion) de façon à ce qu'ils partagent le même symbole dans un diagramme d'activités (voir figure 5.8). Il est donc possible que graphiquement un nœud de décision ait plusieurs arcs entrants alors qu'une de nos règles de cohérence précise que *tout nœud de décision doit avoir exactement un arc entrant*. De la même façon, *un nœud d'interclassement (ou de fusion) doit avoir exactement un arc sortant*.

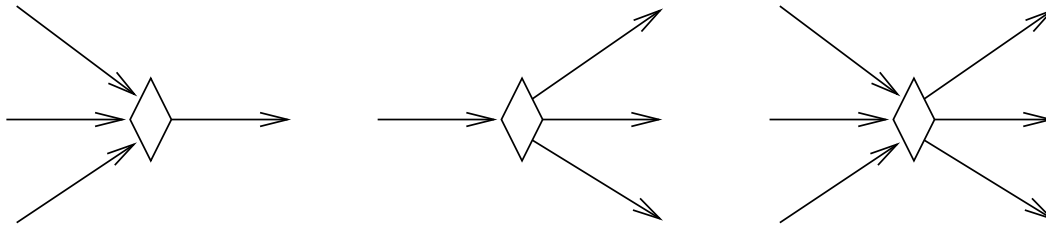


FIG. 5.8: Noeud de fusion, de décision, et les 2 combinés

Notre guide de prévention formule simplement *d'éviter de fusionner un nœud de décision et un nœud d'interclassement*.

5.3.3 Forcer l'expression d'une information supplémentaire

La deuxième catégorie est celle qui consiste en *forcer l'expression d'une information supplémentaire* dans le modèle dans le sens où cette information est non spécifiée car elle est soit optionnelle soit supposée connue car elle possède une valeur par défaut. Le rajout de l'information augmente l'expressivité du modèle. Le modèle peut être incomplet sans l'information, ce qui peut conduire parfois à une incohérence lorsque le modèle est mis en rapport avec un autre modèle.

Cette catégorie de guides correspond à plusieurs cas de figure :

- A) Si l'information est optionnelle ;
- B) Si l'information est supposée connue parce qu'il existe une valeur par défaut ;
- C) Si une propriété doit être appliquée ;
- D) Si on veut éviter de définir une partie de modèle vide.

Nous allons à présent détailler ces quatre catégories.

A) **Si l'information est optionnelle, il s'agit de rendre explicite** un lien graphique.

Certains modèles sont intentionnellement laissés incomplets de manière temporaire, tant que le modelleur n'a pas apporté au modèle l'information adéquate. Mais si cette information manque au modèle final, elle peut se traduire par une incohérence. Parmi les moyens pour rendre un modèle complet, il faut citer le fait de forcer la représentation des liens graphiques qui doivent être précisés.

Parmi plusieurs exemples possibles, en voici deux :

Entre un composant et une interface Un composant définit un comportement en termes d'interfaces fournies ou requises. Considérons cette règle de cohérence : *Une interface fournie par un composant doit être soit implantée par le composant lui-même ou par un des classificateurs qui le réalisent, ou alors elle doit être le type d'un port fourni du composant.*

Un modèle incomplet peut laisser une interface sans composant qui la réalise ou sans composant qui la requiert. 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* (voir figure 5.9) *ou depuis un des classificateurs qui le réalisent via un connecteur de délégation en explicitant le type du port requis du composant* (voir figure 5.10). De la même façon, pour les interfaces fournies, il est conseillé de *représenter la relation de réalisation du composant vers l’interface fournie* (non représentée dans la figure 5.9) *ou de l’interface fournie vers le port fourni par le composant*. Ce guide est appliqué dans la figure 5.10 où les interfaces sont bien reliées aux ports fournis et requis du composant, alors que dans la figure 5.9, il manque la précision de l’interface fournie par le composant.

De plus, pour rajouter une bonne pratique à appliquer de manière systématique à tous les diagrammes, on définit comme convention le placement des interfaces requises à droite et des interfaces fournies à gauche (cf. [11]).

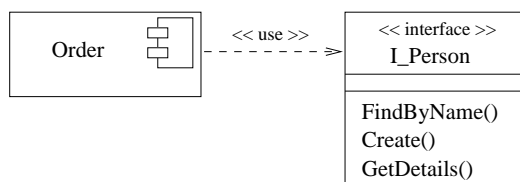


FIG. 5.9: Un composant avec une interface requise, spécifiée par dépendance d’usage

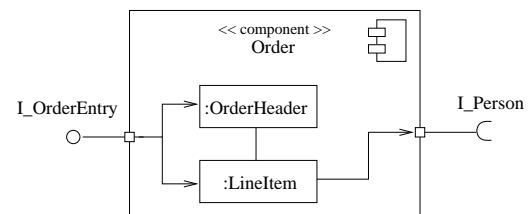


FIG. 5.10: Un composant avec ses interfaces spécifiées par l’intermédiaire de ports

Entre une collaboration et le classificateur rattaché La notion d’occurrence de collaboration définie dans [59], qui a été renommée en “utilisation de collaboration” (*collaboration use* en anglais) dans [61, 62] donne lieu à quelques précisions pour différencier la notation d’une collaboration, de celle d’une utilisation de collaboration et pour pouvoir faire le lien entre les deux notations.

Une utilisation de collaboration représente une utilisation particulière d’une collaboration pour exprimer les relations qu’il existe entre des propriétés d’un classificateur. Une utilisation de collaboration représente l’application du patron décrit par une collaboration à un contexte spécifique, en faisant la correspondance entre des entités spécifiques de ce contexte et les rôles de la collaboration. Selon le contexte, ces entités peuvent être des caractéristiques structurelles d’un classificateur, des spécifications d’instance, ou même des rôles dans une collaboration plus générale.

Il peut y avoir plusieurs utilisations 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 utilisations de la même

ou de différentes collaborations. Les figures 5.11 et 5.12 montrent des exemples de collaboration.

La collaboration “vente” (**Sale**) définit les deux rôles “acheteur” (**buyer**) et “vendeur” (**seller**) qui doivent être joués par des instances. La figure 5.12 montre la collaboration **BrokeredSale** qui est une collaboration entre trois rôles (**broker**, **producer**, **consumer**). **BrokeredSale** consiste en deux utilisations de la collaboration **Sale**, nommées **wholesale** et **retail**, et montrées par une ellipse en pointillés. L'utilisation de la collaboration “vente en gros” (**wholesale**) indique une vente (collaboration **Sale**) dans laquelle le courtier (**broker**) joue le rôle de l'acheteur et le producteur (**producer**) joue le rôle du vendeur. De la même façon, dans l'utilisation de collaboration “vente en détail” (**retail**), le courtier joue le rôle du vendeur, et le consommateur (**consumer**) celui d'acheteur.

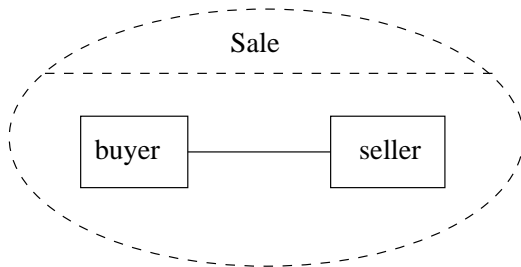


FIG. 5.11: La collaboration **Sale**

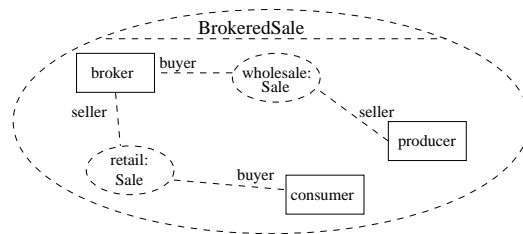


FIG. 5.12: La collaboration **BrokeredSale**

L'incohérence possible tient au non-respect de cette règle : *Tout rôle de la collaboration doit être délimité dans l'utilisation de la collaboration à un élément connectable du classificateur ou de l'opération.* Une incohérence serait qu'un des rôles de la collaboration **Sale** ne soit pas délimité dans l'utilisation de la collaboration **wholesale**. Concernant le première notation que nous voulons illustrer à savoir le lien entre une collaboration et le classificateur rattaché, nous conseillons, chaque fois que cela est possible, de *faire figurer la flèche de dépendance de la collaboration vers le classificateur qu'elle représente avec le mot clé optionnel « occurrence », et de montrer explicitement les délimitations de rôle par des dépendances.* Ceci permet de préciser que la collaboration est utilisée dans le classificateur et de plus, les délimitations de rôle, sont montrées explicitement par des dépendances. Si un troisième rôle était défini dans la collaboration **Sale**, alors il faudrait vérifier que toutes les délimitations de rôle existent bien dans l'utilisation de la collaboration.

Sur la figure 5.13 tirée de [59], la collaboration **Sale** est reliée au classificateur **BrokeredSale**, qui se trouve être ici aussi une collaboration, ici par souci de clarté, seulement représentée partiellement (il manque le rôle **consumer** de la figure 5.12). La deuxième notation qui peut être utilisée ne montre pas explicitement les délimitations de rôle mais juste le lien entre l'utilisation de collaboration et le classificateur rattaché. Nous conseillons, chaque fois que cela est possible, de *faire figurer la flèche*

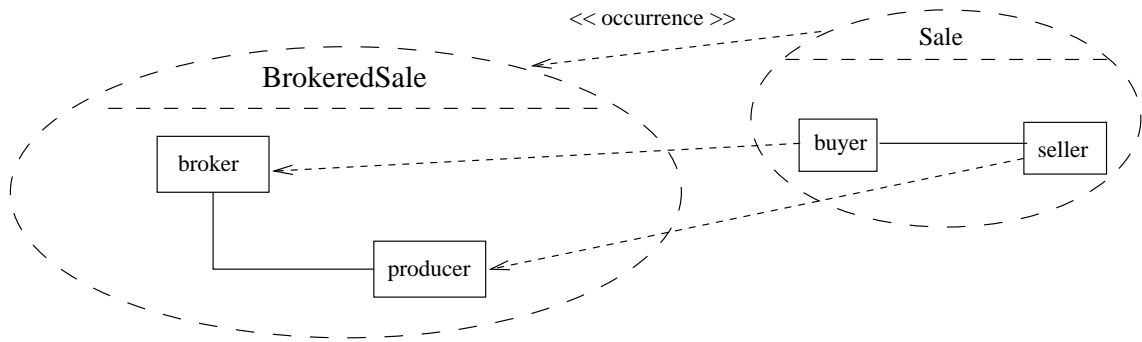


FIG. 5.13: La collaboration `Sale` utilisée par le classificateur `BrokeredSale`

de dépendance de l'utilisation de collaboration vers le classificateur par lequel elle est utilisée avec le mot clé optionnel « represents ». Ceci permet de préciser que le classificateur est rattaché à cette utilisation de collaboration. Sur la figure 5.14 tirée de [59], l'utilisation de collaboration anonyme `:RealizeDisplayBehavior` est reliée au classificateur `Window`.

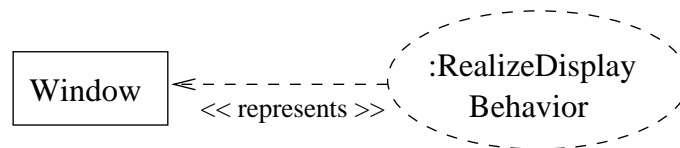


FIG. 5.14: Une utilisation de collaboration reliée à un classificateur avec le mot clé « represents »

Selon le degré de détail que l'on désire, on pourra adopter l'une ou l'autre des deux notations.

- B) **L'information dans le modèle de départ n'est pas spécifiée car supposée connue parce qu'il existe une valeur par défaut.** Il s'agit alors de la définir explicitement en affichant la valeur par défaut. Le problème des valeurs par défaut, outre le fait qu'au bout d'un certain temps on puisse en oublier leur valeur, est qu'elles peuvent masquer des incohérences.

Prenons l'exemple simple de la visibilité des éléments d'un paquetage. 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 par défaut visible en dehors du paquetage dans le cadre d'une importation d'élément (cas de la figure 5.15) ou d'une importation de paquetage.

Supposons que le concepteur ait oublié de donner à la classe B la visibilité privée. La règle de cohérence suivante ne pourrait pas être vérifiée : *l'élément importé doit être visible par l'espace de nommage qui importe l'élément*. Si la visibilité n'est pas

précisée, le problème qui se pose de plus ici est que l'incohérence de la tentative d'importation est masquée.

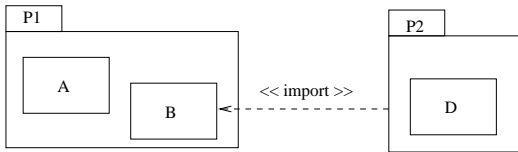


FIG. 5.15: Éléments empaquetable sans visibilité

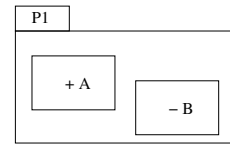


FIG. 5.16: Éléments empaquetable avec visibilité

Afin de prévenir un accès non autorisé à un élément du paquetage, le guide de prévention conseille de *préciser systématiquement la visibilité des éléments d'un paquetage et ne pas laisser d'élément sans visibilité spécifiée explicitement*. Dans ce cas (figure 5.16), si le guide avait été appliqué et qu'un signe '-' avait été mis devant le nom de la classe B, l'importation de l'élément B n'aurait pas été possible.

- C) Il peut aussi s'agir de **forcer l'application d'un mot clé à un élément si la propriété doit être appliquée**. C'est l'exemple du mot clé `{abstract}` qui doit être appliqué si la classe est une classe abstraite de la conception.

Dans l'exemple de la figure 5.17, la non-précision de la propriété `{abstract}` pourrait faire croire que la classe `Vehicle` n'est pas abstraite et qu'elle est donc instanciable. Or une bonne conception devrait préciser que cette classe est abstraite et donc par définition qu'elle ne peut pas être instanciée et que seules ses classes filles peuvent l'être. En précisant la propriété `{abstract}` sur la classe `Vehicle` (figure 5.18), l'incohérence prévenue est celle où l'on chercherait à l'instancier dans un diagramme de communication par exemple.

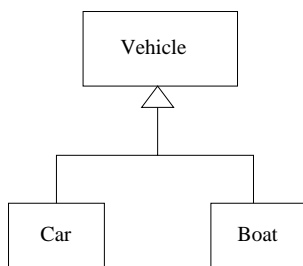


FIG. 5.17: Classe abstraite non explicite

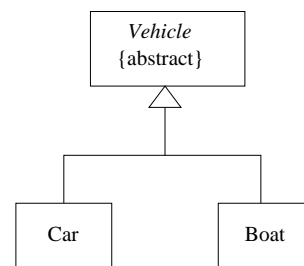


FIG. 5.18: Classe abstraite explicite

- D) Enfin, on peut donner l'exemple d'un **guide évitant de définir une partie de modèle vide qui ne servirait à rien**. Dans un diagramme d'interaction, 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 si la contrainte d'interaction est évaluée à vrai, soit rien ne se produit, c'est-à-dire que l'on ne rentre pas dans le comportement optionnel. Mais un fragment combiné d'opérateur d'interaction `opt` qui lui-même serait vide n'a pas de

sens (voir figure 5.19) puisque si l'on définit quelque chose à faire optionnellement, il faut que ce traitement soit effectivement spécifié.

En généralisant ceci à tous les fragments combinés, on peut formuler le guide suivant : *nous conseillons de toujours faire figurer au moins une occurrence d'événement à l'intérieur d'un opérande d'interaction d'un fragment combiné.*

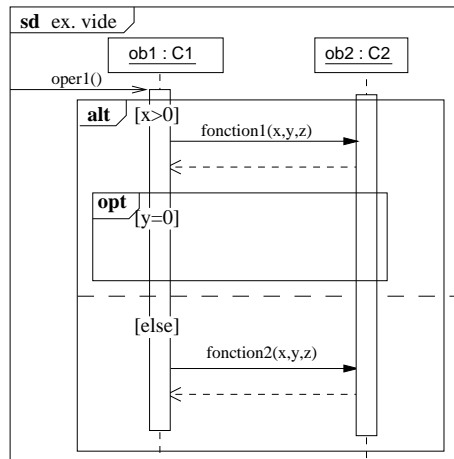


FIG. 5.19: Fragment optionnel vide

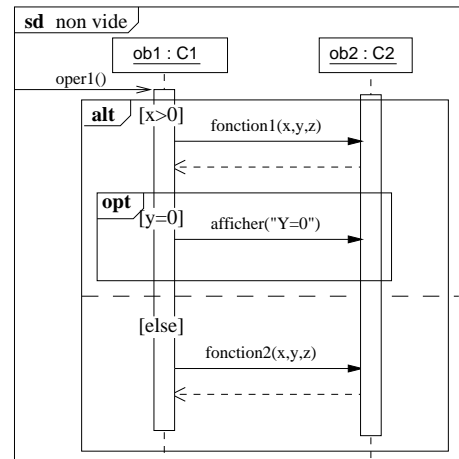


FIG. 5.20: Fragment optionnel non vide

La figure 5.20 montre un exemple de l'application de ce guide.

Parmi les recommandations générales à ces deux premières sections (5.3.2 et 5.3.3), on peut rajouter un guide qui est celui de préférer utiliser la dernière notation définie par la spécification d'UML et de délaissier les anciennes notations (comme par exemple « uses », « includes », « extends » des diagrammes de cas d'utilisation UML 1.x qui ont été remplacées en UML 2.0 par « use » et « extend »).

Ceci est aussi illustré dans le guide qui va suivre concernant la relation de manifestation. Le guide proposé dit que : *lorsque l'on veut lier un artefact à l'élément empaquetable qu'il manifeste, on conseille de faire figurer explicitement le lien de dépendance avec le mot clé « manifest » de l'artefact vers l'élément empaquetable.* Auparavant (versions d'UML 1.x), on utilisait le mot clé « implementation », mais comme il y avait plusieurs emplois possibles, un mot clé singulier a été proposé.

Pour conclure, le but de cette première catégorie de guides est d'éviter l'implicite et donc de rendre explicite une information utile en levant des sous-entendus (valeurs par défaut non représentées, par exemple).

5.3.4 Adopter des conventions de nommage

La troisième catégorie de guides de prévention sont des conventions de nommage, auxquelles nous conseillons de se tenir, une fois adoptées. Deux exemples illustratifs sont fournis.

5.3.4.1 Exemple de l'alias optionnel de la relation d'importation d'élément

Dans l'exemple du chapitre 3 sur les relations (section 3.2.1.2), nous avons présenté une nouvelle règle de cohérence concernant l'importation d'élément. *"Si un alias est spécifié, aucun nom d'élément appartenant à l'espace de nommage importateur ne doit être identique à l'alias spécifié."*

Nous avons défini deux guides de prévention pour l'alias optionnel qui peut être donné à cette relation d'importation d'élément. A l'origine, la possibilité de donner un alias correspondant au nom de l'élément importé dans l'espace de nommage source de la relation d'importation d'élément répond au besoin de rebaptiser un élément dans un autre espace de nommage et d'éviter des conflits de nommage.

Mais en pratique, et dans le cas où on ne spécifie pas d'alias, il est possible de créer dans l'espace de nommage importateur un élément de même nom que l'élément importé. Mais on perd l'utilité de l'importation d'élément, puisque l'on devra faire appel au nom qualifié de l'élément importé pour le différencier du nouvel élément créé dans l'espace de nommage importateur. De ce fait, nous avons défini ce guide : *Si aucun alias n'est spécifié, nous conseillons de ne pas créer dans l'espace de nommage importateur un élément de même nom qu'un élément déjà importé.* La figure 5.21 montre un exemple qui ne respecte pas ce guide.

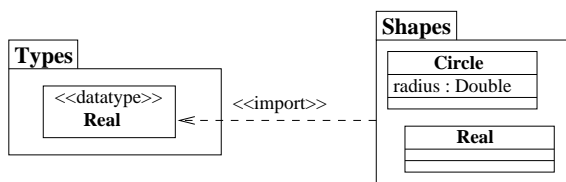


FIG. 5.21: Importation d'élément sans alias

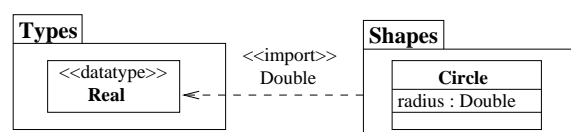


FIG. 5.22: Importation d'élément avec alias

Si l'on considère maintenant le cas où un alias est déjà spécifié, on peut formuler ce guide afin toujours d'éviter un conflit de nommage : *Si un (ou plusieurs) des alias est(sont) déjà spécifié(s), nous conseillons que les éléments que l'on pourrait ajouter à l'espace de nommage importateur ne portent pas le même nom que l'(les) alias déjà existant(s).*

Dans la figure 5.22, tirée de [59] qui respecte bien ce dernier guide, il vaut donc mieux ne pas ajouter de classe nommée `Double` dans l'espace de nommage importateur, à savoir dans le paquetage `Shapes`.

5.3.4.2 Exemple des portes

Un autre exemple concerne les portes (*gate* en anglais) dans les diagrammes d'interaction qui correspondent aux points de connexion entre un message extérieur à un fragment d'interaction et un message appartenant à ce fragment d'interaction.

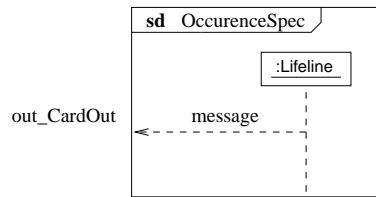


FIG. 5.23: Porte nommée d'une spécification d'occurrence

Cet exemple n'est qu'un exemple parmi d'autres pour montrer que bien que cette construction soit peu utilisée, elle n'échappe pas aux conventions de nommage (guide de style) que l'on peut formuler. Le guide précise que *les portes peuvent être identifiées par un nom s'il a été spécifié ou les portes peuvent être nommées avec un identificateur construit en concaténant la direction du message avec le nom du message (par exemple, out_CardOut) comme conseillé dans la spécification d'UML [59]*. La connaissance de la direction du port encourage sa bonne utilisation. La figure 5.23 est un exemple générique.

5.3.5 Veiller à l'agencement graphique des éléments dans un diagramme

La quatrième catégorie de guides concerne l'agencement graphique des éléments dans un diagramme qui, s'il est bien pensé, permet de le rendre plus lisible du premier coup d'œil. Nous ne donnerons pas ici d'illustration du degré d'illisibilité que l'on peut atteindre, mais nous en appelons aux premières expériences de modélisation de chacun. Un seul exemple de guide sera pris concernant les ensembles de généralisation.

Un ensemble de généralisation (*Generalization Set* en anglais) décrit la façon de partitionner l'ensemble des classificateurs qui sont enfants d'un même parent via une relation d'héritage.

Le fait de répéter le nom de l'ensemble de généralisation pour chaque fils multiplie à la fois le nombre de flèches de généralisations et la quantité d'information présente dans le modèle, en disséminant les catégories (voir figure 5.24).

Il est simplement une question de bon sens "graphique" de vouloir simplifier les diagrammes et d'appliquer ce guide : *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.*

Ceci permet de plus de pouvoir préciser la propriété qui s'applique à l'ensemble de généralisation (voir figure 5.25). La propriété `{disjoint}` signifie que l'intersection des sous-types est vide et la propriété `{incomplete}` signifie que la réunion des sous-types n'est pas égale au type. Si l'on réunit toutes les instances de l'ensemble de généralisation `EntertainmentProfile`, on n'obtient pas tous les profils existants de loisirs pour une personne. Quatre couples de propriétés sont exclusivement possibles en UML 2.0. La spécification des couples permet la vérification des propriétés.

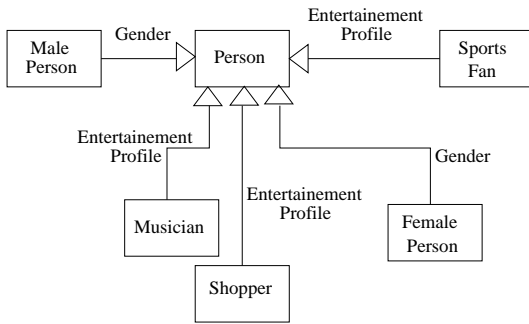


FIG. 5.24: Arbre de généralisation non agencé

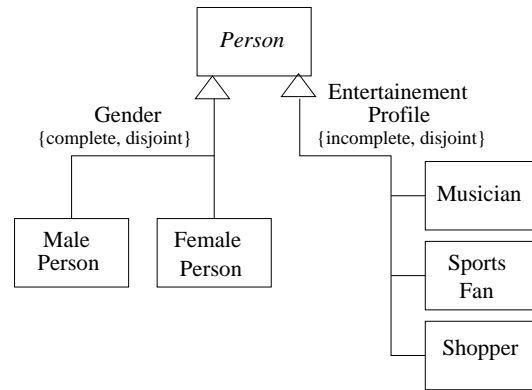


FIG. 5.25: Arbre de généralisation réagencé par ensemble de généralisation

5.3.6 Appliquer des guides de modélisation

La cinquième catégorie regroupe des guides de modélisation. Il est à noter que certains guides permettent de prévenir des incohérences d'ordre dynamique dont deux exemples sont présentés.

5.3.6.1 Nœud final d'activité ou nœud final de flot

Dans les diagrammes d'activités UML 2.0, un nœud final de flot (*flow final node*) détruit tous les jetons qui arrivent à lui mais n'a aucun effet sur les autres flots de l'activité. Au contraire, lorsqu'un jeton atteint un nœud final d'activité (*activity final node*) tous les flots de l'activité sont stoppés.

Plaçons nous 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, mais 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.

Un exemple de guide de prévention est illustré dans la figure 5.26 extraite de [59] et donne un conseil sur le choix d'emploi entre un nœud final d'activité et un nœud final de flot. Le guide propose de faire la chose suivante : *“S'il n'est pas souhaitable d'arrêter tous les flots dans une activité, il est préférable d'utiliser un nœud final de flot plutôt qu'un nœud final d'activité.”*

Dans cet exemple, on veut simplement éviter que certains comportements de l'activité ne soient pas prématurément stoppés alors qu'ils sont encore en cours.

On prend comme hypothèse que plusieurs composants peuvent être construits et installés avant de délivrer l'application résultante. Le nœud d'action **Build Component** a des occurrences itérativement pour chaque composant. Quand le dernier composant est construit, la fin de l'itération de construction est indiquée par une fin de flot. Cependant, même si toute la construction des composants est arrivée à sa fin, d'autres comportements sont en cours d'exécution. Quand le dernier composant a été installé, l'application est délivrée. Quand la délivrance de l'application est terminée (nœud d'ac-

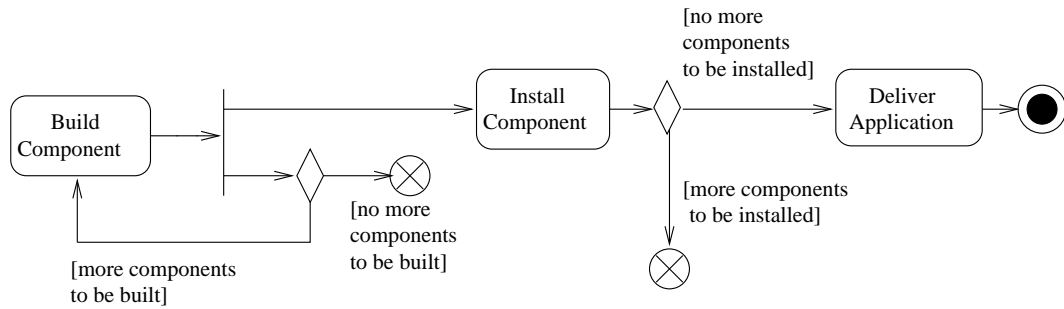


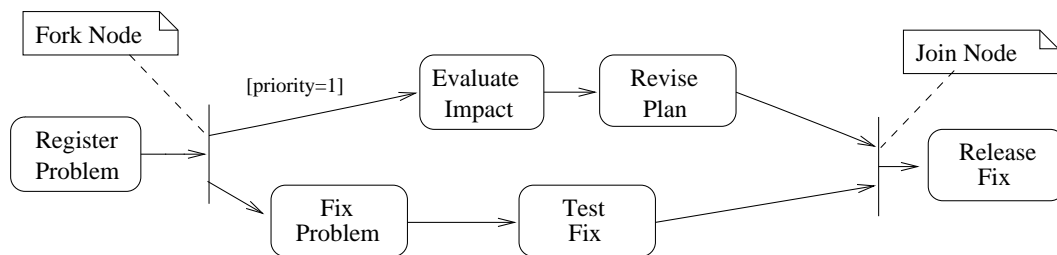
FIG. 5.26: Diagramme d'activités avec deux nœuds finaux de flots

tion *Deliver Application*), le flux de contrôle est passé au nœud final d'activité, ce qui indique que le traitement de l'activité est terminé.

5.3.6.2 De l'usage des conditions de garde dans les diagrammes d'activités

La sémantique de la spécification du diagramme d'activités de la figure 5.27 est que chaque jeton qui arrive à un nœud de bifurcation (*fork node*) est dupliqué sur ses arcs de sortie. Les jetons offerts par l'arc d'entrée sont tous offerts aux arcs de sortie. Quand un jeton offert est accepté sur tous les arcs de sortie, des duplicatas du jeton sont faits et une copie traverse chaque arc.

Ce diagramme d'activités présente néanmoins un risque d'incohérence, lié au fait que l'on peut assigner aux arcs de sortie d'un nœud de bifurcation (*fork node* en anglais) des conditions de garde.

FIG. 5.27: Arc d'activité avec condition de garde sortant d'un nœud de bifurcation (*fork node*), non protégé

Si la condition de garde n'est pas vérifiée, aucun jeton ne sera transmis par la branche supérieure et n'arrivera à l'arc d'entrée du nœud d'union, créant ainsi un blocage du flot de données. Un guide de prévention pourrait être explicité comme une alternative à ce blocage potentiel.

La figure 5.28 extraite de [59] illustre l'application de précautions d'usage concernant l'emploi de conditions de gardes, formulées dans ce guide : *“Si des conditions de garde sont utilisées sur des arcs de sortie d'un nœud de bifurcation, les modeleurs devraient s'assurer qu'aucun nœud d'union (join node) du flot “aval” ne dépend de l'arrivée de*

jetons passant par l'arc avec condition de garde. Si ce cas ne peut pas être évité, alors un nœud de décision (decision node) et un nœud d'interclassement ou de fusion (merge node) devraient être introduits pour prévenir la non-satisfaction de la condition de garde et pouvoir retransmettre le jeton au nœud d'union aval.”

La figure 5.28 montre en réalité une solution qui est le résultat de l'application du guide.

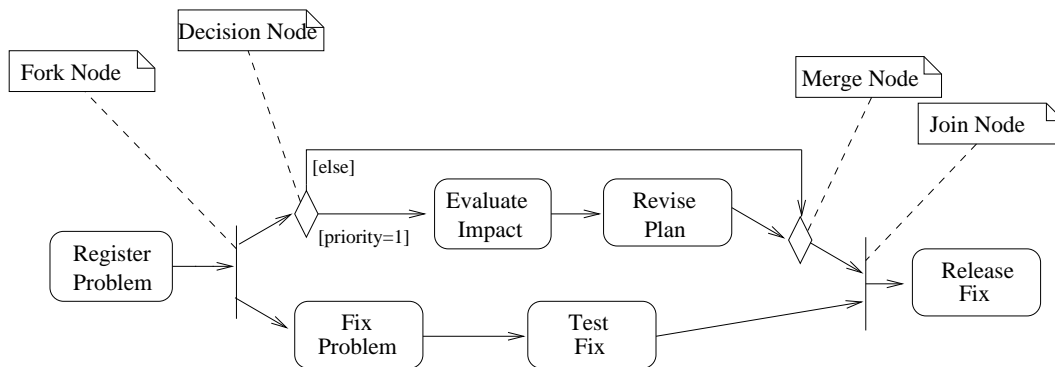


FIG. 5.28: Arc d'activité avec condition de garde sortant d'un nœud de bifurcation (*fork node*), protégé par un nœud de décision et un nœud de fusion

Mentionnons que ce guide n'est pas seulement applicable pour les arcs de sortie de nœud de bifurcation mais aussi pour les conditions de garde que l'on peut rajouter aux arcs entre deux simples nœuds d'activité.

5.4 Protection

5.4.1 Principes

Les techniques de protection ont pour but de réduire la gravité d'une incohérence. Au préalable, il faut d'abord identifier le critère qui nous permet d'évaluer la gravité et ses métriques, comme vu dans le chapitre précédent. Nous considérons deux cas possibles :

- la gravité est associée à l'impact de l'incohérence sur le code dérivé des modèles,
- la gravité est associée à la difficulté de détection des incohérences dans les modèles par revue.

Dans le premier cas, il faut considérer qu'actuellement les outils AGL ne prennent en compte pratiquement que le diagramme de classes pour générer le code, alors que d'autres informations utiles sont présentes dans les autres diagrammes. La mesure de l'impact sur le code est donc fortement dépendante des possibilités des générateurs de code actuels. Lors des estimations par expertise, les experts nous ont cependant donné leur vision sur cet impact en connaissant la relation entre éléments des modèles UML et instructions du code. Dans le deuxième cas, comme nous l'avons déjà signalé, la détection est facilitée lorsque des informations redondantes existent dans les modèles. En effet, le

style de modélisation a un fort impact sur la capacité de détection. Un diagramme d'activités contenant plus d'une dizaine d'actions commencerait à être difficilement lisible et rendrait la détection manuelle d'une incohérence du type de celle présentée en section 5.3.6.2 certainement plus coûteuse en temps. C'est en ce sens que certains des guides de prévention déjà présentés contribuent aussi à faciliter la détection manuelle ou automatique par un outil. Ainsi, nous avons proposé des guides de protection qui à l'origine, ont été pensés pour faciliter la détection d'une incohérence, ou si on l'exprime de manière positive, pour s'assurer qu'une règle de cohérence est vérifiée. Ils seront appliqués selon le niveau de criticité de l'application auquel ils sont associés : plus l'application est critique, plus le nombre de guides à appliquer sera important. Cependant, nous n'avons pas établi cette correspondance : quel guide pour quel niveau de criticité ?

Les guides de protection étant moins nombreux que ceux de prévention, nous n'en illustrerons que quelques-uns. De plus, quand on tente de classifier les guides de protection identifiés dans [54], on se rend compte que les trois grandes catégories correspondent à des catégories de guides de prévention déjà vues. La première (section 5.4.2) est celle qui consiste à rajouter une information : faire figurer un champ optionnel, exprimer le type ou la visibilité, exprimer explicitement un nom. La deuxième catégorie se réfère à ce qu'il faut éviter comme par exemple avoir recours à l'emploi d'expressions opaques pour les spécifications de valeur qui ne seront pas évaluables et qui empêcheront de vérifier une règle. *Nous conseillons de ne pas utiliser d'expression opaque pour une spécification de valeur quand cela est possible.* Lorsque ce guide est suivi, la règle suivante concernant les spécifications de valeur peut être vérifiée : *Dans le cas où aucune expression opaque n'apparaît dans la formulation de l'expression, l'expression doit être syntaxiquement correcte.* La dernière catégorie (section 5.4.3) regroupe des guides de modélisation généraux, comme par exemple *utiliser le stéréotype de dépendance le mieux adapté.*

5.4.2 Rajouter une information

Un grand nombre de règles de cohérence ne peuvent être vérifiées que si l'on spécifie des informations optionnelles. Nous pouvons illustrer cette catégorie de guide en prenant comme exemples ceux de la propriété `{readOnly}`, du type et de la visibilité pour les attributs. Un dernier exemple concerne la dépendance de stéréotype « `derive` ».

Propriété `{readOnly}` Un premier guide de protection a été défini pour faciliter la détection d'une incohérence liée à la tentative de modification d'attribut `{readOnly}`. *Nous conseillons de spécifier avec la propriété `{readOnly}` les attributs qui ne doivent pas être modifiés.* On pourrait se demander quelle est la différence entre la définition de la propriété `{readOnly}` dans la spécification d'UML et ce guide. L'application de ce guide est un moyen de s'approprier la définition et de donner un sens à une notation proposée dans la perspective d'éviter une incohérence ou d'en faciliter la détection. L'incohérence qui pourrait être plus facilement détectée correspond à la règle suivante :

“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. Si la propriété `{readOnly}` n’avait pas été précisée, une tentative de modification n’aurait pas été détectable.

Un moyen de protection supplémentaire mis en évidence lors des interviews est qu’il faudrait pouvoir empêcher à l’aide de l’outil dans les machines à états l’utilisation d’un attribut `{readOnly}`. L’outil devrait afficher `{readOnly}` si on tente d’utiliser cet attribut dans le diagramme de machines à états. Ceci requiert de renforcer les outils. L’incohérence est présentée dans la figure 5.29 où l’attribut `jour` est modifié dans le diagramme de machines à états correspondant à la classe `Alerte`, dont les opérations ne sont pas montrées avec leurs arguments par souci de clarté.

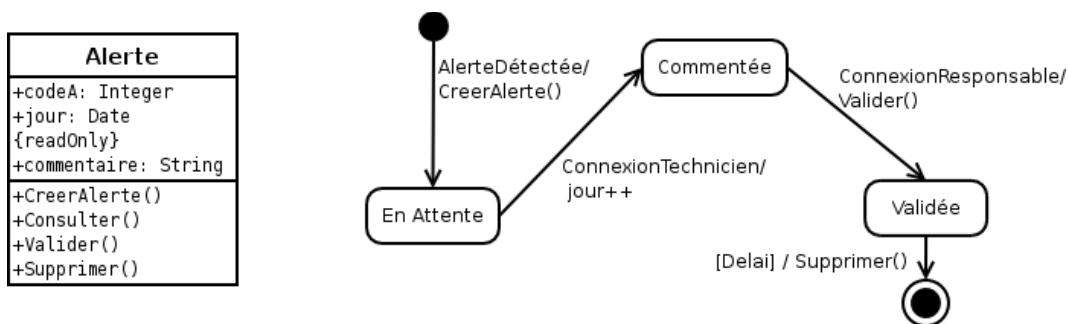


FIG. 5.29: Classe `Alerte` et sa machine à états

Type des attributs Dans le cas où un attribut a une valeur par défaut, cette valeur doit avoir le même type que celui de l’attribut. Afin de faciliter la détection d’une incohérence liée au non-respect de la règle précédente, un deuxième guide de protection a été défini : *Nous conseillons d’exprimer le type de chaque attribut afin de pouvoir détecter toute utilisation incorrecte de celui-ci.* En particulier, l’utilisation de la valeur par défaut d’un attribut incohérente avec la définition de son type pourra être plus facilement détectée.

Visibilité des attributs Le guide de protection qui conseille *d’exprimer la visibilité de l’attribut d’un classificateur pour vérifier la correction de l’accès à l’attribut* est à appliquer systématiquement seulement sur des modèles de conception. Cette information n’est pas généralement pertinente sur des modèles du domaine ou modèles conceptuels en phase d’analyse.

Dépendance de stéréotype « derive » Un exemple de guide de protection à appliquer à chaque fois que l’on utilise une dépendance d’abstraction de stéréotype « `derive` », est illustré dans la figure 5.30. Avant d’exprimer le guide, resituons-en son contexte et les notions concernées. 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).

Il est possible de spécifier de manière formelle quelle expression permet de calculer la source à partir de la cible, mais cela n'est pas obligatoire.

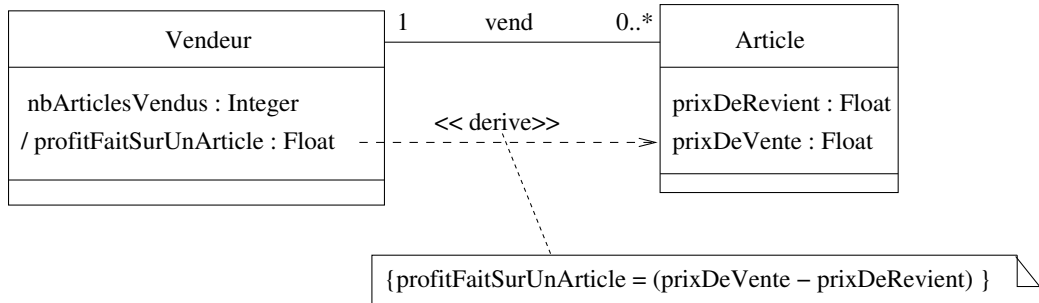


FIG. 5.30: Dépendance d'abstraction de stéréotype « derive »

Une des règles définie dans [54] exprime que : “si l'expression qui sert à calculer l'élément dérivé est explicitée, cette expression doit répondre aux règles énoncées pour les expressions”. Une autre règle énonce que “le type de l'élément source et le type de l'expression éventuellement utilisée pour décrire la dérivation doivent être identiques.”

Afin d'aider les développeurs à détecter plus facilement une incohérence concernant les règles précédentes, nous conseillons “de faire figurer l'expression de calcul de l'élément dérivé.” Le respect de ce guide de protection permet de vérifier le fait que l'élément est effectivement un élément dérivé. Ainsi la vérification des deux règles précédentes 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. Dans cet exemple, il aurait été difficile de vérifier la cohérence de l'attribut `profitFaitSurUnArticle`, sans l'information donnée dans la note.

5.4.3 Appliquer des guides de modélisation

Afin d'illustrer cette catégorie, nous allons prendre un exemple lié aux ports dans les diagrammes de structures composites et un autre concernant les connecteurs.

Ports dans les diagrammes de structure composite 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. De plus, un port peut être redéfini lorsque le classificateur qui le contient est spécialisé.

Nous conseillons de *préciser au maximum le fait qu'un port fournit ou requiert une interface* afin de faciliter la détection des erreurs associées aux règles suivantes :

- *Les interfaces requises d'un port doivent être fournies par les éléments auxquels le port est connecté.*
- *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.*
- *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.*

Remarque Un port comportemental est noté en reliant le port à un symbole d'un état comme montré dans la figure 5.31. Dans ce cas, le port comportemental “p” fournit l'interface `powertrain`.

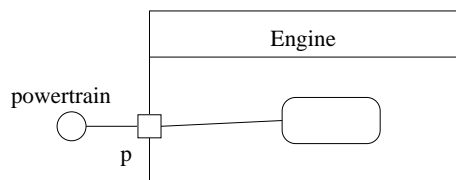


FIG. 5.31: Représentation d'un port comportemental

- *Les interfaces requises ou fournies par le port doivent être des interfaces connues du classificateur qui contient le port.*

Ce qui est formulé dans le guide comme “préciser au maximum” se rapproche de la première catégorie de guides de protection, mais comme cette précision permet de vérifier plusieurs règles, nous avons considéré qu'il s'agit d'un guide de modélisation plus général.

Connecteurs dans les diagrammes de structure composite 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.

Un guide a été défini comme suit : *Lorsqu'un connecteur est l'instance d'une association nous conseillons de l'indiquer.*

Il s'agit de faire figurer le nom de l'association afin de vérifier les règles suivantes :

- *Le champ `classname` d'un connecteur nommé doit être le nom d'exactly une association.*

- *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.*
- *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.*

Un exemple d'application de ce guide est visible dans la figure 3.13 (sous figure (ii)) de la section 3.2.1.4.

5.5 Conclusion

Prévenir la présence d'une incohérence revient à réduire la probabilité d'occurrence des incohérences, et nous préconisons d'appliquer des guides de prévention. Naturellement, ces guides ne se substituent pas aux autres bonnes pratiques de modélisation visant à d'autres objectifs complémentaires (lisibilité, etc.).

L'ensemble des guides sont reportés dans notre document [54] dont la rubrique justification des guides est destinée à consigner les éléments d'explication pouvant apparaître à l'avenir dans un outil de vérification automatique. Ces explications pourraient prendre la forme d'un message d'erreur argumenté en cas de détection d'une incohérence, proposant différentes pistes d'explication de l'incohérence pour faciliter sa résolution. [28] souligne comme une des caractéristiques les plus importantes d'un canevas d'application de gestion de la cohérence, la gestion même des règles de cohérence, et notamment la définition des règles. L'élaboration des règles devrait permettre de renseigner les champs suivants :

1. un nom descriptif pour l'incohérence ;
2. une description brève et précise de l'incohérence ;
3. un message de diagnostic contextuel expliquant la cause de l'incohérence ;
4. une recommandation d'actions de réparation de l'incohérence ;
5. un niveau de gravité de l'incohérence (faible/forte, facilement/difficilement détectable, etc.) ;
6. une expression dans le langage basé-règle choisi dans le cadre de travail de gestion de la cohérence, pour la détection et la réparation automatique de l'incohérence.

Les items 1 à 3 sont décrits dans notre document. L'item 5 correspond à une part de l'estimation que nous avons complété par la vraisemblance. Nous y ajoutons des guides de prévention et de protection. L'item 6 est étudié dans le cadre de la thèse conduite au LESIA par M. Hugues Malgouyres. L'item 4 n'est pas actuellement abordé par le LESIA. Enfin signalons que si un travail systématique d'identification a été fait, des guides ont été développés pour illustrer leurs principes par type d'incohérences.

Les guides de protection permettent, en complément de l'action des guides de prévention, de faciliter la détection d'incohérences. La frontière entre certains guides de prévention et de protection peut sembler parfois un peu floue, mais l'ensemble des guides de protection constitue une catégorie de guides bien particulière quand on s'applique à associer au guide de protection la ou les règles de cohérence dont la vérification est facilitée. Enfin, un autre moyen de faire de la protection est de renforcer la vérification (*le checking* en anglais) faite par les outils de telle sorte que les développeurs d'outils et des checkers intégrés prennent en compte non plus certains types d'incohérences, mais toutes celles qui ont été identifiées [54].

Le traitement des risques par réduction a été présenté comme un moyen pour prévenir le risque de la présence d'incohérences dans les modèles UML et pour faciliter la détection et donc l'extraction des incohérences rémanantes. Notre approche fondée sur le management du risque permet de justifier que c'est un moyen d'améliorer l'assurance de la sûreté des applications critiques en cherchant à diminuer les valeurs estimées de risques élevés. L'emploi de guides de prévention dans des sociétés telles que Thalès Avionics montre que le nombre d'incohérences faites par les développeurs est diminué. De plus, l'application dans un contexte "plus souple" des guides déjà existants et présentés dans ce chapitre et dans la perspective de la formation de futurs concepteurs est loin d'être inutile et constitue une motivation supplémentaire pour enrichir ce travail.

Chapitre 6

Synthèse, contributions majeures et perspectives

6.1 Synthèse

Les systèmes logiciels posent aujourd'hui des questions complexes concernant leur développement et leur maintenance. Dans le domaine des technologies à objets, le langage UML s'est imposé comme le langage graphique de spécification pouvant servir de support à la conception des systèmes à objets. Du fait de la richesse des constructions et des vues offertes par UML, un des problèmes majeurs auxquels sont confrontés les concepteurs est la maîtrise de la cohérence des modèles.

La définition du concept de cohérence n'est déjà pas un sujet facile et cette notion peut être considérée dans différents contextes et depuis différentes perspectives, selon que l'on cherche à vérifier à un instant précis ou à assurer dès le début d'un processus de développement la cohérence des modèles. Nous nous sommes placés dans cette perspective d'assurance d'une bonne utilisation du langage par la maîtrise des constructions d'UML. Nous avons défini le concept de cohérence comme étant le respect des propriétés associées au langage UML, que tout modèle doit respecter.

Face aux difficultés que pose l'utilisation du langage UML, l'objectif de cette thèse était de proposer un cadre d'étude complet basé sur le management des risques liés au langage lui-même, pour faciliter la maîtrise de la cohérence des modèles UML.

La première étape de ce travail a consisté à identifier les risques possibles d'incohérence, par l'étude de la spécification du langage UML et le recensement des événements dommageables que constituent les incohérences dans une modélisation UML. Nous avons exprimé un ensemble de règles de cohérence, tout en pensant à des moyens de prévention et de protection, pour réduire la vraisemblance et la gravité des incohérences.

Dans une deuxième phase, l'estimation des risques a permis d'évaluer un ensemble de règles représentatives d'un tiers des constructions du langage. Les interviews d'experts et les revues manuelles de modèles ont donné des éléments principalement qualitatifs permettant de mettre en valeur certaines constructions "à risque". Les critères d'éva-

luation de la vraisemblance d'une incohérence et de sa gravité ont été respectivement associés à la probabilité d'occurrence d'une incohérence et à la difficulté de sa détection ou à son impact sur le code.

Sur la base du travail effectué en parallèle du travail d'identification, quelques moyens de traitement des risques jugés inacceptables car correspondant à des niveaux de vraisemblance et de gravité élevés, ont été finalement proposés. Des catégories de guides et prévention et de protection ont été identifiées et illustrées.

6.2 Contributions majeures et perspectives

L'identification exhaustive des règles de cohérence que doit respecter tout modèle UML a mené à l'établissement d'un document conséquent disponible sur internet, qui constitue une base de données de règles toujours en cours de maintenance. Il comporte actuellement plus de 650 règles de cohérence. L'ambition finale du document est de regrouper à la fois toutes les règles de cohérence, des illustrations d'un modèle incohérent et du modèle cohérent associé pour chacune ainsi que des guides. Conscients de l'ampleur du travail de maintenance d'un tel document pour les versions futures du standard UML pour avoir participé à son élaboration, et conscients de l'enjeu pour la communauté UML de disposer d'un tel document, ce travail devrait être poursuivi par une équipe de personnes motivées. De plus, la gestion de ces règles devrait s'intégrer à un canevas d'application plus général de gestion de la cohérence. Beaucoup de travaux seraient à entreprendre pour améliorer la capacité de détection des outils et l'adéquation de leur base de règles avec le métamodèle d'UML, et pour faciliter la visualisation des incohérences détectées. Actuellement les travaux de Hugues Malgouyres pour l'établissement d'un prototype d'outil d'aide à la vérification de modèles UML s'intègrent dans ce cadre.

L'étude des événements dangereux (ou initiateurs) et des causes réelles d'introduction d'une incohérence par un utilisateur n'a pas été menée. Pourtant cette étape mériterait aussi une attention particulière pour comprendre les aspects cognitifs qui conduisent à introduire, laisser, ou oublier une incohérence.

La deuxième étape consistant en l'estimation du risque est un vaste terrain d'expérimentations possibles. Des interviews concernant les diagrammes de classes ont été menées, et la campagne d'interviews va être poursuivie avec les contacts qui ont déjà été établis. Afin qu'elle revête un caractère systématique et qu'elle embrasse la totalité du langage UML, cette campagne devrait être poursuivie en l'étendant de manière prioritaire aux diagrammes de machines à états, puis aux diagrammes de séquence et d'activités. Sur la base d'un panel plus large d'experts UML, des mesures de similitude pourraient être faites entre les valeurs estimées de gravité et de degré de vraisemblance par les différentes personnes pour une même règle de cohérence et pour chaque construction évaluée. Ceci permettrait de confirmer par l'expérience les constructions à risque

qui ont déjà été identifiées.

L'estimation du risque est suivie par une étape d'*évaluation* qui hiérarchise les risques, et une étape d'*acceptation*. Ces deux étapes n'ont été qu'effleurées. En particulier, il serait très utile d'effectuer une analyse bénéfices/coûts sur les constructions UML, vis-à-vis de la sécurité, en prenant la cohérence comme critère. En effet, si nous avons souligné le point de vue négatif (existence d'incohérences) et son coût (guides, outils de vérification), nous n'avons pas traité des aspects positifs des constructions d'UML qui permettent de révéler les incohérences. En effet, le fait que les incohérences soient exprimées dans les modèles et donc soient traitables est une chance. Sinon, elles existeraient dans les données informelles (textes écrits en langage naturel) sans être perceptibles, tout en étant nuisibles.

Concernant le traitement des risques par réduction, un travail systématique d'identification de guides de prévention et de protection reste à compléter. Une des principales difficultés à été la confrontation assez tardive avec des modèles UML industriels et donc la limitation de nos guides à des cas plutôt conventionnels.

Il serait également nécessaire d'établir la correspondance entre valeurs estimées et guides pour savoir quels guides doivent être employés selon le niveau de criticité de l'application. De plus, nous avons mentionné que la cohérence n'est pas une exigence permanente. Il conviendrait d'établir les règles qui doivent être respectées selon les étapes du développement. Par ailleurs, une fois une incohérence détectée, il serait utile de fournir une assistance à la réparation pour ramener le modèle dans un état cohérent.

Enfin, sûrs de la valeur pédagogique et positive de l'apprentissage par l'erreur, nos travaux pourraient donner lieu à des supports d'enseignement du langage UML par le biais des incohérences, en présentant les constructions et les risques possibles.

Ainsi, ce mémoire contribue à l'ensemble des travaux menés au sein de l'équipe Systèmes Embarqués Critiques au LESIA-INSA sur le langage UML et constitue une base de travail qui donne d'ores et déjà lieu à des thèses en cours.

Annexe A

Interviews

A.1 Feuille-type d'estimation d'une règle par interview

Règle n°

1. Difficulté de compréhension de la règle

Cette règle vous semble :

- A) être très facile à comprendre, ou
- B) être compréhensible avec un peu d'effort, ou
- C) nécessiter un effort très important de compréhension.

2. Fréquence d'utilisation de la construction

Dans vos modèles, la construction support de l'incohérence est utilisée :

- A) de très nombreuses fois, ou
- B) assez souvent, ou
- C) de façon ponctuelle, ou
- D) jamais.

3. Probabilité d'occurrence de l'incohérence

En supposant que la construction est utilisée, l'incohérence peut-elle se rencontrer :

- A) très fréquemment, ou
- B) assez fréquemment, ou
- C) parfois, ou
- D) pratiquement jamais.

4. Guide de prévention

Quelle bonne pratique de modélisation pourrait éviter cette incohérence ?

5. Difficulté de détection

Dans un modèle, la détection de cette incohérence vous semble :

- A) être évidente, ou
- B) demander un effort mais être faisable, ou
- C) nécessiter une investigation trop importante pour y arriver.

6. Confiance dans la réponse précédente

Votre confiance dans votre réponse précédente sur la difficulté de détection

- A) est bonne ou très bonne, ou
- B) est moyenne, ou
- C) est faible.

7. Guide de protection

Quelle bonne pratique de modélisation permettrait de faciliter cette détection ?

8. Impact de la présence d'une incohérence sur le code dérivé

Si cette incohérence est présente dans un modèle,

- A) il y a de très fortes chances que le code dérivé soit erroné, ou
- B) il y a peu de chances que le code dérivé soit erroné, ou
- C) il n'y a pas d'impact, c'est-à-dire que le code dérivé sera correct.

Commentaire :

A.2 Règles utilisées pour l'interview

A.2.1 Liste des concepts de niveau méta-modèle considérés

1. Éléments :

Named Element, Multiplicity, Constraint, Element Properties, Instance Specification, Redefinable Element, Operation, Property, Property Attribute, Date Type, Interface, Reception, Association Class, Template Parameter, Behavior, Signal.

2. Relations :

Element Import, Generalization, Generalization Set, Abstraction, Derive Stereotype, Substitution, Create Stereotype, Association, Template Binding.

A.2.2 Énoncé textuel des règles

Les règles présentées ici sont tirées de [54] et la numérotation correspond à la version 1.1 de ce document, disponible sur internet. Les termes entre parenthèses renvoient au nom anglais de la construction du métamodèle considérée, et éventuellement à un sous-groupe de règles de cette construction (CONSTRUCTION, *sous-groupe en italique*).

A.2.2.1 Éléments

- RÈGLE 12 (NAMED ELEMENT) : Un élément ne peut appartenir qu'à un seul espace de nommage en même temps. [Règle dérivée du méta-modèle]
- RÈGLE 17 (MULTIPLICITY) : La borne supérieure doit être supérieure ou égale à la borne inférieure. [[59] p.41]
- RÈGLE 19 (MULTIPLICITY) : Si des valeurs non littérales sont utilisées pour décrire les bornes des multiplicités, alors leurs spécifications doivent être des constantes. [[59] p.41]
- RÈGLE 28 (CONSTRAINT) : L'élément qui contient la contrainte doit avoir accès aux éléments mis en jeu par la contrainte. [tirée de [59] p.54]
- RÈGLE 28 (ELEMENT PROPERTIES) : Toute propriété doit s'appliquer sur un élément pour lequel elle a été définie. [Nouvelle règle][Règle utilisateur]
- RÈGLE 42 (INSTANCE SPECIFICATION) : 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 (INSTANCE SPECIFICATION) : La multiplicité des caractéristiques structurelles dans le classificateur (association) doit être respectée par le nombre de caractéristiques structurelles dans la spécification d'instance. [Nouvelle règle]
- RÈGLE 48 (REDEFINABLE ELEMENT) : 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. [[59] p.70]

• RÈGLE 59 (OPERATION) : Une opération redéfinissant une autre opération doit respecter les propriétés suivantes : [tirée de [59] 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 qui redéfinit est conforme au type du paramètre ou résultat de l'opération redéfinie, c'est-à-dire d'un même type ou d'un sous-type.

• RÈGLE 63 (OPERATION) : 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]¹

Remarque Une opération sans effet de bord est une opération à laquelle on a rajouté la propriété {query}.

• RÈGLE 71 (PROPERTY, *au sens possession*) : 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 (PROPERTY, *au sens possession*) : 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]

• RÈGLE 73 (PROPERTY, *au sens possession*) : 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 79 (PROPERTY, *attribute*) : 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 (PROPERTY, *attribute*) : Les méthodes associées à des opérations ne doivent pas modifier d'attributs qui ont la propriété {readOnly}. [Nouvelle règle]²

• RÈGLE 81 (PROPERTY, *attribute*) : 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]³

• RÈGLE 82 (DATA TYPE) : 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 82 (INTERFACE) : La visibilité de toutes les caractéristiques d'une interface doit être publique. [[59] p.114]

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

¹inter-diagramme, machines à états et classes

²inter-diagramme, classes et (machines à état ou activités)

³inter-diagramme, classes et machines à états

- RÈGLE 94 (RECEPTION) : Une classe passive ne peut pas avoir de réception. [[59] p.394]

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

- RÈGLE 96 (ASSOCIATION CLASS) : 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. [[59] p.118]

- RÈGLE 101 (TEMPLATE PARAMETER) : Le paramètre générique ne doit pas être utilisé ailleurs dans le modèle qu'au sein de l'élément générique. [Nouvelle règle]

Remarque préliminaire à la règle 103 : Dans le cas où le paramètre générique est un classificateur, la syntaxe à respecter est la suivante : [[59] p.556]

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

- RÈGLE 103 (TEMPLATE PARAMETER, *classifier*) : Les champs `classifier-name1` et `classifier-name2` doivent correspondre à un classificateur visible par le classificateur générique. [Nouvelle règle]

- RÈGLE 114 (BEHAVIOR) : Les paramètres du comportement doivent correspondre aux paramètres de la caractéristique comportementale. [[59] p.381]

- RÈGLE 115 (BEHAVIOR) : La caractéristique comportementale implémentée par le comportement doit être une caractéristique (éventuellement héritée) du classificateur contexte du comportement. [[59] p.381]

- RÈGLE 116 (BEHAVIOR) : Si la caractéristique comportementale implantée a été redéfinie dans un ancêtre du classificateur qui contient ce comportement, alors le comportement doit réaliser la dernière caractéristique comportementale redéfinissante. [[59] p.381]

- RÈGLE *non numérotée dans [54]* (SIGNAL) : Si un classificateur définit une réception, alors un signal de même nom que cette réception doit être défini par un autre classificateur. [Nouvelle règle]

A.2.2.2 Relations

- RÈGLE 123 (ELEMENT IMPORT) : Si un alias est spécifié, il ne doit pas être identique au nom d'un des éléments appartenant à l'espace de nommage importateur. [Nouvelle règle]

- RÈGLE 131 (GENERALIZATION) : La hiérarchie des généralisations doit être orientée (*directed* en anglais) et acyclique. [[59] p.62]

- RÈGLE 139 (GENERALIZATION SET) : Le parent d'un ensemble de généralisation qui a la propriété {complete} ne peut pas être instancié. [Nouvelle règle]

- RÈGLE 140 (GENERALIZATION SET) : 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 (GENERALIZATION SET) : 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. [[59] p.62]

- RÈGLE 147 (ABSTRACTION, *Derive stereotype*) : Le type de l'expression éventuellement utilisée pour décrire la dérivation doit être identique ou alors d'un sous-type du type de l'élément source (client de la relation). [Nouvelle règle]

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

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

- RÈGLE 157 (SUBSTITUTION) : 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]

- RÈGLE 162 (CALL STEREOTYPE) : Dans le cas où la cible de la relation de dépendance d'utilisation de stéréotype « call » est une classe, cette classe doit posséder au moins une opération visible par la source de la relation. [Nouvelle règle]

- RÈGLE 163 (CALL STEREOTYPE) : 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]⁴

- RÈGLE 167 (CREATE STEREOTYPE) : S'il existe une relation de composition entre les classes A et B (A étant la classe composée), alors l'objet créateur de l'instance d'une classe B doit être une instance de A ou d'une des classes filles de la classe A. [Nouvelle règle]⁵

- RÈGLE 178 (ASSOCIATION) : Si une fin d'association est une classe alors les autres fins de cette association doivent aussi être des classes. [Nouvelle règle]

- RÈGLE 182 (ASSOCIATION, *avec contrainte X-or*) : Les bornes inférieures des multiplicités des associations du côté du classificateur qui n'est pas commun aux deux associations doivent être 0. [tirée de [13]]

⁴inter-diagramme, classes et interactions

⁵inter-diagramme, classe et séquence

- RÈGLE 184 (ASSOCIATION, *relation association-lien*) : 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 [68], il n'est pas toujours possible d'implanter un modèle en respectant les contraintes.

- RÈGLE 203 (ASSOCIATION, *aggregation*) : Dans le cas d'association n-aire où $n > 2$, les agrégations composites ne peuvent pas apparaître. [tirée de [61] p.37]

- RÈGLE 207 (ASSOCIATION, *composition*) : Toute instance de la classe composante doit appartenir à une et une seule instance de la classe composée. [Nouvelle règle]

- RÈGLE 212 (ASSOCIATION, *composition*) : 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. [[59] p.81]

- RÈGLE 216 (TEMPLATE BINDING) : Chaque paramètre de substitution doit se référer à un paramètre de la signature générique (contenue par l'élément générique). [[59] p.547]

Lexique Anglais Français

activity edge arc d'activité.

activity final node nœud final d'activité.

activity group groupe d'activité.

activity node nœud d'activité.

artefact artefact.

behaviour comportement.

classifier classificateur.

combined fragment fragment combiné.

component composant.

connector connecteur.

decision node nœud de décision.

dependency dépendance.

element import importation d'élément.

event occurrence occurrence d'événement.

flow final node nœud final de flot.

fork node nœud de bifurcation.

gate porte.

generalization set ensemble de généralisation.

inconsistency incohérence.

interaction operand opérande d'interaction.

join node nœud d'union.

manifestation manifestation.

merge node nœud d'interclassement (ou fusion).

namespace espace de nommage.

packageable element élément empaquetable.

pin broche.

property propriété.

risk risque.

risk avoidance évitement du risque.

risk optimization optimisation du risque.

risk retention prise de risque.

risk transfer transfert du risque.

role binding délimitation de rôle.

substitution substitution.

Index

- Action, 47, 48, 52, 89
- Activité, 38
 - arc d', 38, 39, 46, 48, 52
 - groupe d', 38
- Alias, 47, 96
- Analyse bénéfiques/coûts, 79, 81, 109
 - balance, 80
- Artefact, 22, 45, 46, 84, 85, 95
- Cohérence, 16
 - dynamique, 31
 - gestion de la, 35, 87
 - inter-diagrammes, 32, 41, 60
 - intra-diagramme, 60
 - intra-modèle, 29
 - règles de, 24, 28, 32, 33, 36–38, 40–42, 44, 47–49, 52, 54, 58, 62, 66–69, 73–77, 86, 87, 90, 93, 96, 101, 106, 108
 - sémantique, 30
 - statique, 16, 31
 - syntaxique, 30
- Collaboration
 - rôle de la, 92
 - utilisation de, 91
- Composant, 43–45, 74, 90, 91, 98, 103
 - port de, 91
- Composition, 50, 86
- Connecteur, 39, 46, 91, 103–105
 - de délégation, 91
- Délimitation de rôle, 92
- Dépendance
 - d'usage, 91
- Degré de vraisemblance, 60–63, 65, 66, 68, 70, 73, 108
- Design Patterns, 85
- Dommage, 17, 18, 54, 57, 65, 66, 76, 80
- Élément empaquetable, 42, 45, 46, 95
- Ensemble de généralisation, 97
- Espace de nommage, 47
- Événement dommageable, 18, 21, 24, 27, 41, 54, 57, 60, 80, 107
- Formalisme, 21
- Fragment combiné, 95
- Gravité, 60–63, 65, 68, 70, 71, 73, 76, 80, 81, 87, 88, 100, 105
- Guides, 18, 41, 78, 79, 83–87, 89, 90, 94–101
 - de modélisation, 25, 41, 81–83, 85, 88, 98, 101
 - de prévention, 41, 87, 88, 90, 94–96, 98, 99, 101
 - de protection, 41, 81, 87, 88, 101–103
 - de style, 76, 82, 97
 - sur le processus, 76, 82, 84
- Identification, 37, 38, 53, 54
 - des événements dommageables, 41
 - des guides, 109
 - des règles, 87, 108
 - des règles de cohérence, 37
- Importation d'élément, 40, 47, 93, 94, 96
- Incohérence, 16, 21–25, 27–37, 41, 42, 44–46, 52, 54, 55, 77–82, 85–90, 92–94, 98–103, 105–109
- Incomplétude, 21, 58, 88
- Interface, 43, 44, 67, 90, 91, 103, 104

- Interprétation, 21, 22, 30, 31, 37, 55, 69
- Label, 39, 46, 47, 102
- Méta-métamodèle, 19, 20
- Métamodèle, 19–21, 30–33, 35, 36, 38–40, 42–47, 50, 51, 54, 76
- Métrique, 60, 61, 64, 65, 67, 68, 73, 100
valeur de, 60, 63, 65–67, 74
- Manifestation, 45, 46, 95
- Modèle, 16, 18–21, 23, 24, 28, 29, 31–33, 37, 40, 44, 52–55, 59–61, 65, 66, 72–74, 86, 89, 90, 93, 94, 107, 108
cohérent, 53, 67, 108
complet, 90
d'analyse, 29, 85
d'erreur, 23
de conception, 29, 85
de faute, 23, 32, 41
formel, 33, 34
incohérent, 22, 30, 53, 55, 59, 67, 108
incomplet, 44, 91
niveau conceptuel, 19, 20, 32, 40, 51
- Nœud
d'interclassement(ou fusion), 89, 100
d'union, 48, 99
de bifurcation, 39, 99, 100
de décision, 39, 48, 89, 90, 100
final d'activité, 98, 99
final de flot, 98
- Niveau de criticité, 18, 62, 63, 78, 79, 87, 101
- Occurrence d'événement, 95
- Opérande d'interaction, 95
- Paquetage, 40, 42, 47, 93, 94
- Port, 103
- Porte, 96
- Prévention, 41, 88
- Processus de développement, 18, 21, 84, 107
- Protection, 41, 100
- Réalisation, 44, 91
- Réduction
de la gravité, 71, 77, 78, 81, 86, 87, 100
de la probabilité, 81, 105
du degré de vraisemblance, 77, 78, 87
- Redondance, 18, 28, 53, 81
- Risque, 15–18, 21
évaluation du, 18, 24, 109
évitement du, 79, 80
acceptation du, 24, 109
analyse du, 41
cible du, 18
courbe d'acceptabilité du, 78
estimation du, 16, 24, 57, 58, 60–64, 67–69, 75–78, 107–109
identification du, 16, 24, 37, 74
logiciel, 18
optimisation du, 80
prise de, 80
réduction du, 25, 41, 60, 70, 80, 81
résiduel, 79
refus du, 25, 79, 89
traitement du, 24, 25, 41, 57, 69, 71, 77, 79, 81, 82, 87, 106, 108, 109
transfert du, 79
- Sémantique, 21
de vérification, 23
opérationnelle, 23
- Substitution, 44, 67
- Valeur
implicite, 88
optionnelle, 88
par défaut, 90, 93, 102
- Zone ALARP, 70, 71, 79, 80

Bibliographie

- [1] ISO/CEI Guide 51. *Aspects liés à la sécurité - Principes directeurs pour les inclure dans les normes*. International Organization for Standardization, 1999.
- [2] ISO/IEC Guide 73. *Risk Management - Vocabulary - Guidelines for use in standards*. International Organization for Standardization, September 2002.
- [3] DO178B/ED-12 Revision B. *Software considerations in airborne systems and equipment certification*. Requirement and Technical Concepts for Aviation (RTCA) Inc., 1992.
- [4] ECSS-Q-80A. *Assurance Produit des Projets Spatiaux, Assurance Produit des Logiciels*. European Cooperation for Space Standardization, April 1996.
- [5] Center for Chemical Process Safety of the American Institute of Chemical Engineers. *Guidelines for Hazard Evaluation Procedures, Second Edition with Worked Examples*. Library of Congress Cataloging-In-Publication, 1992.
- [6] B. LYONS H.-E. ERIKSSON, M. PENKER and D. FADO. *UML 2 Toolkit*. Wiley Publishing, 2004. OMG press.
- [7] B. H.LISKOV and J. M. WING. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6) :1811–1841, November 1994.
- [8] Lab package of The Experimental Software Engineering Group (ESEG) of the University of Maryland. Procedural Techniques for Perspective-Based Reading of Requirements and Object-Oriented. Internet address : <http://www.cs.umd.edu/projects/SoftEng/ESEG/>.
- [9] C. ALEXANDER. The origins of pattern theory : The future of the theory and the generation of a living world. *IEEE Software*, pages 71–82, September-October 1999, Keynote speech at OOPSLA'96.
- [10] S. W. AMBLER. *Agile Modeling : Effective Practices for Extreme Programming and the Unified Process*. Wiley, 2002. Agile Modeling.
- [11] S. W. AMBLER. *The Elements Of UML 2.0 Style*. Cambridge University Press, 2005. Agile Modeling.
- [12] H. H. AMMAR, S. M. YACOB, and A. IBRAHIM. A Fault Model for Fault Injection analysis of Dynamic UML Specifications. In *Proceedings of the 12th International Symposium on Software Reliability Engineering*. IEEE, 2001.

- [13] P. ANDRÉ, A. ROMANCZUK, J.-C. ROYER, and A. VASCONCELOS. An Algebraic View of UML Class Diagrams. pages 261–276, January 2000. In H. SAHRAOUI, C. DONY editors, Actes de la conférence LMO'2000 ISBN 2-6462-0093-7.
- [14] L. APVRILLE, J.-P. COURTIAT, C. LOHR, and P. DE SAQUI-SANNES. TURTLE : A Real-Time UML Profile Supported by a Formal Validation Toolkit. *IEEE Transactions on Software Engineering*, Vol 30, n° 7, pages 473–487, July 2004.
- [15] E. ASTESIANO and G. REGGIO. An Algebraic Proposal for Handling UML Consistency. In KUZNIARZ et al. [42], pages 62–70.
- [16] F. BARBIER. *UML 2 et MDE, Ingénierie des modèles avec études de cas*. Dunod, 2005.
- [17] H. BAUERDICK, M. GOGOLLA, and F. GUTSCHE. Detecting OCL Traps in the UML 2.0 Superstructure : An Experience Report. *Lectures Notes in Computer Sciences n° 3273*, pages 188–196.
- [18] B. BERENBACH. The Evaluation of Large, Complex UML Analysis and Design Models. In *ICSE '04 : Proceedings of the 26th International Conference on Software Engineering*, pages 232–241, Washington, DC, USA, 2004. IEEE Computer Society.
- [19] R. BHADURI and R. VENKATESH. Formal Consistency of Models in Multi-View Modelling. In KUZNIARZ et al. [43], pages 149–159.
- [20] J.-P. BODEVEIX, T. MILLAN, C. PERCEBOIS, C. LE CAMUS, P. BAZEX, and L. FERRAUD. Extending OCL for verifying UML models consistency. In KUZNIARZ et al. [43], pages 75–90.
- [21] G. BOOCH. *Objects solutions—Managing the Object-Oriented Project*. Reading, MA :Addison-Wesley, 1995.
- [22] G. BOOCH, J. RUMBAUGH, and I. JACOBSON. *The Unified Modeling Language User Guide*. Addison Wesley, 1998.
- [23] G. CAPLAT. Sherlock Environment. Available on Internet. URL : <<http://servif5.insa-lyon.fr/chercheurs/gcaplat/>>.
- [24] G. CAPLAT. *Modélisation cognitive et résolution de problèmes*. Presses Polytechniques et Universitaires Romandes, 2002.
- [25] R. CONRADI, P. MOHAGHEGHI, T. ARIF, L. C. HEGDE, G. A. BUNDE, and A. PEDERSEN. Object-oriented reading techniques for inspection of uml models - an industrial experiment. In *Proceedings of the 17th European Conference on Object-Oriented Programming*, pages 483–500, Darmstadt, Germany, 2003.
- [26] G. CSERTAN, G. HUSZERL, I. MAJZIK, Z. PAP, and A. PATAR. VIATRA - Visual Automated Transformations for Formal Verification and Validation of UML Models. In *Proceedings of the 17th International Conference on Automated Software Engineering*. IEEE, 2002.

- [27] J. DERRICK, D. AKEHURST, and E. BORTEN. A framework for UML consistency. In KUZNIARZ et al. [43], pages 30–45.
- [28] M. ELAASAR and L. BRIAND. An Overview of UML Consistency Management. Technical report, Departement Of Systems And Computer Engineering, Carleton University, August 2004. TR n° SCE-04-18.
- [29] G. ENGELS, R. HECKEL, and J. M. KUSTER. Rule-based specifications of behavioural consistency based on the UML meta-model. In *4th International Conference on the Unified Modeling Language*, Lectures Notes in Computer Sciences n°2185. Springer, 2001.
- [30] R. ESHUIS and R. WIERINGUA. Tool support for verifying UML activity diagrams. *IEEE Transactions on Software Engineering*, Vol.3, n° 7, 2004.
- [31] A. S. EVANS and S. KENT. Meta Modelling semantics of UML : the pUML approach. In *2nd International Conference on the Unified Modeling Language*, 1999. Lectures Notes in Computer Sciences n°1723.
- [32] A. FINKELSTEIN. A Foolish Consistency : Technical Challenges in Consistency Management. In *DEXA '00 : Proceedings of the 11th International Conference on Database and Expert Systems Applications*, pages 1–5, London, UK, 2000. Springer-Verlag. LNCS n°1873.
- [33] E. GAMMA, R. HELM, R. JOHNSON, and J. VLISSIDES. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995.
- [34] J.-C. GEFFROY and G. MOTET. *Design of Dependable Computing Systems*. Kluwer Academic Publishers, 2002.
- [35] J. H. HAUSMANN, R. HECKEL, and S. SAUER. Extended Model Relations with Graphical Consistency conditions. In KUZNIARZ et al. [43], pages 61–74.
- [36] B. HNATKOWSKA, Z. HUZAR, L. KUZNIARZ, and L. TUZINKIEWICZ. A Systematic Approach to Consistency within UML Based Software Development Process. In KUZNIARZ et al. [43], pages 16–29.
- [37] L. JAPEC. Statistics Sweden. The Interview Process Model and the Concept of Interviewer Burden. In *ASA-meeting*, New York, USA, August 2002.
- [38] S.-K. KIM and D. CARRINGTON. A formal object-oriented approach to defining consistency constraints for UML models. In *Proceedings of the Australian Software Engineering Conference*. IEEE, 2004.
- [39] A. KOZLENKOV and A. ZISMAN. Are their design specification consistent with our requirements? In *IEEE Joint International Conference on Requirements Engineering*, Essen, Germany, September 2002. IEEE.
- [40] P. KRISHNAN. Consistency checks for UML. In *7th Asia-Pacific Software Engineering Conference*, Singapore, Singapore, December 2000. IEEE.

- [41] P. KRUCHTEN. *The Rational Unified Process : An Introduction*. Addison-Wesley Professional ; 2^{ième} édition, 2000.
- [42] L. KUZNIARZ, Z. HUZAR, G. REGGIO, J.-L. SOURROUILLE, and M. STARON, editors. *Proceedings of the IEEE Workshop on Consistency Problems in UML-Based Software Development II*. IEEE and Department of Software Engineering and Computer Science of Blekinge Institute of Technology, 2003.
- [43] L. KUZNIARZ, G. REGGIO, J.-L. SOURROUILLE, and Z. HUZAR, editors. *Proceedings of the IEEE Workshop on Consistency Problems in UML-based Software Development*. Department of Software Engineering and Computer Science of Blekinge Institute of Technology, 2002.
- [44] O. LAITENBERGER, C. ATKINSON, M. SCHLICH, and K. EL EM. An experimental comparison of reading techniques for defect detection in UML design documents. *Journal of Systems and Software*, n° 53, pages 183–204, 2000.
- [45] C. LANGE, M. CHAUDRON, J. MUSKENS, L. SOMERS, and E. DORTMANS. An Empirical Investigation in Quantifying Inconsistency and Incompleteness in UML Designs. In KUZNIARZ et al. [42], pages 26–34.
- [46] J.-C. LAPRIE (editor). *Dependability Concepts and Terminology*. IFIP WG 10.4, Dependability Computing and Fault Tolerance, vol. 5, Springer-Verlag, 1992.
- [47] C. LARMAN. *UML 2 et les design patterns, Analyse et conception orientées objet et développement itératif*. 3^{ième} édition, Pearson Education France, 2005.
- [48] N. G. LEVESON. *Safeware : System Safety and Computers*. Addison-Wesley, 1995.
- [49] X. LI, Z. LIU, and H. JIFENG. A formal semantics of UML sequence diagram. In *Proceedings of the Australian Software Engineering Conference*. IEEE, 2004.
- [50] J. LILIUS and I. P. PALTOR. Formalizing UML state machines for model checking. *Lectures Notes in Computer Sciences n° 1723*, 1999.
- [51] B. LITVAK, S. TYSZBEROWICZ, and A. YEHUDIA. Behavioral consistency validation of UML diagrams. In *Proceedings of the 1st International Conference on Software Engineering and Formal Methods*, Brisbane, Australia, September 2003. IEEE.
- [52] W. LIU, S. EASTERBROOK, and J. MYLOPOULOS. Rules Based Detection of Inconsistency in UML Models. In KUZNIARZ et al. [43], pages 106–123.
- [53] H. MALGOUYRES and G. MOTET. A UML Model Consistency Verification Approach Based on Meta-Modeling Formalization. 21st Annual ACM Symposium on Applied Computing, Dijon, France, April 2006.
- [54] H. MALGOUYRES, J.-P. SEUMA VIDAL, and G. MOTET. Règles de cohérence UML 2.0, July 2005. Available on Internet. URL : <<http://www.lesia.insa-toulouse.fr/UML>>.
- [55] R. MARCANO and N. LEVY. Using B formal specifications for analysis and verification of UML/OCL models. In KUZNIARZ et al. [43], pages 91–105.

- [56] G. MOTET and S. GAUDAN. Risques associés aux technologies logicielles. *Revue de l'Electricité et de l'Electronique (REE), Revue de la Société de l'Electronique, de l'Electricité et des Technologies de l'Information et de la Communication*, Vol. 11, December 2005.
- [57] OBJECT MANAGEMENT GROUP. MetaObjectFacility (MOF) Specification, version 1.4, April 2002. Available on Internet. URL : <<http://www.omg.org/technology/documents/formal/mof.htm>>.
- [58] OBJECT MANAGEMENT GROUP. Unified Modeling Language 2.0 Infrastructure Specification, September 2003. Available on Internet. URL : <<http://www.omg.org/uml/>>.
- [59] OBJECT MANAGEMENT GROUP. Unified Modeling Language 2.0 Superstructure Specification, August 2003. OMG Adopted Specification ptc/03-08-02, available on Internet. URL : <<http://www.omg.org/uml/>>.
- [60] OBJECT MANAGEMENT GROUP. Unified Modeling Language Specification, version 2.0, September 2003. Available on Internet. URL : <<http://www.omg.org/uml/>>.
- [61] OBJECT MANAGEMENT GROUP. Unified Modeling Language 2.0 Superstructure Specification, October 2004. Document ptc/04-10-02 (convenience document), available on Internet. URL : <<http://www.omg.org/uml/>>.
- [62] OBJECT MANAGEMENT GROUP. Unified Modeling Language 2.0 Superstructure Specification, August 2005. Document formal/05-07-04, available on Internet. URL : <<http://www.omg.org/uml/>>.
- [63] OBJECT MANAGEMENT GROUP. Unified Modeling Language 2.0 Infrastructure Specification, March 2006. Document formal/05-07-05, available on Internet. URL : <<http://www.omg.org/uml/>>.
- [64] Holger RASCH and Heike WEHRHEIM. Consistency between UML Classes and Associated State Machines. In KUZNIARZ et al. [43], pages 46–60.
- [65] G. SABALIAUSKAITE, F. MATSUKAWA, S. KUSUMOTO, and K. INOUE. An experimental comparison of checklist-based reading and perspective-based reading for uml design document inspection. In *Proceedings of the International Symposium on Empirical Software Engineering*, pages 148–160, Nara, Japan, October 2002. IEEE Computer Society.
- [66] J.-P. SEUMA VIDAL, H. MALGOUYRES, and G. MOTET. UML 2.0 consistency rules identification. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP'05)*, Las Vegas, USA, June 2005. CSREA Press.
- [67] W. SHEN, Y. LU, and W. LOW. Extending the UML metamodel to support software refinement. In KUZNIARZ et al. [42], pages 35–42.
- [68] J.-L. SOURROUILLE and G. CAPLAT. Checking UML Model Consistency. In KUZNIARZ et al. [43], pages 1–15.

- [69] J.-L. SOURROUILLE and G. CAPLAT. Constraint checking in UML modeling. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering SEKE'02*, pages 217–224, New York, USA, 2002. ACM Press.
- [70] J.-L. SOURROUILLE and G. CAPLAT. A Pragmatic View about Consistency Checking of UML Models. In KUZNIARZ et al. [42], pages 43–50.
- [71] R. V. D. STRAETEN, T. MENS, J. SIMMONDS, and V. JONCKERS. Using description logic to maintain consistency between UML models. *Lectures Notes in Computer Sciences n° 2863*, 2003.
- [72] R.V.D. STRAETEN, T. MENS, and J ; SIMMONDS. Maintaining Consistency between UML Models Using Description Logic. In KUZNIARZ et al. [42], pages 71–77.
- [73] R. L. TORO, J.-P. SEUMA VIDAL, H. MALGOUYRES, and G. MOTET. UML Inconsistencies Assessment. In *3rd European Conference on Embedded Real-Time Software*, Toulouse, France, January 2006.
- [74] R. L. TORO, J.-P. SEUMA VIDAL, L.-M. MOTET, H. MALGOUYRES, T. LE SAUX, and G. MOTET. Estimation of UML Inconsistencies in Avionics Domain. In L. KUZNIARZ, G. REGGIO, J.-L. SOURROUILLE, and M. STARON, editors, *Workshop on Consistency in Model Driven Engineering (C@MoDE) in the European Conference on Model Driven Architecture - Foundations and Applications*. Department of Software Engineering and Computer Science of Blekinge Institute of Technology, 2005.
- [75] G. TRAVASSOS, F. SHULL, J. CARVER, and V. BASILI. Reading Techniques for OO Design Inspections. Technical Report CS-TR-4353, University of Maryland Computer Science Department, 2002.
- [76] R. WAGNER, H. GIESE, and U. A. NICKEL. A Plug-in for flexible and Incremental Consistency Management. In KUZNIARZ et al. [42], pages 78–85.
- [77] Y. ZHAO, Y. FAN, X. BAI, Y. VANG, H. CAI, and W. DING. Towards formal verification of UML diagrams based on graph transformation. In *Proceedings of the International Conference on E-Commerce Technology for Dynamic E-Business*. IEEE, 2004.

Maîtrise de la cohérence des modèles UML d'applications critiques

APPROCHE PAR L'ANALYSE DES RISQUES LIÉS AU LANGAGE UML

Résumé : Le langage UML permet la description de systèmes informatiques complexes dont la sécurité est parfois critique. Lors de la conception de telles applications, des fautes de modélisation peuvent être introduites dans les diagrammes UML représentant différentes vues du système. Ces fautes peuvent mener à des incohérences entre diagrammes ou au sein même d'un seul diagramme. Outre les propriétés spécifiques de l'application, il convient de vérifier en premier lieu que les modèles sont cohérents vis-à-vis des propriétés induites par l'utilisation du langage UML. L'objectif de cette thèse est de proposer un cadre d'étude du langage, et plus particulièrement de sa version actuelle (UML 2.0), en se plaçant dans une démarche de Management du risque de la présence d'incohérences. Ce travail se décompose en deux étapes principales. La première est une analyse du risque qui comporte une identification des événements dommageables que sont les incohérences, par l'expression des règles de cohérence associées. L'analyse est complétée ensuite par l'estimation du risque associé à l'utilisation de chaque construction d'UML. Cette étape a donné lieu à l'analyse du standard UML et des travaux actuels concernant la définition et la vérification de la cohérence des modèles UML. Des interviews et des retours d'expérience ont été effectués afin d'estimer la probabilité d'occurrence d'une incohérence et sa gravité, aussi bien qualitativement que quantitativement. L'étape ultime a concerné l'aide au traitement des risques, par la proposition de guides de prévention et de protection à mettre en œuvre lors de la modélisation d'applications logicielles critiques.

Mots clés : Analyse du risque, UML, cohérence des modèles, guides.

Management of UML models consistency for safety critical systems

APPROACH BASED ON A UML RISK ANALYSIS

Abstract : UML language allows the description of complex computerized systems whose safety is often critical. During design of such applications, modelling faults can be introduced in UML diagrams representing several views of the system. These faults can lead to inconsistencies between diagrams and within a single diagram. In addition to the properties specific to the application, the models first have to be ensured to be consistent with respect to the properties induced by the use of the UML language. This thesis aims at proposing a framework of study of the actual version of the language (UML 2.0), adopting a risk management approach to consider the presence of inconsistencies. This work consists of two main steps. The first one is a risk analysis which implies an identification of the harmful events that are the inconsistencies, by the expression of the associated consistency rules. The analysis is then completed by a risk estimation of the use of each UML feature. This step led to the analysis of the UML specification and of the actual work concerning the definition and the checking of UML models consistency. Interviews and feedback have been done in order to estimate the probability of occurrence of an inconsistency and its severity, as well qualitatively as quantitatively. The last step addressed the assistance for the risk treatment, advocating the use of prevention and protection guidelines during the modelling of safety critical systems.

Keywords : Risk analysis, UML, model consistency, guidelines.