HAL
archives-ouvertes.fr

Contribution à la modélisation des applications temps réel d'aide à la conduite

Hela Marouane

▶ **To cite this version:**

Hela Marouane. Contribution à la modélisation des applications temps réel d'aide à la conduite. Modélisation et simulation. Université du Havre, 2015. Français. <NNT : 2015LEHA0017>. <tel-01257769>

HAL Id: tel-01257769

https://tel.archives-ouvertes.fr/tel-01257769

Submitted on 18 Jan 2016

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

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

Université du Havre
UFR des Sciences et Techniques

Université de Sfax
Faculté des Sciences Économiques et de Gestion

Contribution à la modélisation des applications temps réel d'aide à la conduite

Thèse de Doctorat
de l'Université du Havre
Spécialité Informatique

Présentée par

Hela MAROUANNE

Le 16 octobre 2015

Devant le jury composé de

Président et rapporteur :	M. Yamine Aït Ameur, Professeur à l'ENSEEIHT
Rapporteur :	M. Sami Faiez, Professeur à l'Université de La Manouba
Co-Directeurs :	M. Rafik Bouaziz, Professeur à l'Université de Sfax, MIRACL
	M. Bruno Sadeg, Professeur à l'Université du Havre, LITIS-Le Havre
Co-Encadrants :	M. Claude Duvallet, MCF à l'Université du Havre, LITIS-Le Havre
	M. Achraf Makni, Maître-Assistant à l'Université de Sfax, MIRACL

Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes–EA 4108

Remerciements

Les travaux présentés dans ce mémoire ont été effectués au Laboratoire de Recherche en Informatique et Multimédia (MIRACL), à la Faculté des Sciences Économiques et de Gestion de Sfax en collaboration avec le Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes (LITIS), à l'université du Havre dans le cadre d'une thèse en cotutelle internationale.

C'est pour moi un plaisir autant qu'un devoir de remercier toutes les personnes qui ont pu contribuer de près ou de loin à l'établissement de ce travail, qui m'ont aidé, m'ont soutenu et ont fait sorte que ce travail ait eu lieu.

Ainsi, j'exprime ma gratitude et je tiens à remercier mes deux directeurs de thèse, Monsieur Rafik Bouaziz, Professeur à la Faculté des Sciences Économiques et de Gestion de Sfax, et Monsieur Bruno Sadeg, Professeur à l'université du Havre. Grâce à eux, j'ai pu entreprendre une thèse dans mon domaine favori : les systèmes d'information. Je tiens à les remercier tous les deux pour ces années de soutien et pour leurs précieux conseils afin que je puisse mener à bien ce projet. Je tiens également à remercier Monsieur Claude Duvallet, Maitre de conférence à l'université du Havre, d'avoir accepter de co-encadrer cette thèse. Sans lui, je n'aurais pas pu me lancer dans la préparation de cette thèse. J'exprime ici ma profonde gratitude à son égard et l'estime respectueuse que je lui porte. Merci aussi à mon co-encadrant Achraf Makni, Maitre assistant chez l'Institut Supérieur d'électronique et de communication de Sfax. Je le remercie pour ses encouragements et ses aides, notamment pendant les périodes de doute.

Je remercie également les membres de jury de me faire l'honneur d'accepter l'évaluation de ce modeste travail. Qu'ils trouvent ici l'expression de ma profonde gratitude.

Je saisis aussi l'occasion pour remercier tout le corps professoral et administratifs de la Faculté des Sciences Économiques et de Gestion de Sfax et de l'université du Havre.

Table des matières

Remerciements **II**

Introduction générale **1**

I Étude de l'existant et contexte **7**

1 Systèmes d'aide à la conduite **9**
1.1 Introduction . . . 9
1.2 Systèmes de sécurité routière . . . 10
1.2.1 Systèmes de sécurité passive . . . 11
1.2.2 Systèmes de sécurité active . . . 12
1.3 Caractéristiques des systèmes d'aide à la conduite . . . 21
1.3.1 Contraintes temps réel . . . 21
1.3.2 Imprécision . . . 22
1.3.3 Fiabilité . . . 22
1.3.4 Sécurité . . . 22
1.3.5 Tolérance aux fautes . . . 23
1.4 Méthodes pour la tolérance aux fautes . . . 23
1.4.1 Redondance matérielle . . . 24
1.4.2 Redondance logicielle . . . 25
1.4.3 Synthèse . . . 28
1.5 Conclusion . . . 28

2 Réutilisation et patrons d'ingénierie **31**
2.1 Introduction . . . 31
2.2 Réutilisation . . . 32
2.2.1 Définition . . . 32
2.2.2 Processus de réutilisation . . . 32
2.2.3 Composants réutilisables . . . 34
2.3 Patrons d'ingénierie . . . 39
2.3.1 Comparaison entre les composants réutilisables . . . 39
2.3.2 Différents types de patrons . . . 42
2.4 Patrons de conception . . . 45
2.4.1 Formalismes de description des patrons . . . 45

2.4.2 Profils UML pour les patrons de conception . . . 49
2.4.3 Synthèse . . . 58
2.5 Patrons de conception temps réel . . . 59
2.5.1 Patrons de conception définis par Rekhis *et al.* . . . 59
2.5.2 Patrons de conception définis par Alho *et al.* . . . 60
2.5.3 Patrons de conception définis par Sarika *et al.* . . . 61
2.5.4 Patrons de conception définis par Gerdes *et al.* . . . 62
2.5.5 Synthèse . . . 63
2.6 Conclusion . . . 63

II Contribution 65

3 Intégration d'un système de BDTR dans les systèmes d'aide à la conduite 67
3.1 Introduction . . . 67
3.2 Systèmes de bases de données temps réel . . . 68
3.2.1 Caractéristiques des données TR . . . 68
3.2.2 Caractéristiques des transactions TR . . . 69
3.2.3 Synthèse . . . 71
3.3 Intégration d'un système de BDTR dans les systèmes d'aide à la conduite . . . 71
3.3.1 Nouvelle architecture des systèmes d'aide à la conduite . . . 72
3.3.2 Modèle du système de BDTR . . . 73
3.3.3 Estimateurs des données manquantes . . . 75
3.4 Validation de l'intégration d'un système de BDTR dans les systèmes d'aide à la conduite . . . 78
3.4.1 Système SQLite3 . . . 79
3.4.2 Simulateur PreScan . . . 80
3.4.3 Résultats et discussion . . . 81
3.5 Conclusion . . . 87

4 Profil UML pour les patrons de conception temps réel 91
4.1 Introduction . . . 91
4.2 Langage de modélisation UML . . . 92
4.3 Profil UML . . . 93
4.4 Profil UML-RTDB2 . . . 94
4.4.1 Extensions pour la spécification des patrons de conception TR . . . 95
4.4.2 Extensions pour l'instanciation des patrons de conception TR . . . 103
4.5 Définition des contraintes OCL . . . 108
4.6 Conclusion . . . 112

5 Patrons de conception pour les systèmes d'aide à la conduite 113
5.1 Introduction . . . 113
5.2 Description des systèmes d'aide à la conduite . . . 114
5.2.1 Système de régulateur de vitesse à contrôle de distance . . . 115
5.2.2 Système d'avertissement de sortie involontaire de voie . . . 117

5.2.3 Système SAFERIDER 120
5.3 Modélisation des patrons de conception 121
5.3.1 Processus de création de patrons de conception 121
5.3.2 Construction du patron *"Acquisition des données"* 125
5.3.3 Construction du patron *"Contrôle de données"* 139
5.3.4 Construction du patron *"Action"* 143
5.4 Conclusion 150

6 Réutilisation des patrons de conception 151
6.1 Introduction 151
6.2 Processus de réutilisation 152
6.2.1 Choix des variantes à sélectionner 152
6.2.2 Adaptation du modèle réduit du patron 153
6.2.3 Exemple de réutilisation de patrons 154
6.3 Mise en œuvre des patrons de conception proposés 163
6.3.1 Eclipse 163
6.3.2 Présentation de l'outil ADAS-Patterns 164
6.4 Conclusion 169

7 Évaluation des patrons de conception proposés 171
7.1 Introduction 171
7.2 Métriques d'évaluation 172
7.2.1 Métriques de réutilisabilité 172
7.2.2 Métriques de réutilisation 177
7.3 Évaluation des patrons de conception 181
7.3.1 Évaluation du patron *"Acquisition des données"* 183
7.3.2 Évaluation du patron *"Contrôle de données"* 185
7.3.3 Évaluation du patron *"Action"* 186
7.4 Conclusion 188

Conclusion générale 189

Publications 195

Références 197

Annexe A Relations linguistiques et règles d'unification 213

Annexe B Modélisation des applications d'aide à la conduite 219

Liste des figures

1.1 Architecture d'un système d'aide à la conduite. 13
1.2 Redondance modulaire triple TMR. 24
1.3 Classification des méthodes des commandes tolérantes aux fautes. 26

2.1 Ingénierie pour la réutilisation et ingénierie par la réutilisation [Cauvet et al., 2001]. 34
2.2 Classification des patrons. 42
2.3 Exemple de diagramme de classes d'une instance du patron de conception pour le domaine de contrôle de processus [Reinhartz-Berger and Sturm, 2009]. . . 56

3.1 Nouvelle architecture des systèmes d'aide à la conduite. 73
3.2 Modèle de données temps-réel. 75
3.3 Étapes de développement des systèmes d'aide à la conduite dans PreScan [Leneman et al., 2008]. 81
3.4 Intégration d'un système de BDTR dans ACC et LDW sous Matlab/Simulink. 89

4.1 Définition du stéréotype << *optional* >>. 96
4.2 Définition du stéréotype << *mandatory* >>. 97
4.3 Définition du stéréotype << *extensible* >>. 98
4.4 Définition du stéréotype << *sensor* >>. 100
4.5 Définition du stéréotype << *derived* >>. 100
4.6 Définition du stéréotype << *periodic* >>. 101
4.7 Définition du stéréotype << *sporadic* >>. 101
4.8 Définition du paquetage NFP. 103
4.9 Définition du stéréotype << $PatternClass$ >>. 104
4.10 Définition du stéréotype << $PatternAttribute$ >>. 104
4.11 Définition du stéréotype << $PatternOperation$ >>. 105
4.12 Définition du stéréotype << $ApplicationClass$ >>. 106
4.13 Définition du stéréotype << $PatternLifeline$ >>. 107
4.14 Définition du stéréotype << $PatternMessage$ >>. 107
4.15 Définition du stéréotype << $ApplicationLifeline$ >>. 107
4.16 Définition du stéréotype << $ApplicationMessage$ >>. 108
4.17 Mécanisme d'extension de la vue statique. 109
4.18 Mécanisme d'extension de la vue dynamique. 109

5.1 Diagramme de classes du système ACC. 117

5.2 Diagramme de classes du système LDW. 119
5.3 Diagramme de classes du système SAFERIDER. 122
5.4 Diagramme de classes du patron *"Acquisition des données"*. 130
5.5 Diagramme de séquence relatif à la fonctionnalité d'acquisition des données du système ACC. 131
5.6 Diagramme de séquence relatif à la fonctionnalité d'acquisition des données du système LDW. 131
5.7 Diagramme de séquence relatif à la fonctionnalité d'acquisition des données du système SAFERIDER. 132
5.8 Diagramme de séquence du patron *"Acquisition des données"*. 134
5.9 Diagramme de classes du patron *"Contrôle de données"*. 141
5.10 Diagramme de séquence du patron *"Contrôle de données"*. 141
5.11 Diagramme de classes du patron *"Action"*. 147
5.12 Diagramme de séquence du patron *"Action"*. 148

6.1 Processus de réutilisation. 152
6.2 Réutilisation du diagramme de classes du patron *"Acquisition des données"*. . 158
6.3 Réutilisation du diagramme de séquence du patron *"Acquisition des données"*. 159
6.4 Réutilisation du diagramme de classes du patron *"Contrôle de données"*. . . . 160
6.5 Réutilisation du diagramme de séquence du patron *"Contrôle de données"*. . 161
6.6 Réutilisation du diagramme de classes du patron *"Action"*. 162
6.7 Réutilisation du diagramme de séquence du patron *"Action"*. 163
6.8 Choix des variantes dans le diagramme de séquence du patron *"Acquisition de données"*. 165
6.9 Dictionnaire du domaine. 166
6.10 Renommage de la classe *Risk* du patron *"Contrôle de données"*. 166
6.11 Adjonction d'une classe spécifique au patron *"Contrôle de données"*. 167
6.12 Instanciation du diagramme de classes du patron *"Acquisition des données"*. . 168
6.13 Instanciation du diagramme de séquence du patron *"Acquisition des données"*. 168
6.14 Calcul des métriques de réutilisation après la réutilisation du diagramme de classes du patron *"Acquisition des données"*. 169
6.15 Calcul des métriques de réutilisation après la réutilisation du diagramme de séquence du patron *"Acquisition des données"*. 170

7.1 Deux modèles différents de la même application. 182

B.1 Modélisation du diagramme de classes du système FCW sans réutilisation de patrons. 221
B.2 Modélisation du diagramme de séquence du système FCW sans réutilisation de patrons. 222
B.3 Modélisation du diagramme de classes du système FCW par réutilisation de la vue statique du patron *"Acquisition des données"*. 223
B.4 Modélisation du diagramme de classes du système FCW par réutilisation de la vue statique du patron *"Contrôle de données"*. 224
B.5 Modélisation du diagramme de classes du système FCW par réutilisation de la vue statique du patron *"Action"*. 225

B.6 Modélisation du diagramme de séquence du système FCW par réutilisation de la vue dynamique du patron *"Acquisition des données"*. 226
B.7 Modélisation du diagramme de séquence du système FCW par réutilisation de la vue dynamique du patron *"Contrôle de données"*. 227
B.8 Modélisation du diagramme de séquence du système FCW par réutilisation de la vue dynamique du patron *"Action"*. 228
B.9 Modélisation du diagramme de classes du système SASPENCE sans réutilisation de patrons. 231
B.10 Modélisation du diagramme de séquence du système SASPENCE sans réutilisation de patrons. 232
B.11 Modélisation du diagramme de classes du système SASPENCE par réutilisation de la vue statique du patron *"Acquisition des données"*. 233
B.12 Modélisation du diagramme de classes du système SASPENCE par réutilisation de la vue statique du patron *"Contrôle de données"*. 234
B.13 Modélisation du diagramme de classes du système SASPENCE par réutilisation de la vue statique du patron *"Action"*. 235
B.14 Modélisation du diagramme de séquence du système SASPENCE par réutilisation de la vue dynamique du patron *"Acquisition des données"*. 236
B.15 Modélisation du diagramme de séquence du système SASPENCE par réutilisation de la vue dynamique du patron *"Contrôle de données"*. 237
B.16 Modélisation du diagramme de séquence du système SASPENCE par réutilisation de la vue dynamique du patron *"Action"*. 238
B.17 Modélisation du diagramme de classes du système COMPOSE sans réutilisation de patrons. 241
B.18 Modélisation du diagramme de séquence du système COMPOSE sans réutilisation de patrons. 242
B.19 Modélisation du diagramme de classes du système COMPOSE par réutilisation de la vue statique du patron *"Acquisition des données"*. 243
B.20 Modélisation du diagramme de classes du système COMPOSE par réutilisation de la vue statique du patron *"Contrôle de données"*. 244
B.21 Modélisation du diagramme de classes du système COMPOSE par réutilisation de la vue statique du patron *"Action"*. 245
B.22 Modélisation du diagramme de séquence du système COMPOSE par réutilisation de la vue dynamique du patron *"Acquisition des données"*. 246
B.23 Modélisation du diagramme de séquence du système COMPOSE par réutilisation de la vue dynamique du patron *"Contrôle de données"*. 247
B.24 Modélisation du diagramme de séquence du système COMPOSE par réutilisation de la vue dynamique du patron *"Action"*. 248
B.25 Modélisation du diagramme de classes du système BLA sans réutilisation de patrons. 251
B.26 Modélisation du diagramme de séquence du système BLA sans réutilisation de patrons. 252
B.27 Modélisation du diagramme de classes du système BLA par réutilisation de la vue statique du patron *"Acquisition des données"*. 253

B.28 Modélisation du diagramme de classes du système BLA par réutilisation de la vue statique du patron *"Contrôle de données"*. 254
B.29 Modélisation du diagramme de classes du système BLA par réutilisation de la vue statique du patron *"Action"*. 255
B.30 Modélisation du diagramme de séquence du système BLA par réutilisation de la vue dynamique du patron *"Acquisition des données"*. 256
B.31 Modélisation du diagramme de séquence du système BLA par réutilisation de la vue dynamique du patron *"Contrôle de données"*. 257
B.32 Modélisation du diagramme de séquence du système BLA par réutilisation de la vue dynamique du patron *"Action"*. 258
B.33 Modélisation du diagramme de classes du système SAFELANE sans réutilisation de patrons. 261
B.34 Modélisation du diagramme de séquence du système SAFELANE sans réutilisation des patron. 262
B.35 Modélisation du diagramme de classes du système SAFELANE par réutilisation de la vue statique du patron *"Acquisition des données"*. 263
B.36 Modélisation du diagramme de classes du système SAFELANE par réutilisation de la vue statique du patron *"Contrôle de données"*. 264
B.37 Modélisation du diagramme de classes du système SAFELANE par réutilisation de la vue statique du patron *"Action"*. 265
B.38 Modélisation du diagramme de séquence du système SAFELANE par réutilisation de la vue dynamique du patron *"Acquisition des données"*. 266
B.39 Modélisation du diagramme de séquence du système SAFELANE par réutilisation de la vue dynamique du patron *"Contrôle de données"*. 267
B.40 Modélisation du diagramme de séquence du système SAFELANE par réutilisation de la vue dynamique du patron *"Action"*. 268
B.41 Modélisation du diagramme de classes du système RDCW sans réutilisation de patrons. 271
B.42 Modélisation du diagramme de séquence du système RDCW sans réutilisation de patrons. 272
B.43 Modélisation du diagrammes de classes du système RDCW par réutilisation de la vue statique du patron *"Acquisition des données"*. 273
B.44 Modélisation du diagramme de classes du système RDCW par réutilisation de la vue statique du patron *"Contrôle de données"*. 274
B.45 Modélisation du diagramme de classes du système RDCW par réutilisation de la vue statique du patron *"Action"*. 275
B.46 Modélisation du diagramme de séquence du système RDCW par réutilisation de la vue dynamique du patron *"Acquisition des données"*. 276
B.47 Modélisation du diagramme de séquence du système RDCW par réutilisation de la vue dynamique du patron *"Contrôle de données"*. 277
B.48 Modélisation du diagramme de séquence du système RDCW par réutilisation de la vue dynamique du patron *"Action"*. 278

Liste des tableaux

1.1 Comparaison de différents systèmes d'aide à la conduite. 20

2.1 Formalisme de E. Gamma 46
2.2 Critères de spécification de patrons de conception 51
2.3 Critères de réutilisation de patrons de conception 52

3.1 RMSE (Root Mean Squared Error) pour les données manquantes dans les sous-systèmes ACC et LDW. 78
3.2 Exemple de valeurs de données résultant des différentes simulations dans ACC. 83
3.3 Exemple de valeurs de données résultant des différentes simulations dans LDW. 84
3.4 Les valeurs des données acquises à t = 6.9 s. 86
3.5 Exemple de la mise à jour des données dans ACC en utilisant un système de BDTR. 87
3.6 Nombre de transactions et temps de réponse des systèmes avec et sans intégration d'un système de BDTR. 87

5.1 Fonctions élémentaires, concepts et contraintes de la fonctionnalité d'acquisition des données. 126
5.2 Fonctions élémentaires, concepts et contraintes de la fonctionnalité de contrôle des données. 139
5.3 Fonctions élémentaires, concepts et contraintes de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes. 144

7.1 Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de classes du patron *"Acquisition des données"*. 184
7.2 Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de séquence du patron *"Acquisition des données"*. 185
7.3 Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de classes du patron *"Contrôle de données"*. 186
7.4 Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de séquence du patron *"Contrôle de données"*. 186
7.5 Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de classes du patron *"Action"*. 187
7.6 Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de séquence du patron *"Action"*. 187

A.1 Relations linguistiques pour la comparaison des noms de classes, attributs et opérations [BenAbdallah et al., 2004]. 213
A.2 Relations linguistiques pour la comparaison des noms des objets et des messages [BenAbdallah et al., 2004]. 215
A.3 Règles d'unification des diagrammes de classes. 215

Introduction générale

1 Contexte

Ces dernières années, l'industrie automobile a beaucoup contribué à l'amélioration de la sécurité routière par la diminution du nombre d'accidents. Dans ce sens, des véhicules sont dotés de systèmes d'aide à la conduite temps réel, comme les systèmes de prévention de collision et d'aide à la vision de nuit et à la navigation. Le développement de ces systèmes est difficile du fait qu'il est nécessaire de respecter non seulement l'exactitude des résultats, mais aussi les contraintes temporelles liées à la validité des données et à l'échéance des opérations ainsi que la fiabilité et la sécurité de ces systèmes. Un défaut de comportement ou le non-respect d'une contrainte de temps peut provoquer des conséquences catastrophiques (*e.g.,* des blessés sur la route).

Les travaux qui entrent dans le cadre des systèmes d'aide à la conduite portent généralement sur l'étude de la fiabilité de ces systèmes (*e.g.,* ceux de Piao and McDonald [2008]), les méthodes de fusion de données (*e.g.,* ceux de Darms et al. [2010]) et le développement des systèmes (*e.g.,* ceux deSalameh [2011]), mais ne sont pas axés sur la proposition de méthodes pour la conception. Pour cette raison, nous proposons des méthodes pour la modélisation des systèmes d'aide à la conduite.

Ces systèmes deviennent de plus en plus complexes vu qu'ils comprennent de plus en plus de capteurs, de normes (*e.g.,* eCall) et de fonctionnalités, et traitent de plus en plus de données. En conséquence, devant cette complexité rendant leur réalisation et leur développement plus complexes, plus coûteux et moins fiables, l'ingénieur des systèmes d'aide à la conduite a recours à de nouvelles méthodes favorisant la réutilisation. Ces dernières années, une nouvelle approche basée sur la réutilisation des composants a émergé. Elle concerne tout le processus de développement dans l'objectif de réduire les délais de développement et d'améliorer la qualité des systèmes produits. C'est pourquoi, nous nous sommes intéressés à définir une nouvelle méthodologie basée sur la réutilisation pour la modélisation des systèmes d'aide à la conduite.

Étant une forme particulière de composants réutilisables, les "patrons" (Patterns) permettent d'assurer la réutilisation en proposant des connaissances approuvées et réutilisables depuis la phase d'analyse jusqu'à la phase d'implémentation. Dans le cadre de cette thèse, nous nous intéressons plus particulièrement aux patrons de conception. Ces patrons constituent des solutions génériques permettant de réduire la complexité de certains problèmes spécifiques à la conception et le risque d'erreurs qui se propagent dans le code, et par conséquent, de réduire le risque de conséquences graves engendrées par l'occurrence des erreurs de conception.

2 Problématique

Les systèmes d'aide à la conduite évoluent rapidement ; ils comprennent de plus en plus de capteurs et de fonctionnalités et ils traitent un grand volume de données qui doivent être mises à jour régulièrement pour refléter fidèlement l'état courant de l'environnement. Cependant, les systèmes existants ne permettent pas de stocker ces données, et par conséquent, il leur est difficile de gérer une grande quantité de données, ce qui peut engendrer des anomalies lorsque certaines données sont perdues. Si un capteur est défectueux, la donnée ne sera pas mise à jour régulièrement, ce qui provoque des résultats erronés pouvant aboutir à des situations graves. Partant de ce constat, l'objectif de cette thèse est aussi d'améliorer l'architecture des

systèmes existants par l'intégration d'une base de données temps réel [Ramamritham, 1993] permettant de gérer efficacement les contraintes temporelles relatives à un grand volume de données. Cette nouvelle architecture vise à améliorer la tolérance aux fautes avec un temps de calcul minimal, contrairement aux méthodes de tolérance aux fautes existantes qui se basent essentiellement sur des modèles mathématiques complexes exigeant des temps de calcul très longs, et par conséquent, ne conviennent pas aux systèmes temps réel.

La modélisation des systèmes d'aide à la conduite, basée sur l'approche de réutilisation de patrons en tenant compte de la gestion d'une grande quantité de données et leurs contraintes temporelles, induit des défis que nous abordons dans cette thèse :

- Modélisation des patrons satisfaisant toutes les exigences du domaine d'aide à la conduite temps réel. Il s'agit de construire des patrons qui répondent à toutes les exigences de ce domaine (*i.e.,* les aspects fonctionnels, les aspects non fonctionnels et les contraintes temps réel). Cependant, les travaux existants qui ont proposé des patrons spécifiques au domaine temps réel ne conviennent pas au domaine d'aide à la conduite. En effet, certains patrons n'offrent que des solutions destinées à la modélisation des aspects non fonctionnels (*e.g.,* Armoush [2010]), ou bien à la modélisation des systèmes temps réel en général (*e.g.,* Rekhis et al. [2010b]). L'application de ces patrons pour modéliser les systèmes d'aide à la conduite est une tâche difficile car les solutions fournies par ces patrons ne tiennent pas compte de toutes les exigences de ce domaine. Ils rendent la réutilisation plus compliquée ; le concepteur doit spécifier les exigences des systèmes d'aide à la conduite. De plus, ces patrons n'expriment pas les contraintes temps réel et ne modélisent pas la gestion des bases de données temps réel.
- Modélisation d'un système à partir de patrons. Il s'agit de savoir comment (i) appliquer les patrons pour modéliser un système cible qui répond à toutes les exigences et (ii) évaluer la qualité de ces patrons.

Les travaux existants qui ont défini des patrons temps réel ne permettent pas de distinguer les similitudes et les différences entre les systèmes d'un domaine et de garder trace de

l'instanciation des patrons, à l'exception des travaux proposés dans Rekhis et al. [2010c] et Rekhis et al. [2013a].

Sur la base des insuffisances des travaux antérieurs (*e.g.,* Armoush [2010] et Rekhis et al. [2010b]), nos travaux visent à aboutir aux contributions suivantes :

- Définition de patrons de conception modélisant les différentes fonctionnalités des systèmes d'aide à la conduite utilisant les bases de données temps réel, en se basant sur les diagrammes de classes et les diagrammes de séquences d'UML. Ces patrons permettent de modéliser les aspects structurels, les aspects dynamiques, les exigences non fonctionnelles ainsi que les contraintes temporelles des systèmes d'aide à la conduite. La modélisation de ces patrons gagne à se baser sur un processus qui définit les étapes à suivre pour créer les modèles structurels et les modèles dynamiques. Ce processus est à définir pour aider le concepteur de patrons à identifier les exigences du domaine étudié ainsi que les éléments communs et les éléments qui varient entre les systèmes représentatifs du domaine.
- Définition d'un processus de réutilisation des patrons pour modéliser une application cible. En effet, nous avons à construire un outil permettant de guider le concepteur d'applications pour modéliser une fonctionnalité d'un système d'aide à la conduite en réutilisant le patron représentant cette fonctionnalité, tout en assurant que les éléments correspondant à chaque instance du patron sont identifiés. Cet outil doit permettre aux concepteurs de patrons de vérifier la qualité des modèles de chaque patron lors de la modélisation des systèmes en se basant sur des métriques, *i.e.,* de vérifier que le modèle de l'application en cours de développement est conforme au modèle du patron instancié et que le modèle du patron couvre toutes les exigences du domaine.
- Définition des notations permettant de distinguer explicitement les éléments obligatoires des éléments facultatifs et de souligner les aspects temps réel. Ces extensions doivent fournir une sémantique aux différents concepts UML et aider à la réutilisation. En outre, nous avons à définir des notations qui permettent de garder trace des éléments instanciés à partir des patrons pour aider le concepteur à valider les patrons. Nous avons également à définir des contraintes OCL (Object Constraint Langage)

pour exprimer les contraintes temps réel et des contraintes OCL pour assurer une instanciation correcte des patrons.

3 Organisation de la thèse

Ce présent rapport de thèse est organisé en deux parties. La première partie comporte deux chapitres. Le premier chapitre présente un état de l'art sur les systèmes de sécurité passive (*e.g.,* ceinture de sécurité) et les systèmes de sécurité active, ou d'aide à la conduite (*e.g.,* régulateur de vitesse), et plus particulièrement les systèmes d'aide à la conduite. De plus, ce chapitre décrit différents travaux de recherche qui ont proposé des méthodes de tolérance aux fautes des capteurs dans le domaine de l'automobile pour assurer la fiabilité et la sécurité de ces systèmes. Le deuxième chapitre constitue un état de l'art sur la réutilisation et les composants réutilisables, et plus particulièrement les patrons. Nous montrons également les avantages des patrons par rapport aux autres composants. De plus, nous détaillons les travaux qui portent sur la représentation des patrons de conception. Il décrit également les travaux les plus récents qui ont proposé des patrons de conception destinés au domaine temps réel. Le chapitre présente enfin les avantages et les limites des travaux examinés et positionne nos contributions par rapport à ces travaux.

La seconde partie consiste à présenter nos contributions. Elle est composée de cinq chapitres. Le troisième chapitre présente notre proposition d'une nouvelle architecture de systèmes d'aide à la conduite qui intègre une base de données temps réel en vue de gérer efficacement les contraintes temporelles relatives à un grand volume de données et aux traitements. L'intégration de la base permet également d'assurer la tolérance aux fautes avec des performances acceptables (*e.g.,* minimiser le temps de réponse des systèmes). Le quatrième chapitre décrit le profil UML que nous définissons pour améliorer la modélisation et la réutilisation des patrons. Ce profil définit des extensions permettant d'exprimer le critère de variabilité et l'expressivité des modèles de patrons. Il définit également des extensions pour garder trace des instances de patrons dans le modèle du système cible. Le cinquième chapitre est consacré à la proposition de trois patrons de conception afin de modéliser les

fonctionnalités globales, les exigences non fonctionnelles et les contraintes temporelles des systèmes d'aide à la conduite utilisant les bases de données temps réel. La création de ces patrons suit un processus qui se base sur l'analyse du domaine et l'unification d'un ensemble de systèmes appartenant à ce domaine. Quant au sixième chapitre, il considère la réutilisation des patrons proposés, en les appliquant à des exemples de systèmes d'aide à la conduite. Nous proposons également un outil qui permet d'assister le concepteur lors de l'instanciation des patrons pour la modélisation d'un système. Enfin, le septième chapitre évalue la qualité des patrons proposés à travers deux catégories de métriques : des métriques de réutilisabilité et des métriques de réutilisation Rekhis et al. [2013b]. La première catégorie de métriques consiste à estimer la probabilité qu'un patron soit réutilisable pour montrer que le patron couvre toutes les exigences du domaine. La deuxième catégorie permet de mesurer le degré de réutilisation des éléments du patron pour montrer que le système est modélisé en appliquant le patron avec le minimum de modifications de sa solution.

La conclusion de ce rapport présente une synthèse de notre travail et rappelle les contributions de cette thèse. Elle donne également des perspectives à ce travail.

Première partie

Étude de l'existant et contexte

Chapitre 1

Systèmes d'aide à la conduite

1.1 Introduction

De nos jours, le nombre de véhicules circulant sur la route ne cesse d'augmenter, ce qui accroît le nombre d'accidents et les problèmes de sécurité routière. Pour pallier ces problèmes, il est nécessaire d'améliorer les véhicules et l'infrastructure du réseau routier par la proposition de systèmes intelligents : ce sont les Systèmes de Transport Intelligents (STI).

Les STI désignent l'application des nouvelles technologies de l'information et de la communication aux domaines des transports afin d'améliorer la sécurité routière, de réduire le nombre d'accidents, d'optimiser l'utilisation des infrastructures, etc.

Parmi les services qu'offrent les STI, on trouve : (a) le paiement électronique, (b) la gestion du trafic, (c) le transport public de voyageurs, (d) le contrôle du respect de la réglementation et (e) l'aide à la conduite.

Nous nous sommes intéressés plus particulièrement aux systèmes d'aide à la conduite visant à pallier les problèmes liés à la sécurité en améliorant le contrôle du véhicule et de l'infrastructure du réseau routier. Ces systèmes permettent d'assister le conducteur en l'informant sur les conditions de conduite. Pour ces raisons, les constructeurs automobiles sont

aujourd'hui en mesure de développer des systèmes d'aide à la conduite automatisant certaines tâches.

Dans ce chapitre, nous décrivons, dans un premier temps, certains systèmes de sécurité routière passive et active existants ainsi que quelques exemples de systèmes de sécurité active (*i.e.,* systèmes d'aide à la conduite) proposés dans la littérature. Nous présentons aussi les importantes caractéristiques de ces systèmes comme la fiabilité, l'imprécision et la tolérance aux fautes. Vu que la tolérance aux fautes permet d'améliorer la fiabilité et la sécurité des systèmes d'aide à la conduite, nous étudions, ensuite, les différentes méthodes de tolérance aux fautes des capteurs spécifiques aux automobiles. En effet, nous décrivons les deux méthodes existantes dans la littérature : redondance matérielle et redondance logicielle.

1.2 Systèmes de sécurité routière

Les études de l'accidentologie faites par l'Association Prévention Routière (APR) ont montré que la France a enregistré une forte baisse de l'insécurité routière en 2013 par rapport à l'année 2012 : le nombre de tués a diminué de 11% et le nombre de blessés est en baisse de 6,6% [ONISR, janvier 2014]. Ces accidents sont causés par certains facteurs que nous résumons dans les points suivants :

- L'erreur humaine, qui constitue la première cause des accidents. En effet, ce facteur est lié à l'inattention du conducteur, la fatigue, l'inexpérience et le non respect des règles de conduite (*e.g.*, dépassement de la vitesse autorisée et dégradation du contrôle de la trajectoire par le conducteur).
- La perte de contrôle du véhicule, qui est liée généralement à la dynamique du véhicule (*e.g.,* freinage inadapté et effet de survirage ou de sous-virage), à l'infrastructure (*e.g.,* chaussées glissantes et présence d'obstacles fixes sur la route) et aux conditions météorologiques (*e.g.,* neige et brouillard).

Les études de l'accidentologie faites par l'APR ont montré que la réduction du nombre d'accidents de la route passe par la conception et le développement des systèmes de sécurité visant à améliorer la sécurité routière et à assurer la sûreté des passagers.

Nous distinguons, d'une part, les systèmes de sécurité passive (*e.g.,* airbag et ceintures de sécurité) qui réduisent les dangers et limitent les conséquences des accidents et, d'autre part, les systèmes de sécurité active (*e.g.,* régulateur de vitesse à contrôle de distance et limiteur de vitesse intelligent) qui servent à éviter les accidents en avertissant le conducteur des dangers (alertes visuelles, sonores ou haptiques) ou en agissant directement sur les commandes du véhicule (freinage automatique, contrôle d'interdistance, etc.).

1.2.1 Systèmes de sécurité passive

Les systèmes de sécurité passive ou secondaire sont destinés à protéger les occupants du véhicule en cas d'accidents et à réduire la gravité des blessures. Les principaux systèmes de sécurité passive sont :

- La ceinture de sécurité, qui permet de maintenir les occupants sur leurs sièges pour limiter leurs mouvements lors d'un choc. En effet, la ceinture de sécurité évite le risque d'éjection du véhicule. Ainsi, elle empêche les occupants de se heurter la tête contre les éléments de l'habitacle tels que le pare-brise, le volant ou le tableau de bord. Lorsque la ceinture est associée à un prétensionneur pyrotechnique [1] [Muller and Linn, 1998] et un limiteur d'efforts [2] [Renault, 2008], elle permet de transférer l'énergie du choc dans les attaches de ceintures pour réduire la force du choc.
- Le coussin gonflable de sécurité (ou airbag) [Evans, 2004], qui permet de protéger l'occupant lors d'une collision. En effet, ce coussin se gonfle rapidement d'air ou de gaz en cas de choc pour éviter que l'occupant ne heurte la tête contre le volant ou que le passager *avant* ne se projette contre le tableau de bord. Cependant, l'utilisation du coussin gonflable associé à la ceinture assure une sécurité plus efficace [Valeo, 2008].
- L'appui-tête actif (*e.g.,* l'appui-tête actif de Opel [General Motors Corporation, 2008]), qui permet de réduire les risques de blessures cervicales en cas de choc arrière. En cas de ce type de choc, l'appui-tête se déclenche d'une manière mécanique par le mouvement

1. un dispositif qui a pour objectif de tirer la ceinture vers l'arrière lors d'un choc pour bien plaquer le corps du passager au siège.
2. un dispositif qui permet de diminuer la pression de la ceinture sur le thorax en cas de choc.

subi par le dos du passager ou du conducteur qui permet à l'appui-tête de s'avancer pour se plaquer à la tête du conducteur ou du passager.

Ces systèmes de sécurité passive protègent les occupants des véhicules contre les chocs forts. Ainsi, ces systèmes rendent les véhicules eux-mêmes moins dangereux en cas d'accidents. Malgré tout ça, l'offre en matière de sécurité routière reste insuffisante vu que certaines règles de conduite ne sont pas toujours adaptées à nos propres limites physiologiques (acuité visuelle, évaluation des distances, perte d'attention, etc.). Pour ces raisons, les constructeurs d'automobiles cherchent à améliorer la sécurité par le développement de nouveaux systèmes de sécurité destinés à éviter les accidents en apportant une assistance au conducteur. De nombreux systèmes de sécurité active ont été développés pour assister le conducteur. Ils sont largement implantés sur les véhicules actuels. Le but de ces systèmes est d'améliorer la contrôlabilité du véhicule et obtenir le meilleur comportement dynamique possible dans toutes les situations, de la plus courante à la plus imprévue.

1.2.2 Systèmes de sécurité active

Les avancées technologiques de ces dernières années tentent de rendre la conduite plus facile en développant des systèmes de sécurité active ou d'aide à la conduite (ADAS [3]). Ces systèmes, dit "intelligents", ont pour but de faciliter la maîtrise du véhicule et d'optimiser la tâche de conduite en informant le conducteur, suffisamment à l'avance, des situations de danger. Ces systèmes identifient des situations critiques en se basant sur des fonctions de risques bien définies. Par exemple, le système de régulation de vitesse à contrôle de distance permet de détecter un risque de collision en se basant sur le calcul de l'indicateur *temps avant collision* (Time To Collision : TTC).

La figure 1.1 montre l'architecture générale d'un système d'aide à la conduite. Ces systèmes sont basés sur la perception de l'environnement par l'intermédiaire de capteurs proprioceptifs et extéroceptifs implantés dans le véhicule. Les capteurs proprioceptifs (*e.g.,* capteur de vitesse et accéléromètre) permettent d'acquérir des mesures relatives à l'état et la

3. Advanced Driver Assistance Systems

dynamique du véhicule, telles que sa vitesse et son accélération. Par contre, les capteurs extéroceptifs (*e.g.,* radar et caméra), ils permettent de prélever les informations relatives à l'environnement extérieur du véhicule (*e.g.,* les routes et les obstacles). Chaque donnée acquise est ensuite transmise à une unité de fusion pour être traitée (*e.g.,* convertir le signal, réduire le bruit et extraire les informations à partir d'une image). Si des capteurs produisent des données redondantes ou complémentaires, ces données sont fusionnées (*e.g.,* par moyenne) par l'unité de fusion pour obtenir une image précise de l'état actuel. Par la suite, les données traitées par l'unité de fusion sont transmises vers le module de décision, qui permet de les traiter, d'analyser la situation courante et de choisir les actions en fonction de l'environnement, du comportement du conducteur et de l'état du véhicule. Enfin, ces actions sont envoyées vers le module d'action, qui produit, soit des alertes (visuelles, sonores et/ou haptiques), soit des actions automatiques sur certains organes pour tenter d'éviter ou de limiter les effets de l'accident.

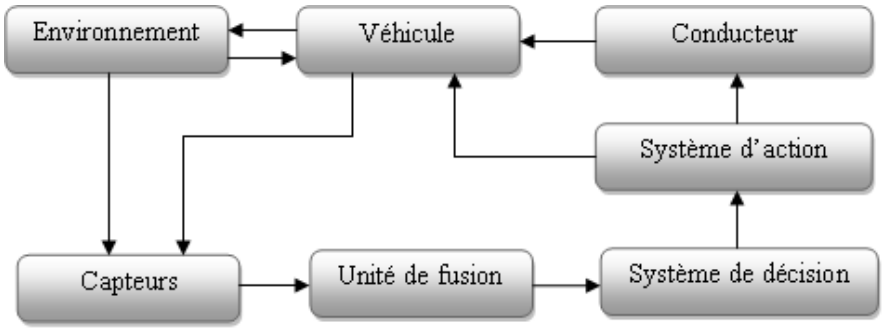


FIGURE 1.1 : Architecture d'un système d'aide à la conduite.

Les systèmes d'aide à la conduite peuvent être classés selon le contrôle en :

- des systèmes de contrôle latéral, permettant de maintenir la trajectoire du véhicule et d'éviter les collisions. Ils permettent la prévention des changements involontaires de voie tels que le système d'alerte de franchissement de ligne et le système d'aide au changement de voie.
- des systèmes de contrôle longitudinal, permettant de réguler la vitesse et maintenir une distance de sécurité entre les véhicules tels que le régulateur de vitesse à contrôle de distance et le système d'alerte pour la prévention de collision *avant*.

- des systèmes de contrôle mixte, permettant d'assurer le contrôlé longitudinal et le contrôle latéral.

Dans la suite de cette section, nous citons quelques exemples de systèmes d'aide à la conduite existants dans la littérature. Nous présentons quelques systèmes d'aide au guidage longitudinal, et quelques systèmes assurant le contrôle latéral. Enfin, nous citons quelques exemples de systèmes assurant en même temps le guidage longitudinal et latéral.

1.2.2.1 Système de contrôle longitudinal

Régulateur de vitesse adaptatif

Le régulateur de vitesse adaptatif à contrôle de distance, ou ACC (Adaptive Cruise Control) est un système d'assistance permettant une conduite plus sûre et plus confortable. Le système ACC est conçu pour être utilisé sur des autoroutes et il ne fonctionne qu'à partir de 30 km/h. Il est intégré et testé dans les voitures modernes tels que BMW [Prestl et al., 2000]. ACC s'apparente à un régulateur de vitesse standard, où une vitesse est mémorisée par le conducteur et le système agit sur l'accélérateur pour maintenir cette vitesse. En supplément, il maintient une distance de sécurité entre le véhicule et un autre qui le précède. Ceci est rendu possible grâce à un télémètre embarqué (radar ou laser, selon la marque du véhicule) qui permet de contrôler la scène routière à l'avant du véhicule. En effet, le capteur *radar* ou *laser* détecte la présence d'un véhicule sur la même voie et mesure la distance du véhicule qui précède ainsi que sa vitesse. De plus, ACC détermine, à partir de l'accélération latérale et de l'angle au volant, la trajectoire du véhicule.

Si le véhicule détecté roule à une vitesse inférieure à celle du véhicule équipé d'ACC, alors le système décélère automatiquement pour conserver la distance de sécurité prédéterminée et avertit le conducteur par une alerte visuelle et sonore si le conducteur doit intervenir. Dans le cas où la voie est à nouveau libre, le système ré-accélère automatiquement jusqu'à la vitesse programmée par le conducteur.

Système de risque de Collision *Avant*

Le système d'alerte de risque de collision *avant*, ou FCW (Frontal Collision Warning) a pour objectif d'éviter les collisions. Son principe de fonctionnement s'appuie sur une surveillance de la route à l'aide de capteurs *laser* [Biral et al., 2010a] ou de caméras [Raphael et al., 2011], qui sont installés à l'avant du véhicule. Ces capteurs permettent de détecter des obstacles et de déterminer la distance entre le véhicule et ces obstacles. Si le système détecte que le véhicule s'approche trop près de l'obstacle, alors il fournit des alertes au conducteur par le biais d'une interface homme-machine afin que le conducteur réagisse et par conséquent évite la collision.

Système d'alerte de vitesse excessive en virage

Le système d'alerte de vitesse excessive en virage (SAVV ou CSW : Cruve Speed Warning) [Biral et al., 2010b] consiste à avertir le conducteur lorsque sa vitesse est inadaptée en s'approchant d'un virage. Cela va atténuer le danger de perte de contrôle. Son principe réside dans l'utilisation d'un récepteur GPS, un capteur de vitesse, et un capteur de lacet afin de localiser le véhicule, et des cartes routières numériques préexistantes pour extraire les caractéristiques de la route. En combinant les informations sur la localisation du véhicule et les caractéristiques de la route avec des facteurs externes (*i.e.*, conditions météorologiques et situation du trafic), la vitesse maximale autorisée pour le virage est estimée. Si le véhicule franchit la vitesse limite autorisée pour un virage, alors le système avertit le conducteur suffisamment à l'avance pour lui permettre d'adapter sa vitesse avant d'entrer dans le virage.

Saspence

Le sous-projet Saspence (SAfe SPEed and Safe DistaNCE) [Bertolazzi et al., 2010] a été introduit au sein du projet PReVENT [4] (PReVENTive and Active Safety Applications). L'objectif principal de ce sous-projet est de développer un système capable d'aider le conducteur à maintenir la vitesse maximale autorisée et la distance de sécurité selon les conditions de conduite données (*i.e.,* géométrie de la route, situation du trafic et conditions météorolo-

4. www.prevent-ip.org

giques). Ce système sert à éviter les risques d'accidents liés à une vitesse excessive et à une distance inappropriée à une situation particulière.

Le module de décision du système consiste à estimer une trajectoire optimale et à calculer une manœuvre de référence (*i.e.,* calculer la vitesse maximale et la distance de sécurité) en se basant sur la fusion des données relatives à l'état du véhicule et à son environnement (géométrie de la route, obstacles, etc.). Si le système de décision détecte une situation critique en comparant la manœuvre de référence avec la manœuvre actuelle, alors il fournit des alertes au conducteur (alertes visuelles, sonores ou vibrations de la pédale d'accélération).

1.2.2.2 Système de contrôle latéral

SAFELANE

Le sous-projet SAFELANE [Amditis et al., 2010] fait partie du projet européen PReVENT. SAFELANE a pour objectif de développer un système permettant d'assister le conducteur en cas de sortie de voie involontairement. Ce système est principalement prévu pour fonctionner sur autoroutes, routes nationales et départementales et dans des conditions défavorables.

Le système SAFELANE est composé d'un système de capteurs, d'actionneurs et d'un composant de décision. Le développement de SAFELANE est basé sur la fusion des données provenant du système de vision et des cartes digitales afin de donner une image précise de l'environnement. Ces données sont ensuite transmises vers le système de décision adaptatif. Ce dernier permet d'analyser ces données, de détecter la voie et d'estimer la trajectoire la plus probable. Au cas où le système de décision détecte une sortie involontaire de voie, il déclenchera une alerte (sonore ou haptique) ou bien une action active, en contrôlant la direction du volant.

Lateral Safe

Le sous-projet Lateral Safe [Amditis et al., 2008] a été étudié au sein du projet PReVENT. Il a pour but de développer un système d'aide à la conduite permettant d'éviter les situations critiques en latéral. Il consiste à assister le conducteur en élaborant des alertes pour accroître

la sécurité et éviter les risques de collisions. Ce système fournit des fonctions basées sur les données issues des radars de gamme courte et longue portée, des caméras et des capteurs proprioceptifs. Ces fonctions analysent la situation et envoient les décisions à l'interface homme-machine (visuelle et sonore) afin d'avertir/d'informer/d'alerter le conducteur de la situation actuelle du trafic, si nécessaire. Ces fonctions sont résumées dans les points suivants :

- une fonction d'avertissement de collision latérale, ou LCW (Lateral Collision Warning), qui avertit le conducteur de la présence d'un obstacle et d'un risque imminent de collision. Cette fonction n'est activée que lorsque la vitesse du véhicule est supérieure à 20 km/h.
- une fonction d'avertissement de collision latérale et arrière, ou LRM (Lateral and Rear Monitoring), qui améliore le perception du conducteur. Elle vise à réduire les risques de collision latérale et en arrière, particulièrement en cas d'une mauvaise visibilité ou d'inattention du conducteur.
- une fonction d'aide au changement de voie, ou LCA (Lane Change Assistant), qui assiste le conducteur en cas de changement de voie.

Système d'alerte de franchissement de lignes

L'avertisseur de sortie de voie, ou LDW (Lane Departure Warning), fournit un contrôle latéral en avertissant le conducteur d'un franchissement involontaire des marquages au sol sans actionner le clignotant [Mobileye, 2007].

Le fonctionnement de ce système se base sur la détection des marquages sur la voie à l'aide d'un capteur vidéo. Ce système fonctionne lorsque le véhicule franchit le marquage sans que le clignotant ne soit activé. Lorsqu'il détecte une sortie de voie involontaire, il avertit le conducteur à l'aide d'une alerte acoustique ou des vibrations (un vibreur situé sous le siège du conducteur, sur le même côté où le franchissement de la ligne a lieu ou une légère torsion au volant).

1.2.2.3 Système de contrôle longitudinal et latéral

Système d'alerte pour prévenir les sorties de route

Le système d'alerte pour prévenir les sorties de route, ou RDCWS (Road Departure Crash Warning System) a été réalisé dans le cadre du projet RDCW FOT [LeBlanc et al., 2006]. Il sert à fournir une assistance longitudinale en délivrant un avertissement sur la vitesse maximale autorisée à l'approche d'un virage ainsi qu'une assistance latérale, en avertissant le conducteur s'il fait un déplacement latéral excessif.

Ce système est constitué des deux sous-systèmes suivants : (i) le système d'alerte de vitesse excessive en virage, CSW (Curve Speed Warning) et (ii) le système d'alerte d'une dérive latérale, LDWS (Lateral Departure Warning System). Ce dernier fournit une assistance latérale en alertant le conducteur d'une dérive involontaire de la route à cause de l'inattention ou de la fatigue du conducteur.

LDWS utilise les informations issues d'un capteur vidéo et des algorithmes de détection de marquages, en combinaison avec les informations à propos de la dynamique du véhicule (*i.e.,* vitesse et position) pour déterminer la position et l'orientation du véhicule. Cela va permettre de calculer un indicateur de risque qui est le temps de sortie de voie, ou TLC [5] ($TLC = \frac{DLC}{v}$, avec DLC la distance au bord de la voie et v la vitesse du véhicule). Le système délivre une alerte dans le cas où la valeur de TLC est inférieure à une valeur *seuil*. L'avertissement peut être acoustique ou haptique (*i.e.,* une vibration du siège du conducteur ou une légère torsion sur le volant).

Navigation Aided Intelligent Cruise Control

Le projet NAICC (Navigation Aided Intelligent Cruise Control) consiste à élaborer un système permettant la détection d'une situation critique, relative à une vitesse et/ou à une trajectoire du véhicule, inadaptée à la condition de conduite [Lauffenburger, 2002]. Ce système a évolué de la manière suivante : (i) dans un premier temps, il a été considéré comme un système d'information du conducteur, permettant d'informer le conducteur de la consigne

5. Time to Line Crossing

adéquate à la situation détectée, (ii) puis, il est devenu un système d'assistance active permettant de contrôler la vitesse et/ou la trajectoire du véhicule.

NAICC intègre l'assistance longitudinale en régulant la vitesse du véhicule et l'assistance latérale en contrôlant la trajectoire. L'assistance longitudinale sert à calculer la vitesse appropriée à la situation de conduite (virage, intersection, etc.) en fonction de la localisation du véhicule sur la route. Par la suite, le système consiste à définir les caractéristiques de l'action de freinage, tels que la distance et le potentiel de freinage. Cette assistance se base principalement sur les informations provenant des cartes digitales, de la localisation du véhicule et de la dynamique du véhicule. L'assistance latérale est fondée sur la localisation du véhicule afin d'estimer une référence de trajectoire. Cette référence est déterminée à l'aide de la dynamique du véhicule et de la situation de conduite.

SAFERIDER

Le projet SAFERIDER[6] vise à fournir de l'assistance à la conduite destinée particulièrement aux véhicules de type "deux-roues" motorisés, appelés "Advanced Rider Assistance Systems (ARAS)" [Bekiaris et al., 2009]. Ce projet consiste également à améliorer la sécurité des motards et à réduire le nombre d'accidents.

SAFERIDER se décompose en cinq fonctions :

1. Fonction d'alerte de vitesse excessive permettant d'avertir le motard lorsque la vitesse limite autorisée est dépassée.
2. Alerte de virage permettant d'avertir le motard d'une situation dangereuse en se basant sur la comparaison de la manœuvre actuelle et une manœuvre de référence.
3. Alerte de collision en avant permettant d'avertir le motard d'un risque de collision avec un obstacle situé à l'avant de la moto.
4. Alerte sur intersection permettant d'avertir le motard sur le risque de collisions avec des obstacles fixes et mobiles sur les intersections.

6. www.saferider-eu.org

5. Aide au changement de voie, ou LCS (Lane Change Support) permettant d'alerter le motard en cas de changement de voie, et par conséquent d'un risque d'accident.

Le principe de ces fonctions réside dans le calcul d'une manœuvre de référence optimale en prenant en compte la dynamique de la moto, les caractéristiques de la route et l'état des obstacles situés à l'avant de la moto. Si le système détecte une situation critique (en comparant la manœuvre actuelle à la manœuvre de référence optimale), alors il génère des avertissements au motard par le biais des interfaces homme-machine visuelles et haptiques (au moyen des vibrations au niveau de l'accélérateur et au niveau des gants).

1.2.2.4 Synthèse

Nous récapitulons dans le tableau 3.5 une analyse effectuée sur les systèmes d'aide à la conduite présentés précédemment. Nous nous intéressons particulièrement aux critères suivants : les modes de contrôle et les différentes sorties.

Tableau 1.1 : Comparaison de différents systèmes d'aide à la conduite.

	Différentes sorties		**Mode de contrôle**	
	Système d'alerte	Système de réaction automatique	Contrôle longitudinal	Contrôle latéral
ACC	oui	oui	oui	**non**
FCW	oui	**non**	oui	**non**
CSW	oui	**non**	oui	**non**
Saspence	oui	**non**	oui	**non**
Lateral safe	oui	**non**	**non**	oui
Safelane	oui	oui	**non**	oui
LDW	oui	**non**	**non**	oui
RDCW	oui	**non**	oui	oui
NAICC	oui	oui	oui	oui
SAFERIDER	oui	**non**	oui	oui

Le tableau 3.5 montre que tous les systèmes d'aide à la conduite se basent sur la génération des alertes par le biais des composants d'interface homme-machine pour avertir le conducteur d'une situation critique. Il montre également que quelques systèmes se basent simultanément

sur la génération des alertes à travers les composants d'interface homme-machine et sur l'exécution des actions automatiques par l'activation des organes du véhicule pour éviter les dangers.

1.3 Caractéristiques des systèmes d'aide à la conduite

Le développement des systèmes d'aide à la conduite nécessite la définition des besoins fonctionnels en termes de fonctions souhaitées et de contraintes opérationnelles. Le développement de ces systèmes exige l'utilisation d'une combinaison de composants matériels et logiciels. Ces composants sont confrontés à des contraintes strictes en termes de ressources (*e.g.,* la taille de la mémoire et les ressources d'entrée et de sortie) car l'espace est limité dans les véhicules. En outre, ces systèmes sont des systèmes temps réel critiques, qui nécessitent le respect des contraintes temporelles, la tolérance à l'imprécision, la tolérance aux fautes et un haut niveau de fiabilité et de sécurité. Ces caractéristiques sont résumées dans les paragraphes suivants.

1.3.1 Contraintes temps réel

Les systèmes d'aide à la conduite font partie des systèmes temps réel qui sont définis comme suit : "*Un système temps réel est un système dont le comportement dépend non seulement de l'exactitude des traitements effectués, mais également de l'instant où les résultats de ces traitements sont fournis*" [Stankovic and Ramamritham, 1988]. Le temps est souvent une contrainte importante dans les systèmes d'aide à la conduite. En effet, ces systèmes doivent respecter des contraintes temporelles qui sont définies sous forme d'échéances des traitements et/ou de durées de validité des données. Le non respect de ces contraintes peut entrainer des conséquences graves sur le système.

1.3.2 Imprécision

Attendu que les systèmes temps réel doivent respecter des contraintes du temps, ils peuvent parfois tolérer des résultats imprécis ou incomplets plutôt que des résultats exacts mais obtenus en retard. Cette imprécision doit être bornée pour éviter les situations critiques dues à des valeurs erronées.

1.3.3 Fiabilité

La fiabilité est l'aptitude d'un système à continuer son fonctionnement d'une manière correcte pendant une période de temps bien définie et dans des conditions données [OMG, 2004]. Cette propriété peut être déterminée en termes de fausses alertes positives et de fausses alertes négatives. Les fausses alertes positives sont des avertissements qui sont déclenchés dans des situations normales. Ce type d'alertes peut être une source de distraction pour le conducteur car ce dernier réagit à une situation dangereuse inexistante. Les fausses alertes négatives se produisent lorsqu'aucun avertissement n'est déclenché dans des situations dangereuses. Ces anomalies peuvent conduire à des situations graves dans lesquelles le conducteur ou le système ne réagit pas pour éviter le danger. Ainsi, un taux élevé de fausses alertes conduit à une dégradation des performances de conduite et à une diminution de la fiabilité des systèmes d'aide à la conduite.

1.3.4 Sécurité

La sécurité est l'aptitude d'un système à éviter de faire apparaître, dans des conditions données, des situations critiques ou catastrophiques [Avizienis et al., 2004]. Avec la demande de véhicules plus sûrs et la diminution des dangers, la sécurité est donc devenue un élément primordial pendant la conception des systèmes d'aide à la conduite. Pour assurer la sécurité, ces systèmes doivent intervenir ou alerter le conducteur en cas de détection d'un risque. Par exemple, dans un système d'alerte de risque de collision, la sécurité dépend de la probabilité

de collision. Cela implique que l'absence de collision introduit un haut niveau de sécurité du véhicule et son environnement.

1.3.5 Tolérance aux fautes

La tolérance aux fautes est la capacité d'un système à maintenir ses fonctions malgré l'occurrence de fautes, tels que les défauts matériels (*e.g.,* des fautes dues à l'interaction du système avec l'environnement) et logiciels (*e.g.,* erreur de spécification) [Blanke et al., 2001]. Les systèmes d'aide à la conduite sont constitués d'une combinaison de composants matériels et logiciels où une défaillance peut conduire à des situations critiques, ce qui peut avoir un impact sur la sécurité et la fiabilité des systèmes d'aide à la conduite. Par conséquent, ces systèmes doivent être tolérants aux fautes. Ainsi, la tolérance aux fautes permet d'assurer la fiabilité et la sécurité des systèmes d'aide à la conduite, deux importantes propriétés dans ces systèmes [Piao and McDonald, 2008]. Pour cette raison, nous nous sommes intéressés à l'amélioration de la tolérance aux fautes des systèmes d'aide à la conduite, et nous proposons deux approches que nous décrivons dans la section suivante.

1.4 Méthodes pour la tolérance aux fautes

Plusieurs travaux se sont intéressés à la proposition de méthodes pour la tolérance aux fautes pour les capteurs (*e.g.,* [Sangha et al., 2011] [Varrier et al., 2013]) et pour les actionneurs (*e.g.,* [Németh et al., 2012] [Sename et al., 2013]).

Les capteurs constituent les premiers composants responsables de la production des valeurs nécessaires pour les traitements, le contrôle et la prise de décision. Ainsi, les défauts des capteurs peuvent conduire à des valeurs différentes des valeurs réelles, ce qui peut conduire à une dégradation de la performance du système et/ou à des situations dangereuses. Par conséquent, il est important que les systèmes puissent tolérer des fautes des capteurs pouvant se produire au niveau matériel (*e.g.,* des défauts mécaniques) ou logiciel (*e.g.,* erreurs

de conception des composants logiciels). C'est pourquoi, plusieurs travaux ont développé des méthodes pour la tolérance aux fautes des capteurs. Parmi ces travaux, il y a la redondance matérielle et la redondance logicielle.

1.4.1 Redondance matérielle

La redondance matérielle consiste à dupliquer les capteurs pour tolérer les fautes pouvant se produire au niveau de ce type de composant (par exemple, si un premier capteur tombe en panne, le second prend la relève permettant ainsi au système d'être encore disponible). Il existe deux catégories de redondance matérielle :

- La redondance statique ou passive, basée sur le mécanisme de vote pour masquer les fautes sans avoir recours aux méthodes de détection d'erreur [Isermann et al., 2002]. En effet, cette méthode utilise le principe de la majorité des votes. La forme la plus commune et simple de la redondance statique est appelée *redondance modulaire triple* TMR (Triple Modular Redundancy) [Zou and Xu, 2009]. La technique TMR, comme le montre la figure 1.2, utilise trois capteurs en parallèle produisant le même signal d'entrée et un composant *voteur*, qui compare les signaux de sortie et décide du résultat correct à la majorité des votes. Un résultat correct signifie qu'aucun ou qu'un seulement des capteurs défaille. Un résultat erroné implique que deux ou les trois capteurs défaillent. Cependant, le principal problème de cette méthode se pose lorsque le composant *voteur* défaille. Il entraine en effet une défaillance du système. La technique TMR est un cas spécial de redondance modulaire NMR (N-Modular Redundancy) [Lombardi et al., 2001].

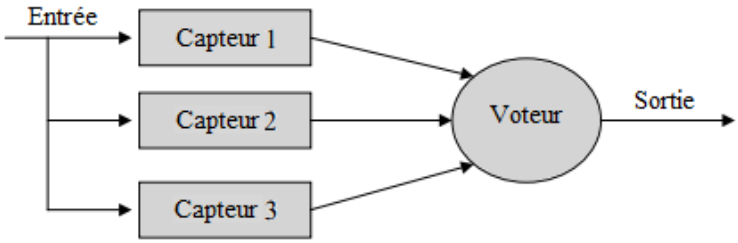


FIGURE 1.2 : Redondance modulaire triple TMR.

- La redondance dynamique ou active permettant de détecter la présence de défauts et d'effectuer des actions pour supprimer les composants défectueux. Cette méthode nécessite au moins deux capteurs et la détection de défauts pour chaque capteur plutôt que le masquage des fautes [Zou and Xu, 2009]. La redondance dynamique peut être réalisée par l'utilisation de deux modules où le premier est en service, le deuxième module fonctionnant seulement en cas d'erreur. Lorsque le deuxième module fonctionne en permanence, cette méthode est appelée *"hot standby"* [Isermann et al., 2002]. Lorsque le module de secours ne devient opérationnel qu'en cas d'erreur, elle est appelée *"cold standby"* [Isermann et al., 2002]. Cependant, la redondance dynamique exige plus de traitement de l'information. En outre, cette technique ne convient pas pour les applications temps réel [Anwar, 2010].
- La redondance mixte repose sur le mécanisme de vote et sur la détection d'erreurs [Isermann et al., 2002]. Cette technique demande plus de composants, une configuration du système à chaque fois et plus de traitements, ce qui rend l'architecture du système très complexe.

La principale difficulté de la redondance matérielle réside dans les problèmes de coût et de taille causés par l'installation de matériels supplémentaires. Or, les systèmes d'aide à la conduite exigent des contraintes strictes en termes de ressources limitées, tels que la taille, le coût et le poids. C'est pourquoi, la redondance matérielle ne présente pas une solution efficace pour les automobiles. Pour pallier ces problèmes, plusieurs travaux ont proposé des méthodes de commande tolérantes aux fautes qui mettent en œuvre un mécanisme de détection de défauts, puis de traitement de ceux-ci afin de reconfigurer le système sans avoir à dupliquer les composants matériels.

1.4.2 Redondance logicielle

La redondance logicielle se base sur l'utilisation d'un système de commande tolérant aux fautes qui vise à maintenir la stabilité du système et des performances acceptables en dépit de la présence de défauts. Ce système possède la capacité de s'adapter automatiquement à des défauts pouvant impacter ses composants.

Comme illustré par la figure 1.3, les systèmes de commande tolérant aux fautes peuvent être classés en deux grandes familles [Zhanga and Jiang, 2008] : les systèmes de commande tolérants passifs (Passive Fault Tolerant Control System) et les systèmes de commande tolérants actifs (Active Fault Tolerant Control System).

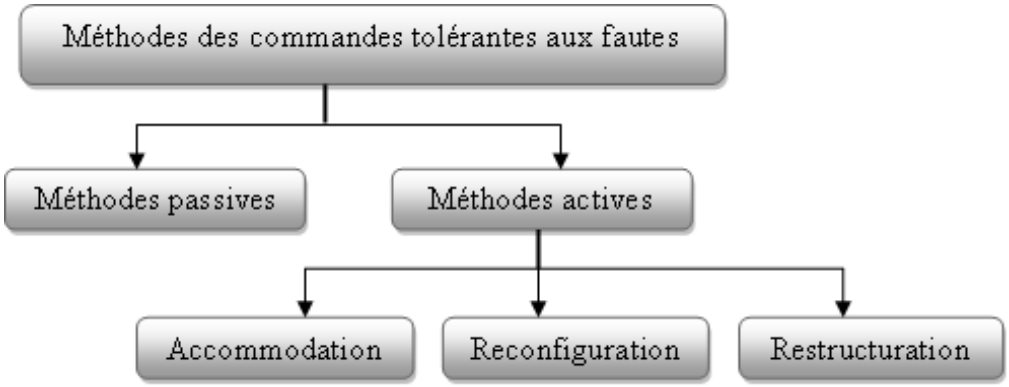


FIGURE 1.3 : Classification des méthodes des commandes tolérantes aux fautes.

1.4.2.1 Systèmes de commande tolérants passifs

Un système de commande tolérant passif est basé sur des techniques de contrôle robuste afin d'assurer que le système en boucle fermée est robuste aux incertitudes et à certains défauts connus a priori [Oudghiri, 2008]. Ces défauts sont considérés lors de la conception du système de commande. Par conséquent, cette approche ne demande pas d'informations en ligne sur les défauts [Steffen et al., 2009] et permet de tolérer les défauts sans modifier les paramètres du contrôleur. Ce qui fait que cette approche ne nécessite aucun module de détection de défauts, ni de reconfiguration du système. Mais, l'application de cette approche est limitée en raison de ses inconvénients. En effet, un système de commande tolérant passif possède une capacité limitée de tolérances aux fautes vu qu'il ne considère qu'un nombre restreint de défauts. Il semble donc que l'approche passive soit suffisante dans les applications qui présentent une classe de défauts restreints et connus [Oudghiri, 2008]. En outre, plus le nombre de défauts sera important et leur impact sur le système grave, moins le système de commande tolérant passif sera capable de tolérer ces défauts, ce qui augmente la dégradation du système.

Plusieurs stratégies adaptées à l'approche passive ont été proposées pour tolérer les défauts des capteurs pour les systèmes automobiles. Parmi ces stratégies, on trouve : la technique de l'inégalité matricielle linéaire (LMI : Linear Matrix Inequality) [Tong et al., 2011], la commande robuste H_∞ [Varrier et al., 2013] et la commande H_2/H_∞ [Habibinia et al., 2013].

1.4.2.2 Systèmes de commande tolérants actifs

L'approche active de commande tolérante aux fautes consiste à réagir à des défauts imprévus affectant le système en modifiant les lois de commande (*i.e.,* modifier la structure et/ou certains paramètres) de manière à assurer la stabilité du système [Oudghiri, 2008]. Cette approche exige des techniques de détection, d'isolation et d'accommodation de ces défauts (*i.e.,* adaptation des paramètres du régulateur) [Blanke et al., 2001] ou reconfiguration du système (*i.e.,* modification de la structure du système pour compenser les défauts) [Boskovic and Mehra, 2004] ou restructuration (*i.e.,* modification des paramètres du régulateur et de la structure du système) [Staroswiecki and Gehin, 2001]. Le principal mode de fonctionnement d'un système de commande tolérant actif s'appuie généralement sur les deux modules suivants :

- Un module de détection et d'isolation de défauts (FDI : Fault Detection and Isolation), qui permet de détecter et de localiser les défauts. De plus, il permet d'estimer les paramètres d'état du système. À l'apparition d'un défaut, le module FDI fournit en ligne au module FTC (Fault Tolerant Control) les informations relatives au défaut détecté (*e.g.,* type de défaut et instant d'apparition) et l'état du système.
- Un module de tolérance aux défauts (FTC : Fault Tolerant Control), basé sur un mécanisme de reconfiguration (ou d'accommodation ou de restructuration) pour maintenir la stabilité et les performances du système. La reconfiguration (ou l'accommodation ou la restructuration) dépend des informations délivrées par le module FDI.

Plusieurs stratégies, utilisant l'approche active pour les défauts de type capteurs dans les véhicules, ont été développées dans la littérature. Parmi ces stratégies, on peut citer : les réseaux de neurones [Sangha et al., 2012], les filtres (*e.g.,* filtre de kalman et le filtre

Luenberger) [Gietelink, 2007] [Tabbache et al., 2013], les observateurs [Abdo et al., 2009] [Viehweider et al., 2012] et la logique floue [Quet and Salman, 2007].

1.4.3 Synthèse

À partir des travaux existants concernant la détection et l'isolation des fautes (FDI/FTC), nous constatons que ces méthodes ne nécessitent pas une configuration de matériels supplémentaires, qui peut s'avérer coûteuse à cause des limitations de l'espace dans les véhicules. En revanche, ces méthodes reposent sur des algorithmes et des modèles mathématiques complexes, ce qui rend le comportement des systèmes plus complexe en termes de charge de calcul et de taille de la mémoire [Gietelink, 2007]. En outre, un résultat en retard du module FDI risque de conduire à une dégradation des performances des systèmes temps réel critiques qui doivent respecter leurs contraintes temporelles [Gietelink, 2007]. En revanche, l'approche passive ne demande, ni un module de détection, ni une reconfiguration du système, ce qui permet de gagner du temps par rapport à l'approche active. Mais, l'approche passive ne permet de traiter qu'un nombre restreint et connu de fautes. Pour pallier les problèmes de ces deux approches, nous nous sommes intéressés, dans nos travaux, à définir une méthode de tolérance aux fautes basée sur l'approche passive, que nous avons adaptée. En effet, l'approche que nous proposons ne nécessite pas des calculs complexes et ne pose pas de limites quant au nombre de fautes lors de la conception du contrôleur.

1.5 Conclusion

Nous avons donné dans ce chapitre un aperçu sur les systèmes industriels d'aide à la conduite et sur quelques systèmes qui existent dans la littérature. Ces systèmes permettent d'améliorer la sécurité routière en fournissant une aide au conducteur sous forme passive ou active. En effet, ces systèmes permettent d'avertir le conducteur, suffisamment à l'avance, d'un danger, soit par le contrôle des commandes du véhicule, soit par la génération des alertes à travers une interface homme-machine. Nous avons également montré les importantes

caractéristiques des systèmes d'aide à la conduite. Vu que la tolérance aux fautes possède comme objectif d'améliorer la fiabilité et la sécurité des systèmes d'aide à la conduite, nous nous sommes intéressés à la présentation de différents travaux de recherche qui ont proposé des méthodes de tolérance aux fautes dans le domaine de l'automobile. Les travaux présentés dans ce chapitre reposent essentiellement sur les techniques de tolérance aux fautes affectant les capteurs, étant donné que ces derniers sont les composants qui produisent les données nécessaires aux traitements et à la prise de décision. De ce fait, une défaillance de ce type de composant peut provoquer des résultats différents des résultats réels, et par conséquent, des conséquences graves.

Pour conclure, nous affirmons que les exigences d'un système d'aide à la conduite sont nombreux : respect des contraintes temps réel, maintien de la fiabilité et la sécurité, maintien de la tolérance aux fautes, etc. Le développement d'un système qui répond à toutes ces exigences est compliqué. Pour pallier ce problème, il parait nécessaire d'avoir des méthodes de conception basées sur l'approche de réutilisation. Le chapitre suivant présente les travaux qui portent sur la réutilisation à base de "patrons" de conception.

Chapitre 2

Réutilisation et patrons d'ingénierie

2.1 Introduction

Les systèmes d'aide à la conduite évoluent rapidement et deviennent de plus en plus complexes *i.e.,* de plus en plus de composants (*e.g.,* capteurs et actionneurs), de plus en plus de normes (*e.g.,* eCall), des contraintes temps réel, la gestion d'un grand volume de données et de traitements, etc. Cette complexité et l'accélération du rythme d'évolution de ces systèmes rendent leur conception de plus en plus difficile et coûteuse. Pour réduire cette complexité, de nouvelles méthodologies, basées sur la réutilisation, sont apparues ces dernières années. Les méthodes de réutilisation à base de patrons se sont de plus en plus imposées, depuis la phase d'analyse jusqu'à la phase d'implémentation. Ces méthodes permettent d'améliorer la qualité de développement des systèmes d'aide à la conduite. Elles visent à rendre le développement plus rentable et moins coûteux.

Dans ce chapitre, nous présentons, dans un premier temps, l'approche de réutilisation pour la conception de systèmes. Nous nous sommes intéressés, ensuite, à décrire les différents types de composants réutilisables. Puis, nous montrons que les patrons de conception, étant une forme particulière de réutilisation, constituent un des éléments les plus pertinents pour la modélisation des systèmes d'aide à la conduite, au travers d'une étude comparative des

différents composants. De plus, nous illustrons, dans ce chapitre, la représentation des patrons et nous détaillons également quelques travaux qui proposent des formalismes permettant de décrire les patrons. Nous étudions aussi quelques travaux qui définissent des extensions utilisées pour modéliser les points similaires et les points différents au niveau des patrons. Nous décrivons, ensuite, quelques travaux qui se sont intéressés à la définition des extensions pour garder une trace de l'instanciation des patrons. Nous donnons, enfin, un aperçu sur les travaux qui se sont concentrés sur la définition des patrons dédiés au domaine temps réel, tout en positionnant nos contributions.

2.2 Réutilisation

2.2.1 Définition

La réutilisation consiste à construire un nouveau système à partir des composants réutilisables (*e.g.,* composants, patrons et canevas) existants ayant été testés et mis en œuvre dans des projets antérieurs [Rolland, 2005]. Cette technique s'oppose aux méthodes de développement classiques qui permettent de construire un nouveau système en partant de rien (from scratch). De ce fait, la réutilisation représente aujourd'hui un enjeu majeur de recherche en ingénierie des systèmes d'information. Elle vise à réduire le temps de productivité et à améliorer la qualité de développement. Pour cela, elle est appliquée depuis la phase d'analyse des besoins jusqu'à la phase d'implémentation. Lorsque la réutilisation est appliquée à un domaine particulier, elle permet de répondre à des besoins propres à ce domaine et de mieux maîtriser la multiplication des connaissances.

2.2.2 Processus de réutilisation

L'approche de réutilisation inclut deux phases complémentaires : le développement de composants réutilisables (design for reuse) et le développement de systèmes par réutilisation de composants (design by reuse) comme illustré par la figure 2.1. Le développement de

composants réutilisables a pour objectif de développer des modèles pour la spécification des systèmes réutilisables permettant la représentation des connaissances génériques. La spécification des systèmes réutilisables est mise en œuvre en suivant les tâches d'identification des ressources réutilisables, de spécification, d'organisation et d'implémentation de composants [Cauvet et al., 2001]. L'identification consiste à abstraire les éléments réutilisables en se basant sur l'analyse de produits existants. Dans ce cas, cette analyse s'appuie essentiellement sur deux techniques : l'analyse de similarité [Spanoudakis and Constantopoulos, 1996] [Rekhis et al., 2010a] et la recherche par analogie [Falkenhainer et al., 1989] (*i.e.,* évaluer les ressemblances sémantiques, telle que la nature des relations entre objets, etc.). Une fois que les éléments réutilisables sont identifiés, leur réutilisation doit être associée à une spécification claire et pertinente. La spécification permet de fournir des informations utiles en vue de faciliter leur utilisation (*e.g.,* contraintes d'utilisation et description fonctionnelle du composant). Lorsqu'il s'agit d'une collection de composants réutilisables, il est nécessaire d'exploiter les différents types de relations qui existent entre ces composants pour mieux les utiliser afin de construire des architectures plus importantes. Enfin, les composants doivent être implémentés afin d'automatiser leur réutilisation (gestion automatique de la recherche, de la sélection, de l'adaptation, etc.).

Quant au développement des systèmes d'information par réutilisation, il vise à développer des méthodes et des outils permettant d'exploiter les composants réutilisables pour construire des systèmes d'information [Cauvet et al., 2001]. En effet, cette phase passe par les tâches de recherche, de sélection, d'adaptation et d'intégration. La recherche de composants consiste à trouver un candidat parmi plusieurs en tenant compte des besoins des utilisateurs. Il est donc nécessaire de comparer les composants candidats pour sélectionner le plus adéquat. Lorsque le composant est sélectionné, il doit être adapté au contexte du système à développer en se basant sur différentes techniques, tels que l'instantiation, la spécialisation et le paramétrage [Cauvet et al., 2001]. Enfin, le composant doit être intégré au système.

Dans notre travail, nous nous sommes focalisés sur l'application des deux phases du processus de réutilisation afin de concevoir des composants réutilisables spécifiques aux systèmes

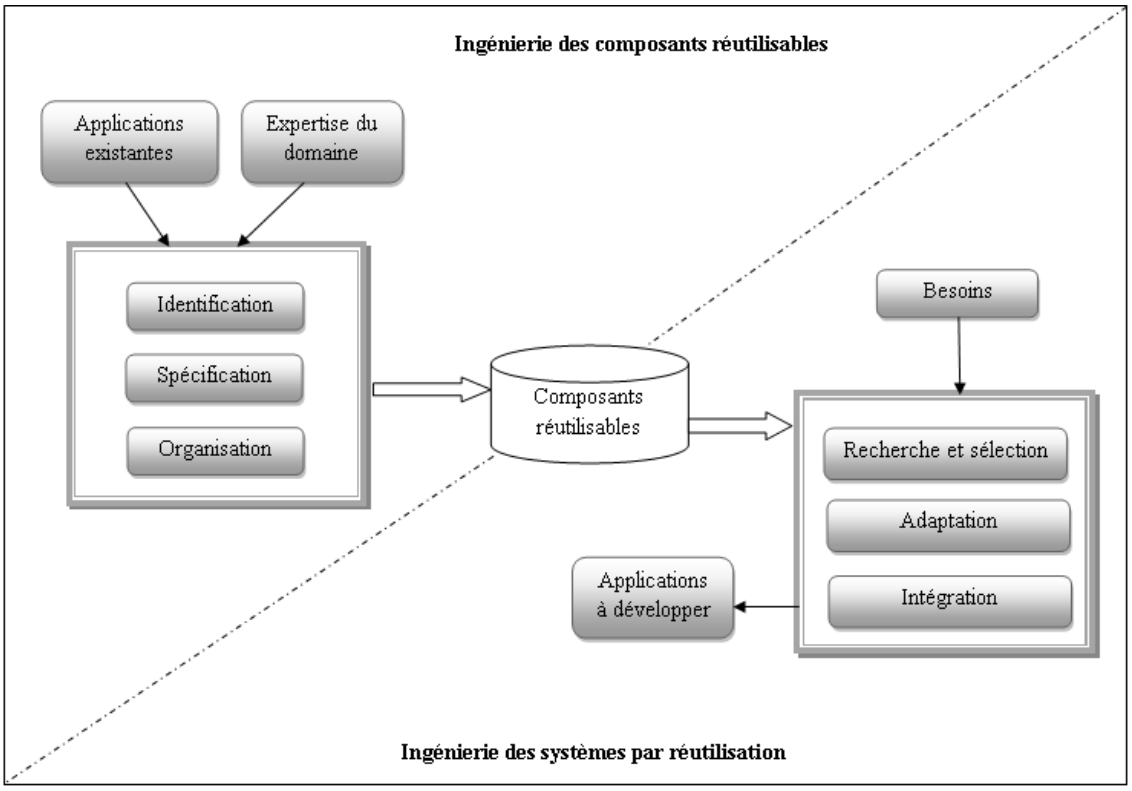


FIGURE 2.1 : Ingénierie pour la réutilisation et ingénierie par la réutilisation [Cauvet et al., 2001].

d'aide à la conduite, d'une part, et d'assister le concepteur dans l'utilisation de ces composants pour développer son système, d'autre part.

2.2.3 Composants réutilisables

Pour introduire la réutilisation dans toutes les phases de développement des systèmes d'information, de nombreux types de composants réutilisables ont été proposés. Nous présentons dans ce qui suit les composants les plus usuels : les composants logiciels, les composants métier, les *frameworks*, les modèles de domaine et les patrons.

2.2.3.1 Composants logiciels

Dans [Szyperski, 2002], un composant logiciel est défini comme *« une unité de composition présentant des interfaces contractuelles et des dépendances explicites avec son contexte »*. Dans [Etienne and Bouzefrane, 2006], un composant logiciel est défini comme : *« une entité logicielle de calcul entièrement indépendante, interagissant avec son environnement au moyen d'interfaces bien définies »*. Pour qu'un composant soit réutilisé dans plusieurs applications, il est nécessaire qu'il soit interopérable au niveau plate-forme et langage. L'interopérabilité de plates-formes signifie qu'un composant peut communiquer avec d'autres composants présents sur différents types de plates-formes. Quant à l'interopérabilité de langage, elle désigne le fait qu'un composant est indépendant de son langage d'implantation. Ainsi, l'interopérabilité permet d'assurer la compatibilité des logiciels, et par conséquent, leur intégration sera plus facile. De nombreuses technologies supportant les composants logiciels ont été proposées, tels que EJB, CORBA, COM/DCOM/.NET. Les composants logiciels sont appliqués pendant la phase d'implantation d'une application. Ils permettent la réutilisation du code source, mais ils ne permettent pas d'appliquer la réutilisation lors des phases d'analyse des besoins et de conception.

2.2.3.2 Composants *métier*

Bien que l'approche à base de composants ait été adoptée en premier lieu dans les activités de programmation, de nouvelles propositions d'unités réutilisables commencent à émerger afin de répondre à des problèmes lors de la phase de recensement des besoins, d'analyse et de conception. Les composants *métier* s'intègrent dans cette démarche. Un composant *métier* est défini de plusieurs manières. Une définition donnée par [Barbier, 2002] est la suivante : *« A business component models and implements business logic, rules and constraints that are typical, recurrent and comprehensive notions characterizing a domain or business area. Within software engineering, business components are key abstractions that are captured during the domain engineering activity. They are software artefacts in the sense that they are not part of reality but embody and represent recurrent invariants relevant to requirements,*

particularly at the earliest phases of development ». Une autre définition est donnée dans [Heineman and Councill, 2001] : *« A business component is a software component that provides functions in a business domain. (...) for example, an order management application is part of the Enterprise Resource Planning (ERP) business domain »*.

On peut donc définir un composant *métier* comme une unité de réutilisation des connaissances au sein d'un même domaine d'applications, dès les phases d'analyse et de conception. En effet, il met l'accent sur la représentation, l'implémentation et le déploiement d'un concept de domaine dans un système logiciel. Ce type de composant diffère des composants logiciels par la prise en considération explicite d'un domaine d'applications. Les composants *métier* sont représentés par des diagrammes UML : diagrammes de classes, diagrammes d'états-transitions, diagrammes d'activités ou diagrammes de composants. Par ailleurs, l'intégration de cette forme de composants logiciels est une tâche difficile. En fait, la collaboration entre composants a souvent été présente en termes syntaxiques tandis que les véritables enjeux sont l'interopérabilité sémantique [Barbier et al., 2002]. En effet, les mécanismes de gestion de conflits sémantiques permettent d'assurer que les concepts échangés par les composants métier ont une signification commune. Ainsi, l'absence de ces mécanismes peut réduire la réutilisation de ces composants.

2.2.3.3 Frameworks

Un *framework*[1] est défini dans [Johnson, 1997] comme étant *« un modèle réutilisable pour tout ou partie d'un système. Il est représenté par un ensemble de classes abstraites et le moyen avec lequel leurs instances interagissent »*. Selon cette définition, on peut déduire qu'un *framework* est considéré comme une application semi-complète. En effet, il permet de capturer l'architecture d'un ensemble d'applications relatives à un domaine donné, et permet de définir une structure globale de ces applications. Cette structure est exprimée à l'aide d'un ensemble de classes abstraites et coopérantes ; un *framework* définit des collaborations entre les classes le constituant, contrairement aux bibliothèques de classes. De ce fait, un

1. cadriciel, architecture ou canevas

framework doit s'adapter à toutes les applications du domaine visé par instanciation et/ou spécification des éléments de sa structure globale pour créer une application complète. En effet, la structure et le comportement fournis peuvent être réutilisés sans aucune modification ou les adapter en modifiant et ajoutant des éléments spécifiques à l'application considérée. Par conséquent, l'utilisation du *framework* consiste à faciliter la réalisation d'une application et à simplifier sa maintenance [Gamma et al., 1995]. En général, les *frameworks* sont classés en deux catégories :

- Les *frameworks* de domaine ou *frameworks* verticaux, qui capturent l'expertise d'un domaine particulier. Ils sont donc dépendants de ce domaine. Il existe plusieurs propositions de *frameworks* dédiés pour différents domaines d'application : les applications multimédia (*e.g.,* [Ha et al., 2013] [Lopez-Otero et al., 2014]), les bases de données temps réel [Idoudi et al., 2009], l'ingénierie de l'interaction Homme-Machine (*e.g.,* [Bhojan et al., 2014]), etc.
- Les *frameworks* d'application ou *frameworks* horizontaux, qui capitalisent les connaissances d'une large variété d'applications appartenant à différents domaines, tels que les *frameworks STRUTS* [Goodwill, 2003].

Un *framework* permet de réutiliser des éléments de conception et du code source car il permet de conserver l'architecture d'une famille d'applications et les modes de collaboration. L'inconvénient majeur d'un *framework* réside dans le fait qu'il dépend de l'environnement de développement dans lequel il est défini. En outre, un *framework* est constitué de plusieurs dizaines de classes, ce qui rend sa compréhension difficile.

2.2.3.4 Modèle de domaine

« Un modèle de domaine exprime des connaissances de haut niveau réutilisables dans le développement de tous les systèmes d'un même champ d'application » [Semmak, 1998]. Selon cette définition, on peut déduire qu'un modèle de domaine présente un espace de référence pour développer toutes les applications appartenant à ce domaine. Il existe plusieurs propositions de modèles de domaine, allant des modèles de domaine *orientés objets* aux abstractions de domaine issues de l'intelligence artificielle [Nellborn, 1999]. Un modèle de

domaine *orienté objet* sert à décrire les concepts, les liens entre ces concepts et les contraintes d'un domaine spécifique. Il permet également de décrire les similarités et les différences entre les connaissances des systèmes logiciels dans un domaine.

Les modèles de domaine sont applicables pendant les phases amont d'un processus de développement pour représenter les connaissances relatives à l'expression des problèmes d'un domaine particulier. Il présente aussi une base pour créer d'autres composants réutilisables, telles que les architectures à base de composants logiciels [John and Dörr, 2003]. L'utilisation des modèles de domaine est considérée importante dans le cas où la structure et le comportement sont soumis à de nombreuses modifications. Cette forme de composant réutilisable permet donc de diminuer le coût total et le temps nécessaire pour effectuer ces modifications.

2.2.3.5 Patrons

Le terme *patron* est la traduction du terme anglais « pattern » qui désigne un modèle de génie logiciel à objet et un modèle simplifié d'une structure. Le développement de logiciels par la réutilisation de patrons constitue une nouvelle approche pour capitaliser les expériences des experts et obtenir des logiciels de meilleure qualité. Il a été introduit par Beck et Cunningham [Beck and Cunningham, 1987] dans le domaine de l'informatique. Ils ont proposé une adaptation à la programmation orientée objet du langage de patrons, introduit par C. Alexander [Alexander, 1979] dans le domaine de l'architecture des bâtiments.

Un patron est défini par C. Alexander, dans le contexte de l'architecture et de l'urbanisation, ainsi *« Chaque patron décrit à la fois un problème qui se produit très fréquemment dans votre environnement et l'architecture de la solution à ce problème de telle façon que vous puissiez utiliser cette solution des millions de fois sans jamais l'adapter deux fois de la même manière »*. Il définit également le patron comme *« une règle en trois parties, exprimant une relation entre un contexte, un problème et une solution »*.

De manière générale, un patron est défini comme une solution aux problèmes récurrents dans un contexte [Schmidt et al., 1996] :

- Contexte : se réfère à un ensemble de situations récurrentes.
- Problème : définit un but à atteindre.
- Solution : permet de résoudre un problème en l'adaptant à un contexte particulier.

Les patrons permettent de bénéficier des avantages de la réutilisation pour toute les étapes d'ingénierie des systèmes d'information dès les phases d'analyse et de conception [Rieu et al., 1999b].

Les patrons jouent un rôle d'aide à l'analyse et à la conception, en offrant des solutions éprouvées aux problèmes de modélisation rencontrés. De plus, ils permettent de présenter les systèmes d'une manière simple, en masquant les détails non nécessaires. Ainsi, les patrons offrent un vocabulaire commun aux concepteurs dans le but d'assurer une meilleure compréhension.

Dans le paragraphe suivant, nous allons comparer les différents composants présentés dans ce paragraphe afin de cerner le composant le mieux adapté à l'ingénierie des systèmes d'aide à la conduite.

2.3 Patrons d'ingénierie

2.3.1 Comparaison entre les composants réutilisables

Plusieurs critères permettent de comparer les différents composants réutilisables (*e.g.,* [Cauvet et al., 2001]). Dans cette thèse, nous retenons les quatre critères présentés dans [Rieu, 1999] : couverture, portée, granularité et techniques d'implantation.

- *Couverture.* La couverture représente le degré de généralité des composants. Ces derniers peuvent être généraux, dans le cas où le problème traité est fréquent dans de nombreux domaines d'applications, ou spécifiques, si le problème traité est spécifique à un domaine particulier. Les *frameworks*, les composants *métier* et les composants logiciels peuvent encapsuler les connaissances relatives à un domaine particulier, comme ils peuvent être destinés à différents domaines. Les patrons peuvent être utilisés dans

n'importe quel domaine (patrons généraux [Gamma et al., 1995]), mais peuvent également être appliqués à un domaine particulier (patrons temps réel (*e.g.,* [Rekhis et al., 2010b] [Marouane et al., 2010]). Cependant, les modèles de domaine sont dédiés à un domaine spécifique [Barbier et al., 2004].

- *Portée.* La portée d'un composant réutilisable est évaluée en fonction de l'étape de l'ingénierie (analyse, conception, implémentation) à laquelle il s'adresse. Les *frameworks* peuvent être utilisés pendant les étapes d'analyse ou durant la conception architecturale des systèmes. Les patrons sont réutilisables lors des phases d'analyse, de conception et d'implantation. De même, les composants *métier* s'orientent vers ces phases en amont du développement des systèmes d'information [Barbier, 2002]. Quant aux composants logiciels, ils sont dédiés à la phase d'implantation.
- *Granularité.* Elle mesure le plus souvent le nombre d'entités (*e.g.,* classes et modules) constituant un composant. Elle permet d'identifier le degré de complexité de l'architecture que ce composant propose. Elle peut être faible (nombre d'entités inférieur à 10), moyenne (nombre d'entités inférieur à 100) ou forte (nombre d'entités supérieur à 100). Les patrons offrent des diagrammes constitués d'un nombre limité d'objets ou de classes vu qu'ils présentent des solutions à des problèmes restreints. Par exemple, les patrons d'analyse de Coad [Coad et al., 1996], ou ceux de conception proposés dans [Gamma et al., 1995], proposent des diagrammes de deux à six classes. De même, les modèles de domaine sont des structures de faible granularité comme les patrons. Quant aux composants *métier* et les composants logiciels, ils peuvent avoir une granularité allant de quelques classes à quelques dizaines. Au contraire, la majorité des *frameworks* offrent des architectures constituées d'une grande variété de classes, ce qui les rend trop rigides et très difficiles à réutiliser.

  Les patrons fournissent des unités légères et très modulaires. De ce point de vue, la réutilisation des patrons présente un avantage par rapport aux autres types de composants.
- *Techniques d'implantation.* Les techniques d'implantation déterminent si un composant dépend ou non d'un langage de programmation [Rieu, 1999]. À l'exception des patrons d'implantation dépendant de leurs cadres d'implantation, les patrons sont in-

dépendants d'un langage de programmation. De même, les modèles de domaine et les composants *métier* le sont aussi. Ainsi, la possibilité de réutiliser ces composants devient plus large. Par contre, les composants logiciels sont liés aux techniques d'implantation et/ou à une plate-forme logicielle [Rieu, 1999]. Les *frameworks* de nature conceptuelle sont indépendants des techniques d'implantation, alors que les *frameworks* logiciels sont liés à un langage de programmation.

Sur la base des critères de comparaison évoqués ci-dessus, nous constatons que la réutilisation des patrons est la plus adaptée à l'ingénierie des applications d'aide à la conduite temps réel. Elle peut être appliquée à toutes les étapes de développement d'un système, *i.e.,* depuis la phase d'expression des besoins jusqu'à la phase d'implantation. Les patrons sont de faible granularité ; ils sont généralement constitués d'un nombre réduit d'unités modulaires et légères. Ainsi, ils proposent une structure simple et facile, et pourraient donc être facilement réutilisés et appliqués sur un domaine d'application particulier. Par ailleurs, le concept de patron peut être considéré comme une notion générique que l'on peut appliquer pour représenter de manière uniforme une variété de composants (métier, architecturaux ou logiciels) [Barbier et al., 2004]. En outre, l'intégration d'un problème-type et d'une solution dans un même patron représente une aide à la recherche et à l'intégration. Cela permet également d'améliorer la fiabilité des logiciels. Enfin, les patrons permettent de :

- capitaliser un savoir-faire utilisé plusieurs fois pendant le développement logiciel, vu qu'ils constituent des composants réutilisables,
- assister le concepteur en respectant le domaine d'application, favorisant la documentation, etc.

De même, les patrons offrent un vocabulaire commun qui assure la communication de l'expertise au sein d'une équipe de développement. Ils fournissent les concepts d'un langage d'analyse ou de conception afin de faciliter la manipulation et la communication des spécifications et des modèles entre les concepteurs et les développeurs. Pour ces raisons, nous nous sommes intéressés plus particulièrement à la réutilisation à base de patrons. Les caractéristiques et les différents types de patrons sont détaillés dans le paragraphe suivant.

2.3.2 Différents types de patrons

De nombreux critères (*e.g.,* [Cauvet et al., 2001]) permettent de classifier les patrons proposés dans la littérature (*cf.* figure 2.2) : classification selon le type de connaissances (les patrons permettent de capitaliser des connaissances de type produit ou processus [2]), classification selon la couverture et classification selon la portée.

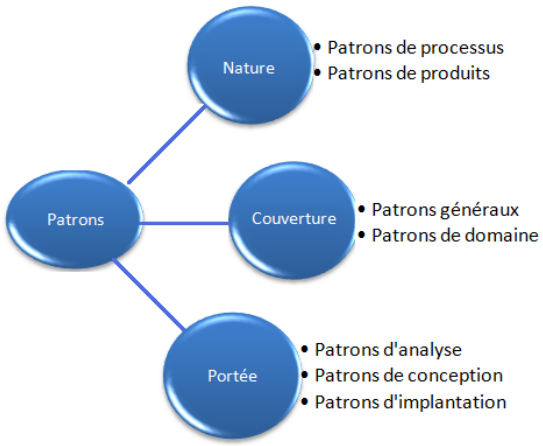


FIGURE 2.2 : Classification des patrons.

2.3.2.1 Classification selon le type de connaissances

Selon le type de connaissances rassemblées par les patrons, nous distinguons deux grandes classes de patrons : les patrons de processus (*e.g.,* [Kouroshfar et al., 2009] [Vuori et al., 2011]) et les patrons de produits (*e.g.,* [Rekhis et al., 2010b] [Marouane et al., 2010] [Sarika et al., 2013]). Les patrons de processus décrivent un savoir-faire sous forme d'actions et/ou de tâches qui permettent de guider le développement des systèmes. Ce type de patron est une démarche de développement qui permet de répondre à des exigences spécifiques. Par contre, les patrons de produits décrivent un savoir. Ils sont dédiés à être réutilisés et intégrés dans le système en cours de développement.

2. Un patron de type *produit* correspond à la description des résultats à atteindre. Un patron de type *processus* correspond plutôt à la description du chemin à suivre pour atteindre un résultat.

2.3.2.2 Classification selon la couverture

Les patrons sont classés en fonction de leur degré d'indépendance vis-à-vis du domaine. Les patrons généraux ou multi-domaines (*e.g.,* patrons de Gamma [Gamma et al., 1995]) couvrent plusieurs domaines d'applications et offrent des structures applicables à n'importe quel domaine, alors que les patrons de domaine proposent des solutions spécifiques à un domaine particulier. Bien que les patrons de domaine se caractérisent par un niveau de spécificité bien défini, il est plus facile de les adapter aux exigences du domaine en cours de développement, comparativement aux patrons généraux [Port, 1998]. Mais, il est important que les patrons de domaine soient les plus généraux possibles pour ce domaine afin d'être réutilisables quelle que soit l'application de ce domaine. Dans ce cadre, de nombreux travaux se sont intéressés aux patrons de domaine. Par exemple, il y a les patrons destinés aux systèmes de gestion de Workflow (*e.g.,* [Götz, 2009] [Dashtban, 2012]) et les patrons dédiés au domaine temps réel (*e.g.,* [Marouane et al., 2010] [Pekka and Jari, 2013] [Sarika et al., 2013]), etc.

2.3.2.3 Classification selon la portée

Comme nous l'avons mentionné dans le paragraphe 2.3.1, les patrons peuvent être notamment classés, selon leur utilisation lors des étapes de l'ingénierie, en patrons d'analyse, patrons de conception et patrons d'implantation.

Patrons d'analyse

Les patrons d'analyse sont utilisés lors de la phase d'analyse des besoins d'une application. Ils aident les concepteurs à construire des modèles qui représentent mieux les exigences fonctionnelles et non fonctionnelles des applications. Ils permettent d'identifier les problèmes récurrents dans l'expression de ces exigences et de proposer des solutions générales et réutilisables pour résoudre les problèmes identifiés [Fowler, 1997]. En effet, ces patrons permettent de représenter ce que le système doit faire. Dans ce contexte, il existe de nombreuses propo-

sitions de patrons d'analyse *e.g.,* les patrons proposés dans [Fernandez and Yuan, 2010] et les patrons définis dans [Marouane et al., 2010].

Patrons de conception

Les patrons de conception (design patterns), introduits par Gamma, sont les plus connus et utilisés parmi les patrons d'ingénierie [Gamma et al., 1995]. Ils capturent la connaissance liée à la conception en identifiant les objets intervenants, leurs rôles et leurs collaborations. Cependant, ils ne permettent pas d'exprimer les comportements fonctionnels des systèmes. Ils proposent une solution efficace et réutilisable à des problèmes de conception. En effet, les patrons de conception permettent d'exprimer la manière dont le système doit répondre à ses exigences. Ces patrons sont indépendants des langages de programmation, ce qui accroît leur réutilisation. Par ailleurs, ils peuvent être représentés par un formalisme contenant une rubrique de fragments de code qui montre la façon de les implémenter à la manière du formalisme de représentation des patrons de conception proposé par [Gamma et al., 1995].

Patrons d'implantation

Les patrons d'implantation ou idiomes sont des patrons de bas niveau. Généralement, ils sont spécifiques à un langage de programmation. Ils décrivent la façon d'implémenter des astuces pour ce langage. Par exemple, ils permettent d'implémenter des mécanismes absents dans des langages, comme la simulation de l'héritage multiple en Java [Rieu et al., 1999a]. L'application des idiomes permet de réduire les erreurs ainsi que le temps de développement. Dans la littérature, les idiomes les plus connus sont ceux du langage C++ [Coplien, 1992] et du langage Java [Langr, 1999].

La complexité croissante des systèmes d'aide à la conduite et leur évolution de plus en plus rapide rendent leur conception plus difficile, plus coûteuse et moins fiable. En effet, il est difficile de spécifier et de concevoir les différents composants et les contraintes temporelles relatives à la validité des données et aux échéances des traitements. En d'autres termes, le concepteur doit surmonter différents problèmes tels que la gestion d'un grand volume de données et de traitements, le listage des bons objets intervenant dans le système, l'identification de leurs propriétés et de leurs collaborations, etc. En outre, les systèmes d'aide à la conduite

sont mis en œuvre sur différentes plate-formes, mais les méthodes de conception sont absentes. Ainsi, les erreurs de conception peuvent provoquer une défaillance du système qui peut mener à des conséquences graves. Les travaux de recherche menés dans le domaine de l'aide à la conduite ne se sont intéressés jusqu'à maintenant qu'à certaines problématiques, telles que la gestion de la fiabilité (*e.g.,* [Gietelink, 2007]) et la fusion des données (*e.g.,* [Ghahroudi and Sabzevari, 2009] [Darms et al., 2010]), et n'exploitent pas les principes de réutilisation pour traiter les problèmes de conception de ces systèmes. Pour ces raisons, la réutilisation des patrons de conception nous semble la plus adaptée pour aider à obtenir une bonne conception et réduire le risque d'erreurs propagées dans le code.

Dans la suite du chapitre, nous allons décrire les travaux existant qui portent sur la représentation et l'instanciation des patrons de conception et présenter quelques travaux proposant des patrons de conception destinés au domaine temps réel.

2.4 Patrons de conception

Dans cette section, nous présentons des formalismes de description des patrons. Nous décrivons également quelques extensions nécessaires à la représentation des patrons.

2.4.1 Formalismes de description des patrons

Généralement, un patron est défini comme une solution à un problème récurrent dans un contexte donné. Pour représenter le triplet {problème, solution, contexte}, plusieurs formalismes ont été proposés dans la littérature. Nous entendons par formalisme la forme, la structure et la syntaxe utilisées pour représenter les patrons. On distingue deux catégories de formalismes : les formalismes narratifs et les formalismes structurés. Contrairement aux formalismes narratifs (*e.g.,* [Bass et al., 2004]) qui sont peu structurés et informels, les formalismes structurés (*e.g.,* [Fowler, 1997]) permettent une meilleure représentation des patrons. Ces formalismes structurés, constitués chacun d'un ensemble de rubriques qui lui sont

propres, sont quasiment équivalents. Cependant, ils différent par le nombre de rubriques, le degré de détails de celles-ci et le type de patrons qu'ils représentent. Le formalisme de M. Fowler ([Fowler, 1997]) est destiné, par exemple, à la représentation des patrons d'analyse. Le formalisme de E. Gamma [Gamma et al., 1995] est dédié à la description de patrons de conception. Nous présentons ce formalisme par les rubriques décrites dans la tableau 2.1.

Tableau 2.1 : Formalisme de E. Gamma [Gamma et al., 1995].

Rubrique	**Signification**
Nom du patron	Identifie le patron par un nom significatif ou une phrase.
Classification	Permet d'organiser les patrons en fonction de deux critères : le rôle, qui désigne ce que fait le patron (création, structuration, comportement), et le domaine, qui indique si le patron s'applique à une classe ou à un objet.
Intention	Indique le problème de conception qui concerne le patron ou son but.
Alias	Énumère d'autres noms connus du patron, s'il en existe.
Motivation	Illustre un scénario d'application du patron pour résoudre un problème de conception. Elle permet de faciliter la compréhension de la solution du patron.
Indication d'utilisation	Décrit des situations où on peut appliquer le patron.

Structure	Représente la structure du patron en utilisant des diagrammes de classes. Elle décrit aussi les interactions entre les classes en utilisant les diagrammes de collaboration.
Participants	Définissent les classes intervenant dans le patron et leurs responsabilités.
Collaborations	Décrivent la manière dont les constituants du patron interagissent pour assurer leurs responsabilités.
Conséquences	Présentent comment le patron atteint ses objectifs et décrivent les compromis et les impacts de son utilisation.
Implémentation	Présente la technique d'implémentation du patron.
Exemple de code	Donne des fragments de code illustrant la méthode utilisée pour implémenter le patron dans un langage objet (*e.g.,* langage C++).
Utilisations connues	Donnent des exemples d'application du patron dans des systèmes réels.
Patrons apparentés	Énumère les patrons reliés avec le patron considéré et leurs différences.

Pour augmenter le taux de réutilisation, un formalisme idéal doit employer les principes d'abstraction et de variabilité. Il est à noter que la majorité des formalismes existants possèdent des limites qui réduisent la réutilisation. En effet, les rubriques dédiées à la sélection et l'adaptation des patrons sont décrites, le plus souvent, sous forme de textes non structurés, qui sont mal adaptées à des techniques automatisées d'utilisation de patrons. Cependant, seules les rubriques représentant les réalisations des patrons sont formelles ou sous forme de modèles semi-structurés tels que les diagrammes UML. Enfin, la plupart des relations

déterminant les ensembles de patrons interdépendants ne sont pas suffisamment claires pour favoriser la réutilisation. D'où l'apparition de nombreuses recherches qui s'intéressent à :

- la formalisation des rubriques de sélection dont la finalité est de faciliter la recherche de patrons satisfaisant les exigences du concepteur [Berrut and Front-Conte, 2001],
- l'uniformisation des formalismes de représentation de patrons produits et processus, de façon à les combiner au sein d'un même système de patrons [Gzara et al., 2000],
- la définition de relations inter-patrons pour améliorer la structuration des systèmes de patrons (ou catalogues de patrons) [Eden, 2000].

Citons, par exemple, le formalisme P-SIGMA [Conte et al., 2001] qui permet une uniformisation des formalismes existants dans la littérature. Ce formalisme décrit chaque patron en trois parties : *Interface*, *Réalisation* et *Relation*. Chaque partie contient un ensemble de rubriques décrites sous forme textuelle ou formelle.

La partie *Interface* contient des rubriques permettant de faciliter la sélection des patrons de conception. En effet, cette partie contient cinq rubriques :

1. *Identifiant* : précise le nom du patron.
2. *Classification* : définit l'intention du patron.
3. *Problème* : détaille le problème à résoudre par le patron.
4. *Contexte* : cite les situations d'utilisation du patron.
5. *Forces* : définit les motivations d'application du patron.

La partie *Réalisation* est composée de rubriques spécifiant la solution en termes de modèles et/ou de démarches afin de résoudre le problème soulevé par le patron. Ces rubriques sont définies comme suit :

1. *Solution* : décrit la solution du problème, généralement à l'aide de modèles semi-structurés (*e.g.,* diagrammes de classes).
2. *Cas d'application* : indique des exemples d'instanciation de la solution. Cette rubrique est optionnelle mais fortement conseillée pour faciliter la compréhension de la solution.
3. *Conséquences* : discute les avantages et les inconvénients de l'application de patrons.

Enfin, la partie *Relation* permet d'expliciter les différents types de relations entre patrons. En effet, les relations, définies dans ce qui suit, permettent de positionner le patron considéré par rapport à d'autres patrons existants.

- *Utilise :* le patron P1 utilise le patron P2 si une partie des problèmes soulevés par P1 peuvent être soulevés par P2.
- *Raffine :* le patron P1 raffine le patron P2 si le problème soulevé par P1 constitue une spécialisation de celui soulevé par P2. Le patron P2 peut donc résoudre le problème posé par P1.
- *Requiert :* le patron P1 requiert un patron P2, donc l'application de P2 doit être un pré-requis à l'application de P1 et P2 doit apparaître dans le contexte de P1.
- *Alternative :* le patron P1 est une alternative au patron P2 si P1 traite le même problème et s'adresse au même contexte que P2, mais définit une solution différente.

Il convient de noter que le formalisme P-Sigma nous semble un moyen efficace de description de patrons, vu qu'il offre une meilleure structuration des patrons, et facilite leur sélection grâce aux rubriques définies dans la partie *Interface*. Le formalisme P-Sigma permet également de faciliter l'identification de l'ensemble des patrons liés au patron cible par une relation de type définie dans la partie *Relation*. C'est pourquoi nous allons adopter ce formalisme pour décrire les patrons de conception que nous proposons pour la modélisation du domaine d'aide à la conduite.

2.4.2 Profils UML pour les patrons de conception

Les patrons de conception ont été décrits, soit dans le langage naturel, qui est trop imprécis et ambigu, soit par des langages formels (*e.g.,* ceux proposés dans [Bottoni et al., 2009] et [Bayley and Zhu, 2012]), qui sont complexes et difficiles à comprendre par les concepteurs qui n'ont pas de connaissances approfondies en mathématiques. Les méthodes basées sur le langage naturel et sur les langages formels rendent la réutilisation des patrons plus complexe. Pour remédier à ces difficultés, des notations visuelles et graphiques basées sur UML, tel que DPML [3] [Mapelsden et al., 2002], ont été proposées pour bien visualiser, spécifier et

3. Design Pattern Modeling Language

documenter les *artefacts* des systèmes. En effet, UML offre divers diagrammes qui présentent plusieurs perspectives ou vues de l'architecture du système, à différents niveaux d'abstraction. Par ailleurs, UML présente l'avantage d'être un standard pour la modélisation orientée objet. Cependant, ces notations n'offrent pas une sémantique précise aux différents concepts (*e.g.,* classes et attributs). C'est ainsi qu'UML fournit des mécanismes d'extensions qui permettent d'offrir plus de sémantique aux différents concepts des modèles à travers la définition de stéréotypes, de contraintes et de valeurs marquées : ce sont les profils UML.

Dans la suite, nous détaillons les travaux les plus récents qui proposent des profils UML pour les patrons de conception. Nous distinguons trois catégories de profils. La première catégorie vise à représenter des modèles spécifiant la solution de patrons (*e.g.,* celui défini dans [Arnaud, 2008]). La deuxième catégorie consiste à représenter des instances de patrons (*e.g.,* ceux définis dans [Dong et al., 2007] et [Loo et al., 2012]. Quant à la troisième catégorie, elle permet de représenter, à la fois, la spécification des patrons et leurs instances (*e.g.,* ceux définis dans [Reinhartz-Berger and Sturm, 2009] et [Rekhis et al., 2013a]). Ces profils sont évalués en se basant sur les critères décrits dans les tableaux 2.2 et 2.3.

2.4.2.1 Profils UML pour la spécification des patrons

Profil de Arnaud.

Arnaud [Arnaud, 2008] a proposé des extensions d'UML, décrits ci-dessous, pour une représentation explicite de la variabilité pendant la modélisation de patrons :

— Les stéréotypes << *variation* >> et << *variant* >>, qui sont associés aux cas d'utilisation, définissent respectivement un cas d'utilisation comme point de variation et variante. Ces deux stéréotypes sont également associés aux interactions dans le diagramme de séquence. Le type de variation entre les variantes est exprimé par le stéréotype << *alternative* >>.

— Les stéréotypes << *obligatoire* >> et << $facultative$ >>, qui sont associés aux cas d'utilisation, expriment respectivement les fonctionnalités obligatoires (*i.e.,* les

Tableau 2.2 : Critères de spécification de patrons de conception.

	Critères	**Signification**
Spécification de patrons	Variabilité	La variabilité est définie comme « *la capacité d'un système à être changé, personnalisé et configuré en fonction d'un contexte spécifique* » [Gurp et al., 2001]. Le concept de "Variabilité" consiste à représenter les éléments (*e.g.,* classes et attributs) qui varient selon le contexte d'une application particulière. Ces éléments variables sont représentés dans des points de variation qui constituent « *des endroits du système où il y a une variation* » [Czarnecki and Eisenecker, 2000], à réaliser par des variantes.
	Consistance	Un patron de conception est décrit souvent par plusieurs vues complémentaires : la vue statique, modélisée par exemple par le diagramme de classes, et la vue dynamique, modélisée par exemple par le diagramme de séquence. Ces vues sont généralement dépendantes. Par exemple, à une classe fondamentale correspond un objet du diagramme de séquence, qui est aussi un élément fondamental. Ainsi, le fait d'assurer la cohérence entre les différentes vues conduit à des patrons corrects.
	Expressivité	Pour avoir des patrons de conception bien documentés et compréhensibles, il est nécessaire de les représenter par des notations visuelles et expressives, tel que le langage UML, d'une part, et de différencier les notations de modélisation de celles utilisées pour l'instanciation, d'autre part.

fonctionnalités communes aux applications étudiées) et les fonctionnalités optionnelles (*i.e.,* les fonctionnalités qui varient d'une application à une autre).

— Le stéréotype << *fragment statique* >> indique la vue statique associée à un cas d'utilisation. Le concepteur peut associer aux fragments, des notes << *A Faire* >> associées à n'importe quel élément imitable dont le contenu peut être des textes ou des algorithmes. Ces notes consistent à exprimer des propriétés génériques (implémentation d'opérations, adjonction d'attributs, etc.) pour attirer l'attention du concepteur

Tableau 2.3 : Critères de réutilisation de patrons de conception.

	Critères	**Signification**
Réutilisation de patrons	Traçabilité	Ce critère vise à permettre d'identifier facilement les éléments de chaque patron de conception lorsque de nombreux patrons sont instanciés et composés pour modéliser une application particulière. Il s'agit également de distinguer entre les éléments jouant le rôle d'éléments de patron de ceux ajoutés par le concepteur selon le contexte de l'application cible.
	Composition	Une instance d'un patron de conception peut être intégrée dans une application spécifique constituée d'autres instances de patrons. En effet, cette intégration consiste à mettre en relation les éléments de cette instance avec des éléments composant les autres instances pour construire le modèle complet de l'application considérée. La représentation explicite de la composition de patrons a pour objectif d'identifier les éléments communs aux instances de patrons composés.

d'applications sur des aspects qui sont extrêmement dépendants du contexte d'imitation pour figurer plus formellement dans la solution.

Les extensions proposées permettent d'exprimer la variabilité dans les vues fonctionnelles, dynamiques et statiques. Le diagramme de cas d'utilisation constitue la base du processus d'instanciation ; le concepteur de l'application sélectionne une variante d'une fonctionnalité pour commencer l'imitation de la solution du patron et l'adapter à son contexte. Cependant, le diagramme de cas d'utilisation est trop abstrait et ne peut pas être utilisé en tant que modèle d'entrée pour l'instanciation de patron. En effet, il se situe à un niveau élevé d'abstraction et donc le concepteur ne peut pas identifier, par exemple, les méthodes ou les attributs relatifs à ses exigences.

Le profil d'Arnaud n'est pas assez expressif pour deux raisons. Premièrement, la vue statique est représentée par des paquetages très élémentaires comprenant une ou deux classes, qui réduit la compréhension de la solution de patron et rend sa composition plus difficile.

Deuxièmement, Arnaud, bien qu'il ait proposé un processus d'instanciation de patrons de conception, n'a pas défini des notations permettant la visualisation des éléments du patron dans son instance. Ainsi, le profil qu'il a défini ne permet pas de garder trace des éléments du patron lors de son instanciation. Par conséquent, ce profil ne vérifie pas les deux critères d'instanciation des patrons : la traçabilité et la composition.

2.4.2.2 Profils UML pour la réutilisation des patrons

Profil de Dong *et al.*

Le profil proposé par Dong *et al.* [Dong et al., 2007] ne propose que l'adjonction de notations au diagramme de classes pour distinguer les éléments instanciés à partir d'un patron de ceux ajoutés par le concepteur pendant la modélisation d'une application spécifique. Ce profil inclut trois stéréotypes $<< PatternClass >>$, $<< PatternAttribute >>$ et $<< PatternOperation >>$ qui identifient respectivement les classes, les attributs et les opérations appartenant à chaque instance de patrons. Chaque stéréotype dispose d'une valeur marquée dont le nom possède le format : **role@name[instance]**. Le champ **role** indique le rôle joué par un élément appartenant au modèle de l'application dans le patron. Le champ **name** indique le nom du patron à partir duquel un élément est instancié. Le champ **instance** indique le nombre d'instances d'un patron au sein d'une application. Ce champ peut ne pas figurer si le patron n'est instancié qu'une seule fois.

Le profil défini facilite la réutilisation de la vue statique des patrons de conception, mais il ne permet pas d'exprimer la variabilité lors de la spécification des patrons et ne s'intéresse pas à l'instanciation de la vue dynamique. Il propose des extensions d'UML pour représenter explicitement des éléments instanciés à partir des patrons. Il permet d'indiquer le rôle joué par une classe dans chaque patron vu que cette classe peut constituer, en même temps, l'instance de nombreuses classes appartenant à différents patrons. Ce profil propose également des valeurs marquées qui ne donnent des informations que sur le nom du patron, le rôle joué par l'élément instancié et le nombre d'instances du patron. Ainsi, il assure le critère de traçabilité et facilite la composition des patrons. Cependant, les stéréotypes $<< PatternAttribute >>$

et << *PatternOperation* >> peuvent être source de confusion lorsque deux rôles d'attributs ou d'opérations définis dans des rôles de classes différents portent le même nom. Cette ambiguïté peut être source de confusion pour les outils lors de la validation des instances de patrons. De plus, le profil proposé n'est pas assez expressif vu qu'il définit seulement des extensions pour l'instanciation des patrons et non pas pour leur modélisation.

Profil de Loo *et al.*

Loo *et al.* ont proposé dans [Loo et al., 2012] un profil qui facilite la spécialisation et l'instanciation des diagrammes de séquence. Ils définissent quatre stéréotypes décrits ci-dessous :

- Le stéréotype << *patternRole* >> indique les rôles joués par les objets et les messages instanciés à partir d'un patron.
- Le stéréotype << *patternInteractionFragment* >> indique les fragments combinés composés d'interactions qui sont instanciées à partir d'un patron.
- Le stéréotype << *patternEngage* >> précise les fragments combinés composés de nouvelles interactions ajoutées à un patron.
- Le stéréotype << *patternDisengage* >> précise les fragments combinés composés d'interactions supprimées d'un patron.

Le profil proposé dans [Loo et al., 2012] définit la propriété *fragmentType* du stéréotype << *patternInteractionFragment* >> dans le but de montrer s'il s'agit d'un seul fragment combiné ou d'une variante d'un fragment parmi d'autres versions. Néanmoins, la variabilité est exprimée pendant la modélisation d'un patron et non pas pendant son instanciation. En outre, cette variabilité peut être exprimée par le fragment combiné *« alt »* (alternative) du standard UML 2.0 [OMG, 2005b], auquel cas, il est inutile de définir des notations. Aussi, ce profil n'exprime que partiellement la variabilité, car il ne propose pas d'extensions pour représenter des points de variation dans la vue statique. De plus, le profil ne définit pas de notations pour exprimer les interactions obligatoires et les interactions facultatives lors de la spécification d'un patron. Bien que ce profil précise les interactions supprimées d'une instance d'un patron par le stéréotype << *patternDisengage* >>, il ne vérifie pas suffisamment le critère d'expressivité.

2.4.2.3 Profil UML pour la spécification et la réutilisation de patrons

Profil de Berger *et al.*

Berger *et al.* [Reinhartz-Berger and Sturm, 2009] ont proposé une approche de modélisation de domaine basée sur l'application d'ADOM [4]-UML. Ce profil facilite la spécification de la vue statique et de la vue dynamique de patrons spécifiques à un domaine et leur instanciation. Il présente quatre stéréotypes permettant d'exprimer les points communs entre les applications d'un domaine et les différences entre elles :

- Le stéréotype << *optional single* >> indique qu'un élément (*e.g.,* classe, attribut, opération, objet, fragment combiné et message) peut apparaître au plus une fois dans l'instance du patron.
- Le stéréotype << *optional many* >> indique qu'un élément peut être instancié plusieurs fois ou omis lors d'une instanciation d'un patron.
- Le stéréotype << *mandatory single* >> indique qu'un élément doit être obligatoirement instancié une et une seule fois à chaque instanciation d'un patron.
- Le stéréotype << *mandatory many* >> indique qu'un élément doit être obligatoirement instancié au moins une fois lors d'une instanciation d'un patron.

Chaque stéréotype possède deux étiquettes *« min »* et *« max »* qui définissent respectivement le nombre minimal et le nombre maximal d'instances d'un élément du patron dans un modèle d'application. Ces stéréotypes, avec leurs étiquettes, permettent d'exprimer la variabilité des patrons.

De plus, le profil ADOM-UML permet une représentation explicite des éléments instanciés à partir de patrons. En effet, à chaque élément instancié à partir d'un patron lors de la modélisation d'une application spécifique du domaine, un stéréotype est créé avec le nom du rôle joué par l'élément dans le patron. Par exemple, dans l'application de contrôle de processus (figure 2.3), la classe *« WaterController »* joue le même rôle que la classe *« Controller »* du patron, donc elle est stéréotypée << *Controller* >>. Ce profil améliore la lisibilité des modèles d'applications vu que les stéréotypes proposés permettent d'identifier facilement les éléments instanciés à partir d'un patron. Cependant, l'application de ces stéréotypes entraine

4. Application-based DOmain Modeling

une ambiguïté lorsqu'un élément du modèle joue des rôles d'éléments appartenant à différents patrons et portant les mêmes noms.

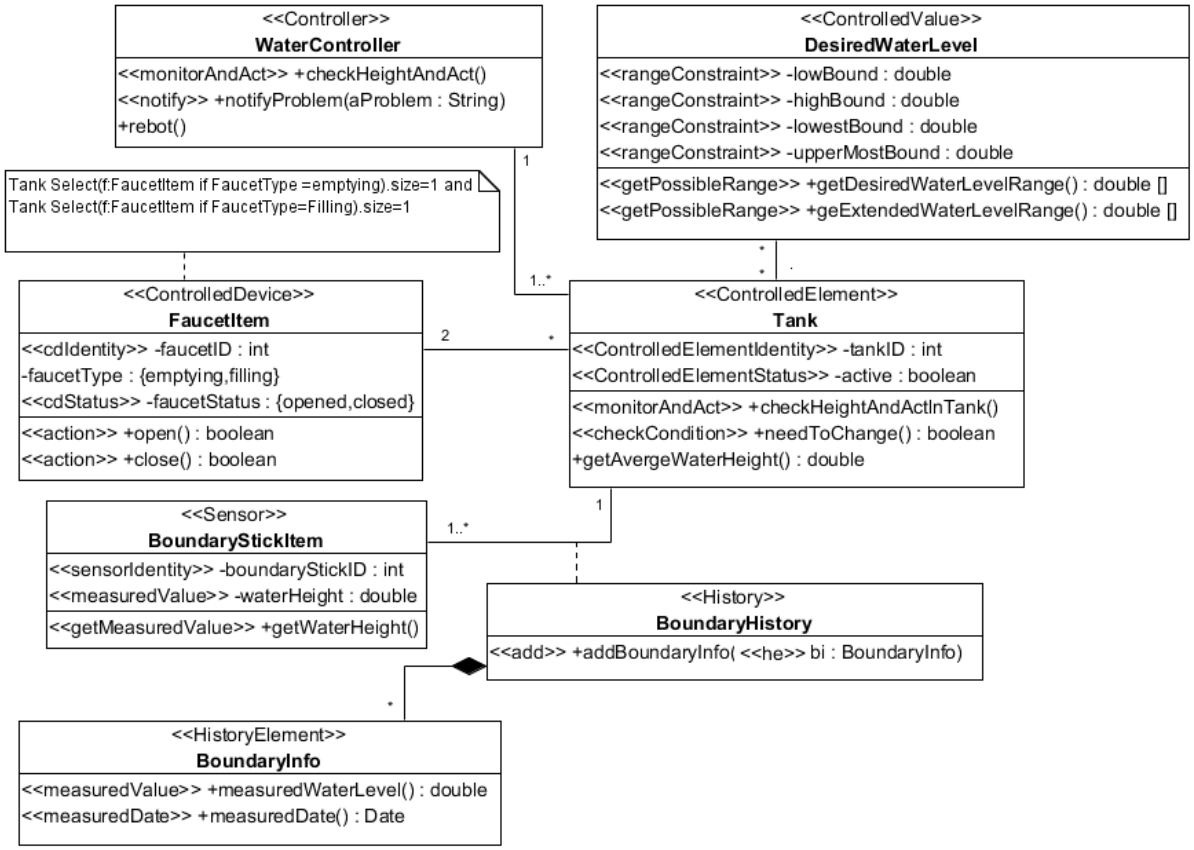


FIGURE 2.3 : Exemple de diagramme de classes d'une instance du patron de conception pour le domaine de contrôle de processus [Reinhartz-Berger and Sturm, 2009].

Profil de Rekhis *et al.*

Les auteurs de [Rekhis et al., 2013a] ont proposé un profil UML appelé UML-RTDP [5]. Il consiste à présenter des notations permettant d'exprimer explicitement la modélisation et la réutilisation des patrons de conception dédiés au domaine temps réel.

Pour représenter la spécification des patrons, le profil UML-RTDP inclut les stéréotypes << *mandatory* >> et << *optional* >> qui expriment respectivement les éléments (*e.g.,* classes, objets et interactions) communs entre les applications temps réel et les éléments optionnels qui peuvent être omis lors de l'instanciation des patrons.

5. UML profile for RT Design Patterns

Contrairement aux stéréotypes << *mandatory single* >>, << *mandatory many* >>, << *optional single* >> et << *optional many* >> [Reinhartz-Berger and Sturm, 2009], les stéréotypes << *mandatory* >> et << *optional* >> ignorent le détail concernant le nombre d'instances d'un élément vu que les applications temps réel couvrent un large spectre d'activités. Par conséquent, la détermination du nombre d'instances d'un élément dans une application particulière est une tâche difficile. De plus, le profil UML-RTDP présente les stéréotypes suivants :

— le stéréotype << *extensible* >>, qui indique qu'une classe d'une instance d'un patron peut être étendue par l'adjonction d'attributs et/ou d'opérations. Ce stéréotype est appliqué aux méta-classes suivantes : *Class*, *Interface* et *ClassAssociation* ;

— Le stéréotype << *variable* >>, qui indique que l'implémentation des méthodes varie selon l'instanciation d'un patron.

En plus des stéréotypes proposés pour la modélisation des patrons de conception, UML-RTDP présente des stéréotypes permettant une représentation explicite de l'instanciation de la vue statique et de la vue dynamique des patrons. Ces stéréotypes sont décrits ci-dessous :

— Le stéréotype << $PatternClass$ >> indique qu'une classe est instanciée à partir d'un diagramme de classes d'un patron.

— Le stéréotype << $PatternLifeline$ >> indique qu'un objet est instancié à partir d'un diagramme de séquence d'un patron.

— Le stéréotype << $PatternInteraction$ >> indique qu'une interaction est instanciée à partir d'un diagramme de séquence d'un patron.

À chaque stéréotype sont associées deux propriétés. *PatternName*, qui précise le nom du patron à partir duquel un élément est instancié, et *ParticipantRole*, qui spécifie le nom du rôle joué par l'élément dans le patron. Ainsi, ces deux propriétés permettent d'enlever toute ambiguïté pouvant se produire lorsque un élément joue simultanément des rôles différents dans des patrons différents.

Le profil UML-RTDP présente l'avantage d'obtenir des patrons de conception flexibles et bien documentés. En effet, ce profil contient des stéréotypes permettant au concepteur de savoir quels sont les éléments qui doivent être présents dans une instance du patron et quels

sont les éléments à supprimer dans cette instance, et assurent donc le critère de traçabilité. Ce profil définit également des stéréotypes expressifs du fait qu'il est facile de différencier les notations utilisées durant la modélisation d'un patron de celles utilisées durant sa réutilisation. Cependant, l'usage du profil UML-RTDP reste limité car il définit des notations permettant d'identifier seulement les classes, les objets et les interactions jouant un rôle dans le patron, mais ne précise pas les autres éléments tels que les attributs, les opérations et les messages. Il ne permet pas, par exemple, de différencier les attributs instanciés à partir d'un patron de ceux ajoutés par le concepteur selon le contexte de l'application à modéliser.

2.4.3 Synthèse

Les travaux présentés dans ce chapitre se sont focalisés sur la proposition des extensions dédiées, soit à la modélisation des patrons, soit à leur réutilisation, soit aux deux. Les extensions pour la représentation de la modélisation des patrons ne sont pas assez expressives vu qu'elles ne différencient pas clairement les extensions utilisées lors de la modélisation des patrons de celles utilisées lors de leur instanciation, comme c'est le cas des extensions destinées à la réutilisation des patrons. Par contre, les extensions destinées à la fois à la modélisation et à la réutilisation sont expressives du fait qu'elles permettent de distinguer clairement les extensions appliquées pendant la modélisation des patrons de celles utilisées pendant leur instanciation. En outre, ces extensions facilitent la composition des patrons puisqu'elle précisent le rôle joué par un élément dans chaque patron si cet élément représente simultanément l'instance de plusieurs éléments de différents patrons. Cependant, ces extensions n'identifient pas tous les éléments qui jouent un rôle dans les patrons. Ceci engendre des difficultés pour vérifier si des éléments appartenant aux modèles d'applications sont instanciés à partir des patrons ou non. Ainsi, le critère de la traçabilité de l'instanciation des patrons n'est exprimé que partiellement.

Nous considérons qu'il est nécessaire de définir un profil UML pour la modélisation des patrons de conception dédiés au domaine d'aide à la conduite et leur instanciation en tenant

compte des critères présentés ci-dessus et des spécificités temps réel. Ceci permet d'obtenir des patrons flexibles, bien documentés et compréhensibles.

2.5 Patrons de conception temps réel

La conception et la réalisation des Systèmes Temps Réel (STR) sont complexes et coûteuses. En effet, les concepteurs de ces systèmes doivent prendre en considération les aspects fonctionnels ainsi que les contraintes temps réel. Pour réduire cette complexité et améliorer la qualité des STR développés, de nouveaux travaux, basés sur la définition des patrons de conception Temps Réel (TR), ont été entamés. Nous présentons, dans cette section, les travaux les plus récents qui définissent des patrons de conception destinés au domaine TR.

2.5.1 Patrons de conception définis par Rekhis *et al.*

Rekhis *et al.* ont proposé deux patrons de conception destinés aux STR. Le premier concerne la fonctionnalité d'acquisition de données à partir de l'environnement [Rekhis et al., 2010b]. Ce patron concerne la modélisation des données gérées par les STR et l'expression des contraintes temporelles liées aux données et aux traitements. Le deuxième correspond à la fonctionnalité de traitement et de contrôle des données [Rekhis et al., 2010c]. Ce patron concerne la modélisation des traitements et le contrôle des données TR ainsi que les contraintes temporelles.

Chaque patron est représenté par (i) un diagramme de classes pour modéliser l'aspect structurel des STR (*i.e.,* les classes, leurs attributs et leurs relations) et (ii) un diagramme de séquence pour décrire les invocations des opérations entre les objets identifiés. Ces patrons comprennent les éléments communs entre divers STR et les éléments qui varient d'un système à un autre. La variabilité est donc vérifiée dans ces patrons. De plus, ces deux patrons décrivent les exigences non fonctionnelles qui permettent d'améliorer la qualité des systèmes développés.

Cependant, ces patrons concernent le domaine TR en général et sont décrits à un niveau d'abstraction élevé. D'une part, le concepteur doit ajouter des éléments (*e.g.,* des entités, leurs attributs et leur relations) qui ne figurent pas dans les modèles de patrons, selon le contexte de l'application à modéliser. Ainsi, l'instanciation des patrons est complexe, ce qui engendre un système qui ne répond pas exactement à tous ses exigences. En fait, ces patrons ne représentent pas clairement certains concepts des applications TR, à savoir les données sensorielles et les données dérivées (*i.e.,* ces données sont représentées par une seule classe et une relation réflexive). Cette représentation n'est pas expressive pour certains systèmes tels que les systèmes d'aide à la conduite, vu que les données sensorielles et les données dérivées n'ont pas les mêmes relations et sont traitées d'une manière différentes. Par exemple, dans les systèmes d'aide à la conduite, les données sensorielles, mises à jour périodiquement, sont traitées par une unité de fusion, et par la suite, elles sont utilisées par le contrôleur pour calculer sporadiquement les données dérivées nécessaires pour analyser la situation. Donc, il serait utile de modéliser une classe pour les données sensorielles et une autre pour les données dérivées afin de représenter explicitement leurs relations.

De plus, les patrons proposés ne représentent pas clairement les contraintes temps réel (*e.g.,* les délais des tâches) malgré leur importance ; si une contrainte n'est pas prise en compte (*e.g.,* échéance d'une tâche), elle peut entrainer la défaillance du système. Il est donc nécessaire de prendre en compte ces contraintes dans les patrons.

2.5.2 Patrons de conception définis par Alho *et al.*

Alho *et al.* ont proposé des patrons TR qui présentent des solutions pour la tolérance aux fautes. Ces patrons visent à améliorer la fiabilité des systèmes de contrôle temps réel distribués [Pekka and Jari, 2013]. En effet, le premier patron a pour objectif de réduire le niveau de couplage entre les modules (*e.g.,* services, composants ou processus) en utilisant un *middleware orienté données*. Le deuxième patron a pour but de mettre en place un gestionnaire de services qui peut démarrer, surveiller et arrêter les modules d'un système pour assurer un fonctionnement correct et une exécution en temps réel. Quant au troisième patron, il a pour

objectif de gérer les erreurs en évitant toute manipulation complexe (*e.g.,* éviter d'écrire un code trop long).

Les patrons proposés modélisent les exigences non fonctionnelles, qui permettent d'améliorer la qualité des systèmes développés, mais ne s'intéressent pas aux aspects fonctionnels qui sont indispensables pour le développement des systèmes. De plus, les auteurs de [Pekka and Jari, 2013] ont décrit les solutions de ces patrons sous forme textuelle, ce qui rend leur compréhension et leur réutilisation plus difficiles, car ambigües.

2.5.3 Patrons de conception définis par Sarika *et al.*

Les auteurs de [Sarika et al., 2013] ont proposé un patron de conception temps réel dédié aux systèmes distribués basés sur les architectures orientées services (SOA : Service Oriented Architecture). Ce patron a pour objectif de réduire la complexité des requêtes composées des services WEB dans les systèmes distribués d'une manière efficace. En effet, il décrit comment gérer la composition des services WEB, l'invocation dynamique des services et l'adjonction de nouveaux services dans les architectures SOA distribuées, à l'aide des modèles de programmation *MapReduce*. Ce dernier permet de décomposer les requêtes en des sous-requêtes simples et d'envoyer chaque sous-requête à un serveur pour la traiter.

Le patron défini dans [Sarika et al., 2013] présente une solution optimale qui aide à développer des architectures SOA dynamiques et évolutives dans les STR distribués. Il concerne le diagramme de classes afin de modéliser l'aspect structurel du patron, et le diagramme de séquence afin de représenter l'aspect comportemental du patron. Cependant, ce patron ne permet pas de distinguer les éléments obligatoires des éléments facultatifs. Ainsi, la variabilité, qui est un critère important pour faciliter l'instanciation du patron, n'est pas vérifiée. Ce patron est également spécifique aux architectures SOA dans les systèmes distribués, et, il est difficile de l'adapter pour modéliser une application qui n'appartient pas au domaine des SOA.

2.5.4 Patrons de conception définis par Gerdes *et al.*

Les auteurs de [Gerdes et al., 2013] ont décrit un système de patrons qui offre des patrons de conception et des idiomes appropriés pour le développement de programmes parallèles pour les systèmes embarqués temps réel afin de faciliter l'analyse du temps d'exécution dans le pire cas. Ce système offre quatre patrons de conception :

- Le patron « Parallélisme des tâches » (Task Parallelism Pattern) : il a pour objectif de décomposer un problème en des tâches indépendantes et exécutées simultanément de manière à minimiser le temps d'exécution.
- « Parallélisme des tâches périodiques » (Periodic Task Parallelism) : il a pour objectif de décomposer un programme séquentiel en des tâches qui sont exécutées en parallèle. Chaque tâche est exécutée périodiquement avec différentes périodes.
- « Données Parallèles » (Data Parallel) : il a pour objectif de décomposer une opération de mise à jour en des des tâches qui sont exécutées en parallèle.
- « Chaîne de traitement » (Pipeline) : il a pour objectif de découper l'exécution des instructions de calcul en différentes étapes et de fournir un mécanisme par lequel chaque étape du «pipeline» peut envoyer des éléments de données à l'étape suivante.

Les patrons de conception proposés dans [Gerdes et al., 2013] aident à fournir une analyse du temps d'exécution dans le pire cas des programmes parallèles. Ainsi, ces patrons sont destinés au parallélisme et au contrôle de concurrence car les STR doivent ordonner les tâches concurrentes pour pouvoir accélérer l'exécution. Ils offrent des indices sur lesquels des informations d'annotation sont nécessaires pour réduire la surévaluation dans l'analyse du temps d'exécution dans le pire cas, et de favoriser une analyse du temps statique. Ces patrons sont représentés par un formalisme composé d'un ensemble de rubriques décrites en langage naturel. Ceci n'exprime pas clairement les solutions des patrons et rend leur compréhension difficile. Cette représentation ne favorise donc pas la réutilisation de ces patrons.

2.5.5 Synthèse

Les travaux présentés précédemment se concentrent sur la définition des patrons de conception TR. Parmi ces travaux, il y a ceux qui expriment les exigences non fonctionnelles relatives aux mécanismes propres au domaine TR (*e.g.,* tolérance aux faute, ordonnancement des tâches, parallélisme et synchronisation des horloges). Ces patrons permettent aux concepteurs de développer des STR de très bonne qualité et satisfaisant les contraintes temporelles. Néanmoins, ils ne permettent pas de décrire la structure des STR (*i.e.,* les participants, leurs attributs, leurs opérations et leurs relations) et leur comportement.

Pour ces raisons, nous proposons dans cette thèse des patrons de conception TR qui prennent en compte (a) la description de la structure des applications d'aide à la conduite et leur comportement, (b) la modélisation de leurs aspects non fonctionnels, (c) le critère de variabilité et (d) les contraintes temps réel.

2.6 Conclusion

Ce chapitre nous a permis, dans un premier temps, de montrer que la réutilisation constitue une solution prometteuse et pertinente dont l'objectif est d'aider les ingénieurs d'applications à la conception des systèmes et l'amélioration de leur qualité. Dans un deuxième temps, l'étude comparative que nous avons réalisée a montré que les patrons, et particulièrement les patrons de conception, constituent la forme de réutilisation la plus pertinente pour la modélisation des systèmes d'aide à la conduite, afin de réduire leur complexité. Dans un troisième temps, nous avons présenté des travaux qui se concentrent sur la définition des notations pour exprimer la variabilité des patrons et garder trace de leur instanciation. Nous avons également donné un aperçu sur les travaux qui définissent des patrons de conception dédiés au domaine temps réel. Nous avons montré les avantages et les limites de leur utilisation et présenté le positionnement de nos travaux par rapport à ceux étudiés. Cette étude nous a permis de tirer profit des travaux existants pour la spécification et la modélisation des pa-

trons TR destinés aux systèmes d'aide à la conduite intégrant un système de base de données temps réel, pour pouvoir gérer un grand volume de données.

Dans le chapitre suivant, nous proposons une nouvelle architecture des systèmes d'aide à la conduite qui intègrent des systèmes de bases de données temps réel afin d'améliorer leur fiabilité et leur sécurité.

Deuxième partie

Contribution

Chapitre 3

Intégration d'un système de BDTR dans les systèmes d'aide à la conduite

3.1 Introduction

Dans le chapitre 1, nous avons étudié des méthodes de tolérance aux fautes dédiées au domaine automobile. Il ressort de cette étude que les méthodes basées sur l'approche passive sont les plus adéquates pour les systèmes d'aide à la conduite. En effet, ces méthodes compensent les défauts des capteurs sans qu'il soit nécessaire d'installer du matériel supplémentaire, contrairement aux méthodes basées sur la redondance matérielle, qui ajoute une complexité et des coûts supplémentaires. Elles tolèrent également les défauts avec le minimum de calcul, contrairement aux techniques de FDI (Fault Detection and Isolation) qui nécessitent des temps de calcul élevés et ne conviennent pas aux systèmes temps réel (STR), y compris les systèmes d'aide à la conduite. Ces systèmes manipulent de grandes quantités de données issues de plusieurs capteurs et nécessitent leur stockage. Ces données doivent être mises à jour régulièrement pour refléter fidèlement l'état de l'environnement. Ainsi, les systèmes d'aide à la conduite doivent être gérés par des systèmes de Bases de Données Temps Réel (systèmes de BDTR) qui garantissent le respect des contraintes temporelles relatives aux données et aux traitements TR. Ce constat nous amène à proposer dans ce chapitre

une nouvelle méthode de tolérance aux fautes basée sur une approche passive pour tolérer les défauts des capteurs causant des valeurs manquantes (*i.e.,* un signal nul) qui peuvent se produire dans les systèmes d'aide à la conduite. Notre méthode se base sur l'intégration d'un système de BDTR qui vise à améliorer la tolérance aux fautes des systèmes d'aide à la conduite ainsi que leur fiabilité et leur sécurité. Les systèmes de BDTR sont décrits dans la première section de ce chapitre. La deuxième section présente l'intégration d'un système de BDTR dans les systèmes d'aide à la conduite et son impact sur ces systèmes.

3.2 Systèmes de bases de données temps réel

[Yuan, 1997] a défini les systèmes de BDTR de la manière suivante : *"A real-time database system combines the features from both real-time systems and database systems ; it not only has to satisfy the time constraints required by a real-time system, but also has to maintain the data consistency required by a database system"*. Selon cette définition, un système de BDTR est considéré comme étant un Système de Bases de Données (SBD) qui gère un grand volume de données tout en respectant les contraintes temporelles relatives aux données contenues dans la base et aux transactions [Ramamritham et al., 2004]. En effet, il supporte toutes les fonctionnalités des systèmes de gestion de bases de données classiques (la gestion d'accès aux données structurées, le respect des propriétés ACID [1] des transactions, etc.), mais il doit en plus garantir le respect des contraintes de temps sur les données et les transactions.

3.2.1 Caractéristiques des données TR

Un système de BDTR est conçu pour gérer deux types de données temps réel qui possèdent un intervalle de validité en dehors duquel leur valeur est obsolète :

- Les données sensorielles qui correspondent aux données acquises par les capteurs. Chaque valeur d'une donnée sensorielle doit être mise à jour régulièrement afin de refléter le plus fidèlement possible l'état du monde réel qu'elle représente.

1. ACID : (Atomicité, Cohérence, Isolation, Durabilité)

- Les données dérivées qui sont calculées à partir des données sensorielles [Amirijoo et al., 2006]. Chaque valeur d'une donnée dérivée est mise à jour à chaque fois qu'une donnée sensorielle utilisée pour son calcul est mise à jour [Amirijoo et al., 2006].

Dans [Idoudi et al., 2008a], les auteurs ont défini une donnée TR comme un quadruplet : $d = (d_{value}, d_{estampille}, d_{avi}, mde)$, où d_{value} représente la valeur réelle de la donnée, $d_{estampille}$ indique l'instant où cette valeur est mise à jour, d_{avi} est l'intervalle de validité absolue de la donnée et mde représente l'erreur maximale tolérée entre la valeur réelle de la donnée et la valeur stockée dans la base.

Ces données TR doivent être cohérentes temporellement (*i.e.,* elle doivent toujours être valides). Cette contrainte vient du fait que ces données doivent toujours refléter l'état réel de l'environnement. Ceci exige que les systèmes de BDTR doivent non seulement maintenir la cohérence logique des données, mais doivent également assurer la cohérence temporelle des données. La cohérence temporelle peut avoir deux formes [Ramamritham, 1993] :

- La cohérence temporelle absolue qui est liée au fait que l'image de l'environnement dans le système doit refléter l'état réel de cet environnement. En effet, la cohérence temporelle absolue est maintenue si $(instant_courant - d_{estampille}) \leq d_{avi}$.
- La cohérence temporelle relative concerne les données sensorielles qui servent à dériver d'autres données. Ces données sensorielles sont considérées relativement cohérentes si elles sont temporellement corrélées entre elles. En effet, la cohérence temporelle relative est maintenue si $\forall d, d' \in R, |d_{estampille} - d'_{estampille}| \leq R_{dvr}$, où R est l'ensemble de données utilisées pour dériver une autre donnée et R_{dvr} représente la durée de validité associée à l'ensemble R.

3.2.2 Caractéristiques des transactions TR

Comme les données sont soumises à des contraintes souvent temporelles, les transactions doivent également respecter leurs contraintes temporelles, exprimées souvent sous forme d'échéances. Ces transactions sont classées en trois catégories selon les conséquences engendrées sur l'environnement lorsqu'elles ratent leurs échéances [Ozsoyoglu and Snodgrass, 1995] :

1. Transactions à échéances strictes critiques (hard deadline transactions) : une transaction de ce type qui rate son échéance conduit à des conséquences graves sur l'environnement.
2. Transactions à échéances strictes non critiques (firm deadline transactions) : une transaction qui rate son échéance devient inutile pour le système et sera donc abandonnée.
3. Transactions à échéances non strictes (soft deadline transactions) : une transaction qui rate son échéance n'est pas immédiatement abandonnée par le système, vu qu'elle peut être utilisée même après l'expiration de son échéance, mais la qualité du service offerte est moindre.

Généralement, les transactions TR manipulent des données sensorielles ou des données dérivées [Ramamritham et al., 2004]. Les transactions TR s'exécutent en tenant compte des différents types de données : sensorielles ou dérivées. Les données sensorielles représentent des entités concrètes de l'environnement (*e.g.,* vitesse et direction). En revanche, les données dérivées sont calculées à partir des données sensorielles. Cette distinction entre les deux types de données mène à un deuxième classement des transactions [Ramamritham, 1993] :

— Transactions sensorielles : elles permettent de mettre à jour les données sensorielles et sont exécutées périodiquement pour refléter fidèlement l'état de l'environnement à contrôler. Elles sont composées d'opérations d'écriture seule.

— Transactions déclenchées de mise à jour : ce sont des transactions de mise à jour des données dérivées. Ces transactions sont déclenchées par la mise à jour des données sensorielles. Elles sont donc exécutées sporadiquement. Elles effectuent des opérations de lecture/écriture.

— Transactions *utilisateur* : elles sont exécutées par l'utilisateur ou le système. Elles effectuent des opérations de lecture ou écriture des données non temps réel. Mais, elles effectuent uniquement des opérations de lecture des données TR.

3.2.3 Synthèse

Nous constatons que les systèmes de BDTR ont le potentiel de traiter efficacement un volume important de données tout en respectant les contraintes temporelles relatives aux données et aux transactions. Partant de ce constat, de nombreuses recherches portent sur la conception des systèmes de BDTR pour résoudre différents problèmes, à savoir le contrôle de concurrence des transactions (*e.g.,* [Lindström, 2003]), les systèmes de gestion de bases de données TR distribués (*e.g.,* [Hameed et al., 2012]) et la qualité de service (*e.g.,* [Amirijoo et al., 2006]). Néanmoins, peu de travaux se sont intéressés à la manière dont les systèmes de bases de données temps réel peuvent assurer la tolérance aux fautes, un aspect crucial pour améliorer la fiabilité des systèmes TR qui manipulent un grand volume de données (*i.e.,* éviter les fausses alertes) et leur sécurité (*i.e.,* réduire les conséquences graves).

3.3 Intégration d'un système de BDTR dans les systèmes d'aide à la conduite

Les systèmes d'aide à la conduite ont beaucoup évolué (*e.g.,* le nombre de capteurs et/ou le nombre d'opérations d'acquisition augmentent considérablement) et leur mise en œuvre est devenue plus complexe, car ils nécessitent un accès à un volume important de données. Ces systèmes ne gèrent pas le stockage des données. Par conséquent, l'acquisition de ces données devient très coûteuse en temps, et le risque que les transactions ratent leur échéance augmente. De plus, ces systèmes sont confrontés à des problèmes dans la manipulation d'une grande quantité de données, ce qui peut engendrer des conséquences graves lorsqu'une donnée est perdue. En effet, lorsqu'un capteur est défectueux, la donnée ne sera pas mise à jour régulièrement, ce qui entraîne des traitements incorrects et par conséquent des situations critiques. D'où l'intérêt du stockage dans une base de données temps réel (BDTR) en vue de minimiser le nombre d'accès et d'optimiser les temps d'acquisition. Pour cela, nous proposons une contribution à l'évolution de l'architecture existante des systèmes d'aide à la conduite pour la rendre plus conviviale et pour y intégrer un système de BDTR.

Dans la suite de cette section, nous détaillons la nouvelle architecture de systèmes d'aide à la conduite. Nous montrons également les avantages de l'intégration d'un système de BDTR dans l'amélioration du temps de réponse des transactions et la tolérance aux fautes des systèmes d'aide à la conduite.

3.3.1 Nouvelle architecture des systèmes d'aide à la conduite

Dans cette section, nous proposons une nouvelle architecture de systèmes d'aide à la conduite qui tient compte, à la conception, d'un système de BDTR. Ce système a pour objectif de gérer d'une manière efficace un grand volume de données, leurs contraintes temporelles et celles liées aux transactions. De ce besoin, résulte la nécessité de reconcevoir l'architecture existante des systèmes d'aide à la conduite, décrite dans le premier chapitre, pour intégrer un système de BDTR, comme illustré par la figure 3.1. En effet, les systèmes d'aide à la conduite utilisent des capteurs embarqués dans le véhicule (*i.e.,* capteurs proprioceptifs et capteurs extéroceptifs) qui permettent d'acquérir régulièrement les données relatives au véhicule et à son environnement. Chaque donnée acquise est ensuite transmise périodiquement à une unité de fusion pour être traitée (*e.g.,* convertir le signal, réduire le bruit et extraire les informations à partir d'une image). Puis, elle sera stockée dans une base de données si la différence entre la dernière valeur mise à jour et la valeur stockée dans la base est supérieure à un seuil fixé par le concepteur. Si des capteurs produisent des données redondantes ou complémentaires, l'unité de fusion accède directement à la base pour accéder en lecture aux données et les fusionne (*e.g.,* par moyenne et min/max) si une donnée parmi celles utilisées est mise à jour, et par la suite, la donnée résultante de la fusion est stockée dans la base. Par contre, si à un instant donné toutes les données utilisées pour la fusion ne sont pas mises à jour, la donnée fusionnée est remplacée par la valeur stockée à l'instant précédent. Ensuite, le module de décision (*i.e.,* le contrôleur) accède à la base pour récupérer les données nécessaires pour analyser la situation courante et choisir les actions, à la différence de l'architecture classique des systèmes d'aide à la conduite où le module de décision est lié directement à l'unité de fusion. Enfin, le module d'action reçoit les décisions prises par le module de décision et produit, soit des

alertes visuelles, sonores et/ou haptiques, soit des actions automatiques sur certains organes du véhicule pour tenter d'éviter ou de limiter les effets d'un éventuel accident.

L'intégration de la BDTR dans les systèmes d'aide à la conduite permet de :

- réduire le nombre de transactions de mise à jour des données dérivées. En effet, une donnée dérivée n'est mise à jour uniquement lorsqu'une donnée sensorielle utilisée pour son calcul est mise à jour. Cependant, dans les systèmes d'aide à la conduite existants, la mise à jour des données dérivées est périodique (*i.e.,* une donnée dérivée est calculée même si aucune donnée sensorielle est mise à jour).
- réduire le nombre de transactions de lecture à partir de la base, car il suffit d'accéder à la version de la BDTR qui réside dans la mémoire centrale. En effet, la raison principale pour stocker une version dans la mémoire centrale est d'éviter les délais introduits par les accès disque.

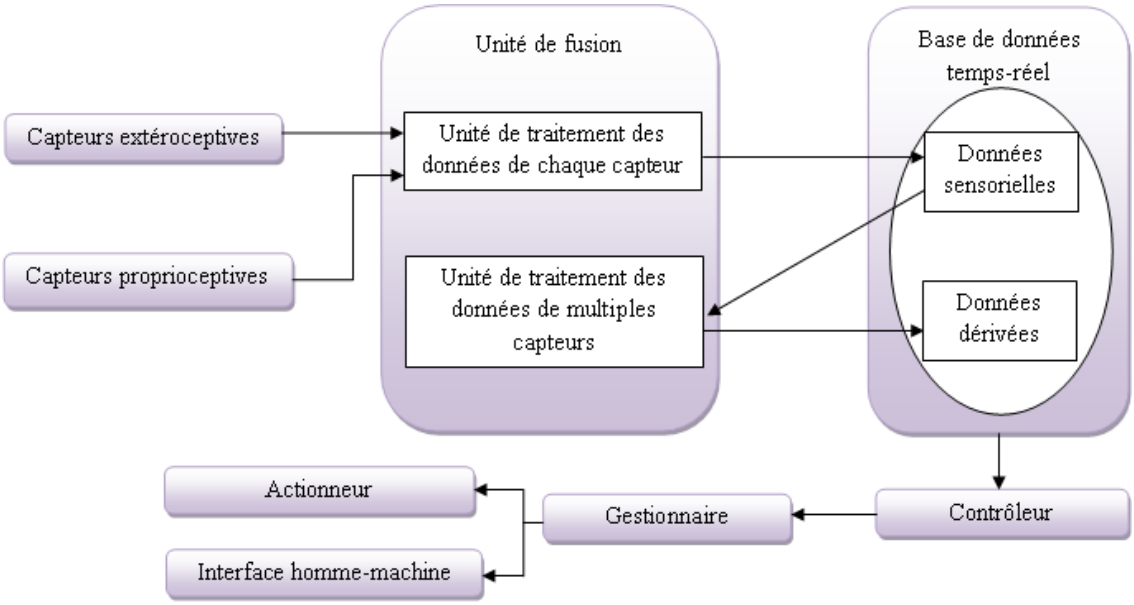


FIGURE 3.1 : Nouvelle architecture des systèmes d'aide à la conduite.

3.3.2 Modèle du système de BDTR

Dans les systèmes d'aide à la conduite existants, toutes les tâches sont exécutées périodiquement. En revanche, avec le système de BDTR, les transactions sont exécutées d'une

manière périodique ou sporadique. En effet, les transactions d'acquisition des données sont périodiques, du fait que les capteurs doivent acquérir régulièrement les données relatives au véhicule et à son environnement pour refléter fidèlement l'état courant du système contrôlé. Tandis que les transactions de mise à jour, elles sont sporadiques vu qu'elles ne sont exécutées que lorsque la différence entre la valeur de la donnée mise à jour et la valeur stockée dans la base est supérieure à un seuil fixé par le concepteur. Ainsi, le système de BDTR permet de minimiser le nombre de transactions de mise à jour, et par conséquent, réduire le temps de leur exécution, contrairement aux systèmes d'aide à la conduite existants où toutes les transactions sont périodiques.

Nous considérons que les transactions de mise à jour sont exécutées en se basant sur la notion de qualité de données (QdD) [Amirijoo et al., 2006] qui indique qu'une donnée stockée dans la base peut posséder un certain écart par rapport à sa valeur dans le monde réel. Cet écart est calculé en faisant la différence entre la valeur courante de la donnée dans la base et sa valeur acquise. Cet écart possède un seuil permettant de déterminer si la transaction de mise à jour peut être exécutée ou écartée. Ce seuil est fixé par le concepteur selon les situations de conduite et les capteurs utilisés. Il doit être le plus petit possible vu que les systèmes d'aide à la conduite sont des systèmes temps réel critiques qui n'autorisent pas des résultats incorrects pouvant entraîner des situations critiques. En effet, une valeur trop grande de ce seuil peut conduire à augmenter le nombre de transactions écartées et même à une dégradation de la qualité de données [Amirijoo et al., 2006]. Pour cela, nous considérons qu'une transaction de mise à jour est écartée si la différence entre la valeur de la donnée qu'elle souhaite mettre à jour et celle stockée dans la base est inférieure à un seuil que nous déterminons par des simulations. Par contre, si cette différence est supérieure à ce seuil, la transaction est exécutée. Dans les deux cas, nous mettons à jour le champ estampille de la donnée [Amirijoo et al., 2006]. Ces constatations sont éprouvées par des simulations qui font l'objet de la section 3.4.

Notre modèle de données est basé sur le modèle proposé dans [Idoudi et al., 2008a] comme le montre la figure 3.2. Cependant, nous considérons dans cette thèse que les données dérivées ne sont pas caractérisées par le champ d_{mde}. En effet, ces données sont calculées à partir

des données sensorielles, et par conséquent, leur mise à jour est déclenchée lorsqu'une des donnée sensorielles est mise à jour. Donc, il n'y a pas d'erreur à calculer ou à prendre en considération pour ce type de données. Selon ce quintuplet, un système de BDTR permet de gérer les contraintes logiques des données (d_{valeur}), les contraintes temporelles ($d_{estampille}$, d_{dva}) et les contraintes de qualité de données (d_{mde}).

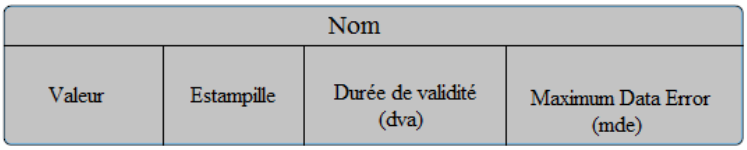


FIGURE 3.2 : Modèle de données temps-réel.

3.3.3 Estimateurs des données manquantes

Les systèmes d'aide à la conduite sont des systèmes temps réel critiques qui sont constitués d'une combinaison de composants matériels et logiciels qui interagissent avec l'environnement grâce à des capteurs et des actionneurs pour effectuer correctement les tâches appropriées. Les défauts d'un ou de plusieurs composants (*e.g.,* capteurs, actionneurs et système) peuvent conduire à des situations critiques. Dans cette thèse, nous considérons les défauts de capteurs vu qu'ils constituent les premiers composants responsables de la production de données nécessaires aux traitements, au contrôle et à la prise de décision. Les défauts de capteurs peuvent conduire à des valeurs différentes des valeurs réelles, ce qui engendre une dégradation de la performance du système et des situations critiques. Ainsi, les systèmes d'aide à la conduite deviennent non fiables et non sécurisés (*e.g.,* [Piao and McDonald, 2008]). Ces deux propriétés peuvent être assurées par l'amélioration de la tolérance aux fautes des systèmes d'aide à la conduite. Pour ce faire, nous proposons une méthode de tolérance aux fautes passive permettant de tolérer différents types de défauts de capteurs (*e.g.,* défauts matériels et défauts logiciels) inconnus causant des données manquantes (*i.e.*, un signal nul), à l'opposé des contrôleurs passifs existants dans la littérature qui ne tolèrent qu'un type prédéfini de défauts. En effet, nous proposons de concevoir un système de BDTR qui permet d'estimer les données manquantes en se basant sur des méthodes d'imputation.

De nombreux travaux se sont intéressés au développement de méthodes d'imputation pour estimer les données manquantes dans un réseau de capteurs dans le domaine des systèmes de transport intelligents (*e.g.,* [Jiang, 2007] [Boyles, 2011]). Ces travaux présentent des études comparatives entre différentes méthodes. Par exemple, certains travaux confirment que les algorithmes *"Espérance-maximisation (EM)"* et *"Augmentation de données"* sont complexes et nécessitent des temps de calculs élevés ([Smith et al., 2003] [Li and Parker, 2008]). Par conséquent, ces méthodes ne peuvent pas être appliquées aux systèmes d'aide à la conduite, qui sont critiques dans la mesure où le temps est un facteur important (*i.e.,* les systèmes d'aide à la conduite doivent produire des résultats avant une échéance donnée). En outre, les méthodes d'imputation proposées pour les réseaux de capteurs (*e.g.,* [Li and Parker, 2008]) ne peuvent pas être utiles pour les systèmes d'aide à la conduite car ces systèmes, sur lesquels notre travail porte, n'utilisent pas de capteurs communicants.

Face à ce constat, nous pensons qu'une méthode d'imputation qui nécessite un temps de calcul minimal et qui fournit une valeur précise est une nécessité absolue pour les systèmes d'aide à la conduite. Pour cela, nous comparons trois méthodes qui sont les plus simples et qui ont un temps de calcul faible :

- Imputation par la moyenne historique : cette méthode consiste à remplacer une valeur manquante par la moyenne de n observations d'un même capteur i, n étant fixé par le concepteur. Cette méthode est simple mais elle est biaisée car elle entraîne une réduction de la dispersion de chacune des variables. La valeur imputée est donnée par : $y_i^* = \bar{y_n}$, où $\bar{y_n}$ est la moyenne de n observations d'un même capteur i.
- Imputation par la valeur précédente : lorsqu'un capteur acquiert une nouvelle valeur d'une donnée, cette valeur remplace celle stockée dans la base. Cependant, si la valeur est manquante, la méthode consiste à remplacer cette valeur par la dernière valeur stockée dans la base. D'où la valeur imputée qui est donnée par : $y_{i,t}^* = y_{i,t-1}$, où $y_{i,t-1}$ est la valeur observée pour un capteur i à l'instant *t-1.*
- Imputation par régression linéaire [Hastie et al., 2009] : cette méthode illustre la relation entre la variable y et les entrées de r capteurs $x = (x_1, ..., x_r)$ en se basant sur l'équation de régression : $y = b_1x_1 + b_2x_2 + ... + b_rx_r + e$, où $B = (b_1, ..., b_r)$ est

l'ensemble des coefficients de régression et e une variable d'erreur. Les coefficients de régression sont déterminés par la résolution de l'équation normale des moindres carrés donnée par : $B = (X^T X)^{-1} X^T Y$, où X est une matrice contenant n enregistrements à partir de r capteurs, X^T est la matrice transposée de X, et Y est un vecteur de n variables.

Pour choisir la méthode la plus appropriée parmi les trois méthodes décrites ci-dessus, nous examinons la précision de chaque méthode en utilisant un indicateur basé sur l'erreur quadratique moyenne (Root Mean Squared Error : RMSE) [Abdella and Marwala, 2005]. Cet indicateur est donné par : $\sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y_i})}{n}}$, où y_i est la valeur réelle, $\hat{y_i}$ représente la valeur estimée et n est le nombre de données manquantes. Il permet de mesurer l'erreur entre la valeur réelle et la valeur estimée. Le paramètre RMSE permet donc d'évaluer les performances d'une prédiction [Abdella and Marwala, 2005]. En effet, le minimum de RMSE indique une variation minimale, ce qui constitue une méthode d'estimation plus précise. Le tableau 3.1 illustre un exemple des valeurs de RMSE pour chaque méthode (*i.e.,* imputation par la moyenne historique, imputation par la valeur précédente et imputation par régression linéaire) appliquée pour estimer la valeur de la vitesse du véhicule contrôlé dans le système régulateur de vitesse à contrôle de distance (Adaptive Cruise Control : ACC) et la valeur de la position du marqueur au sol à gauche dans le système d'alerte de franchissement de lignes (Lane Departure Warning : LDW) du simulateur PreScan [2]. Ce tableau montre que la méthode *"Imputation par la valeur précédente"* est la plus précise (elle possède la valeur de RMSE la plus petite). En outre, les méthodes *"Imputation par la moyenne historique"* et *"Imputation par régression linéaire"* utilisent des données historiques qui peuvent contenir des données estimées. Elles peuvent donc conduire à une dégradation des valeurs estimées. Pour ces raisons, nous choisissons d'appliquer la méthode *"Imputation par la valeur précédente"* pour estimer les valeurs manquantes. Cette méthode est appliquée si la valeur manquante n'est pas présente pour une durée courte. Par contre, dans le cas contraire, nous déclenchons une alerte pour informer le conducteur que le système ne fonctionne pas correctement. La durée pour appliquer la méthode d'estimation est fixée par le concepteur selon les caractéristiques des capteurs utilisés et les conditions de conduite. En effet, si une donnée n'est

2. https ://www.tassinternational.com/prescan

pas présente pour une longue durée, elle est remplacée par la même valeur stockée dans la base. Par exemple, si nous considérons que la valeur de la vitesse du véhicule est manquante pendant une certaine période et la valeur stockée est égale à 100 km/h, le capteur radar détecte que l'inter-distance entre le véhicule et le véhicule précédent (le véhicule précédent a toujours la même vitesse) diminue chaque tous les dixièmes de secondes et devient inférieure à la distance de sécurité prédéfinie. Dans ce cas, la vitesse du véhicule augmente, et par conséquent, un risque de collision est présent, mais le système ne déclenche pas des alertes ou des ordres de commande. Ainsi, la valeur stockée n'est pas correcte dans ce cas, et par conséquent, nous déclenchons une alerte pour informer que le système fonctionne mal.

Tableau 3.1 : RMSE (Root Mean Squared Error) pour les données manquantes dans les sous-systèmes ACC et LDW.

Méthode	**RMSE pour ACC**	**RMSE pour LDW**
Imputation par la moyenne historique	0.075	1.04
Imputation par la valeur précédente	0.024	0.57
Imputation par régression linéaire	1.178	1.51

Pour évaluer les avantages conséquents de l'intégration d'un système de BDTR dans les systèmes d'aide à la conduite, nous proposons d'utiliser un simulateur. Cette évaluation fait l'objet de la section suivante.

3.4 Validation de l'intégration d'un système de BDTR dans les systèmes d'aide à la conduite

Dans cette section, nous présentons l'intégration d'un système de BDTR dans deux sous-systèmes d'aide à la conduite du simulateur PreScan. Nous montrons que ce système de BDTR permet d'améliorer la fiabilité des systèmes d'aide à la conduite par l'estimation des

données manquantes et la gestion des transactions et des données temps réel. Pour le faire, nous proposons d'utiliser SQlite3 [3] pour implémenter le système de BDTR.

3.4.1 Système SQLite3

Généralement, les systèmes de bases de données temps réel sont basés sur des simulations, car peu de prototypes sont proposés, à savoir STRIP (STranford Real-time Information Processor) [Adelberg et al., 1995] et BeeHive [Kim et al., 2002]. Cependant, la plupart de ces systèmes sont propriétaires, tels que TimesTen [TimesTen Team, 1999], BeeHive [Kim et al., 2002] et DeeDS [Andler et al., 1996]. Au contraire, SQLite3 est bien adapté également aux systèmes TR grâce à sa légèreté, sa rapidité et sa forte capacité dans le contrôle. Dans ce sens, de nombreux travaux se sont intéressés à adopter SQLite3 et l'adapter aux systèmes TR, (*e.g.,* [Shuai et al., 2012] [Delight, 2013]).

SQLite3 est un système de gestion de bases de données relationnelles libre écrit en C. Il implémente le standard des langages de requêtes SQL 92 et les propriétés ACID des transactions. Il est portable et ne nécessite aucune configuration et aucune administration. SQLite3 produit une base de données légère, qui ne nécessite pas de reproduire le schéma client-serveur, mais d'être intégrée dans l'application. En effet, SQLite3 génère des bases de données dont chacune est enregistrée dans un fichier qui lui est propre et qui peut être accessible par de nombreux outils comme Matlab. Ceci permet de rendre l'accès aux données plus rapide, plus facile et totalement indépendant de la plateforme utilisée, au contraire des systèmes de gestion de bases de données basés sur le modèle client-serveur, dans lesquels un logiciel *client* interroge un serveur où se trouve la base de données.

SQLite3 est populaire pour les systèmes embarqués vu qu'il est très léger (moins de 300 ko). D'où l'intérêt d'utiliser SQLite3 pour les systèmes d'aide à la conduite qui nécessitent des contraintes strictes en termes de ressources limitées, telle que la taille.

3. (http ://www.sqlite.org/)

3.4.2 Simulateur PreScan

PreScan est un outil intégré dans Matlab/Simulink, qui offre une architecture flexible pour la conception d'un environnement de développement des systèmes d'aide à la conduite et des systèmes de véhicules intelligents (Intelligent Vehicle : IV), qui sont basés sur les technologies de capteurs tels que les radars, les lasers, les caméras, les ultrasons et les GPS [Labibes et al., 2003]. Il permet également de développer et d'évaluer les applications de communication véhicule-à-véhicule (Vehicle to Vehicle : V2V) et véhicule-à-infrastructure (Vehicle to Infrastructure : V2I). Il permet aussi aux constructeurs de tester la conception des systèmes intégrés dans des véhicules dans différentes conditions de conduite.

PreScan est composé des sous-systèmes suivants : régulateur de vitesse adaptatif à contrôle de distance (Adaptive Cruise Control : ACC), système avancé de freinage d'urgence (Advanced Emergency braking System : AEBS), système d'alerte de franchissement involontaire de ligne (Lane Departure Warning : LDW) et système de protection des piétons (Pedestrian Protection System : PPS). Ces systèmes utilisent des capteurs embarqués dans un véhicule, qui acquièrent des informations relatives au véhicule et à son environnement. Ces informations sont ensuite utilisées pour le traitement et la prise de décision. Ces systèmes avertissent le conducteur, soit par le déclenchement des alertes visuelles, soit par l'exécution de commandes.

Comme illustré par la figure 3.3, PreScan permet de développer des systèmes d'aide à la conduite et de les valider en suivant quatre étapes :

1. Construire un scénario : l'interface graphique (GUI) de PreScan permet aux utilisateurs de créer et modifier des scénarios de trafic routier en utilisant une base de données des tronçons routiers, des composants d'infrastructure (les panneaux de signalisation, etc.) et les acteurs (piétons, voitures, etc.).
2. Modéliser le système de capteurs : un véhicule est équipé de différents types de capteurs (radars, caméras, lasers, etc.).
3. Modéliser le système de contrôle : il s'agit de concevoir et de vérifier les algorithmes de traitement des données, de fusion des données de capteurs et de prise de décision en se basant sur Matlab/Simulink.

4. Exécuter l'expérience : il s'agit de visualiser en 3D l'expérience pour permettre aux utilisateurs de vérifier les résultats.

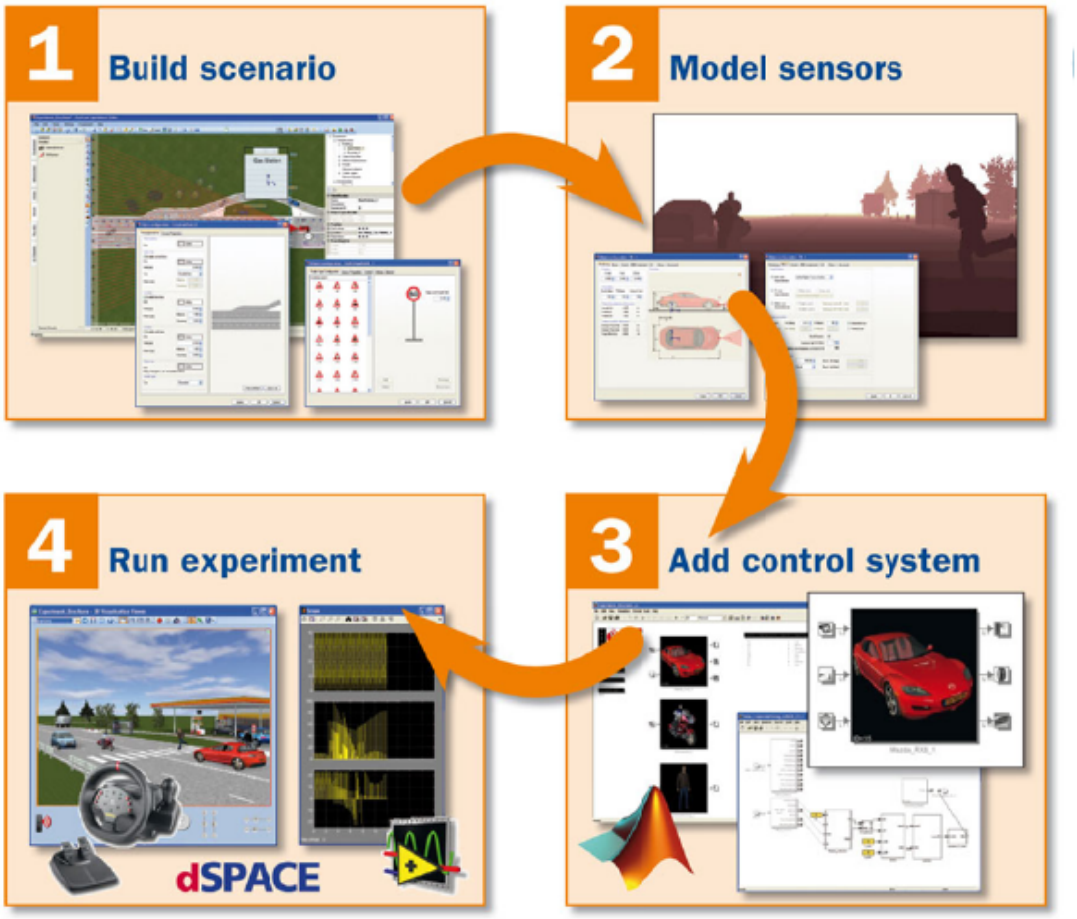


FIGURE 3.3 : Étapes de développement des systèmes d'aide à la conduite dans PreScan [Leneman et al., 2008].

3.4.3 Résultats et discussion

Dans ce paragraphe, nous nous sommes focalisés sur l'intégration d'un système de BDTR au sein de l'architecture des deux systèmes ACC et LDW de PreScan. L'intégration de ce système a fait l'objet du développement d'un modèle sous forme de schéma-bloc à l'aide du logiciel Matlab/Simulink, comme illustré par la figure 3.4. Le bloc *RTDBS* (Real Time DataBase System) intègre toutes les fonctionnalités de gestion de bases de données tout en respectant les contraintes temporelles et la qualité de données (QdD) (*e.g., insertion des*

données dans une base de données tout en respectant l'intervalle de validité et la QdD). Dans la partie qui suit, nous présentons les résultats de la validation.

3.4.3.1 Choix du seuil d'imprécision

La conception du système de BDTR est basée sur le modèle de données défini dans [Idoudi et al., 2008a] et sur la gestion des transactions qui prend en compte la QdD. En effet, ce système vise à stocker les données de ACC et celles de LDW dans la même BDTR tout en respectant l'intervalle de validité de chaque donnée et le critère d'imprécision [Amirijoo et al., 2006] vu que certains systèmes TR tolèrent un certain degré d'imprécision [Amirijoo et al., 2006]. Ces données ne sont mises à jour que si l'erreur sur la donnée (*i.e.,* la différence entre la valeur de la donnée mise à jour et celle stockée dans la base) est supérieure à un seuil d'imprécision que nous déterminons par des simulations selon les conditions de conduite et les capteurs utilisés. Par exemple, si on considère la valeur de la vitesse du véhicule équipé du système, nous constatons que cette donnée possède des valeurs très proches pendant une période. Il est donc inutile de mettre à jour la valeur à chaque instant. Dans ce sens, nous avons choisi pour chaque donnée la plus petite valeur de seuil d'imprécision de façon à minimiser le nombre de transactions de mise à jour et obtenir des résultats similaires aux résultats réels (i.e., résultats obtenus par la simulation sans utiliser un système de BDTR). Pour déterminer les valeurs de seuil appropriées pour chaque donnée, nous avons effectué des simulations pour les deux sous-systèmes ACC et LDW avec différentes valeurs. Le tableau 3.2 illustre, d'une part, les résultats des simulations dans ACC sans l'utilisation de BDTR et, d'autre part, les résultats des simulations avec différentes valeurs de seuil d'imprécision pour quelques données de la base. De même, le tableau présente, d'une part, les résultats des simulations dans LDW sans l'utilisation de BDTR et, d'autre part, les résultats des simulations avec différentes valeurs de seuil d'imprécision pour quelques données de la base.

Le tableau 3.2 montre que les simulations avec les valeurs de seuil 1m/s pour la vitesse du véhicule, 1m/s pour la vitesse du véhicule précédent et 0.01s pour le temps inter-véhiculaires conduisent à des résultats (consistants en des actions freinage) similaires aux résultats réels

obtenus par une simulation sans BDTR. De même, le tableau 3.3 montre que les simulations avec les valeurs de seuil égales à 0.03 m pour chaque donnée conduisent à des résultats (*i.e.,* alerte) similaires aux résultats réels obtenus par une simulation sans BDTR. Cependant, les deux tableaux montrent que les simulations avec d'autres valeurs de seuil engendrent des résultats qui différent des résultats réels, ce qui peut conduire à dégrader les performances du système. Pour ces raisons, nous fixons les valeurs du seuil de l'imprécision comme suit : (a) 1m/s pour la vitesse du véhicule courant, (b) 1m/s pour la vitesse du véhicule précédent, (c) 0.01s pour le temps inter-véhiculaires, (d) 0.03 m pour la distance du véhicule à la ligne gauche de la voie et (e) 0.03 m pour la distance du véhicule à la ligne droite de la voie.

Tableau 3.2 : Exemple de valeurs de données résultant des différentes simulations dans ACC.

Temps(s)	**ACC sans système de BDTR**	**ACC avec système de BDTR**			
	Valeur freinage (%)	**Seuil d'imprécision pour chaque donnée**			**Valeur freinage (%)**
		V-VéhACC [4] **(m/s)**	**V-VéhPrec** [5] **(m/s)**	**Inter** [6] **(s)**	
2.6	33.01	1	1	0 .01	33.01
		1	2	0.01	29
		2	2	0.02	29
12.8	49.15	1	1	0.01	49.15
		1	2	0.01	38.78
		2	2	0.02	25.56

4. Vitesse du véhicule équipé d'ACC, issue du capteur de vitesse.
5. Vitesse du véhicule détecté par le capteur radar LRR.
6. Temps inter-véhiculaires, issu du capteur radar LRR.
7. L'alerte est égale 1 si le système détecte une sortie involontaire de la voie ; l'alerte est égale à 0 si la situation est normale.
8. La distance du véhicule à la ligne gauche de la voie déterminée par le traitement de l'image acquise par le capteur caméra fait par l'unité de fusion.
9. La distance du véhicule à la ligne droite de la voie déterminée par le traitement de l'image acquise par le capteur caméra fait par l'unité de fusion.

Tableau 3.3 : Exemple de valeurs de données résultant des différentes simulations dans LDW.

Temps(s)	**LDW sans système de BDTR**	**LDW avec système de BDTR**		
	Alerte [7]	**Seuil d'imprécision pour chaque donnée**		**Alerte**
		L-dis [8] **(m)**	**R-dis** [9] **(m)**	
17.61	1	0.03	0.03	1
		0.08	0.08	0
		0.1	0.1	0
		0.2	0.2	0
25.01	0	0.03	0.03	0
		0.1	0.1	1
		0.15	0.15	1
		0.4	0.4	1

3.4.3.2 Évaluation des performances du système de BDTR dans les systèmes d'aide à la conduite

Dans cette partie, nous avons voulu montrer comment un système de BDTR peut améliorer la tolérance aux fautes et les performances des systèmes d'aide à la conduite. Dans le cadre de notre travail, nous nous sommes intéressés uniquement aux défauts matériels (capteurs défectueux) qui causent des signaux nuls. Nous définissons alors deux scénarios :

- Scénario 1 : il se produit dans le système ACC lorsque le capteur de vitesse ne fonctionne pas à des instants différents. Dans ce cas, la valeur de la vitesse n'est pas mise à jour et par conséquent le signal est nul. La défaillance du capteur peut être causée par un problème mécanique.
- Scénario 2 : ce scénario a été testé dans le système LDW. Il se produit lorsque le capteur *caméra* est défectueux à certains instants. Dans ce cas, le signal est nul et les informations à extraire du traitement d'une image sont absentes. La défaillance du capteur *caméra* peut être causée par une exposition à une température élevée ou à une forte humidité.

Ces scénarios de conduite sont testés et évalués dans le simulateur, d'une part, sans intégration du système de BDTR et, d'autre part, avec intégration du système de BDTR. La simulation des deux scénarios dans les systèmes ACC et LDW sans intégration du système de

BDTR permet d'obtenir des instants où les systèmes génèrent des fausses alertes. En effet, la simulation du scénario 1 montre les instants où le capteur de vitesse est défectueux. Pendant ces instants (*e.g.,* 2.3s, 5.7s, 6.9s, 11.8s et 27.8s), la vitesse du véhicule est présumée nulle. Ainsi, le contrôleur ne peut pas déterminer l'action appropriée (freinage ou accélération), et par conséquent, le système produit de fausses alertes. C'est pourquoi, le système devient non sécurisé et non fiable. De même, la simulation du scénario 2 nous a permis d'obtenir des instants (*e.g.,* 1.51s, 6.71s, 11.8s et 26.81s) où le capteur caméra est défectueux, et par conséquent, le système est incapable de déterminer s'il y a une sortie de voie involontaire ou non. Pour maintenir ces systèmes fiables et sécurisés malgré la présence des défauts de capteurs, nous avons intégré un système de BDTR pour nous permettre d'estimer les valeurs manquantes qui ont lieu pendant les instants présentés pendant les simulations des scénarios 1 et 2 sans intégration d'un système de BDTR. En fait, lorsqu'un capteur est défectueux, nous appliquons la méthode d'estimation *"Imputation par la valeur précédente"* (cf. Section 3.3.3) pour obtenir des résultats plus précis et le système fonctionne correctement. Pour illustrer comment l'utilisation d'un système de BDTR peut aider le système à fonctionner malgré la présence de défauts, nous prenons l'exemple d'un défaut qui survient à l'instant t = 6.9s (Tableau 3.4). À cet instant, la valeur de la vitesse du véhicule équipé d'ACC est nulle vu que le capteur de vitesse est défectueux. En appliquant la méthode d'estimation *"Imputation par la valeur précédente"*, cette valeur est remplacée par celle stockée dans la BDTR à l'instant t = 6.8s, comme illustré dans le tableau 3.5. En effet, nous constatons que notre méthode permet d'obtenir des résultats quasi identiques aux résultats réels obtenus par des simulations sans intégration du système de BDTR. Ainsi, notre méthode permet d'assurer la fiabilité et la sécurité des systèmes d'aide à la conduite sans nécessiter des calculs. Par conséquent, elle ne demande pas un temps de calcul différent de celui de l'approche décrite dans [Gietelink, 2007] qui propose une méthode FDI (Fault Detection and Isolation) pour tolérer les défauts d'un capteur en utilisant le simulateur PreScan. Cette méthode nécessite des coûts de calcul très élevés (des formules complexes et des algorithmes de filtre). De plus, l'identification des modes de défaillances est un processus qui nécessite beaucoup de temps [Gietelink, 2007].

Les tableaux 3.4 et 3.5 montrent que les valeurs des données : *Vitesse du véhicule, issue du TIS2*[10] et *Temps inter-véhiculaires, issu de TIS2* ont les mêmes valeurs à l'instant t = 6.8s. Ainsi, la différence entre leurs valeurs stockées dans la base et leurs valeurs acquises est inférieure à la valeur du seuil d'imprécision. En conséquence, leurs valeurs ne sont pas mises à jour et le système utilise les valeurs stockées dans la base à l'instant t = 6.8s. Mais, nous mettons à jour l'estampille 6.8s par 6.9s. Cependant, pour les valeurs des données à l'instant t = 6.9s (*Vitesse du véhicule détecté par le TIS1*[11], *Temps inter-véhiculaires, issu de TIS1*, *Temps inter-véhiculaires résultant de la fusion* et *Vitesse résultante de la fusion*), la différence entre leurs valeurs et les valeurs stockées dans la base à l'instant t = 6.8s est supérieure aux valeurs de seuil d'imprécision. Pour cette raison, les valeurs à t = 6.8s (Tableau 3.5) sont remplacées par les valeurs acquises à t = 6.9s (Tableau 3.4).

Tableau 3.4 : Les valeurs des données acquises à t = 6.9 s.

Nom de la donnée	**Valeur**
Vitesse du véhicule équipé d'ACC	signal null
Vitesse du véhicule détecté par les capteurs radars	19
Temps inter-véhiculaires	0.11
Vitesse du véhicule détecté par le TIS1	19
Vitesse du véhicule détecté par le TIS2	0
Temps inter-véhiculaires issu de TIS1	0.11
Temps inter-véhiculaires issu de TIS2	5

D'après le tableau 3.5, nous constatons que le nombre de transactions a diminué, et par conséquent, le temps d'exécution des transactions s'est réduit (tableau 3.6). En effet, le système ACC traite 2365 transactions lorsqu'il n'intègre pas un système de BDTR (simulateur réel). Par contre, il ne traite que 1650 transactions (transactions de mise à jour) lorsqu'il intègre un système de BDTR. Ceci est dû du fait que les données sont mises à jour uniquement lorsque la différence entre la valeur acquise est celle stockée dans la base est supérieure à la valeur du seuil d'imprécision et les données dérivées ne sont mise à jour que lorsque une donnée sensorielle utilisée dans leur calcul est mise à jour. De même, le système LDW traite 2580

10. Capteur radar à coute portée
11. Capteur radar à longue portée

Tableau 3.5 : Exemple de la mise à jour des données dans ACC en utilisant un système de BDTR.

Nom de la donnée	**Valeur**	**Unité**	**Estampille**
Vitesse du véhicule équipé d'ACC	26	m/s	~~6.8~~ 6.9
Vitesse du véhicule détecté par les capteurs radars	~~15~~ 19	m/s	~~6.8~~ 6.9
Temps inter-véhiculaires	~~0.15~~ 0.11	s	~~6.8~~ 6.9
Vitesse du véhicule détecté par le TIS1	~~15~~ 19	m/s	~~6.8~~ 6.9
Vitesse du véhicule détecté par le TIS2	0	m/s	~~6.8~~ 6.9
Temps inter-véhiculaires issu de TIS1	~~0.15~~ 0.11	s	~~6.8~~ 6.9
Temps inter-véhiculaires issu de TIS2	5	s	~~6.8~~ 6.9

transactions dans le simulateur réel (i.e., le système n'intègre pas un système de BDTR). Cependant, il ne traite que 1865 transactions lorsqu'il intègre un système de BDTR. La minimisation du nombre de transactions de mise à jour entraîne la diminution du temps de réponse des systèmes, comme illustré dans le tableau 3.6.

Tableau 3.6 : Nombre de transactions et temps de réponse des systèmes avec et sans intégration d'un système de BDTR.

	ACC		**LDW**	
	Sans un système BDTR	Avec un système BDTR	Sans un système BDTR	Avec un système BDTR
Nombre de transactions	2365	1450	2580	1865
Temps de réponse (seconds)	1487.95	226.19	1427.72	980

3.5 Conclusion

Dans ce chapitre, nous avons remodelé l'architecture existante des systèmes d'aide à la conduite pour intégrer un système de BDTR qui permet de manipuler efficacement un grand nombre de données. Il vise également à garantir les contraintes temporelles relatives aux données et aux transactions. C'est ainsi que nous nous sommes intéressés, dans ce chapitre, à montrer, à travers deux systèmes du simulateur PreScan, que l'intégration d'un système

de BDTR dans les systèmes d'aide à la conduite permet, d'une part, d'améliorer la tolérance aux fautes de ces systèmes en se basant sur la méthode *"Imputation par la valeur précédente"*. Cette méthode ne nécessite pas de calculs énormes et permet de donner des résultats précis (*i.e.*, des résultats presque identiques aux résultats réels), ce qui évite la génération de fausses alertes. D'autre part, le système de BDTR permet l'exécution des transactions d'une manière sporadiques au lieu des exécutions périodiques en se basant sur la QdD. C'est pourquoi le nombre de transactions de mise à jour est réduit, et par conséquent, le temps d'exécution des transactions l'est aussi.

La gestion d'un grand volume de données et leurs contraintes temporelles rend la conception des systèmes d'aide à la conduite reposant sur l'utilisation des systèmes de BDTR plus complexe. Pour remédier à cette complexité, nous proposons des patrons de conception spécifiques à ces systèmes. La modélisation de ces patrons est assurée par le biais d'un profil UML qui fait l'objet du chapitre suivant.

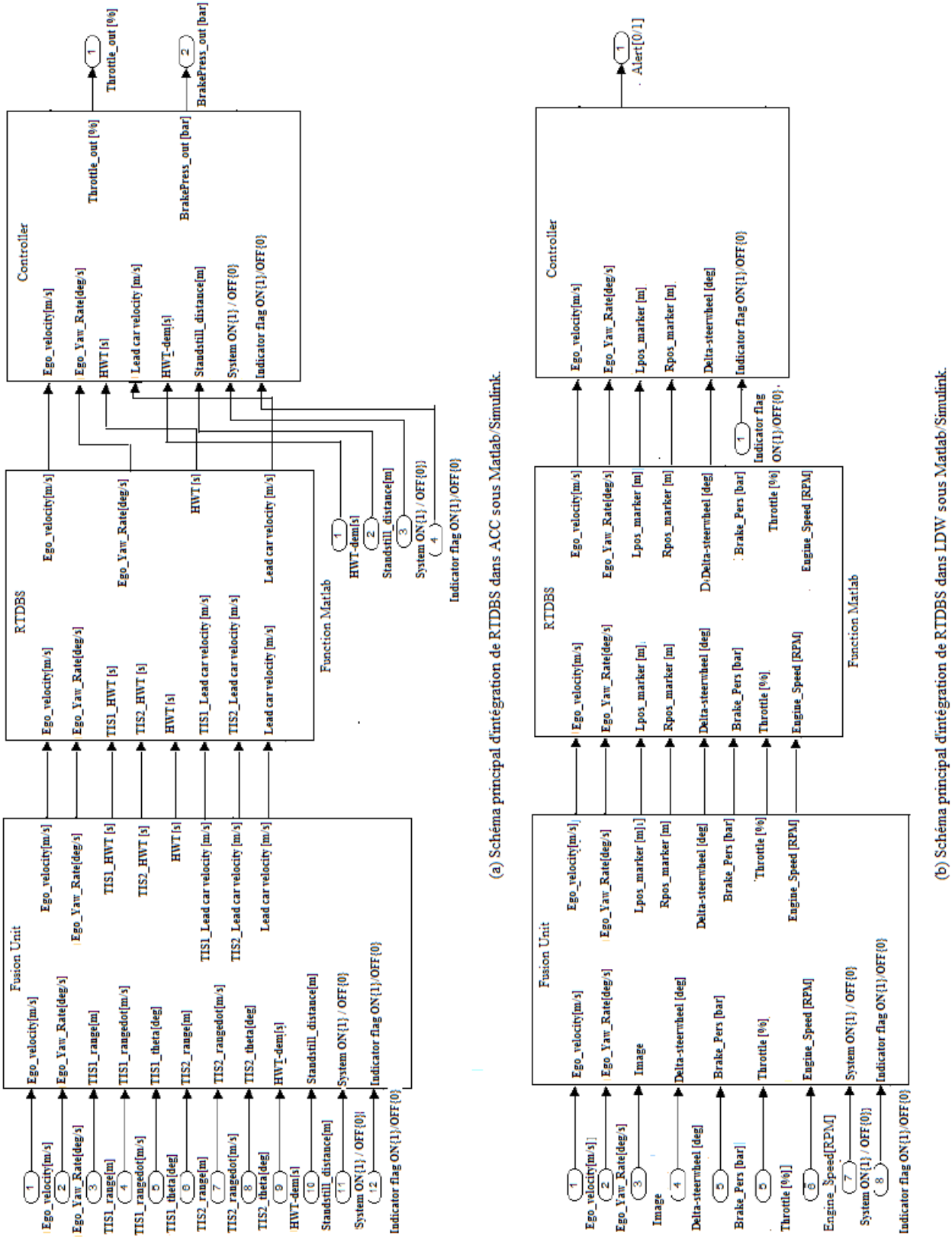


FIGURE 3.4 : Intégration d'un système de BDTR dans ACC et LDW sous Matlab/Simulink.

Chapitre 4

Profil UML pour les patrons de conception temps réel

4.1 Introduction

Nous avons présenté, dans le chapitre 2, quelques profils UML (*e.g.,* [Dong et al., 2007] et [Loo et al., 2012]) destinés à la modélisation des patrons de conception. Ces profils semblent insuffisants pour expliciter les critères de modélisation des patrons, en particulier la manière dont on peut spécifier correctement la variabilité et combiner les critères de modélisation et d'instanciation dans le même profil. Face à ce constat, nous proposons dans ce chapitre un profil UML, nommé UML-RTDB2 (UML-Real Time DataBase 2), une extension du profil UML-RTDB [Idoudi et al., 2008b] qui exprime les aspects des bases de données temps réel. Le profil que nous proposons se distingue par la prise en compte de la représentation de la variabilité ainsi que des critères de modélisation et d'instanciation des patrons.

4.2 Langage de modélisation UML

Le langage UML [OMG, 2007] offre un grand nombre de concepts et de diagrammes pour spécifier des modélisations dans un grand nombre de domaines d'application. Néanmoins, il arrive des cas où ce langage ne puisse pas capturer certaines propriétés spécifiques à un domaine particulier. C'est le cas par exemple des systèmes d'aide à la conduite, pour lesquels il est important de modéliser des contraintes temporelles telles que l'échéance et la périodicité, ce qui n'est pas supporté par les concepts de base d'UML.

D'autre part, UML propose des notations graphiques et visuelles, sous forme de diagrammes, pour bien spécifier et visualiser les éléments du système à développer. Par conséquent, il garantit le critère d'expressivité pour les patrons. De plus, UML offre un ensemble de concepts pour exprimer la variabilité dans les diagrammes.

1. Dans les diagrammes de classes, la variabilité est exprimée par l'utilisation des composants suivants :
   - Les relations d'héritage représentent les points de variation définis par une classe abstraite regroupant les propriétés communes entre différentes applications [McGregor, 2004] et un ensemble de sous-classes qui constituent les différentes variantes. À ces relations d'héritage sont associées des contraintes dont *{incomplet}* et *{partition}* (ou *{XOR}*). La contrainte *{incomplet}* autorise l'adjonction de nouvelles sous-classes lors de l'instanciation du modèle, alors que la contrainte *{partition}* indique qu'une seule sous-classe peut être instanciée parmi l'ensemble de sous-classes.
   - Les interfaces peuvent être réalisées de différentes manières selon l'utilisation des classes qui les implémentent.
   - Les classes *template* d'UML représentent des classes génériques comportant un ensemble de paramètres, instanciés de manière spécifique par chaque système.
2. Dans les diagrammes de séquences, la variabilité est également représentée grâce à l'utilisation des fragments combinés avec les deux opérateurs d'interaction :

(a) L'opérateur {alt} définit un choix parmi des fragments d'interaction (opérandes), *i.e.,* l'opérande choisi sera exécuté.

(b) L'opérateur {opt} définit un opérande optionnel qui peut s'exécuter ou pas.

Les concepts de base d'UML permettent d'exprimer la variabilité, et par conséquent, il est inutile de définir des extensions dans ce cas. Cependant, ils ne sont pas suffisants pour exprimer explicitement la variabilité. Et, comme nous l'avons mentionné dans le chapitre 2, la variabilité est un critère essentiel dans la modélisation de patrons de conception TR de meilleure qualité, flexibles et compréhensibles. C'est pourquoi, il est nécessaire de différencier clairement les éléments communs et les éléments variables.

En outre, ces concepts ne permettent pas de garantir le critère de traçabilité des patrons de conception du fait qu'ils ne peuvent pas marquer les éléments appartenant à chaque instance. Par conséquent, il est difficile de valider l'instanciation des patrons.

C'est notamment pour ces raisons qu'il est devenu nécessaire d'étendre UML, grâce aux mécanismes d'extension, à l'aide des stéréotypes, des valeurs marquées (tagged values) et des contraintes [OMG, 2007], pour tenir compte des critères de modélisation des patrons de conception TR. Ces extensions nous ont amenés à proposer un profil UML qui permet de combler des lacunes du standard UML en termes de sémantique pour servir de pivot à des outils de validation. Dans la section suivante, nous décrivons le concept de profil UML.

4.3 Profil UML

Le profil mécanisme d'extension d'UML a été introduit dans le standard UML 1.3 [OMG, 1999] pour permettre la structuration des extensions UML sous forme de profil. D'après le document d'UML [OMG, 2007], la notion de profil UML est définie comme suit : *"A profile package can specify a related model library and identify a subset of the UML meta model that is applicable for the profile. In principle, profiles merely refine the standard semantics of UML by adding further constraints and interpretations that capture domain-specific semantics and modelling patterns. They do not add any new fundamental concepts"*. Selon cette

définition, un profil UML permet d'étendre des méta-classes existantes sans définir de nouvelles méta-classes. Ce type d'extension est défini par des stéréotypes, des valeurs marquées et des contraintes.

Un stéréotype spécialise les méta-classes du méta-modèle. Il ne définit pas de nouvelles méta-classes mais utilise des méta-classes existantes en leur attribuant une nouvelle sémantique adaptée au contexte de l'application. Un stéréotype est associé à des valeurs marquées et des contraintes. Les valeurs marquées se composent d'un nom de propriété et d'une valeur associée. Quant aux contraintes, elles définissent des restrictions d'utilisation pour la nouvelle méta-classe [Staron and Kuzniarz, 2008]. Ces contraintes sont décrites, soit en langage naturel, soit en langage de contraintes tel qu'OCL (Object Contraintes Language) [OMG, 2012]. Un élément étendu par un stéréotype doit donc respecter les contraintes définies pour ce stéréotype.

4.4 Profil UML-RTDB2

Pour modéliser des patrons de conception TR réutilisables sans ambiguïté, flexibles et compréhensibles, il est nécessaire de définir un langage permettant (i) d'exprimer les contraintes temporelles associées aux données et aux traitements des systèmes reposant sur l'utilisation des bases de données TR, (ii) de spécifier la variabilité au niveau des modèles de patrons et (iii) d'identifier les éléments de patrons instanciés à partir du modèle d'une application spécifique. Nous proposons pour cela le profil UML-RTDB2.

4.4.1 Extensions pour la spécification des patrons de conception TR

4.4.1.1 Extensions pour la représentation de la variabilité

Les patrons de conception sont destinés à être réutilisés pour modéliser différentes applications. Ainsi, il est indispensable de définir un langage de modélisation de la variabilité afin de guider le concepteur d'applications à déterminer les éléments communs entre les applications du domaine et les éléments qui différent d'une application à une autre. Dans ce sens, nous décrivons les stéréotypes que nous proposons pour exprimer la variabilité au niveau structurel ainsi qu'au niveau comportemental.

Stéréotype << optional >>

Nous proposons ce stéréotype pour désigner l'optionalité des éléments d'un diagramme de classes ainsi que les éléments d'un diagramme de séquence. En effet, il indique qu'un élément peut être omis pendant l'instanciation des patrons. Il a la même signification que le stéréotype proposé dans [Clauß and Jena, 2001] qui étend uniquement les méta-classes des diagrammes de classes. Cependant, le stéréotype que nous utilisons permet d'étendre les méta-classes *Class*, *Property*, *Operation*, *Association*, *Lifeline*, *Message* et *InteractionFragment*. La figure 4.1 représente la définition du stéréotype << *optional* >> en utilisant la relation d'extension telle qu'elle est définie dans le paquetage Profiles d'UML 2.2 [OMG, 2009].

Stéréotype << mandatory >>

Nous proposons ce stéréotype pour spécifier les éléments obligatoires, aussi bien au niveau des diagrammes de classes (classes et associations) qu'au niveau des diagrammes de séquence (objets, messages et fragments). En effet, tout élément stéréotypé << *mandatory* >> doit être instancié au moins une fois lors de la réutilisation du patron. Ce stéréotype a la même signification que celle du stéréotype défini par Clauß dans [Clauß and Jena, 2001], où les auteurs proposent l'application du stéréotype pour étendre le méta-modèle du diagramme de classes UML. Néanmoins, nous utilisons le stéréotype << *mandatory* >> pour étendre

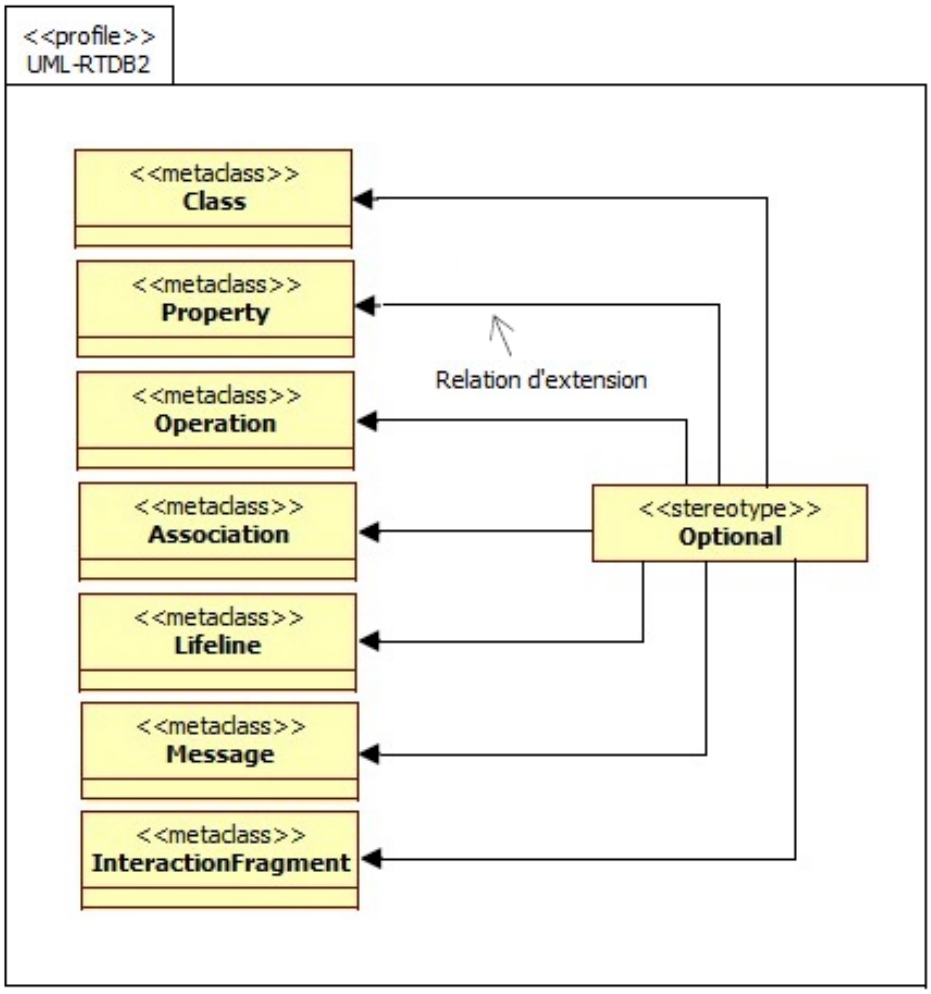


FIGURE 4.1 : Définition du stéréotype << *optional* >>.

les méta-classes *Class*, *Association*, *Lifeline*, *Message* et *Interaction*, comme illustré dans la figure 4.2.

Stéréotype << extensible >>

Ce stéréotype étend la méta-classe *Class* pour spécifier qu'une classe peut être extensible par l'adjonction de nouveaux attributs et/ou opérations pendant l'instanciation des patrons. Il a la même signification que celle du stéréotype proposé dans [Rekhis et al., 2013a]. Ce stéréotype possède deux valeurs marquées *extensibleAttribute* et *extensibleMethod*, qui sont de type booléen, et qui prennent la valeur *Vrai* (par défaut) pour indiquer que le concepteur peut ajouter des attributs (si *extensibleAttribute* est *Vrai*) ou des opérations (si *extensibleMe-*

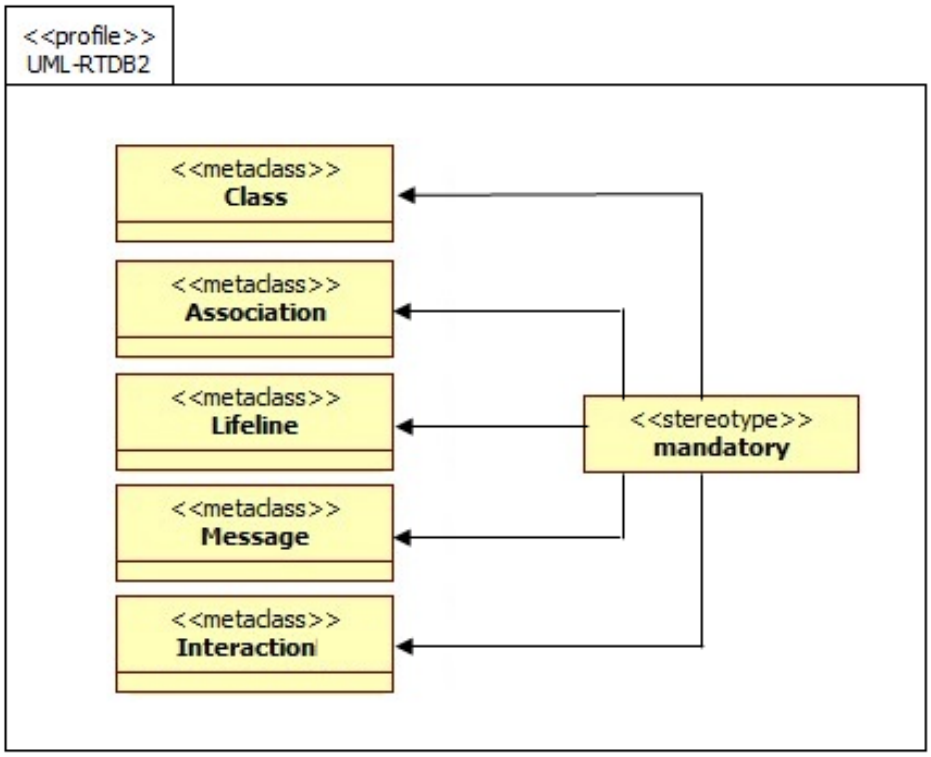


FIGURE 4.2 : Définition du stéréotype << *mandatory* >>.

thod est *Vrai*) dans une instance d'un patron. Nous utilisons des notes associées aux classes uniquement lorsque l'une des valeurs marquées est *fausse* pour indiquer aux concepteurs qu'il est impossible d'ajouter de nouveaux attributs et/ou opérations.

La figure 4.3 montre la définition du stéréotype << *extensible* >> en utilisant la relation d'extension d'UML 2.2.

4.4.1.2 Extensions pour la modélisation des aspects temps réel

Pour répondre aux exigences spécifiques du domaine de modélisation des systèmes temps réel, de nombreux profils UML ont été proposés. Nous citons plus particulièrement :

- Le profil UML-RT [Selic, 1998] [Gherbi and Khendek, 2006] qui se base sur le langage UML pour modéliser les systèmes TR. Il s'agit d'une adaptation de la méthode ROOM

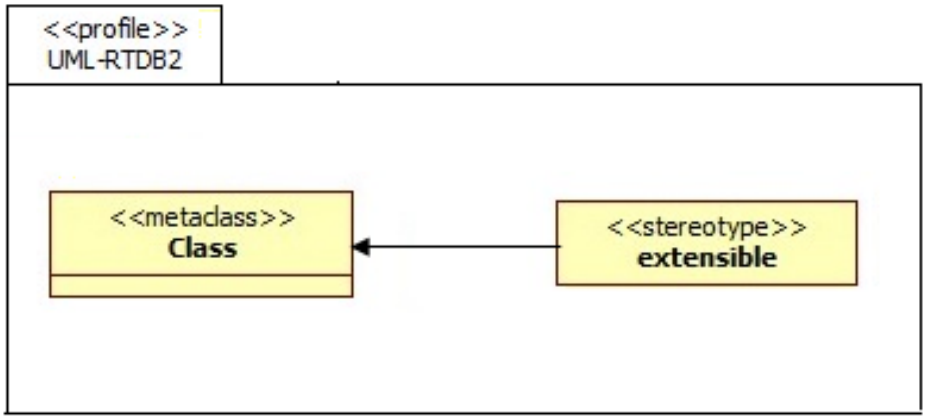


FIGURE 4.3 : Définition du stéréotype << *extensible* >>.

(Real-Time Object_Oriented Modeling) [Selic et al., 1994]. Ce profil introduit des stéréotypes permettant de prendre en compte les aspects contrôle de ces systèmes.

- Le profil TURTLE (Timed UML and RT-LOTOS Environment) qui a été introduit dans [Apvrille et al., 2004], étendu dans [Apvrille et al., 2006] et supporté par l'atelier Open Source TTool [1]. Ce profil permet de mener à bien l'analyse et la conception des systèmes TR et/ou distribués sur la base des diagrammes de classes et d'activités décrivant la structuration du système et son comportement [Apvrille et al., 2006]. Cette conception est validée grâce à une description formelle apportée par le langage RT-LOTOS [Courtiat et al., 2000].
- Le profil SPT (Scheduling Performance and Time [OMG, 2005a]) qui été standardisé par l'OMG pour la modélisation de différents aspects des systèmes TR, tel que la temporisation, les ressources, l'ordonnancement, etc. Cependant, ce profil permet d'annoter un modèle avec des caractéristiques de QdS (Qualité de Service), sans offrir une sémantique formelle et sans faire le lien avec la sémantique de la partie fonctionnelle du modèle UML.
- Le profil MARTE (Modeling and Analysis of Real-Time and Embedded systems) [OMG, 2011] qui remplace le profil SPT pour cibler l'analyse et la modélisation des sys-

1. http ://labsoc.comelec.enst.fr/turtle/

tèmes TR embarqués. Ce profil couvre aussi bien la modélisation des aspects logiciels que matériels. De plus, il assure l'interopérabilité entre les outils de développement.

Les extensions proposées par ces profils ne couvrent pas les besoins de la modélisation des systèmes de BDTR, principalement pour deux raisons. Premièrement, l'architecture des systèmes TR, y compris les systèmes d'aide à la conduite et leur fonctionnement, ne reposent pas sur l'utilisation des BDTR. En effet, un système TR se compose essentiellement (i) d'un module d'acquisition permettant d'acquérir les données à l'aide de capteurs, (ii) d'un module de traitement de ces données et d'élaboration des résultats et (iii) d'un module d'envoi d'ordres de commande à l'environnement à l'aide d'actionneurs. Deuxièmement, ces profils utilisent des stéréotypes UML qui ne prennent en compte, ni les caractéristiques temps réel des BDTR (données temps réel, données dérivées temps réel, etc.), ni de leurs contraintes temporelles (cohérence relative, cohérence absolue), ni la qualité des données, ni encore les contraintes temporelles des transactions. Pour combler ces lacunes, *Idoudi et al.* ont proposé dans [Idoudi et al., 2009] un profil, nommé UML-RTDB (UML Real-Time DataBase) permettant de modéliser les caractéristiques des systèmes TR qui reposent sur l'utilisation des BDTR au niveau des *frameworks*. Nous adoptons des stéréotypes du profil UML-RTDB afin de prendre en considération les propriétés temporelles des BDTR au niveau des patrons de conception à proposer. Nous détaillons, dans ce qui suit, les stéréotypes utilisés.

Stéréotype << sensor >>

Ce stéréotype étend la méta-classe *Class*. Il sert à modéliser des objets représentant des données temps réel sensorielles (*i.e.,* les données acquises par des capteurs) au niveau du modèle structurel. Ce stéréotype possède deux propriétés *Validity_Duration* et *Maximum_Data_Error*, représentant respectivement la durée de validité d'une donnée et l'erreur maximale tolérée d'une donnée. La figure 4.4 montre la définition du stéréotype << *sensor* >> en utilisant la relation d'extension d'UML 2.2.

Stéréotype << derived >>

Ce stéréotype étend aussi la méta-classe *Class*. Il est utilisé pour annoter des éléments représentant des données dérivées (*i.e.,* données calculées à partir des données sensorielles).

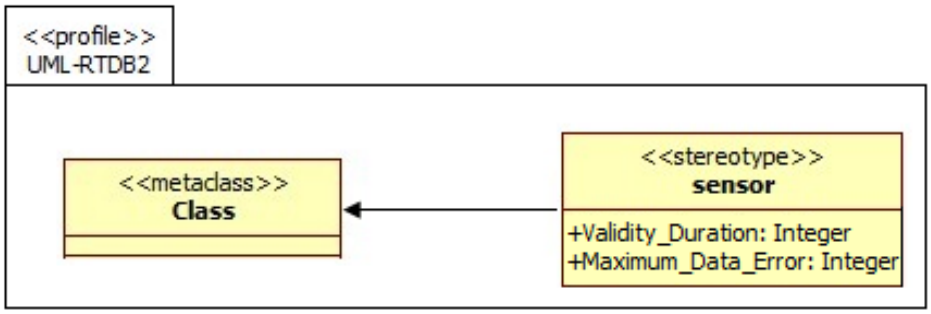


FIGURE 4.4 : Définition du stéréotype << *sensor* >>.

Il est caractérisé par la propriété *Validity_Duration* qui représente la durée de validité de la donnée sensorielle qui a la petite valeur. La figure 4.5 présente la définition du stéréotype << *derived* >>.

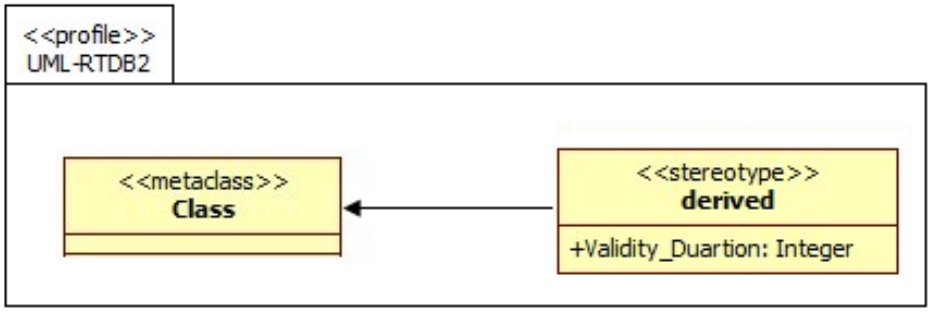


FIGURE 4.5 : Définition du stéréotype << *derived* >>.

Stéréotype << periodic >>

Ce stéréotype constitue une extension des méta-classes *Operation* et *Message*. Il vise à spécifier les opérations des diagrammes de classes et les messages des diagrammes de séquence qui s'exécutent périodiquement. Ce stéréotype possède deux propriétés : *Deadline* (*i.e.,* spécification de l'échéance de l'opération ou du message temps réel) et *Period* (*i.e.,* spécification de la périodicité de l'opération ou du message temps réel). La définition de ce stéréotype est présentée dans la figure 4.6.

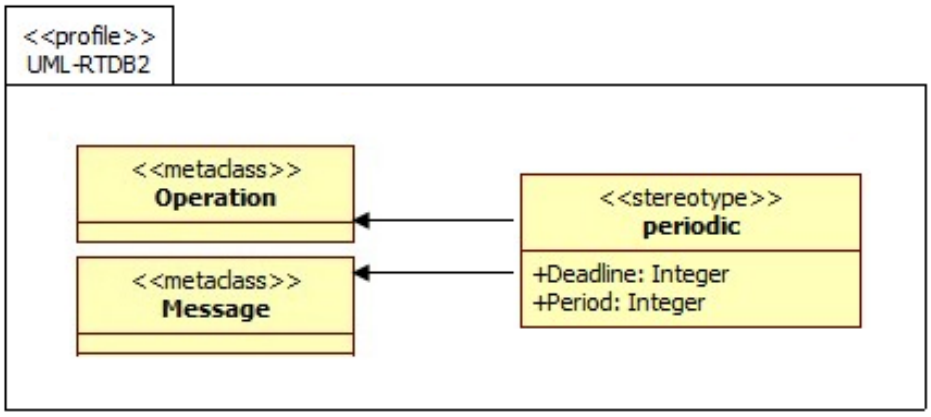


FIGURE 4.6 : Définition du stéréotype << *periodic* >>.

Stéréotype << sporadic >>

Ce stéréotype constitue aussi une extension des méta-classes *Operation* et *Message*. Il indique que ces éléments sont exécutés d'une manière sporadique. Il est caractérisé par deux propriétés : *Deadline* et *Trigged_Time*. La propriété *Deadline* désigne l'échéance de l'opération ou du message temps réel, alors que la propriété *Trigged_Time* donne l'instant d'exécution de l'opération ou du message temps réel, qui se déclenche à la demande. La définition de ce stéréotype est présentée dans la figure 4.7.

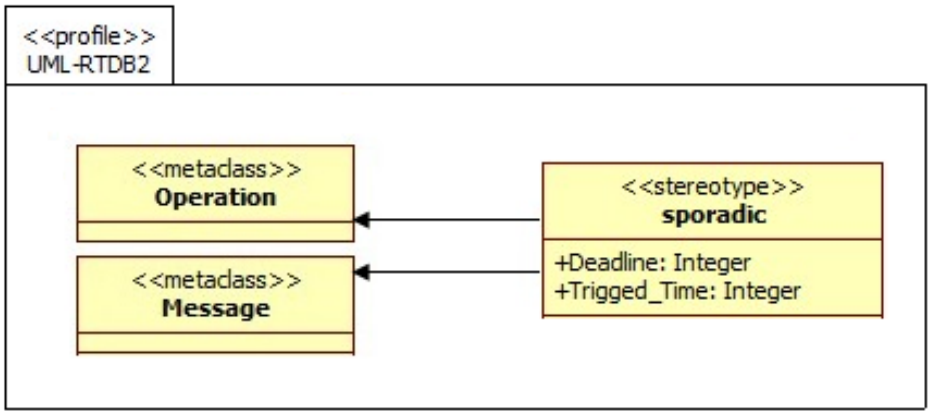


FIGURE 4.7 : Définition du stéréotype << *sporadic* >>.

Les extensions proposées pour le profil UML-RTDB [Idoudi et al., 2009] permettent de spécifier les contraintes temporelles associées aux données et aux transactions des systèmes TR se basant sur l'utilisation des BDTR. Cependant, elles ne permettent pas la représentation

des contraintes non fonctionnelles (*e.g.,* qualité de service), malgré leur importance dans l'amélioration de la qualité du développement. Un système doit donc satisfaire des contraintes, tels que les temps d'exécution, la puissance de calcul, les échéances, l'utilisation de la mémoire, etc. Ceci implique une utilisation optimale des ressources logicielles et matérielles, tout en respectant les contraintes temporelles. Ainsi, garantir le bon fonctionnement et respecter les contraintes non fonctionnelles sont devenus des défis pour le développement logiciel. Donc, il est nécessaire d'adopter des extensions pour spécifier ces contraintes. Pour ce faire, nous adoptons des stéréotypes du paquetage *NFP* (Non Functional Properties) du profil MARTE qui est le standard de l'OMG le plus récent pour la modélisation des systèmes embarqués TR. Nous présentons, ci-après, les stéréotypes utilisés appartenant au paquetage NFP.

4.4.1.3 Extensions pour la modélisation des contraintes non fonctionnelles

Le paquetage NFP de MARTE introduit un ensemble d'extensions permettant de préciser différentes propriétés non fonctionnelles. Ce paquetage sert à qualifier et à typer de manière standard les propriétés spécifiées par le stéréotype $<< nfp >>$ qui constitue une extension de la méta-classe *Proprety* (cf. Figure 4.8). Pour cela, ce paquetage étend les types de données d'UML par les principaux types manipulés dans le domaine temps réel embarqué (*e.g.,* le débit et la fréquence). Ces types standards sont fournis par la librairie *NFP-Types*. Cette librairie englobe un ensemble de types de données destinées aux propriétés non fonctionnelles. Parmi ces types, nous distinguons notamment *NFP_Duration*, *NFP_Frequency* et *NFP_Percentage*. Ces types sont spécifiées par le stéréotype $<< nfpType >>$ qui spécialise le stéréotype $<< TupleType >>$, qui constitue une extension de la méta-classe *DataType*.

La figure 4.8 montre un extrait du diagramme définissant le paquetage NFP contenant les stéréotypes ($<< nfp >>$ et $<< nfpType >>$) que nous importons pour modéliser les propriétés non fonctionnelles au niveau des patrons.

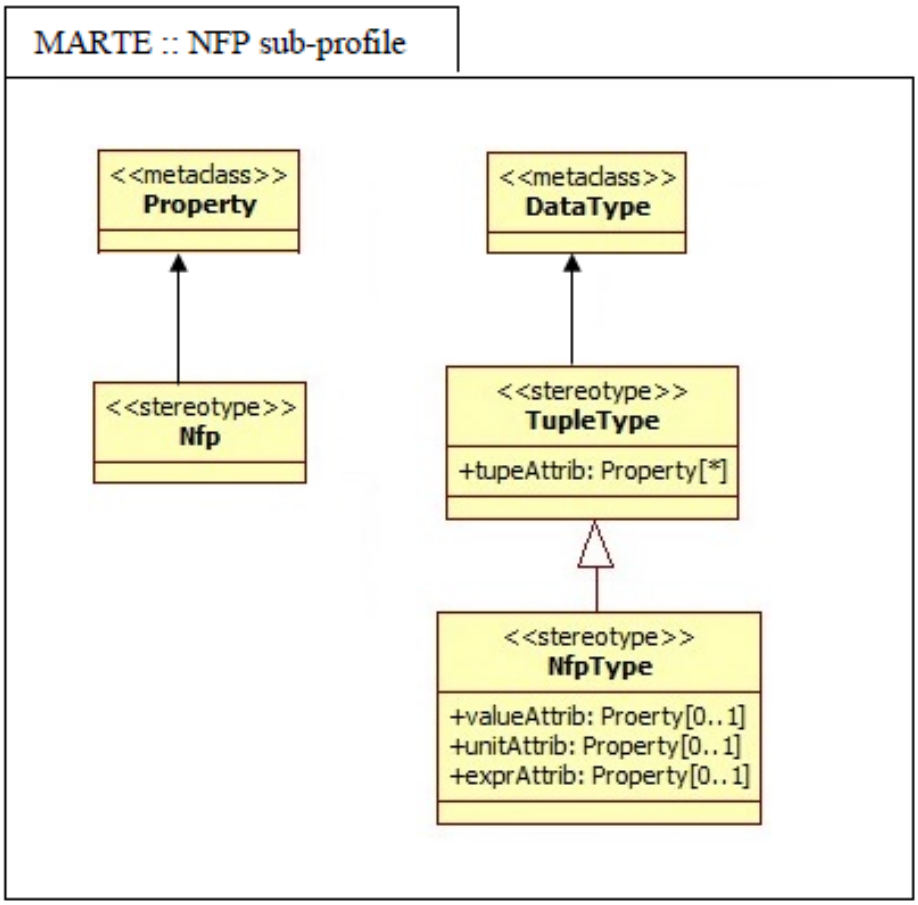


FIGURE 4.8 : Définition du paquetage NFP.

4.4.2 Extensions pour l'instanciation des patrons de conception TR

Une fois les patrons créés, ils peuvent être réutilisés par les concepteurs pour modéliser des systèmes spécifiques. Pour cela, il est nécessaire de différencier les éléments instanciés à partir de patrons de ceux ajoutés par le concepteur de l'application selon ses besoins. Donc, il suffit de garder une trace des éléments d'un patron issus de son instanciation. Par conséquent, nous adaptons certains stéréotypes, que nous détaillons ci-après, pour assurer la traçabilité de l'instanciation de la vue statique et de la vue dynamique d'un tel patron.

Stéréotype << PatternClass >>

Ce stéréotype constitue une extension de la méta-classe *Class*. Il vise à indiquer que la classe est instanciée à partir d'un patron et qu'elle n'est pas ajoutée par le concepteur d'application. La définition de ce stéréotype est illustrée par la figure 4.9.

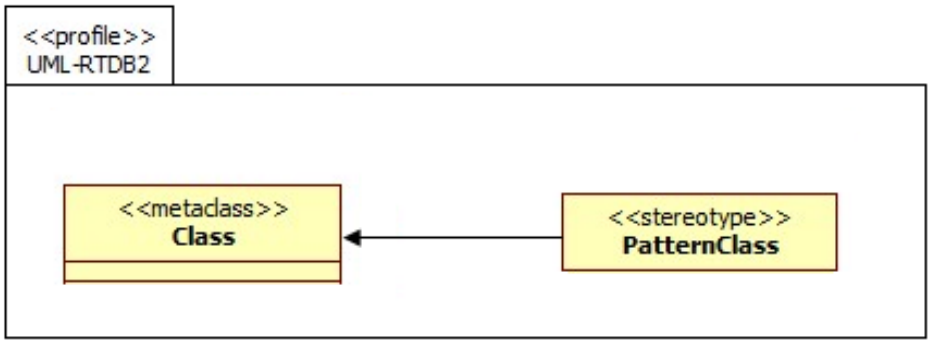


FIGURE 4.9 : Définition du stéréotype << *PatternClass* >>.

Stéréotype << PatternAttribute >>

Ce stéréotype étend la méta-classe *Proprety*. Il est utilisé pour marquer les attributs instanciés à partir d'un patron. La figure 4.10 présente la définition du stéréotype << *PatternAttribute* >>.

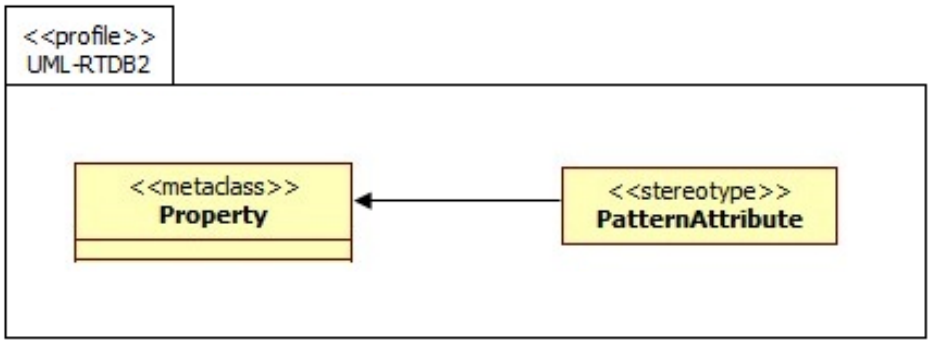


FIGURE 4.10 : Définition du stéréotype << *PatternAttribute* >>.

Stéréotype << PatternOperation >>

Ce stéréotype étend la méta-classe *Operation*. Il est utilisé pour différencier les opérations

instanciées à partir d'un patron de celles ajoutées par le concepteur. La définition de ce stéréotype est présentée dans la figure 4.11.

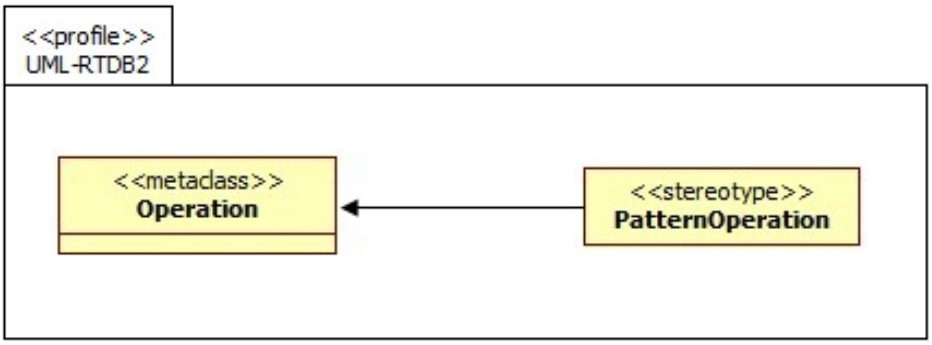


FIGURE 4.11 : Définition du stéréotype << *PatternOperation* >>.

Ces stéréotypes ont les mêmes significations que les stéréotypes définis dans [Dong et al., 2007] mais possèdent des propriétés différentes. En effet, à chaque stéréotype, nous associons une valeur marquée dont le nom possède le format suivant : *{PatternName, role}*, où le champ *PatternName* indique le nom du patron à partir duquel l'élément est instancié, et le champ *role* indique le rôle joué par l'élément dans le patron.

Les stéréotypes << *PatternAttribute* >> et << *PatternOperation* >> peuvent être source de confusion lorsque deux rôles d'attributs ou d'opérations définis dans des rôles différents, de classes différentes portant le même nom, ce qui peut engendrer des problèmes de la validation des instances de patrons. Pour cette raison, nous proposons d'ajouter *ClassName* pour les deux stéréotypes << *PatternAttribute* >> et << *PatternOperation* >>. Ainsi, ces deux stéréotypes disposent de la valeur marquée suivante : *{PatternName, ClassName.role}* pour indiquer que l'attribut ou l'opération est associée à la classe dont le nom dans le patron est *ClassName*. L'information *ClassName* permet d'éviter les ambiguïtés lorsque deux rôles d'attributs ou d'opérations définis dans deux classes différentes ont le même nom.

Stéréotype << ApplicationClass >>

Ce stéréotype étend la méta-classe *Class*. Il est utilisé pour indiquer explicitement qu'il s'agit d'une classe spécifique ajoutée par le concepteur d'application lors de la modélisation

de son système. L'objectif d'utiliser ce stéréotype est d'ajouter plus de sémantique au modèle et de faciliter sa validation. La définition de ce stéréotype est illustrée dans la figure 4.12.

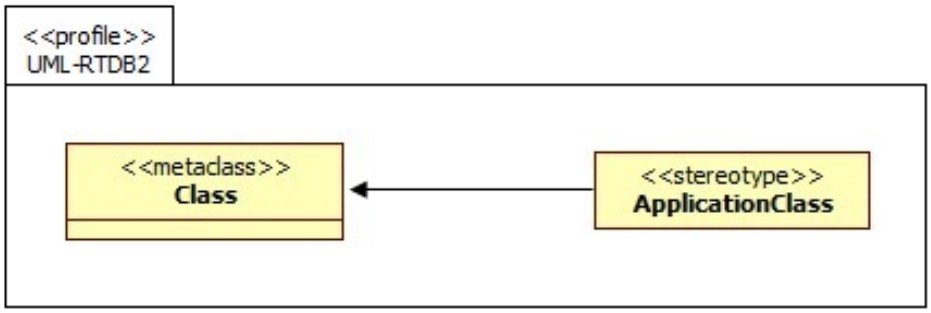


FIGURE 4.12 : Définition du stéréotype << *ApplicationClass* >>.

Stéréotype << PatternLifeline >>

Ce stéréotype constitue une extension de la méta-classe *Lifeline* (c.f. Figure 4.13). Il vise à marquer les objets instanciés à partir du diagramme de séquence d'un patron. Ce stéréotype possède la même valeur marquée que le stéréotype << *PatternClass* >>. Il a la même signification que le stéréotype défini dans [Rekhis et al., 2013a], où les auteurs se sont intéressés uniquement aux objets et aux interactions d'un patron. Cependant, les messages constituent des éléments essentiels du diagramme de séquence, qui sont considérés par les outils lors de la validation des patrons. Pour cela, nous proposons le stéréotype << *PatternMessage* >>. Mais, il n'est pas nécessaire de spécifier explicitement la correspondance entre les interactions d'un patron de ceux spécifiques à une instance de ce patron ; il suffit de marquer les messages et les objets associés à l'interaction, et par la suite elle sera identifiée directement.

Stéréotype << PatternMessage >>

Ce stéréotype constitue une extension de la méta-classe *Message*. Il est utilisé pour différencier explicitement les messages instanciés d'un patron de ceux ajoutés par le concepteur selon les spécificités de l'application à modéliser. Ce stéréotype dispose de la même valeur marquée que les stéréotypes << *PatternClass* >> et << *PatternLifeline* >>. La définition de ce stéréotype est représentée dans la figure 4.14.

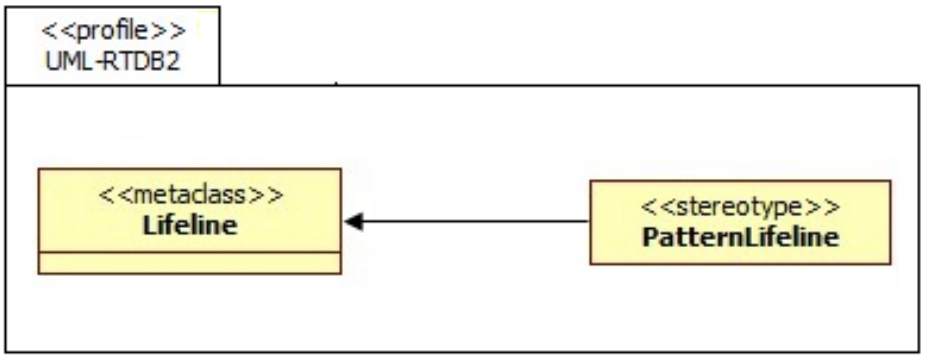


FIGURE 4.13 : Définition du stéréotype << *PatternLifeline* >>.

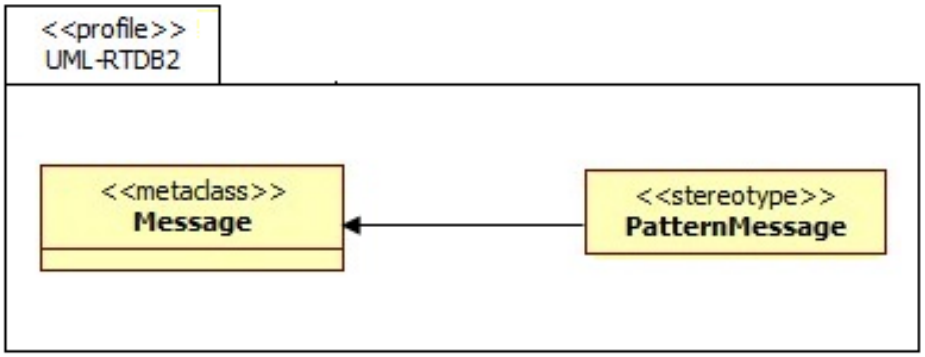


FIGURE 4.14 : Définition du stéréotype << *PatternMessage* >>.

Stéréotype << ApplicationLifeline >>

Ce stéréotype étend la méta-classe *Lifeline*. Il est utilisé pour indiquer explicitement qu'il s'agit d'un objet spécifique ajouté par le concepteur d'application lors de la modélisation de son système. La définition de ce stéréotype est illustrée par la figure 4.15.

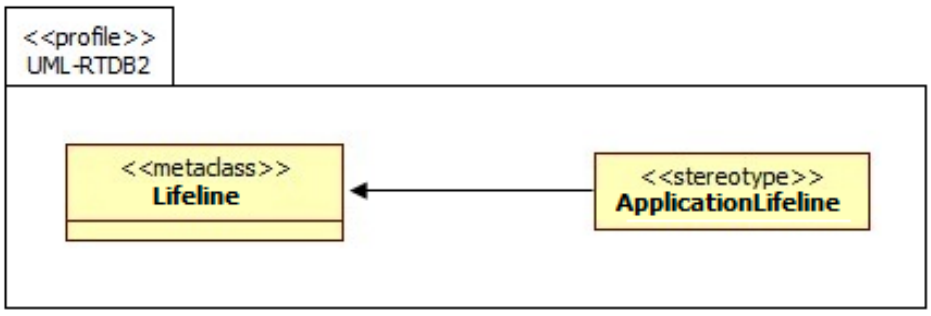


FIGURE 4.15 : Définition du stéréotype << *ApplicationLifeline* >>.

Stéréotype << ApplicationMessage >>

Ce stéréotype étend la méta-classe *Message*. Il est utilisé pour indiquer explicitement qu'il s'agit d'un message spécifique ajouté par le concepteur d'application lors de la modélisation de son système. La définition de ce stéréotype est illustrée par la figure 4.16.

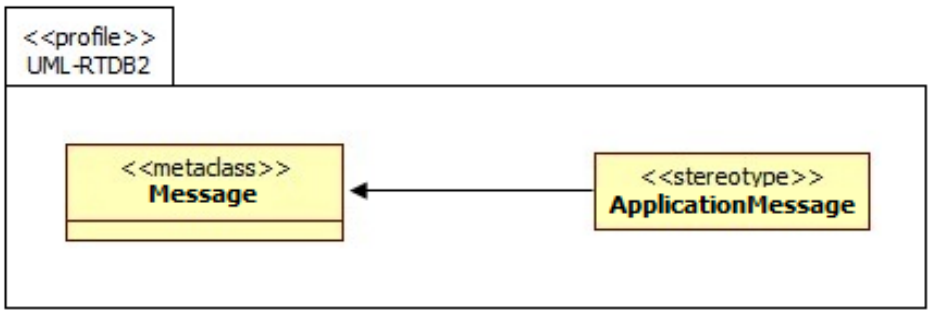


FIGURE 4.16 : Définition du stéréotype << *ApplicationMessage* >>.

Les stéréotypes << *ApplicationLifeline* >> et << *ApplicationMessage* >> ont pour objectif d'ajouter plus de sémantique aux éléments de l'instance du diagramme de séquence du patron afin de faciliter sa validation.

L'ensemble des stéréotypes du profil UML-RTDB2, que nous proposons pour la représentation des patrons de conception, ainsi que leurs relations avec les méta-classes d'UML 2.2 sont présentés dans la figure 4.17, qui représente la vue structurelle, et la figure 4.18, qui représente la vue dynamique.

4.5 Définition des contraintes OCL

La représentation de la variabilité permet d'obtenir des patrons flexibles et compréhensibles. Cependant, la définition des éléments optionnels et des points de variation peut être une source d'incohérence pour les modèles. Par exemple, lorsqu'une classe non optionnelle hérite d'une classe optionnelle, il est probable que le patron contienne une incohérence vu que les éléments sont dépendants les uns des autres. Il nous a paru donc nécessaire de vérifier pour chaque élément optionnel (respectivement obligatoire), que les éléments qui en

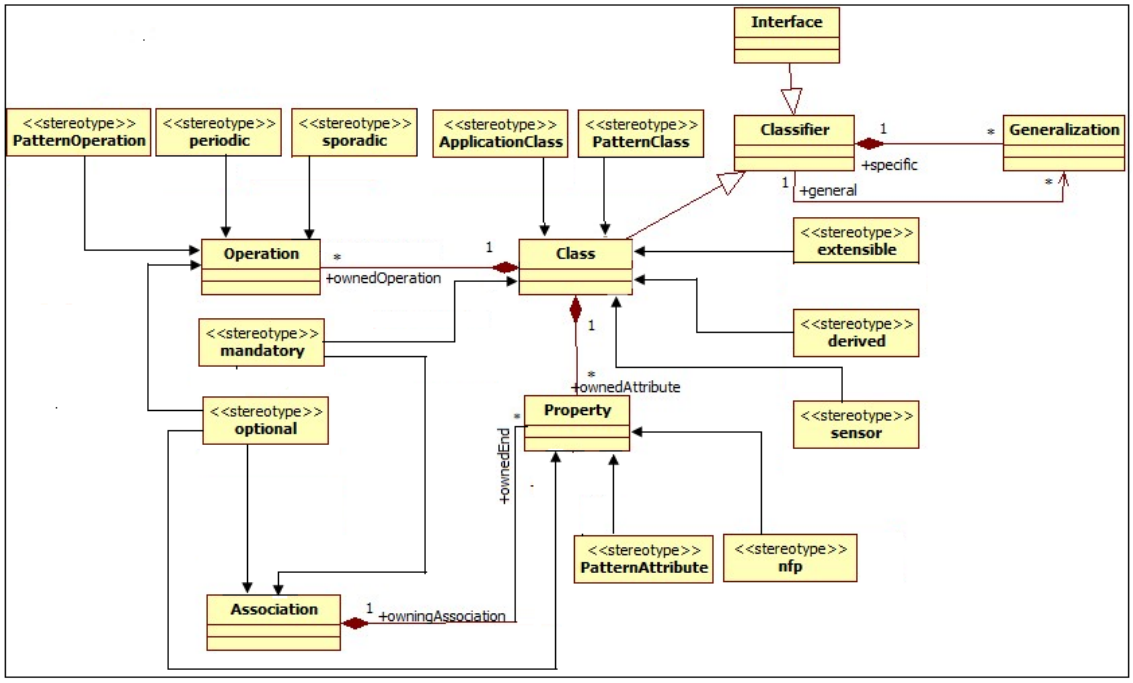


FIGURE 4.17 : Mécanisme d'extension de la vue statique.

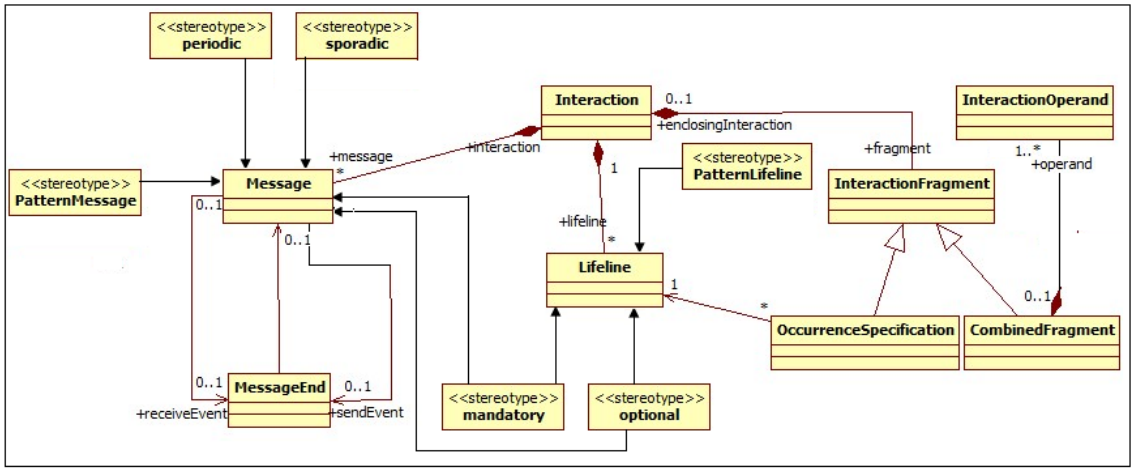


FIGURE 4.18 : Mécanisme d'extension de la vue dynamique.

dépendent sont également optionnels (respectivement obligatoires). Nous proposons pour cela de définir un ensemble de contraintes exprimées en langage OCL, permettant d'assurer une grande richesse et une grande extensibilité d'expression des contraintes. Ces contraintes permettent de décrire la cohérence entre les éléments de la vue statique de patrons, d'une part, et la cohérence entre les éléments de la vue dynamique de patrons, d'autre part. Pour

définir ces contraintes, nous nous basons sur les méta-modèles des diagrammes de classes et de séquences d'UML 2.2.

L'expression de ces contraintes OCL repose sur l'utilisation d'une opération auxiliaire *isStereotyped(S)* exprimée en OCL dans [Ziadi et al., 2003] :

```
Context Construct : :Class : :isStereotyped(S :string) :Boolean
isStereotyped = self.extensions − > exists(E | E.ownedEnd.type.name = S)
```

Cette opération indique qu'un élément E du méta-modèle est stéréotypé par le paramètre S.

Pour assurer la cohérence des éléments de la vue structurelle, nous appliquons les contraintes OCL suivantes définies dans [Rekhis et al., 2013a] :

- **Contrainte C1 :** Chaque association, liée à une classe optionnelle (*i.e.,* stéréotypée << *optional* >>), est optionnelle. Elle est exprimée en OCL comme suit :

```
Context Class inv :
self. isStereotyped ('optional') implies self. ownedAttribute − > forall(a |
a.owingAssociation.isStereotyped ('optional'))
```

- **Contrainte C2 :** Chaque classe qui hérite des caractéristiques (attributs et/ou opérations) d'une super-classe optionnelle doit être optionnelle. Dans le méta-modèle d'UML 2.2, une relation d'héritage est modélisée par la méta-classe *Generalization* qui est liée à la méta-classe *Classifier* par deux associations pour représenter la classe générique et la classe spécialisée de la généralisation. Cette contrainte est exprimée en OCL comme suit :

```
Context Classifier inv :
self.Generalization − > forAll (c | c.general.isStereotyped ('optional')) implies
self.isStereotyped ('optional')
```

Dans [Rekhis et al., 2013a], les auteurs définissent des contraintes OCL qui expriment la cohérence au niveau de la vue structurelle d'un patron, d'une part, et entre la vue structurelle

et la vue comportementale, d'autre part. Néanmoins, ils ne définissent pas de contraintes de dépendances entre les éléments variables du diagramme de séquence. Pour cette raison, nous proposons de définir des contraintes OCL qui permettent d'assurer la cohérence des éléments de la vue comportementale.

- **Contrainte C3** : Chaque message, associé à un objet optionnel, doit être optionnel. Cette contrainte est formalisée en OCL comme suit :

```
Context Lifeline inv :
self.isStereotyped ('optional') implies
self.interaction.message − > forAll (m | m.isStereotyped ('optional'))
```

- **Contrainte C4** : Chaque message, objet et fragment d'interaction, associés à une interaction optionnelle, doivent être optionnels. Cette contrainte est formalisée en OCL comme suit :

```
Context Interaction inv :
self.isStereotyped ('optional') implies
self.lifeline − > forAll (l | l.isStereotyped ('optional'))
and self.message − > forAll (m | m.isStereotyped ('optional'))
and self.fragment − > forAll (f | f.isStereotyped ('optional'))
```

- **Contrainte C5** : Chaque message, associé à un fragment d'interaction optionnel, doit être optionnel. Cette contrainte est formalisée en OCL comme suit :

```
Context InteractionFragment inv :
self.isStereotyped ('optional') implies
self.EnclosingInteraction.message − > forAll (m | m.isStereotyped ('optional'))
```

- **Contrainte C6** : Lorsqu'un message est obligatoire (*i.e.,* stéréotypé << *mandatory* >>), les deux objets liant ce message (objet émetteur et objet récepteur) doivent être obligatoires. De même, si ce message fait partie d'un fragment d'interaction, alors ce fragment doit être obligatoire. Cette contrainte est formalisée en OCL comme suit :

```
Context Message inv :
self.isStereotyped ('mandatory') implies
self.SendEvent.Message.Interaction.lifeline -> forAll (l | l.isStereotyped ('mandatory'))
and self.ReceiveEvent.Message.Interaction.lifeline -> forAll (l1 | l1.isStereotyped ('mandatory'))
and self.Interaction.InteractionFragment -> forAll (f | f.isStereotyped ('mandatory'))
```

4.6 Conclusion

Dans ce chapitre, nous avons présenté nos propositions dans la définition des extensions apportées aux concepts d'UML sous forme d'un profil UML, nommé UML-RTDB2 pour assurer une bonne modélisation des patrons. Ce dernier permet, d'une part, d'exprimer la variabilité et les aspects des bases de données temps réel et, d'autre part, d'identifier les éléments instanciés à partir de patrons de conception. Ce profil à été enrichi par des contraintes OCL dont certaines sont empruntées et d'autres sont proposées par nous mêmes. Ces contraintes permettent de vérifier la cohérence des patrons de conception que nous proposons. La modélisation de ces patrons fait l'objet du chapitre suivant.

Chapitre 5

Patrons de conception pour les systèmes d'aide à la conduite

5.1 Introduction

L'étude des patrons de conception TR (*e.g.,* ceux de [Gerdes et al., 2013] et [Sarika et al., 2013]) montre que les auteurs ne prennent pas en considération la modélisation des systèmes TR qui reposent sur l'utilisation de bases de données TR (BDTR). En effet, les applications actuelles, y compris les systèmes d'aide à la conduite sur lesquels portent nos travaux gèrent un grand volume de données. En outre, ces patrons ne s'intéressent pas à la modélisation des exigences des systèmes d'aide à la conduite. Toutes ces raisons nous amènent à définir dans ce chapitre des propositions visant à permettre une modélisation des patrons de conception dédiés aux systèmes d'aide à la conduite qui reposent sur l'utilisation d'une BDTR. En effet, ces patrons doivent permettre (i) de modéliser les données gérées par ces systèmes, (ii) d'exprimer les contraintes relatives aux données et aux traitements et (iii) de respecter la qualité de service exigée par ces systèmes.

La modélisation de ces patrons est une tâche difficile à cause de la complexité de l'identification des exigences du domaine. C'est pourquoi, nous avons besoin de suivre une démarche

méthodique pour la spécification des patrons de meilleure qualité. Dans ce sens, nous distinguons, d'une part, les processus qui se basent uniquement sur l'analyse du domaine [Arnaud, 2008] et, d'autre part, les processus qui se basent en plus sur l'analyse d'un ensemble d'applications de ce domaine [Rekhis et al., 2010a].

Les processus qui se basent uniquement sur l'analyse du domaine permettent d'identifier les éléments obligatoires et les éléments facultatifs dans un patron. Mais, ils ne procèdent pas à l'analyse des exigences fonctionnelles et non fonctionnelles du domaine malgré l'importance de ces exigences dans l'amélioration de la qualité des modèles générés. En revanche, les processus qui se basent simultanément sur l'analyse du domaine et sur les différentes applications de ce domaine permettent, d'une part, d'identifier les exigences fonctionnelles et non fonctionnelles et, d'autre part, de définir des règles pour expliquer comment déterminer les éléments communs entre les applications représentatives du domaine et les éléments qui varient d'une application à une autre. Cette nécessité est évidente pour le domaine d'aide à la conduite qui est caractérisé par une grande variété de systèmes. Pour cela, nous spécifions nos patrons en suivant les étapes décrites par le processus proposé par [Rekhis et al., 2010a] et en appliquant un ensemble de règles d'unification décrites dans [Rekhis et al., 2010a] et [Marouane et al., 2013].

Afin d'illustrer la modélisation de ces patrons de conception pour les systèmes d'aide à la conduite, nous étudions des exemples de ces systèmes, dont la description est donnée dans la section 5.2. La section 5.3 traite de la modélisation des patrons de conception pour les systèmes d'aide à la conduite.

5.2 Description des systèmes d'aide à la conduite

Les auteurs de [Roberts and Johnson, 1996] sont arrivés à la conclusion que trois applications sont suffisants pour représenter un domaine. Par conséquent, nous décrivons dans cette section les trois systèmes utilisant une base de données temps réel suivants : (1) le système régulateur de vitesse à contrôle de distance (Adaptive Cruise Control system : ACC) [Tass,

2012] représentatif des systèmes de contrôle longitudinal, (2) le système d'avertissement de sortie involontaire de voie (Lane Departure Warning system : LDW) [Tass, 2012] représentatif des systèmes de contrôle latéral et (3) le système SAFERIDER [Bekiaris et al., 2009] représentatif des systèmes de contrôle à la fois longitudinal et latéral.

5.2.1 Système de régulateur de vitesse à contrôle de distance

Le régulateur de vitesse adaptatif est un système d'assistance permettant une conduite plus sûre et plus confortable. L'ACC est conçu pour être utilisé notamment sur des autoroutes jusqu'à une vitesse approximative de 200 km/h. Il vise à maintenir la vitesse souhaitée par le conducteur et à régler la distance inter-véhiculaire. Ceci est possible grâce à deux types de capteurs actifs embarqués dans le véhicule : des capteurs radars et des capteurs proprioceptifs (capteur de vitesse et capteur de la vitesse de lacet). Les capteurs proprioceptifs permettent de mesurer les données relatives à l'état du véhicule. Quant aux capteurs radars, ils permettent de mesurer la distance ainsi que la vitesse du véhicule détecté. Ils sont de deux types :

- Capteur radar à longue portée (Long Range Radar : LRR), permettant de détecter la présence de véhicules situés devant à une distance allant jusqu'à 150 mètres et à un angle maximal de 9° du véhicule équipé d'ACC.
- Capteur radar à courte portée (Short Range Radar : SRR), permettant de détecter des véhicules qui se trouvent à une distance maximale d'environ 30 m, dans un champ allant jusqu'à 80°.

Chaque capteur acquiert des données toutes les 10 ms, avec une grande précision, et les transmet à une unité de fusion par le biais d'un bus CAN. Cette unité permet de traiter les données et de calculer la valeur de vitesse et la distance inter-véhiculaire par la fusion des données acquises par les capteurs radars.

Toute donnée transmise au contrôleur est sauvegardée dans la base de données. Chaque mesure, acquise par un capteur ou calculée, est caractérisée par une valeur, une unité, une durée de validité (*i.e.,* l'intervalle de temps pendant lequel la donnée est considérée valide) et une estampille (*i.e.,* l'instant de mise à jour de la donnée). Pour les données acquises, il

faut préciser l'erreur maximale (*i.e.,* une erreur maximale tolérée entre la valeur réelle et la valeur stockée dans la base). Cette erreur maximale détermine si la transaction qui souhaite mettre à jour la valeur d'une donnée doit être exécutée (la différence entre la valeur réelle et la valeur stockée dans la base est supérieure à un seuil prédéfini) ou pas (la différence entre la valeur réelle et la valeur stockée dans la base est inférieure au seuil).

À chaque fois qu'une donnée capteur est mise à jour, le contrôleur procède à l'analyse de la situation, en accédant à la base de données pour disposer de toutes les données récentes (sensorielles et dérivées). Une fois que le contrôleur identifie un changement de la situation, il détermine l'action appropriée selon le fait qu'il existe un danger d'un certain degré, ou qu'un danger disparait. En effet, le contrôleur calcule la valeur désirée de la décélération/accélération à donner au véhicule pour atteindre la vitesse convenable. Il envoie la valeur de décélération ou d'accélération au gestionnaire afin d'activer les composants d'interface homme-machine et/ou l'actionneur appropriés (*i.e.,* freins ou accélérateur). Si le contrôleur détecte que le véhicule qui précède roule à une vitesse inférieure à celle du véhicule équipée d'ACC, le système décélère automatiquement en activant l'actionneur automatique (*i.e.,* les freins) pour conserver la distance de sécurité prédéterminée. Si le véhicule qui précède roule avec une vitesse supérieure à celle du véhicule équipée d'ACC, alors le contrôleur envoie la valeur d'accélération appropriée au contrôleur du moteur pour accélérer le véhicule.

Lorsque l'intervention du conducteur est nécessaire, le gestionnaire active l'afficheur d'avertissement présent au niveau du tableau de bord pour générer une alerte visuelle afin que le conducteur prenne des actions correctives à temps. Lorsqu'il n'y a plus de véhicule détecté, la vitesse programmée (la vitesse de croisière) se rétablit automatiquement. À chaque alerte émise est associé un ensemble de données indiquant la durée, le taux de répétition et la priorité. Après avoir exécuté les actions suite à l'alerte, par le conducteur ou par les actionneurs automatiques (*i.e.,* les freins et l'accélérateur) qui ont un temps de réponse court, la vitesse du véhicule sera ajustée en conséquence. Chaque action (action prise par le conducteur ou action automatique) est caractérisée par une durée qui doit être respectée avant que le risque ne se produise. La figure 5.1 montre le diagramme de classes du système ACC.

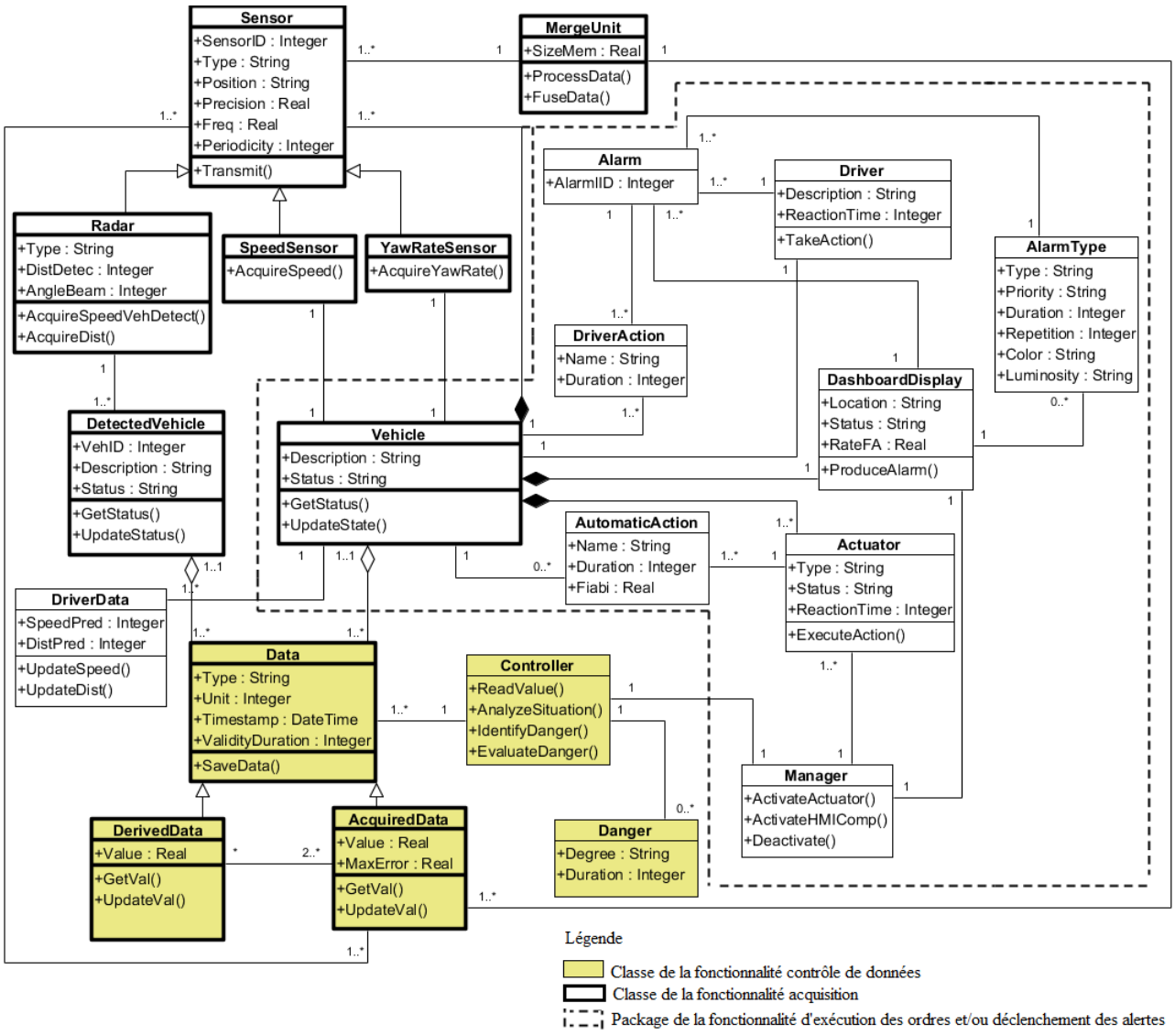


FIGURE 5.1 : Diagramme de classes du système ACC.

5.2.2 Système d'avertissement de sortie involontaire de voie

Le système d'avertissement de sortie involontaire de voie, ou LDW (Lane Departure Warning system), dont le diagramme de classes est représenté dans la figure 5.2, a pour objectif d'aider le conducteur à éviter les sorties involontaires de voie. Ce système est conçu pour être utilisé sur les autoroutes, et par conséquent, il est opérationnel à partir d'une vitesse de 60 km/h. Le système LDW utilise deux types de capteurs embarqués dans le véhicule :

des capteurs proprioceptifs et une caméra. Les capteurs proprioceptifs actifs permettent de fournir régulièrement les mesures relatives à l'état et la dynamique du véhicule. Quant à la caméra, elle permet de scruter périodiquement la voie devant le véhicule et de détecter les marquages au sol. En effet, cette caméra permet de capturer des images de la scène observée. Dans ce cas, le système doit être en mesure d'analyser le contenu des images acquises. Cette analyse est faite par une unité de fusion afin de reconnaître, à un certain niveau de précision, la position du véhicule et la position des marquages de voie. Cela va permettre de calculer la distance du véhicule par rapport à la ligne gauche et à la ligne droite de la voie. Le contrôleur compare cette distance à un seuil prédéfini (0.3 m) pour identifier le comportement du véhicule. Si le véhicule traverse les marquages au sol ($distance < 0.3$), le contrôleur ordonne au gestionnaire d'activer l'actionneur de direction et/ou les éléments d'interface homme-machine appropriés. Cet ordre dépend de la gravité et de la nature du risque identifié et évalué par le contrôleur. Si le contrôleur détecte une sortie involontaire de voie, alors il envoie la valeur à appliquer par l'actionneur de direction. Cependant, lorsque l'intervention du conducteur est nécessaire, le gestionnaire active l'afficheur localisé dans le tableau de bord pour alerter le conducteur par le biais de messages visuelles. Ces messages ont pour objectif d'informer le conducteur d'un danger potentiel pour qu'il prenne des actions correctives à temps afin de rétablir la situation normale. Chaque alerte est caractérisée par une durée, une couleur, une luminosité et un taux de répétition.

Toutes les données temps réel sont enregistrées périodiquement (toutes les 10 ms) dans une base de données. Chaque donnée (acquise ou calculée) est caractérisée par une valeur, une unité, une durée de validité et une estampille. Pour les données acquises, il faut préciser l'erreur maximale qui peut être tolérée. Une donnée acquise temps réel est mise à jour si la différence entre la valeur réelle et la valeur stockée dans la base est supérieure à l'erreur maximale.

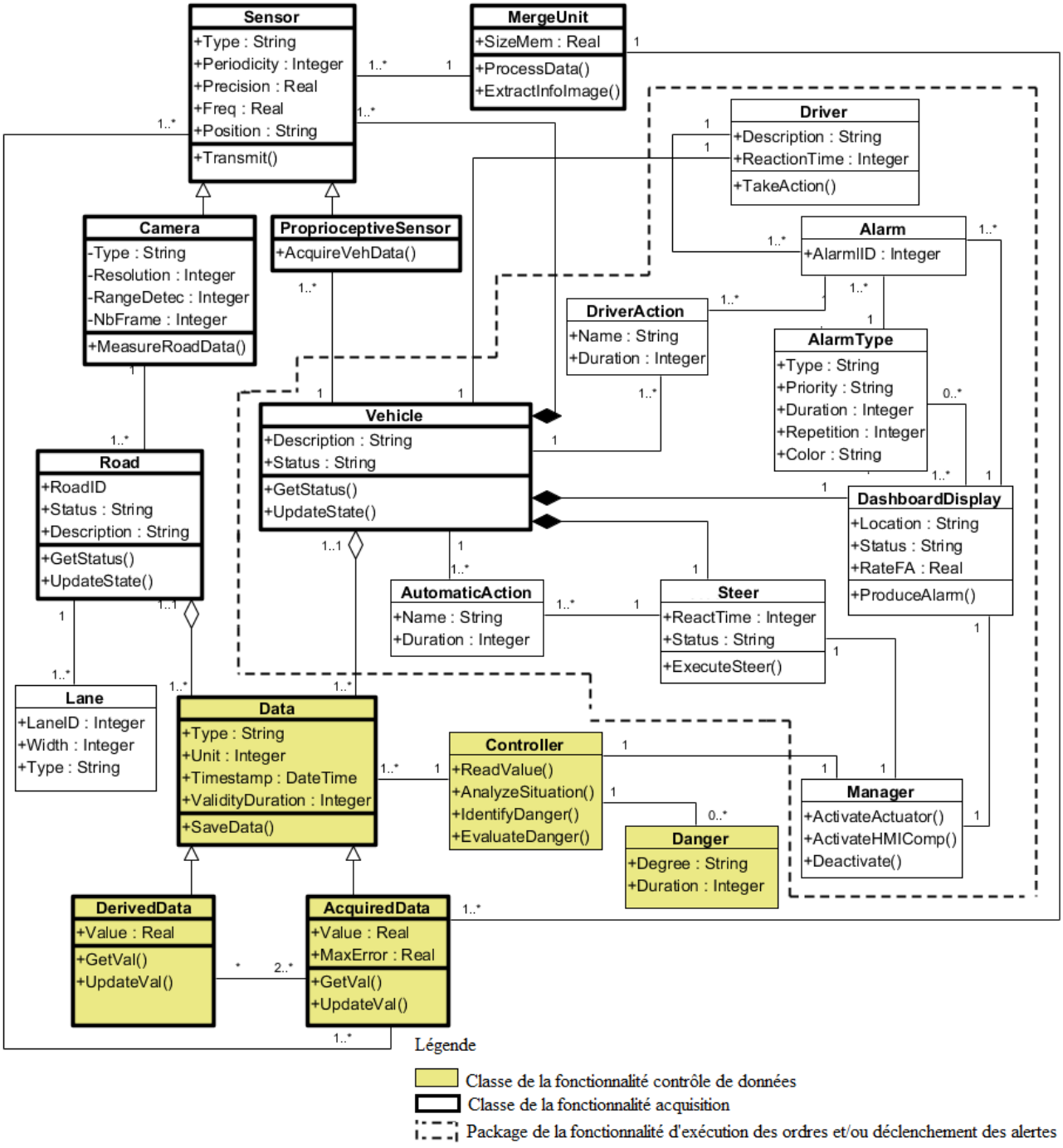


FIGURE 5.2 : Diagramme de classes du système LDW.

5.2.3 Système SAFERIDER

Le système SAFERIDER vise à fournir de l'aide à la conduite destinée aux motards afin d'améliorer leur sécurité et leur confort. Son principe de fonctionnement réside dans l'évaluation et l'analyse de la situation de conduite en prenant en compte des données relatives à la dynamique de la moto, aux caractéristiques de la route et à l'état des obstacles situés devant la moto. Ces données sont acquises périodiquement par des capteurs embarqués dans la moto. En effet, le système est composé par (i) des capteurs proprioceptifs actifs qui fournissent des données concernant l'état et la dynamique de la moto (*e.g.,* vitesse et position), (ii) un capteur laser actif qui fournit des données relatives aux objets détectés et (iii) une caméra passive qui fournit des images des routes. Les données acquises sont transmises périodiquement vers une unité de fusion qui permet de les analyser (*e.g.,* calcul de la vitesse de la moto à partir de la vitesse de chaque roue) et d'extraire à partir des images certaines informations, tels que le nombre de voies de la route et la géométrie de la route. Par la suite, elles sont stockées dans la base pour être utilisées par l'ACM (ARAS Control Module) pour analyser et évaluer la situation de conduite. Chaque mesure (*i.e.,* mesure sensorielle ou mesure dérivée) est caractérisée par un type, une valeur, une unité, l'instant de mise à jour de cette donnée, une durée de validité, une valeur minimale et une valeur maximale. Les mesures sensorielles (données temps réel acquises) sont caractérisées aussi par un maximum d'erreur (seuil) pour que la mise à jour de ce type de données soient exécutées uniquement si la différence entre la valeur réelle et la valeur stockée dans la base est supérieure à ce seuil fixé par le concepteur.

Lorsque le contrôleur identifie un risque, il évalue sa nature pour recommander des alertes à un gestionnaire. Ce dernier permet d'activer les composants d'interface homme-machine correspondants pour fournir les alertes au motard, qui doit réagir à temps pour éviter le danger. Après avoir exécuté la commande, la moto rétablit sa situation normale. Chaque action prise par le conducteur est caractérisée par une durée. En effet, ces composants permettent de générer des alertes (a) visuelles (textes et symboles) à travers un afficheur graphique intégré au tableau de bord avec différentes couleurs et tailles selon le degré de gravité du risque, (b) haptiques (au moyen des vibrations au niveau du siège et au niveau des gants) et/ou (c) audio

par le biais d'un haut-parleur intégré dans la casque. Chaque alerte est caractérisée par un niveau de gravité, une priorité, un taux de répétition et une durée. Le contrôleur déclenche ces alertes dans les conditions suivantes :

- Si la vitesse de la moto dépasse la vitesse limite.
- Si le système détecte la présence d'un obstacle devant la moto et la distance entre cet obstacle et la moto est inférieure à une distance de sécurité.
- Si le système détecte un risque de collisions avec des obstacles fixes et mobiles sur les intersections.
- Si le système identifie un changement de voie involontaire, et par conséquent un risque de collision.

La figure 5.3 représente le diagramme de classes du système SAFERIDER.

5.3 Modélisation des patrons de conception

Dans cette section, nous nous sommes intéressés à la création des patrons de conception dédiés à la modélisation des systèmes d'aide à la conduite reposant sur l'utilisation d'une base de données temps réel. Pour créer les solutions de ces patrons en termes de vue statique (diagramme de classes) et de vue dynamique (diagramme de séquence), nous adoptons le processus défini dans [Rekhis et al., 2010a] pour guider le concepteur dans la création des patrons de conception destinés à la modélisation des systèmes d'aide à la conduite, mais avec de nouvelles règles que nous proposons. Nous décrivons, dans la suite, ce processus.

5.3.1 Processus de création de patrons de conception

Dans ce paragraphe, nous détaillons le processus proposé dans [Rekhis et al., 2010a], qui permet de guider les concepteurs dans la création des patrons de conception TR. En effet, ce processus est constitué des étapes suivantes :

- **Étape 1 :** Elle consiste à étudier différents systèmes appartenant au domaine de l'aide à la conduite pour identifier les différentes fonctionnalités de ce domaine. Cette

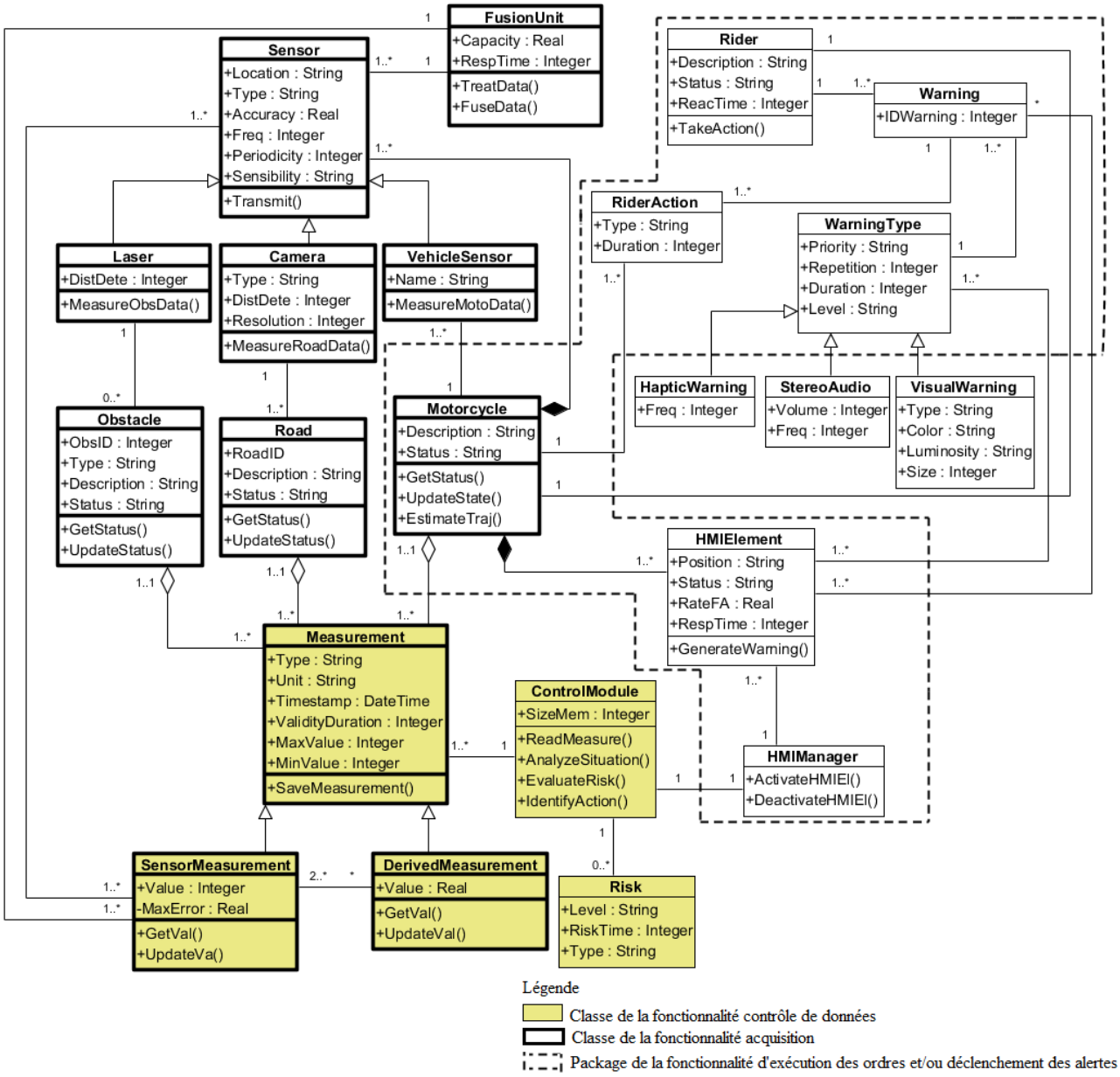


FIGURE 5.3 : Diagramme de classes du système SAFERIDER.

étude nous a permis de dégager trois principales fonctionnalités. La première concerne l'acquisition des données relatives à un véhicule et à son environnement extérieur à l'aide des capteurs. La deuxième consiste en l'analyse de ces données, l'évaluation des situations de conduite et la détermination des actions appropriées. Quant à la troisième fonctionnalité, elle correspond à l'exécution des ordres de commandes par

des actionneurs et/ou au déclenchement des alertes par le biais des composants de l'interface homme-machine. Nous modélisons pour chaque fonctionnalité un patron qui capture ces connaissances.

- **Étape 2 :** Elle correspond à l'identification des contraintes et des concepts (les objets et leurs caractéristiques) relatifs à chaque fonctionnalité identifiée dans l'étape précédente. À titre d'exemple, le capteur, les objets observés, l'unité de fusion et les mesures représentent les concepts relatifs à la fonctionnalité d'acquisition des données. L'identification de ces concepts et des contraintes se base sur la décomposition de chaque fonctionnalité en fonctions élémentaires. Par exemple, le raffinement de la fonctionnalité d'acquisition des données nous amène à distinguer une fonction élémentaire qui analyse un fait (*des capteurs proprioceptifs acquièrent périodiquement des mesures relatives à l'état et à la dynamique du véhicule à contrôler*) et donne des consignes pour faciliter l'identification des concepts associés à cette fonctionnalité : *Capteur proprioceptif*, *Mesure* et *Véhicule*. De plus, cette fonction élémentaire permet de déterminer les mesures qui doivent être rafraîchies périodiquement. L'identification des concepts est effectuée en examinant les « noms » utilisés dans les fonctions élémentaires.
- **Étape 3 :** Elle concerne la modélisation d'une application en fragments (*e.g.,* diagrammes de classes) dont chacun représente une fonctionnalité identifiée à l'étape 1. Cette modélisation est facilitée par les concepts déjà identifiés dans l'étape précédente. En effet, pour une application $A(i)$, chaque classe jouant le rôle d'un concept de domaine est ajoutée au fragment de cette application. Par exemple, une classe jouant le rôle du concept *contrôleur* est associée au fragment de l'application relatif à la fonctionnalité de contrôle de données. Après avoir ajouté toutes les classes jouant les rôles des concepts de domaine, d'autres classes qui leur sont reliées peuvent être ajoutées au fragment par le concepteur.

  Par ailleurs, l'identification du diagramme de séquence relatif à une fonctionnalité est basée sur l'application de la règle suivante : si un objet du diagramme de séquence d'une application correspond à une instance d'une classe du diagramme de classes de cette application, alors cet objet est transféré au diagramme de séquence. De plus, tous

les messages ainsi que les objets en relation avec cet objet sont également transférés vers ce diagramme de séquence.

- **Étape 4 :** Elle consiste à unifier les fragments d'applications d'une même fonctionnalité en se basant sur un ensemble de règles d'unification. En effet, il s'agit de combiner les éléments des fragments (*e.g.,* classes, attributs, opérations, objets et messages) de façon à obtenir un modèle complet qui représente un patron de conception. Nous adoptons pour cela, les règles proposées dans [Rekhis et al., 2010a] pour la construction d'un diagramme de classes complet (cf. Annexe A) et les règles que nous proposons dans [Marouane et al., 2013] pour la construction d'un diagramme de séquence complet (cf. paragraphe 5.3.2.2), constituant les solutions d'un patron de conception. Nous avons proposé les règles d'unification pour la création des diagrammes de séquence vu que Rekhis et *al.* ne se sont intéressés qu'à la proposition des règles pour la construction des diagrammes de classes. L'unification permet d'identifier les éléments communs et les éléments variables entre les fragments relatifs à une application qui constituent respectivement des éléments obligatoires et des éléments optionnels dans les solutions d'un patron (diagramme de classes et diagramme de séquence). Pour déterminer les correspondances sémantiques, nous proposons un nouveau dictionnaire de domaine qui contient pour chaque élément de conception (*e.g.,* nom d'une classe, nom d'un attribut et nom d'une opération) ses synonymes, ses antonymes ou ses variations. Ce dictionnaire peut être enrichi d'une manière semi-automatique par le concepteur de patrons pour déterminer d'autres relations linguistiques entre paires de termes (*e.g.,* nom d'une classe et nom d'un attribut). Par exemple, le concepteur peut introduire une relation de synonymie entre deux termes dans le dictionnaire.

  Pour faciliter la détermination de la relation linguistique entre deux termes, Rekhis *et al.* ont proposé dans [Rekhis et al., 2010a] une étape de pré-traitement des termes. En effet, si le nom d'un élément (classe, attribut, etc.) est formé de plusieurs mots dont la première lettre est en majuscule, les auteurs de [Rekhis et al., 2010a] ont proposé de traiter chaque mot à part afin de vérifier s'il existe une relation linguistique entre ce mot et un mot d'un autre élément. Par exemple, la décomposition des noms de classes

DerivedData et *CalculatedData* permet de vérifier qu'il y a deux mots synonymes (*Derived* et *Calculated*).

Dans la suite de ce chapitre, nous allons illustrer les étapes du processus de construction des diagrammes de classes et de séquence de chaque patron associé à une fonctionnalité du domaine. Ces patrons sont décrits en utilisant les rubriques suivantes du formalisme P-SIGMA [Conte et al., 2001] :

- Nom : pour préciser le nom du patron.
- Problème : pour indiquer le problème à résoudre par le patron.
- Contexte : pour détailler les situations d'utilisation du patron.
- Solution : pour présenter les diagrammes de classes et les diagrammes de séquence afin de modéliser respectivement la vue statique et la vue dynamique d'un patron.

5.3.2 Construction du patron *"Acquisition des données"*

Le processus de création de patrons est illustré à travers la construction de la vue statique (cf. paragraphe 5.3.2.1) et de la vue dynamique (cf. paragraphe 5.3.2.2) du premier patron, appelé *"Acquisition des données"*, qui concerne la modélisation des exigences de la fonctionnalité d'acquisition des données TR à l'aide des capteurs.

5.3.2.1 Construction du diagramme de classes

Comme nous l'avons mentionné dans le paragraphe 5.3.1, l'étude des applications représentatives du domaine de l'aide à la conduite nous a permis d'identifier une première fonctionnalité qui correspond à l'acquisition des données TR. Après avoir identifié cette fonctionnalité, une étape de raffinement est nécessaire pour déterminer les concepts et les contraintes associés à cette fonctionnalité. Le raffinement a permis de décomposer cette fonctionnalité en cinq fonctions élémentaires, comme illustré dans le tableau 5.1. En outre, nous présentons dans ce tableau les concepts et les contraintes associés à chacune de ces fonctions.

Tableau 5.1 : Fonctions élémentaires, concepts et contraintes de la fonctionnalité d'acquisition des données.

Fonctions élémentaires	**Concepts**	**Contraintes**
F1 : Des capteurs proprioceptifs acquièrent périodiquement des mesures relatives à l'état du véhicule à contrôler.	Capteur proprioceptif, Mesure, Véhicule contrôlé	Les mesures doivent être rafraîchies périodiquement.
F2 : Des capteurs extéroceptifs acquièrent périodiquement des mesures relatives à l'état de l'environnement extérieur du véhicule.	Capteur Extéroceptif, Mesure, Environnement extérieur	Les mesures doivent être rafraîchies périodiquement.
F3 : Des capteurs transmettent périodiquement les mesures à une unité de fusion.	Capteur, Mesure, Unité de fusion	Les mesures sont transmises périodiquement.
F4 : L'unité de fusion traite les mesures acquises et calculent des mesures à partir des mesures acquises.	Unité de fusion, Mesure acquise, Mesure calculée	Le traitement sur les données doit être effectué avant que la durée de validité de chaque donnée expire.
F5 : L'unité de fusion transmet périodiquement les mesures pour qu'elles soient stockées.	Unité de fusion, Mesure	Les mesures doivent être rafraîchies périodiquement.

Après avoir identifié les concepts relatifs à la fonctionnalité d'acquisition des données TR, nous déterminons les fragments d'applications associés à cette fonctionnalité en précisant les classes jouant les rôles de *Capteur*, *Capteur proprioceptif*, *Capteur extéroceptif*, *Unité de fusion*, *Mesure*, *Mesure acquise*, *Mesure calculée*, *Véhicule contrôlé* et *Environnement extérieur* (Les classes en gras dans les figures 5.1, 5.2 et 5.3). De plus, les classes qui ne sont liées à aucun concept de domaine et qui ont des relations avec les classes représentant des concepts de domaine sont ajoutées aux fragments de l'application. Les fragments sont les suivants :

- Fragment-Cl(ACC) = {Sensor, SpeedSensor, YawRateSensor, Radar, Vehicle, DetectedVehicle, Data, AcquiredData, DerivedData, MergeUnit} ;
- Fragment-Cl(LDW) = {Sensor, ProprioceptiveSensor, Camera, Vehicle, Road, Data, AcquiredData, DerivedData, MergeUnit, Lane} ;

- Fragment-Cl(SAFERIDER) = {Sensor, VehicleSensor, Camera, Laser, Motorcycle, Road, Obstacle, Measurement, SensorMeasurement, DerivedMeasurement, FusionUnit}.

Une fois que les fragments d'applications relatifs à la fonctionnalité d'acquisition sont déterminées, nous générons le diagramme de classes du patron *"Acquisition des données"* modélisant la vue statique en utilisant les règles d'unification illustrées dans le tableau A.3 présenté en Annexe. Ces règles se basent sur une comparaison entre les noms des éléments (classes, attributs et opérations) qui utilise des relations linguistiques comme le montre le tableau A.1 présenté en Annexe A.

La modélisation de la vue statique du patron *"Acquisition des données"*, présentée dans la figure 5.4, est guidée par l'application des règles suivantes :

- N_equiv(MergeUnit(ACC), MergeUnit(LDW), FusionUnit(SAFERIDER)), Att_equiv (MergeUnit(ACC), MergeUnit(LDW), FusionUnit(SAFERIDER)) et Op_equiv(MergeUnit(ACC), MergeUnit(LDW), FusionUnit(SAFERIDER)) : la règle RC1 est appliquée. Ainsi la classe *FusionUnit* est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>.
- La règle RC2 est appliquée, ce qui permet d'ajouter la méthode *FuseData* relative à la classe *FusionUnit* en tant qu'opération stéréotypée << *optional* >>.
- N_equiv(AcquiredData(ACC), AcquiredData(LDW), SensorMeasurement(SAFERIDER)), Att_equiv(AcquiredData(ACC), AcquiredData(LDW), SensorMeasurement(SAFERIDER)) et Op_equiv(AcquiredData(ACC), AcquiredData(LDW), SensorMeasurement(SAFERIDER)) : la règle RC1 est appliquée. En appliquant cette règle, une classe fondamentale *SensorData* est ajoutée au patron stéréotypée << *mandatory* >>.
- N_equiv(DerivedData(ACC), DerivedData(LDW), DerivedMeasurement(SAFERIDER)), Att_equiv(DerivedData(ACC), DerivedData(LDW), DerivedMeasurement(SAFERIDER)) et Op_equiv(DerivedData(ACC), DerivedData(LDW), DerivedMeasurement (SAFERIDER)) : la règle RC1 est appliquée. Ainsi la classe *DerivedData* est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>.

- Att_conf(AcquiredData(ACC), AcquiredData(LDW), SensorMeasurement(SAFERIDER)) : la règle RC3 est appliquée. Les classes *AcquiredData(ACC)*, *AcquiredData(LDW)* et *SensorMeasurement(SAFERIDER)* ont en commun l'attribut *Value* qui peut être de différents types compatibles (*Integer* et *Real*). Par conséquent, cet attribut est ajouté au patron avec le type *String*. La même règle s'applique à l'attribut associé à la classe *DerivedData*.
- N_dist(Data(ACC), Data(LDW), Measurement(SAFERIDER)), Att_equiv(Data(ACC), Data(LDW), Measurement(SAFERIDER)) et Op_equiv(Data(ACC), Data(LDW), Measurement(SAFERIDER)) : la règle RC4 est appliquée car il n'existe pas de relation linguistique unique pour les noms des classes. En effet, certains sont équivalents (Data(ACC), Data(LDW)) et d'autres sont distincts (Data(ACC), Measurement(SAFERIDER)), mais ces classes sont associées au même concept de domaine : *Mesure*. Ainsi, une classe fondamentale *DBMeasurement* est ajoutée au patron stéréotypée << *mandatory* >>. Cette classe possède les attributs fondamentaux : *Type*, *Unit*, *Timestamp* et *ValidityDuration* et une opération fondamentale *StoreData()* qui permet de stocker les données dans une base. Les attributs *MinValue* et *MaxValue* sont spécifiques au système SAFERIDER et ne sont pas pertinents pour le domaine. Ils ne sont donc pas ajoutés au patron, mais la classe *DBMeasurement* est stéréotypée << *extensible* >>.
- N_dist(SpeedSensor(ACC), YawRateSensor(ACC), ProprioceptiveSensor(LDW), VehicleSensor(SAFERIDER)) et Op_equiv(SpeedSensor(ACC), YawRateSensor(ACC), ProprioceptiveSensor(LDW), VehicleSensor(SAFERIDER)) : la règle RC4 est appliquée. Les classes *SpeedSensor*, *YawRateSensor*, *ProprioceptiveSensor* et *VehicleSensor* jouent le même rôle qu'un capteur proprioceptif. Ainsi, une classe stéréotypée << *mandatory* >> ayant un nom relatif au rôle joué par les classes d'applications est ajoutée au patron : *ProprioceptiveSensor*.
- N_dist(Radar(ACC), Camera(LDW, SAFERIDER), Laser(SAFERIDER)), Att_int (Radar(ACC), Camera(LDW, SAFERIDER), Laser(SAFERIDER)) et Op_equiv(Radar(ACC), Camera(LDW, SAFERIDER), Laser(SAFERIDER)) : la règle RC4 est appliquée. Les classes *Radar*, *Camera* et *Laser* jouent le même rôle qu'un capteur

extéroceptif. Ainsi, la classe *ExteroceptiveSensor* est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>. Elle possède un attribut fondamental : *DistanceRange* et une opération fondamentale *MeasureDataElement()* qui permet d'acquérir les mesures relatives à l'objet observé. Cette classe est stéréotypée << *extensible* >> car il existe d'autres attributs qui sont spécifiques aux applications (*e.g.,* l'attribut *AngleBeam* associé au système ACC et qui n'est pas pertinent pour le domaine de l'aide à la conduite).

- N_equiv(Sensor(ACC), Sensor(LDW), Sensor(SAFERIDER)), Att_equiv(Sensor(ACC), Sensor(LDW), Sensor(SAFERIDER)) et Op_equiv(Sensor(ACC), Sensor(LDW), Sensor(SAFERIDER)) : la règle RC1 est appliquée et une classe nommée *Sensor* est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>.
- N_dist(Vehicle(ACC), Vehicle(LDW), Motorcycle(SAFERIDER)), Att_int(Vehicle(ACC), Vehicle(LDW), Motorcycle(SAFERIDER)) : la règle RC4 est appliquée et une classe fondamentale est ajoutée au patron stéréotypée << *mandatory* >> car les classes *Vehicle* et *Motorcycle* jouent le même rôle, qui est le *Véhicule contrôlé*. Le nom de cette classe fondamentale est relatif au rôle joué par ces classes d'applications : *ControlledVehicle*. La même règle RC4 est appliquée pour les classes *DetectedVehicle*, *Road* et *Obstacle* et une classe fondamentale ayant un nom relatif au rôle joué par ces classes : *TrackedElement*.
- La règle RC5 est appliquée, ce qui permet de maintenir dans le patron toutes les relations entre les classes transférées.

5.3.2.2 Construction du diagramme de séquence

Pour modéliser le diagramme de séquence du patron *"Acquisition des données"*, nous commençons par présenter les diagrammes de séquence de chaque application, associés à la fonctionnalité d'acquisition des données temps réel et illustrés par les figures 5.5, 5.6 et 5.7.

Par la suite, nous appliquons les règles d'unification que nous avons proposées [Marouane et al., 2013]. Ces règles se basent sur une comparaison entre les noms des éléments (objets

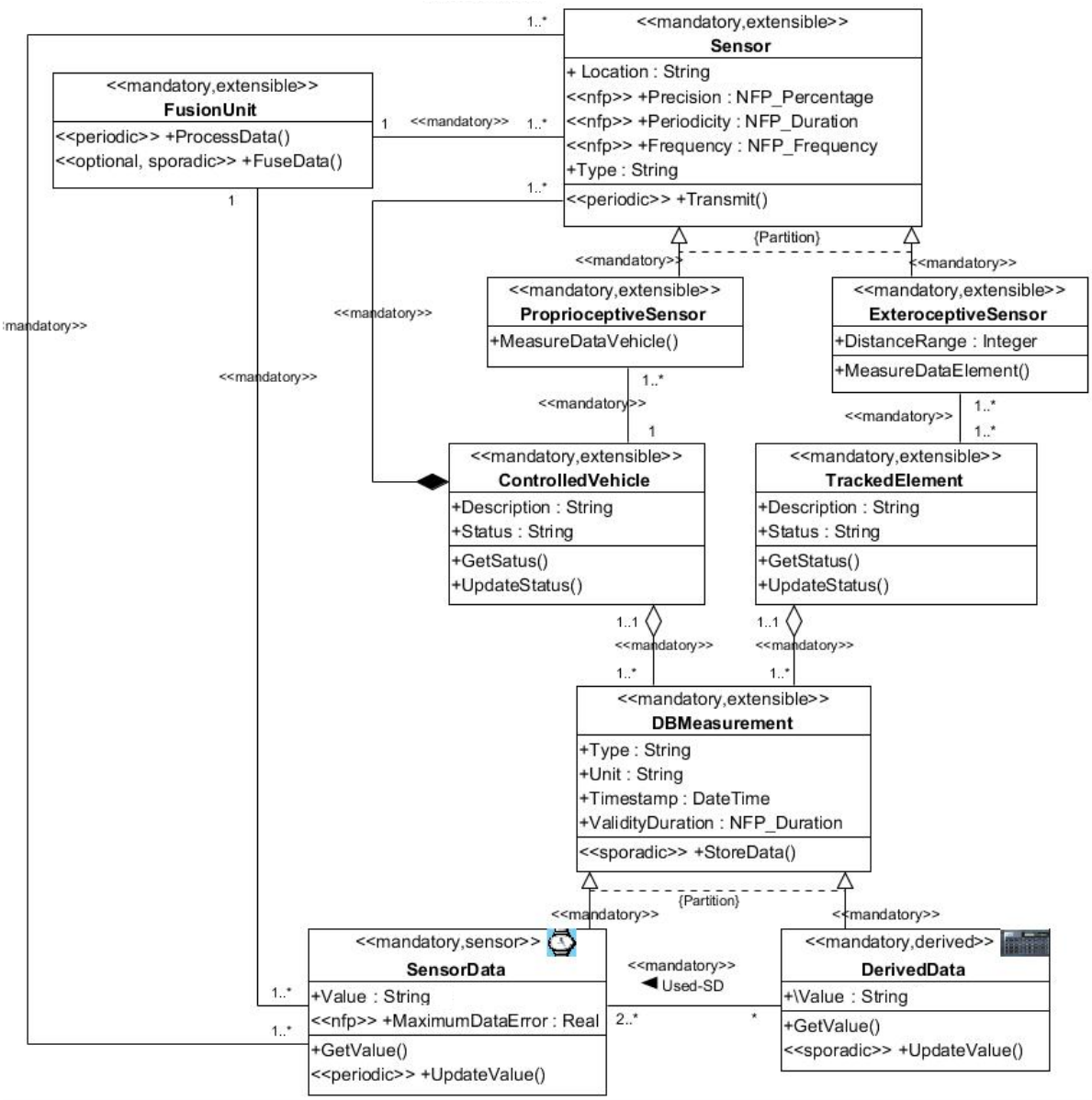


FIGURE 5.4 : Diagramme de classes du patron *"Acquisition des données"*.

et messages) en utilisant des relations linguistiques comme le montre le tableau A.2 présenté en Annexe A. Les règles d'unification que nous avons proposées sont :

- **RS1 :** S'il existe un ensemble d'objets $\{O(Ai),...,O(An)\}$ qui ont des noms équivalents ou synonymes, alors un objet fondamental stéréotypé << *mandatory* >> est ajouté au patron.

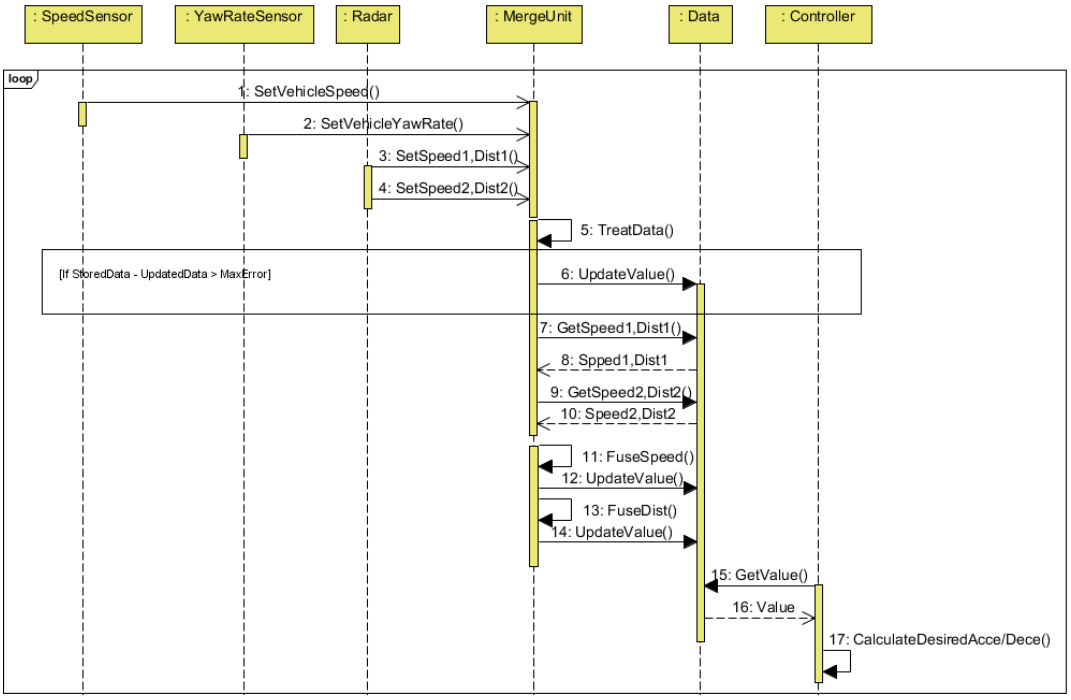


FIGURE 5.5 : Diagramme de séquence relatif à la fonctionnalité d'acquisition des données du système ACC.

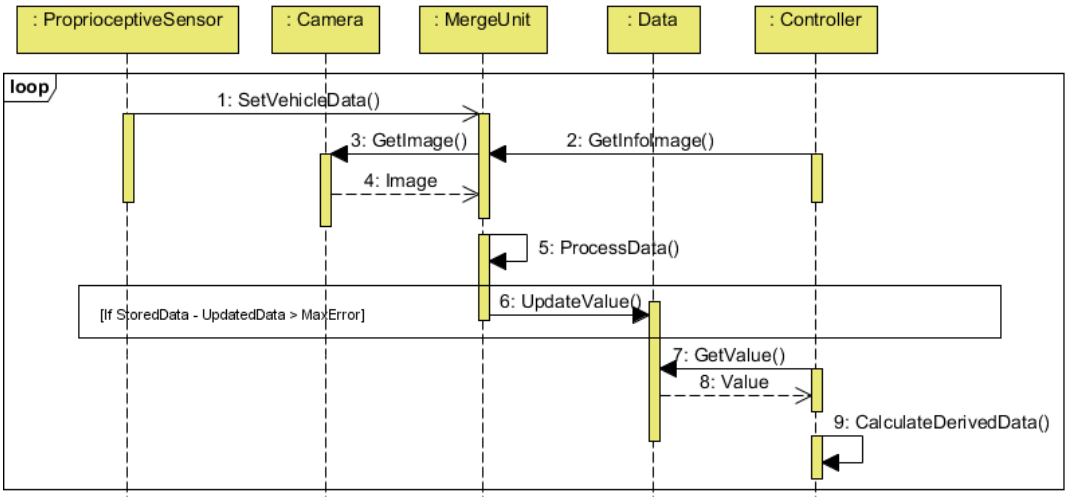


FIGURE 5.6 : Diagramme de séquence relatif à la fonctionnalité d'acquisition des données du système LDW.

- **RS2 :** S'il existe un ensemble d'objets $\{O(Ai),...,O(An)\}$ qui ont des noms distincts mais qui ont des rôles similaires à un même concept de domaine, alors un objet fondamental stéréotypé << *mandatory* >> ayant le nom du concept de domaine est

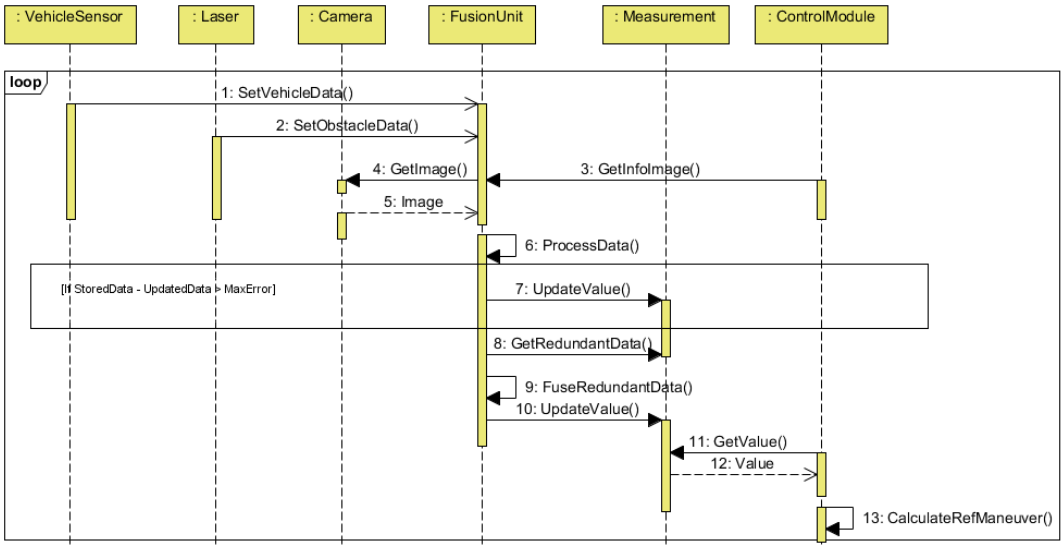


FIGURE 5.7 : Diagramme de séquence relatif à la fonctionnalité d'acquisition des données du système SAFERIDER.

ajouté au patron. À titre exemple, les objets *radar* et *caméra* ont des noms distincts, mais ils jouent le même rôle : *Capteur*. Un objet fondamental avec le nom *Sensor* est alors ajouté au patron.

- **RS3 :** S'il existe un ensemble de messages $\{M(Ai), ..., M(An)\}$ qui ont des noms équivalents ou synonymes, des objets émetteurs (N_equiv$(O_e(Ai), ..., O_e(An))$) équivalents et fondamentaux, et des objets receveurs équivalents (N_equiv$(O_r(Ai), ..., O_r(An))$) et fondamentaux, alors le message est ajouté en tant que message fondamental stéréotypé $<<\ mandatory\ >>$.
- **RS4 :** S'il existe un ensemble de messages $\{M(Ai), ..., M(An)\}$, pertinents pour le domaine, qui ne sont pas communs à toutes les applications et qui ont des noms équivalents (N_equiv$(M(Ai), ..., M(An))$), et si leurs objets émetteurs (N_equiv$(O_e(Ai), ..., O_e(An))$) sont équivalents et fondamentaux, et leurs objets receveurs sont équivalents (N_equiv$(O_r(Ai), ..., O_r(An))$), alors un message optionnel est ajouté au diagramme de séquence stéréotypé $<<\ optional\ >>$. À titre exemple, le message *FuseSpeed()* (cf. figure 5.5) et le message *FuseRedundantData()* (cf. figure 5.7) sont relatifs au même message *FuseData()*. Ces deux messages ne sont présents que dans quelques applications (ratio

de couverture est supérieure à $\frac{1}{2}$). Un message optionnel avec le nom *FuseData()* est alors ajouté au diagramme de séquence du patron.

- **RS5 :** S'il existe un ensemble de messages {$M(Ai)$, ..., $M(An)$} qui ont des objets émetteurs ou des objets receveurs qui constituent des variations d'un concept général, alors les messages sont ajoutés au patron dans un fragment combiné ayant l'opérateur alternative *alt*.
- **RS6 :** S'il existe un ensemble de messages {$M(Ai)$, ..., $M(An)$} ajoutés au patron, alors le fragment associé à ces messages est aussi ajouté au patron.

La modélisation du diagramme de séquence présenté dans la figure 5.8 repose sur l'application des règles précédentes :

- N_equiv(MergeUnit(ACC), MergeUnit(LDW), FusionUnit(Saferifer)) : la règle RS1 est appliquée et un objet *FusionUnit* est ajouté au patron en tant qu'objet stéréotypé << *mandatory* >>. La même règle est appliquée pour l'ensemble d'objets {Controller(ACC), Controller(LDW), ControlModule(SAFERIDER)}, et par conséquent, un objet *Controller* est ajouté au patron en tant qu'objet stéréotypé << *mandatory* >>.
- N_dist(Data(ACC), Data(LDW), Measurement(SAFERIDER)) : la règle RS2 est appliquée. Ainsi, l'objet *DBMeasuremet* est ajouté au patron en tant qu'objet fondamental stéréotypé << *mandatory* >>.
- N_dist(Radar(ACC), SpeedSensor(ACC), YawRateSensor(ACC), Camera(LDW), ProprioceptiveSensor(LDW), VehicleSensor(SAFERIDER), Camera(SAFERIDER), Laser
  (SAFERIDER)) : la règle RS2 est appliquée et un objet *Sensor* fondamental stéréotypé << *mandatory* >> est ajouté au patron. Certains capteurs (*e.g., Radar*, *SpeedSensor* et *YawRateSensor*) sont de type actif, alors que d'autres correspondent à des capteurs passifs (*e.g.,* camera).
- La règle RS3 est appliquée, ce qui permet d'ajouter les messages *ProcessData*, *UpdateValue*, *GetValue* et *CalculateDerivedData* au diagramme de séquence en tant que messages fondamentaux stéréotypés << *mandatory* >>.
- La règle RS4 est appliquée, ce qui permet d'ajouter les messages *GetValue*, *FuseData*, *UpdateValue* en tant que messages optionnels stéréotypés << *optional* >>.

- La règle RS5 est appliquée, ce qui permet d'ajouter les messages *GetValue* et *SetValue* dans un fragment combiné ayant l'opérateur *alt*. Cela est dû principalement au fait que les objets représentant des capteurs actifs émettent le message *SetValue*, alors que les objets de types capteurs passifs reçoivent le message *GetValue*. Les objets émetteurs du message *SetValue* et les objets receveurs du message *GetValue* représentent des variantes (jouent le même rôle d'un capteur) du concept général *Sensor*.
- La règle RS6 est appliquée pour transférer les fragments des messages ajoutés vers le patron.

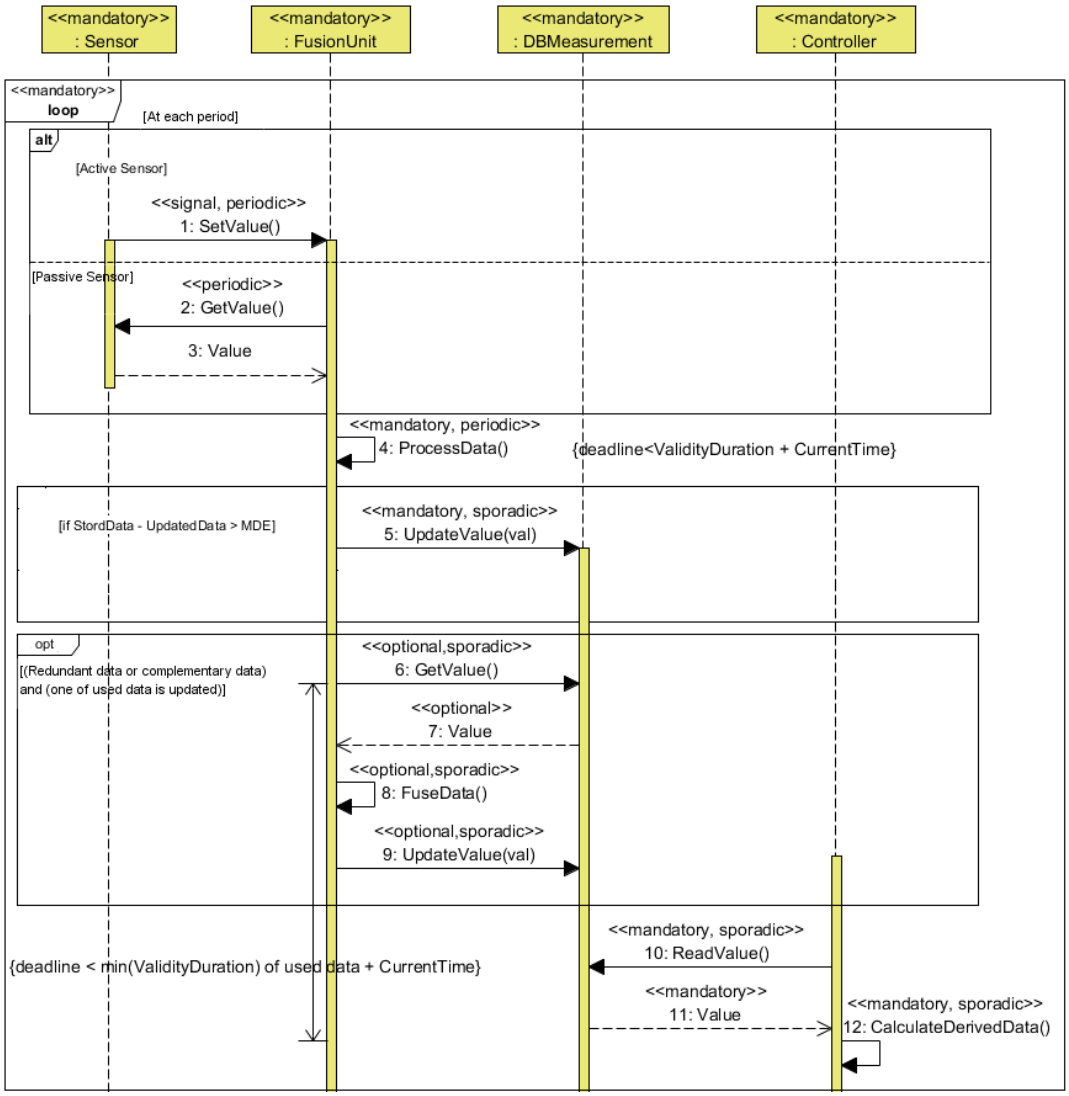


FIGURE 5.8 : Diagramme de séquence du patron *"Acquisition des données"*.

5.3.2.3 Description du patron *"Acquisition des données"*

Nous décrivons à ce niveau le patron *"Acquisition des données"* selon des rubriques retenues du formalisme P-SIGMA [Conte et al., 2001] comme suit :

1. **Nom :** *"Acquisition des données"*, appelé en anglais ADAS-DA (Advanced Driver Assistance Systems Data Acquisition).
2. **Contexte :** Ce patron est appliqué lors de la phase d'acquisition des données TR des systèmes d'aide à la conduite reposant sur l'utilisation des bases de données temps réel.
3. **Problème :** Les systèmes d'aide à la conduite existants utilisent un grand volume de données, ce qui rend leur gestion de plus en plus complexe. Par conséquent, il est difficile de garantir les contraintes temporelles relatives à ces données et aux transactions qui s'exécutent sur ces données. Comment les systèmes d'aide à la conduite peuvent-ils gérer d'une manière efficace un grand volume de données ? Comment peuvent-ils maintenir les contraintes TR ?
4. **Solution :** Les systèmes d'aide à la conduite nécessitent l'utilisation d'un système de base de données temps réel pour gérer un grand volume de données et garantir le respect de leurs contraintes temporelles (durée de validité des données). Un système de base de données temps réel permet d'améliorer la fiabilité, la sécurité et les performances de ces systèmes (cf. chapitre 3).

   Vue Statique : La figure 5.4 présente le diagramme de classes du patron *"Acquisition des données"* représentant les différents participants :

   - *Sensor.* Les capteurs sont classés en deux catégories selon les éléments observés : capteurs extéroceptifs et capteurs proprioceptifs qui constituent des variantes de la classe générale *Sensor*. Cette variation est exprimée par le biais d'une relation d'héritage. Un capteur extéroceptif acquiert des mesures relatives à l'état de l'environnement du véhicule à contrôler (*e.g.,* route et obstacles) via la méthode *MeasureDataElement()* alors qu'un capteur proprioceptif fournit des mesures relatives à l'état du véhicule à travers la méthode *MeasureDataVehicle()*. Ces capteurs transmettent périodiquement les mesures via un bus CAN.

Nous associons alors le stéréotype << *periodic* >> à la méthode *Transmit()* de la classe *Sensor*. Cette classe possède les attributs : (a) *SensorLocation* qui indique la position du capteur, (b) *Precision* qui indique le pourcentage d'erreur d'un capteur lors de la lecture d'une mesure, (c) *Periodicity* qui indique la période d'acquisition des données et (d) *Frequency* qui indique la fréquence de transmission des données. Les attributs *Precision*, *Periodicity* et *Frequency* ne sont pas de type *Integer* ou *Real*, mais ils ont des types composés et indiquent la qualité de service. Nous affectons alors à ces attributs le stéréotype << nfp >> et des types importés de la bibliothèque BasicNFP_types de MARTE [OMG, 2011].

- *FusionUnit.* Cette unité permet de traiter régulièrement les données acquises provenant de chaque capteur (*e.g.,* extraction des informations à partir d'une image) à travers la méthode *ProcessData()* qui est stéréotypée << *periodic* >>. Nous associons à cette méthode une contrainte OCL (*context FusionUnit : :ProcessData() pre.period* $\leq$ *self.DBMeasurement.ValidityDuration*) qui indique que la durée de traitement d'une donnée doit être inférieure à sa durée de validité pour assurer la cohérence temporelle. En outre, cette unité permet de fusionner des données issues de plusieurs capteurs (*e.g.,* par moyenne) à travers la méthode *FuseData()* qui ne s'exécute que si des capteurs fournissent des données complémentaires ou redondantes. Ainsi, nous associons à cette méthode le stéréotype << *sporadic* >>.
- *SensorData, DerivedData.* Les données acquises par des capteurs sont des données sensorielles qui sont représentées par la classe *SensorData* stéréotypée << *sensor* >>. Cependant, les données dérivées sont calculées à partir des données sensorielles. Nous associons donc à la classe *DerivedData* le stéréotype << *derived* >>. Chaque donnée (sensorielle et dérivée) est définie par sa valeur qui représente la dernière valeur mise à jour. Nous associons à la classe *SensorData* un seuil Maximum Data Error (MDE) [Amirijoo et al., 2006] qui représente l'erreur maximale qui peut être tolérée entre la valeur réelle et la valeur stockée dans la base. En effet, une donnée sensorielle est considérée

valide si la différence entre la valeur mise à jour et la valeur stockée dans la base est inférieure ou égale au seuil MDE. Afin de maintenir la cohérence de la base, les données sensorielles doivent être mises à jour périodiquement à l'aide de la méthode *UpdateValue()*. Nous associons alors à cette méthode le stéréotype << *periodic* >>. Cependant, les données dérivées sont mises à jour à travers la méthode *UpdateValue()* à laquelle nous associons le stéréotype << *sporadic* >>. Ceci est dû au fait que la mise à jour des données dérivées est déclenchée par la mise à jour d'une donnée sensorielle utilisée dans son calcul. Nous définissons deux contraintes OCL qui indiquent que la mise à jour d'une donnée doit s'achever avant que la durée de validité n'expire. La première contrainte, (*context SensorData : :UpdateValue() pre :deadline <= self.DBMeasurement.ValidityDuration*), est associée à la classe *SensorData*. Quant à la deuxième contrainte, (*context DerivedData : :UpdateValue() pre :deadline <= self.DBMeasurement.ValidityDuration*), elle est associée à la classe *DerivedData*. De même, nous associons à la classe *SensorData* une deuxième contrainte OCL. Cette contrainte, (*Context DBMeasurement inv : self.MaximumDataError* $\geq$ *self.Sensor.Precision*), indique que l'erreur maximale doit être supérieure au pourcentage d'erreur du capteur lors de l'acquisition de la donnée.

- *DBMeasurement.* Cette classe représente les données TR stockées dans la base, qui sont caractérisées par les attributs suivants : (a) *MeasureType* qui indique le type de la donnée (vitesse, position, etc.), (b) *Unit* qui indique l'unité de la donnée (seconde, mètre, etc.), (c) *Timestamp* qui représente le temps où la valeur de la donnée est obtenue et (d) *ValidityDuration* qui représente l'intervalle en dehors duquel la valeur de la donnée est obsolète. Nous définissons une contrainte OCL relative à la classe *DBMeasurement*. Cette contrainte (*Context Measurement inv : self.ValidityDuration* $\geq$ *self.Sensor.Periodicity*) indique que la durée de validité d'une donnée doit être supérieure à la périodicité du capteur pour maintenir la cohérence de la base [Ramamritham et al., 2004].
- *ControlledVehicle.* Cette classe représente le véhicule à contrôler qui est observé par les capteurs proprioceptifs.

- *TrackedElement.* Cette classe représente l'environnement du véhicule à contrôler (routes, obstacles, etc.) qui est observé par des capteurs extéroceptifs.

Vue dynamique : la vue dynamique du patron *"Acquisition des données"* est représentée par le diagramme de séquence illustré par la figure 5.8. Ce diagramme permet de modéliser les interactions nécessaires pour les mises à jour des données temps réel. En effet, l'unité de fusion traite d'abord, à travers l'opération *ProcessData*, les données provenant des capteurs suite à la réception du signal *SetValue*, ou suite à l'exécution de l'opération *GetValue*. Ces méthodes, (*ProcessData*, *SetValue* et *GetValue*), sont stéréotypées << *periodic* >>, vu que chaque capteur acquiert les données régulièrement. Ensuite, ces données sont mises à jour et stockées dans la base suite à l'exécution de l'opération *UpdateValue* lorsque la valeur de la dernière donnée mise à jour est différente de la valeur stockée dans la base (*i.e.,* $storeddata - updateddata > MDE$). Nous associons alors le stéréotype << *sporadic* >> à l'opération *UpdateValue*. Dans le cas où le système utilise des capteurs qui fournissent des données complémentaires ou redondantes, l'unité de fusion invoque l'opération *FuseData* pour fusionner les données. La mise à jour des données sensorielles déclenche le calcul des données dérivées à travers l'invocation de l'opération *CalculateDerivedData*.

Les opérations *ProcessData* et *UpdateValue* du fragment *([stored data − updated data* $> MDE$*])* doivent être exécutées avant que la durée de validité d'une donnée n'expire. Ainsi, nous définissons la contrainte temporelle suivante : *deadline* $< ValidityDurationofData + CurrentTime$ alors que, les opérations *FusedData*, *UpdatedValue* et *CalculateDerivedData* doivent être exécutées avant que la durée de validité minimale d'une donnée utilisée pour le traitement n'expire. Nous définissons alors la contrainte temporelle suivante : $deadline < min(ValidityDuration\ of\ Data + CurrentTime)$.

5.3.3 Construction du patron *"Contrôle de données"*

La construction du diagramme de classes (cf. paragraphe 5.3.3.1) et du diagramme de séquence (cf. paragraphe 5.3.3.2) du patron *"Contrôle de données"* est aussi basée sur le processus de création des patrons proposé dans [Rekhis et al., 2010a] et enrichi par les règles d'unification que nous avons proposées pour le diagramme de séquence. Ce patron est relatif à la fonctionnalité de traitement, du contrôle de données et de la prise de décision.

5.3.3.1 Construction du diagramme de classes

Le Tableau 5.2 représente les contraintes et les concepts relatifs aux fonctions élémentaires déterminées suite au raffinement de la fonctionnalité du contrôle de données.

Tableau 5.2 : Fonctions élémentaires, concepts et contraintes de la fonctionnalité de contrôle des données.

Fonctions élémentaires	Concepts	Contraintes
F1 : Un contrôleur lit les mesures acquises nécessaires pour le traitement.	Contrôleur, Mesure acquise	La lecture des mesures n'est effectuée que si une mesure est mise à jour.
F2 : Un contrôleur analyse la situation de conduite en utilisant les mesures acquises et les mesures dérivées.	Contrôleur, Mesure acquise, Mesure dérivée	L'analyse de la situation doit être effectuée lorsqu'une mesure utilisée pour l'analyse est mise à jour.
F3 : Un contrôleur détecte un risque et évalue son degré de gravité.	Contrôleur, Risque	La valeur d'une mesure ne doit pas dépasser un seuil prédéfini.
F4 : Un contrôleur décide de l'action appropriée à la situation.	Contrôleur	L'action doit être identifiée avant que le risque ne se produise.

Une fois que les concepts sont identifiés, les classes qui jouent les rôles de ces concepts doivent être identifiées pour déterminer les fragments d'applications associés à cette fonctionnalité, comme illustré dans les figures 5.1, 5.2 et 5.3 (Les classes en jaune). Les classes

qui ne sont liées à aucun concept et qui ont des relations avec les classes identifiées sont alors ajoutées aux fragments d'applications. Ces fragments sont les suivants :

- Fragment-Cl(ACC) = {Controller, AcquiredData, DerivedData, Danger, Data} ;
- Fragment-Cl(LDW) = {Controller, AcquiredData, DerivedData, Danger, Data} ;
- Fragment-Cl(SAFERIDER) = {ControlModule, SensorMeasurement, DerivedMeasurement, Risk, Measurement}.

Après l'étape d'identification des fragments d'applications relatifs à la fonctionnalité de contrôle de données, nous générons le diagramme de classes (5.9) relatif à cette fonctionnalité. Ce diagramme est obtenu en appliquant les règles d'unification suivantes :

- N_equiv(Danger(ACC), Danger(LDW), Risk(SAFERIDER)) et Att_equiv(Danger(ACC), Danger(LDW), Risk(SAFERIDER)) : la règle RC1 est appliquée et la classe *Risk* est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>. La même règle est appliquée pour les classes *AcquiredData* et *SensorMeasurement* et une classe *SensorData* est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>.
- N_equiv(DerivedData(ACC), DerivedData(LDW), DerivedMeasurement(SAFERIDER)), Att_equiv(DerivedData(ACC), DerivedData(LDW), DerivedMeasurement(SAFERIDER)) et Op_equiv(DerivedData(ACC), DerivedData(LDW), DerivedMeasurement (SAFERIDER)) : la règle RC1 est appliquée, ce qui permet d'ajouter au patron la classe *DerivedData* en tant que classe fondamentale stéréotypée << *mandatory* >>.
- N_dist(Controller(ACC), Controller(LDW), ControlModule(SAFERIDER)) et Op_equiv(Controller(ACC), Controller(LDW), ControlModule(SAFERIDER)) : la règle RC4 est appliquée et une classe fondamentale *Controller* est ajoutée au patron en tant que classe stéréotypée << *mandatory* >>. La même règle est appliquée pour les classes (Data(ACC), Data(LDW), Measurement(SAFERIDER)) et une classe *DBMeasurement* est ajoutée au patron en tant que classe stéréotypée << *mandatory* >>.
- La règle RC5 est appliquée, ce qui permet d'ajouter les relations entre les classes d'applications transférées au patron.

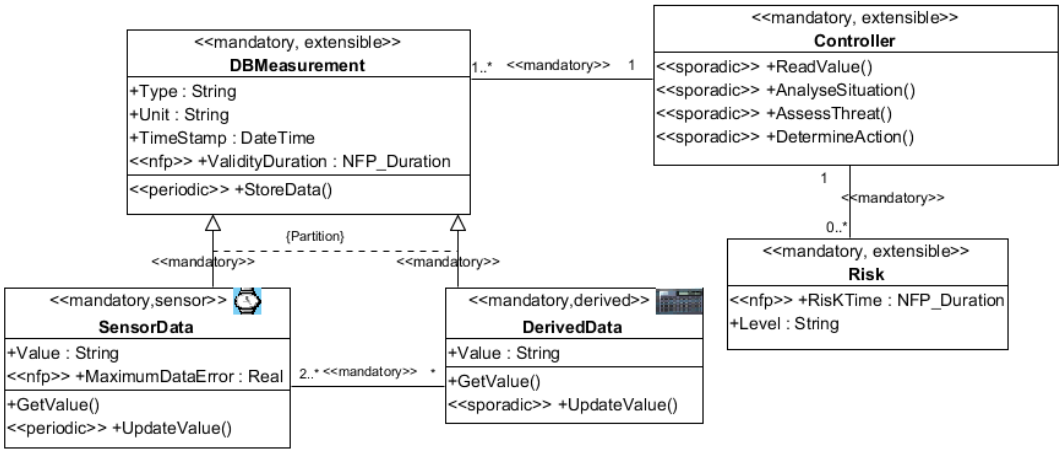


FIGURE 5.9 : Diagramme de classes du patron *"Contrôle de données"*.

5.3.3.2 Construction du diagramme de séquence

De la même manière que la construction du diagramme de séquence du patron *"Acquisition des données"*, la modélisation du diagramme de séquence du patron *"Contrôle de données"* présenté dans la figure 5.10 est illustrée par l'application des règles d'unification.

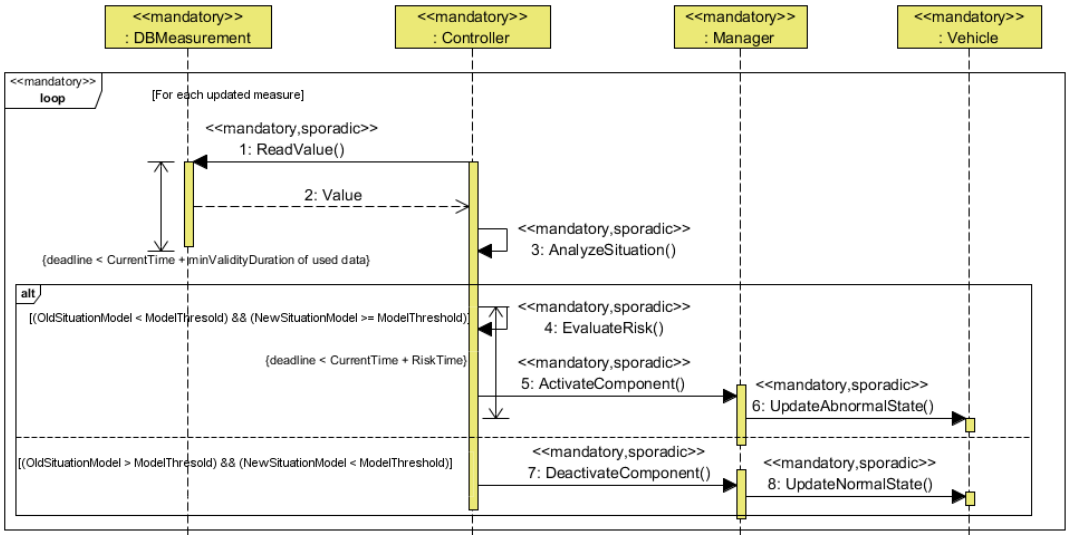


FIGURE 5.10 : Diagramme de séquence du patron *"Contrôle de données"*.

5.3.3.3 Description du patron *"Contrôle de données"*

Nous détaillons à ce niveau la description du patron *"Contrôle de données"* selon le formalisme P-SIGMA comme suit :

1. **Nom :** *"Contrôle de données"*, appelé en anglais ADAS-Controller (Advanced Driver Assistance Systems Controller).
2. **Contexte :** Ce patron est appliqué pendant la phase de contrôle et de traitement des données. Il est appliqué lorsqu'une donnée acquise est mise à jour.
3. **Problème :** Un contrôleur dans un système d'aide à la conduite doit contrôler de manière efficace un grand volume de données pour décider des actions correctives appropriées à la situation de conduite. Donc, comment le contrôleur peut-il contrôler la situation et comment peut-il prendre des décisions ?
4. **Solution :**

   Vue Statique : la vue statique du patron *"Contrôle de données"* est modélisée par le diagramme de classes présenté dans la figure 5.9, qui représente les différents participants de la fonctionnalité de contrôle de données :

   - *Controller.* Le contrôleur accède à la base et récupère les données nécessaires pour analyser la situation de conduite à chaque fois qu'une donnée acquise est mise à jour. Les opérations *ReadValue()* et *AnalyzeSituation()* sont donc stéréotypées << *sporadic* >>. Si un risque est détecté, le contrôleur décide l'action appropriée à cette situation et notifie au gestionnaire d'activer l'actionneur et/ou les composants de l'interface homme-machine. Une fois que la situation normale est rétablie, le contrôleur notifie au gestionnaire de désactiver l'action. Ainsi, les opérations *AssessThreat()* et *DetermineAction()* sont stéréotypées << *sporadic* >>.

     Nous associons à la classe *Controller* deux contraintes OCL. La première, (*context Controller : :AssessThreat() pre :deadline < self.Risk.RiskTime*), indique que l'évaluation du risque doit être effectuée avant que le risque ne se produise. La deuxième, (*context Controller : :DetermineAction() pre :deadline < self.Risk.RiskTime*), indique que le contrôleur doit déterminer l'action corrective avant que le risque ne se produise.

- *SensorData, DerivedData.* La classe *SensorData* représente les données acquises par les capteurs. Elle doit être alors stéréotypée << *sensor* >>. Quant à la classe *DerivedData*, elle représente les données dérivées à partir des données acquises. En effet, le contrôleur utilise les données acquises pour vérifier la situation et calculer les données dérivées.
- *DBMeasurement.* Cette classe représente toutes les données temps réel stockées dans la base.
- *Risk.* Cette classe représente une situation critique (*e.g.,* risque de collision ou sortie involontaire de voie) que le contrôleur peut détecter en analysant l'état du véhicule et son environnement. Cette classe possède deux attributs : (1) *RiskTime* qui indique la durée avant que le risque ne se produise et (2) *Level* qui indique le degré de gravité du risque.

Vue dynamique : la vue dynamique du patron *"Contrôle de données"* est illustrée par le diagramme de séquence présenté dans la figure 5.10. Ce diagramme montre les différentes interactions nécessaires pour contrôler la situation de conduite suite à la mise à jour périodique des données. En effet, le contrôleur est responsable de l'exécution de l'opération *AnalyzeSituation()* suite à l'exécution de l'opération *ReadValue()*. Après l'analyse de la situation, deux alternatives peuvent être suivies, qui sont représentées par le fragment combiné ayant l'opérateur *alt*. La première alternative correspond à la situation où le contrôleur détecte un risque. Il permet alors d'invoquer l'opération *ActiveAction()* suite à l'exécution de l'opération *EvaluateRisk()*. Quant à la deuxième alternative, elle correspond à la situation normale. Dans ce cas, le contrôleur permet d'invoquer l'opération *StopAction()*. Dans les deux cas, l'état de véhicule est mis à jour.

5.3.4 Construction du patron *"Action"*

La construction du diagramme de classes (cf. paragraphe 5.3.4.1) et du diagramme de séquence (cf. paragraphe 5.3.4.2) du patron *"Action"* est aussi illustrée en suivant les étapes du processus de création des patrons proposé dans [Rekhis et al., 2010a] et enrichi par nos

travaux. Ce patron est relatif à la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes.

5.3.4.1 Construction du diagramme de classes

Le raffinement de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes a permis d'identifier les fonctions élémentaires ainsi que leurs concepts et leurs contraintes comme illustré dans le tableau 5.3.

Tableau 5.3 : Fonctions élémentaires, concepts et contraintes de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes.

Fonctions élémentaires	**Concepts**	**Contraintes**
F1 : Un gestionnaire d'action active un actionneur et/ou un composant d'Interface Homme-Machine (IHM).	Gestionnaire d'action, Actionneur, Composant de l'IHM	L'activation d'actionneurs et/ou de composants de l'IHM doit être avant l'échéance.
F2 : Un actionneur exécute l'action appropriée.	Actionneur, Action automatique	L'action doit être exécutée avant l'échéance.
F3 : Un composant de l'IHM génère des alertes visuelles, audio et/ou haptiques au conducteur.	Composant de l'IHM, Alerte, Conducteur	La génération des alertes doit être avant l'échéance.
F4 : Le conducteur prend les actions correctives.	Conducteur, Action conducteur	L'action prise par le conducteur doit être exécutée avant que le risque ne se produise.
F5 : Un gestionnaire d'action désactive l'actionneur et/ou les composants d'IHM.	Gestionnaire d'action, Actionneur, Composant de l'IHM	

Une fois que les concepts sont identifiés, nous déterminons les fragments d'applications relatifs à la fonctionnalité d'exécution des ordres et/ou de déclenchement des alertes. L'identification de ces fragments est basée sur la détermination des classes jouant les mêmes rôles des concepts relatifs à cette fonctionnalité (Les classes du rectangle pointillé présenté dans les figures 5.1, 5.2 et 5.3). À chaque fragment, nous associons les classes qui ont des relations avec

les classes identifiées et qui ne sont liées à aucun concept de domaine. Ainsi, les fragments de chaque application sont les suivants :

- Fragment-Cl(ACC) = {Manager, Actuator, DashboardDisplay, Alarm, AlarmType, Driver, Vehicle, DriverAction, AutomaticAction} ;
- Fragment-Cl(LDW) = {Manager, Steer, DashboardDisplay, Alarm, AlarmType, Driver, Vehicle, DriverAction, AutomaticAction} ;
- Fragment-Cl(SAFERIDER) = {Manager, HMIElement, Warning, Rider, Motorcycle, RiderAction, WarningType, AutomaticAction, VisualWarning, HapticWarning, StereoAudio, VisualDevice}.

L'unification des fragments d'applications relatifs à la fonctionnalité d'exécution et/ou de déclenchement des alertes permet d'obtenir le diagramme de classes du patron présenté dans la figure 5.11. Ce diagramme de classes est obtenu en appliquant les règles d'unification suivantes :

- N_equiv(Manager(ACC), Manager(LDW), HMIManager(SAFERIDER)) et Op_equiv (Manager(ACC), Manager(LDW), HMIManager(SAFERIDER)) : la règle RC1 est appliquée et une classe *Manager* est ajoutée au patron en tant que classe fondamentale et stéréotypée << *mandatory* >>. La même règle est appliquée pour les classes (*VisualAlarm(ACC)*, *VisualAlarm(LDW)*, *Warning(SAFERIDER)*), ce qui permet d'ajouter au patron la classe *WarningSignal* en tant que classe fondamentale stéréotypée << *mandatory* >>.
- N_dist(Vehicle(ACC), Vehicle(LDW), Motorcycle(SAFERIDER)), Att_int(Vehicle(ACC), Vehicle(LDW), Motorcycle(SAFERIDER)) : la règle RC4 est appliquée, ce qui permet d'ajouter la classe *Vehicle* au patron en tant que classe fondamentale stéréotypée << *mandatory* >> vu que les classes *Vehicle* et *Motorcycle* jouent le même rôle d'un concept de domaine qui est le *Véhicule contrôlé*. La même règle est appliquée pour les classes (Driver(ACC), Driver(LDW), Rider(SAFERIDER)) et les classes (DashboardDisplay(ACC), DashboardDisplay(LDW), HMIElement(SAFERIDER)), et par conséquent, les classes *Driver* et *HMIElement* sont ajoutées au patron en tant que classes fondamentales stéréotypées << *mandatory* >>.

- N_var(DriverAction(ACC), DriverAction(LDW), RiderAction(SAFERIDER)), Att_int (DriverAction(ACC), DriverAction(LDW), RiderAction(SAFERIDER)) et Op_int(DriverAction(ACC), DriverAction(LDW), RiderAction(Saferider)) : la règle RC6 est appliquée. Les classes *DriverAction* et *RiderAction* représentent des variations du concept *Action conducteur*. Ainsi, la classe *DriverAction* est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>.
- N_dist(Actuator(ACC), Steer(LDW)), Att_int(Actuator(ACC), Steer(LDW)) et Op_ int(Actuator(ACC), Steer(LDW)) : la règle RC7 est appliquée vu que les deux classes (*Actuator* et *Steer*) sont présentes dans les systèmes ACC et LDW, et jouent le même rôle du concept *Actionneur automatique*. Ainsi, la règle RC4 est appliquée et la classe *AutomaticActuator* est ajoutée au patron en tant que classe optionnelle stéréotypée << *optional* >>.
- La règle RC7 est appliquée, ce qui permet d'ajouter la classe optionnelle *AutomaticAction* au patron en tant que classe stéréotypée << *optional* >>. Ceci est dû au fait que la classe *AutomaticAction* est présente dans certaines applications ($Rdc(AutomaticAction) = \frac{2}{3} \geq \frac{1}{2}$).
- N_equiv(AlarmType(ACC), AlarmType(LDW), WarningType(SAFERIDER)) et Att_ int(AlarmType(ACC), AlarmType(LDW), WarningType(SAFERIDER)) : la règle RC1 est appliquée et la classe *WarningType* est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>.

5.3.4.2 Construction du diagramme de séquence

Le diagramme de séquence du patron *"Action"*, illustré dans la figure 5.12, est obtenu par l'application des règles d'unification d'une manière similaire au patron *"Acquisition des données"*.

5.3.4.3 Description du patron *"Action"*

Nous décrivons le patron *"Action"* selon le formalisme P-SIGMA comme suit :

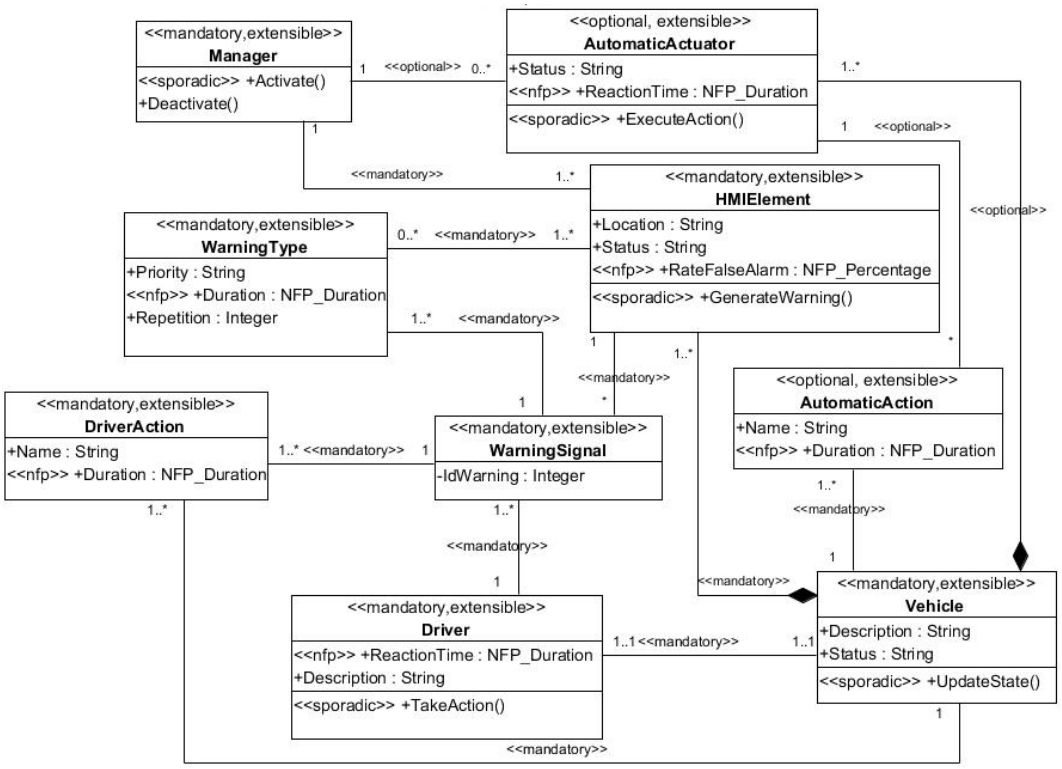


FIGURE 5.11 : Diagramme de classes du patron *"Action"*.

1. **Nom :** *"Action"*, appelé en anglais ADAS-AS (Advanced Driver Assistance Systems Action Subsystem).
2. **Contexte :** Le patron est appliqué uniquement lorsque le contrôleur détecte une situation critique. Il notifie alors le gestionnaire pour activer les composants de l'IHM et/ou un actionneur pour éviter le risque.
3. **Problème :** Comment un système d'aide à la conduite peut-il éviter une situation critique et assurer la sécurité des occupants du véhicule ?
4. **Solution :**

   Vue Statique : la vue statique du patron *"Action"* est modélisée par le diagramme de classes présenté dans la figure 5.12, qui montre les différents participants de la fonctionnalité d'exécution et/ou de déclenchement des alertes :

   - *AutomaticActuator.* Les actionneurs représentent les dispositifs du véhicule responsables de l'exécution des actions mécaniques, tel que le freinage. La classe *AutomaticActuator* possède les attributs suivants : (a) *Status* qui indique l'état

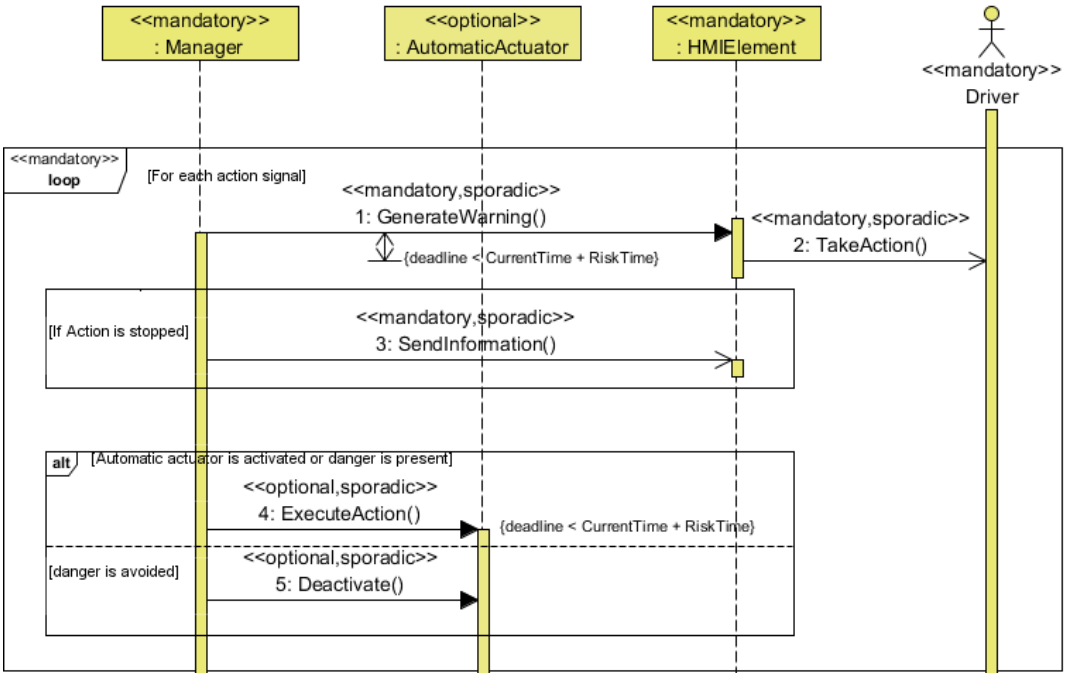


FIGURE 5.12 : Diagramme de séquence du patron *"Action"*.

de l'actionneur (activé ou désactivé) et (b) *ReactionTime* qui représente le temps nécessaire pour que l'actionneur exécute l'action et l'opération *ExecuteAction()*. Cette opération est stéréotypée << *sporadic* >> vu que l'exécution de l'action se produit uniquement si une situation critique est détectée.

Nous associons à cette classe la contrainte OCL « *context AutomaticActuator : :Execute-Action() pre : deadline* $\leq$ *current time + D*, où D est la durée restante avant que le risque ne survienne » pour préciser que l'action doit s'exécuter avant que le risque ne se produise.

- *HMIElement.* Les composants d'IHM sont responsables de la génération des alertes au conducteur. La classe *HMIElement* possède les attributs suivants : (a) *Location*, qui représente la position du dispositif dans le véhicule, (b) *Status*, qui représente l'état du composant (activé ou désactivé), (c) *RateFalseAlarm*, qui indique le pourcentage de présence d'une fausse alerte. En outre, cette classe possède l'opération *GenerateWarning()* qui est stéréotypée << *sporadic* >>, vu que les alertes ne doivent être déclenchées que si une situation critique est détectée.

Nous associons aussi à cette classe la contrainte OCL « *context HMIElement : :Generate-Warning() pre : deadline* $\leq$ *current time + D + self.Driver.ReactionTime*, où D est la durée restante avant que le risque ne survienne » pour indiquer que les alertes doivent être générées avant que le risque ne se produise et en prenant en compte le temps de réaction du conducteur pour éviter le danger.

- *Manager.* Le gestionnaire d'action est responsable de l'activation et de la désactivation des composants de l'IHM et/ou de l'actionneur approprié selon l'action à exécuter.
- *AutomaticAction.* Cette classe représente les différentes actions qui peuvent être exécutées par un actionneur automatique.
- *WarningSignal.* Les alertes sont déclenchées pour informer le conducteur d'un risque et sont générées par le biais des composants de l'IHM.
- *WarningType.* Ces alertes sont généralement classées en trois catégories : des alertes visuelles, des alertes sonores et des alertes haptiques. Elles sont caractérisées par : (i) une priorité qui représente le niveau d'une alerte selon le degré de gravité du risque (*e.g.,* une alerte faible et une alerte élevée), (ii) une durée qui constitue l'intervalle de temps durant lequel une alerte est considérée valide et (iii) un taux de répétition.
- *Driver.* Suite à la réception d'une alerte, le conducteur doit prendre les actions correctives immédiatement pour éviter le danger. La classe *Driver* possède l'opération *TakeAction()* qui est stéréotypée << *sporadic* >> vu que le conducteur ne prend les actions que si une alerte est déclenchée.
- *DriverAction.* Cette classe représente les actions correctives prises par le conducteur.

  Nous associons à cette classe la contrainte OCL « *context DriverAction inv : self.Duration* $\leq$ *self.WarningType.Duration* » pour préciser que le conducteur doit réagir immédiatement pour éviter le risque.
- *Vehicle.* Cette classe possède l'opération *UpdateState()* qui indique que l'état du véhicule change selon les actions exécutées par un actionneur ou par le conducteur.

Vue dynamique : la vue dynamique du patron *"Action"* est modélisée par le diagramme de séquence présenté dans la figure 5.12, qui modélise les interactions effectuées suite à la détection d'une situation critique. En effet, l'objet *Manager* est responsable à l'invocation de deux opérations : (1) *ExecuteAction()* pour activer l'actionneur qui va exécuter l'action et (2) *GenerateWarning()* pour activer les composants de l'IHM qui vont générer les messages d'alerte au conducteur.

5.4 Conclusion

Dans ce chapitre, nous nous sommes intéressés à la modélisation des patrons que nous avons proposés pour les systèmes d'aide à la conduite ainsi que leur représentation. Le patron permettant la modélisation de la fonctionnalité d'acquisition des données temps réel a fait l'objet d'une publication dans [Marouane et al., 2012]. Quant au patron destiné à la modélisation de la fonctionnalité d'exécution des ordres et/ou de déclenchement des alertes, il a fait l'objet de la publication [Marouane et al., 2013].

Dans le chapitre qui suit, nous allons présenter la réutilisation de ces trois patrons en les adaptant à différentes applications d'aide à la conduite qui reposent sur l'utilisation des systèmes de bases de données TR.

Chapitre 6

Réutilisation des patrons de conception

6.1 Introduction

Le chapitre précédent a décrit nos propositions permettant de spécifier les patrons de conception dédiés à la modélisation des systèmes d'aide à la conduite temps réel. De telles spécifications n'ont d'intérêt que si un concepteur d'applications peut extraire une partie de toutes ces informations lors de la réutilisation. C'est dans ce but que nous proposons, dans ce chapitre, un processus de réutilisation pour guider les concepteurs d'applications lors de la réutilisation des solutions de patrons.

La première section de ce chapitre présente le processus de réutilisation et l'illustre par le biais de l'instanciation des patrons que nous avons proposés dans le chapitre précédent. La deuxième section présente l'outil de réutilisation que nous mettons en œuvre.

6.2 Processus de réutilisation

La figure 6.1 présente le processus de réutilisation sous forme d'un diagramme d'activités. Ce processus est composé de deux activités principales détaillées ci-après. La première activité consiste à obtenir un modèle réduit du patron ; il s'agit de supprimer les variantes qui ne répondent pas aux besoins du concepteur d'applications. Cette activité est précédée par l'activité *Choisir une solution d'un patron* qui permet au concepteur d'applications de sélectionner la solution du patron qui résout le problème en s'appuyant sur les rubriques *"Contexte"* et *"Problème"*. La deuxième activité consiste à générer les diagrammes UML correspondant aux instances du patron.

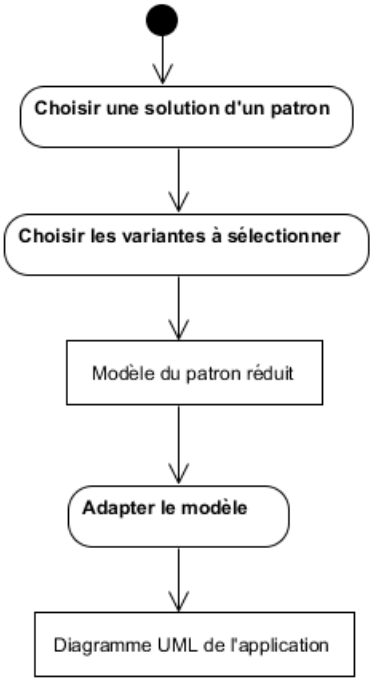


FIGURE 6.1 : Processus de réutilisation.

6.2.1 Choix des variantes à sélectionner

L'activité relative au choix des variantes permet au concepteur d'applications de créer, à partir de la solution du patron choisi comportant les points de variations, un modèle ré-

duit d'un patron plus spécifique à ses exigences, appelé modèle du patron réduit. Ce modèle correspond à la solution de base qu'il devra appliquer à son contexte. Le concepteur d'applications doit alors choisir les variantes qu'il désire réutiliser, tout en respectant ses besoins. Ceci entraîne l'instanciation de tous les éléments en relation avec ces variantes. Par exemple, si des messages, qui constituent des variantes, sont instanciés, alors le fragment combiné, auquel ils sont associés, est aussi instancié. Cependant, les variantes sélectionnées ne sont pas les seules qui figurent dans le modèle cible. En effet, les éléments fondamentaux doivent y figurer.

Pour conclure, cette activité permet de créer un modèle ne comportant que les éléments fondamentaux (*i.e.,* stéréotypés << *mandatory* >>) et les éléments variables choisis par le concepteur.

6.2.2 Adaptation du modèle réduit du patron

Une fois que le modèle du patron réduit est obtenu, le concepteur d'applications peut renommer, redéfinir et ajouter des éléments selon ses besoins tout en respectant les contraintes associées aux éléments instanciés. En effet, le concepteur d'applications peut renommer les éléments du modèle en se basant sur un dictionnaire pour qu'ils restent conformes aux relations linguistiques entre les noms des termes. Toute modification affectant un élément instancié doit respecter, d'une part, les contraintes exprimées implicitement par le(s) stéréotype(s) relatif(s) à cet élément et, d'autre part, les contraintes OCL. Par exemple, il n'est pas possible d'ajouter des attributs et/ou des opérations à des classes qui ne sont pas stéréotypées << *extensible* >>. De plus, il n'est pas possible de supprimer une classe stéréotypée << *mandatory* >>.

Il est également possible au concepteur d'ajouter une autre instance d'un élément instancié, à condition que les contraintes traitant des occurrences d'instanciation de cet élément ne soient pas violées. Par exemple, si une classe fondamentale a une multiplicité de 1, alors une occurrence de cette classe ne peut participer qu'une seule fois à l'association.

D'ailleurs, toute classe instanciée est automatiquement stéréotypée $<< PatternClass >>$ pour assurer le critère de traçabilité. De même, les stéréotypes $<< PatternAttribute >>$, $<< PatternOperation >>$, $<< PatternLifeline >>$ et $<< PatternMessage >>$ sont ajoutés automatiquement aux attributs, aux opérations, aux objets et aux messages. Les stéréotypes exprimant les aspects temps réel et les propriétés non fonctionnelles ($<< sensor >>$, $<< derived >>$, $<< periodic >>$, $<< sporadic >>$ et $<< nfp >>$) sont également ajoutés automatiquement au diagramme de l'application considérée lorsque les éléments du patron associés à ces stéréotypes sont instanciés. Ceci permet au concepteur de respecter les contraintes temporelles lors de la modélisation de son application vu qu'une violation d'une contrainte temporelle peut engendrer des conséquences graves.

Pour conclure, cette activité permet de générer les diagrammes UML comportant les éléments instanciés modifiés par le concepteur selon le contexte de l'application à modéliser, et de faire évoluer ces diagrammes par l'adjonction de nouveaux éléments spécifiques aux besoins de l'application. Les classes spécifiques sont stéréotypées $<< ApplicationClass >>$. Les objets et les messages spécifiques sont également stéréotypés respectivement $<< ApplicationLifeline >>$ et $<< ApplicationMessage >>$.

Une fois que les fragments relatifs à chaque patron sont obtenus, une étape de composition peut être réalisée, c'est à dire mettre en relation les éléments d'un fragment avec d'autres éléments d'autres fragments de façon à générer le modèle complet de l'application cible (cf. travaux de Arnaud [Arnaud, 2008]). Dans le cadre de nos travaux, nous ne nous sommes pas intéressés à la composition des instances des patrons.

6.2.3 Exemple de réutilisation de patrons

Dans cette section, nous illustrons le processus par l'instanciation des patrons que nous avons proposés dans le chapitre précédent pour la modélisation du système *Laterl Safe* Amditis et al. [2008].

Le système *Lateral Safe* a pour objectif d'éviter les situations critiques en latéral. Il consiste à assister le conducteur en élaborant des alertes pour accroître la sécurité et éviter les risques de collisions, grâce à différents capteurs embarqués dans le véhicule :

- Des capteurs proprioceptifs permettant d'observer l'état du véhicule (*e.g.,* la vitesse et la position).
- Des radars de gamme courte placés des deux côtés du véhicule, qui permettent de détecter la présence des objets de chaque côté du véhicule et d'acquérir les informations relatives aux objets détectés, telles que la distance entre les objets et le véhicule, leur position latérale et leur taille. Ces capteurs doivent avoir une couverture latérale suffisante pour détecter le maximum d'objets (distance de détection de 30 m) et disposer d'une forte précision et d'un taux élevé de mise à jour.
- Un radar de longue portée (77 Ghz) placé à l'arrière du véhicule, qui permet de détecter la présence d'objets à l'arrière du véhicule. Ce capteur offre une distance de détection de 24 mètres et une bonne précision.
- Des caméras placées de chaque côté et à l'arrière du véhicule, qui permettent d'observer l'état des objets détectés. Ces caméras sont caractérisées par une distance de détection, une résolution, une précision et un angle de couverture.

Chaque capteur acquiert des données et les transmet périodiquement vers une unité de fusion par le biais d'un bus CAN. Cette unité permet, d'une part, de traiter les données (*e.g.,* extraction des informations à partir des images) et, d'autre part, de calculer la valeur de vitesse et la distance inter-véhiculaires par la fusion des données acquises par chaque capteur radar. Chaque donnée est caractérisée par une valeur, une unité, une durée de validité et une estampille. Concernant les données acquises par les capteurs, elles sont caractérisées par une erreur maximale tolérée entre la valeur réelle et la valeur stockée dans la base.

L'unité de contrôle accède à la base et utilise les données nécessaires pour analyser la situation. Cette unité permet de déterminer les situations à risque en calculant le temps à collision (Time To Collision : TTC). Lorsqu'elle détecte que la valeur de TTC est inférieure à une valeur TTC seuil, elle identifie un risque de collision (critique ou moyen) et envoie un signal à un contrôleur d'interface homme-machine pour activer le composant adéquat.

La génération d'alertes doit se faire avant que le risque ne se produise potentiellement dans un délai inférieur à 1 s. Ces alertes peuvent être visuelles ou sonores. Chaque type d'alerte est caractérisé par un niveau (alerte de niveau 2 si la situation est grave ; alerte de niveau 1 si la situation est moyenne et alerte de niveau 0 si la situation est normale), un taux de répétition et une durée pour réduire le nombre de fausses alertes et le taux de défaillance des composants de l'interface homme-machine. Ces alertes sont transmises au conducteur afin qu'il effectue des actions correctives à temps (il a besoin d'un temps supérieur à 1 s pour réagir après avoir été averti).

La modélisation des fonctionnalités de ce système (*i.e.,* acquisition des données, contrôle des données et exécution des ordres et/ou déclenchement des alertes) en réutilisant les patrons débute par le choix des variantes puis l'adaptation des éléments selon les besoins du système. Cependant, le choix des variantes n'est pas appliqué pour les diagrammes de classes et de séquence du patron *"Contrôle de données"*. Ceci est dû au fait que tous les éléments dans ces diagrammes sont fondamentaux, et doivent donc être instanciés pendant la modélisation d'un système.

6.2.3.1 Réutilisation du patron *"Acquisition des données"*

La figure 6.2 montre l'adaptation du diagramme de classes du patron *"Acquisition des données"* pour modéliser l'acquisition des données par les capteurs du système Lateral Safe. Les classes *Sensor*, *EnvironmentSensor* et *VehicleSensor* correspondent respectivement aux classes *Sensor*, *ProprioceptiveSensor* et *ExteroceptiveSensor* du patron. Les capteurs proprioceptifs et les capteurs extéroceptifs observent respectivement le véhicule et les objets détectés, qui représentent les éléments observés. Ainsi, les classes *EgoVehicle* et *DetectedObject* correspondent respectivement aux classes *ControlledVehicle* et *TrackedElement* du patron. De plus, les classes *FusionUnit*, *DBMeasurement*, *SensorData* et *DerivedData* sont instanciées respectivement par les classes *FusionUnit*, *Measurement*, *AcquiredData* et *CalculatedData*. Les capteurs *Caméras* et les capteurs *Radars* constituent des variantes des capteurs extéro-

ceptifs. Ainsi, les classes *Camera* et *Radar* sont ajoutées au diagramme en tant que classes instanciées de la classe *ExteroceptiveSensor*.

Chaque classe jouant le rôle d'une classe du patron est identifiée par le stéréotype $<< PatternClass >>$ et ses propriétés *PatternName* et *Role*.

Les attributs *Resolution* et *Sensitivity* de la classe *Camera* et l'attribut *Type* de la classe *Radar* représentent des attributs spécifiques du système Lateral Safe. Ces deux classes constituent des sous-classes qui spécialisent la classe *EnvironmentSensor* et elles sont stéréotypées $<< ApplicationClass >>$. De plus, l'attribut *ID* associé à la classe *Object* et les deux attributs *Mark* et *Label* associés à la classe *EnvironmentSensor* constituent des attributs spécifiques du système. De même, les attributs *ComptingPower*, *Frequency* et *Label* de la classe *FusionUnit*, l'attribut *Description* de la classe *VehicleSensor* et l'attribut *Error* de la classe *Measurement* constituent des attributs spécifiques. Cependant, tous les autres attributs correspondent aux attributs du patron.

L'opération *ClassifyObject()* de la classe *FusionUnit* et l'opération *AddObject()* de la classe *Object* sont des opérations spécifiques, alors que toutes les autres opérations représentent des instances des opérations du patron. Ainsi, les attributs et les opérations instanciés du patron sont identifiés respectivement à l'aide des stéréotypes $<< PatternAttribute >>$ et $<< PatternOperation >>$ et de leurs propriétés *PatternName*, *ClassName* et *Role*.

De son côté, la figure 6.3 illustre l'instanciation du diagramme de séquence du patron *"Acquisition des données"*. Elle montre que les objets et les messages instanciés du patron sont respectivement identifiés à travers les stéréotypes $<< PatternLifeline >>$ et $<< PatternMessage >>$. Les objets *Sensor*, *FusionModule*, *Measurement* et *ControlUnit* correspondent respectivement aux objets *Sensor*, *FusionUnit*, *DBMeasurement* et *Controller* du patron. Tous les messages représentés dans le modèle du système correspondent également aux messages du patron. Les fragments combinés associés aux messages instanciés sont alors instanciés.

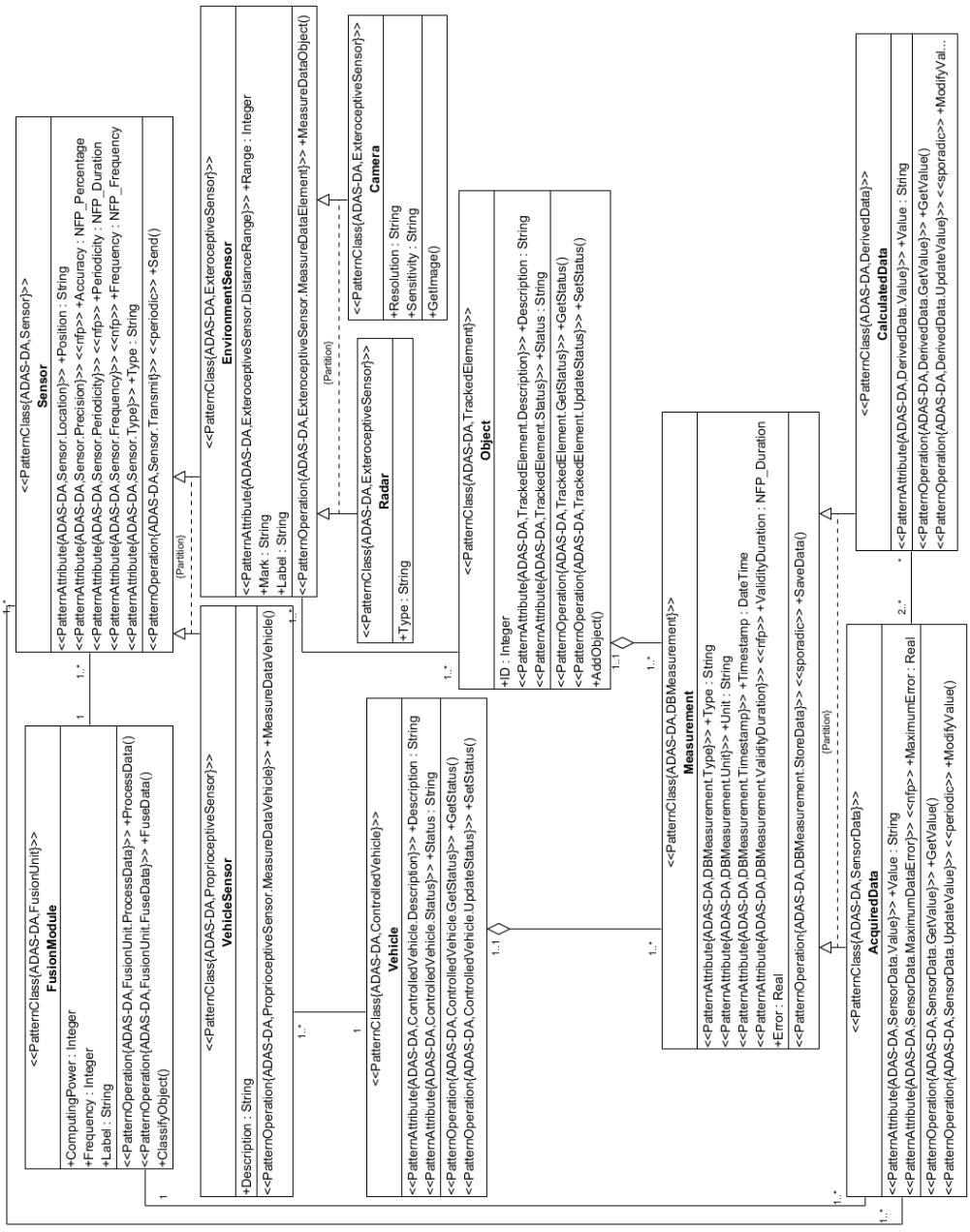


FIGURE 6.2 : Réutilisation du diagramme de classes du patron *"Acquisition des données"*.

6.2.3.2 Réutilisation du patron *"Contrôle de données"*

La figure 6.4 illustre la réutilisation du patron *"Contrôle de données"* pour modéliser la fonctionnalité du contrôle et la prise des décisions du système Lateral Safe. On y trouve que les classes, les attributs et les opérations instanciés du patron sont identifiés respec-

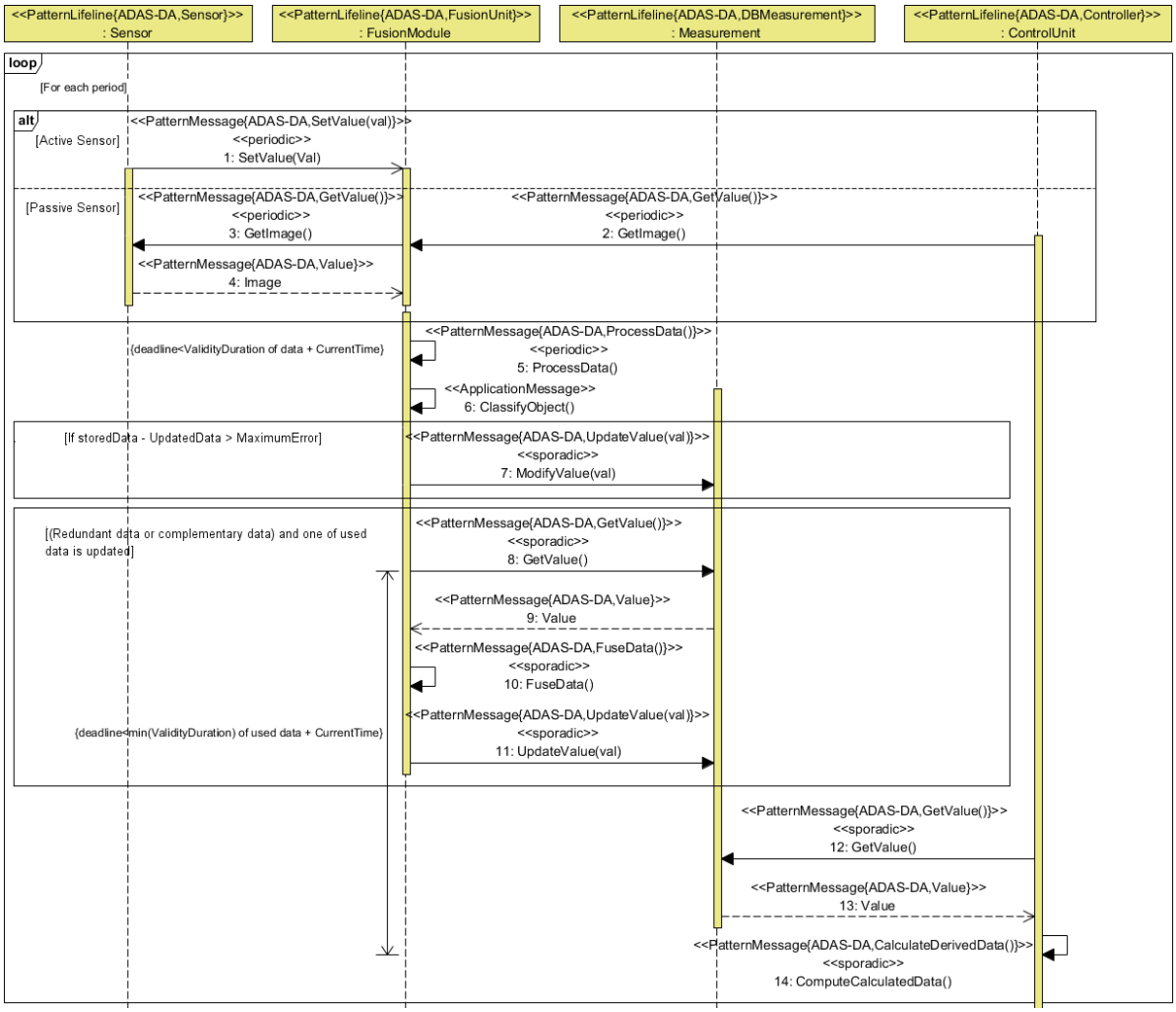


FIGURE 6.3 : Réutilisation du diagramme de séquence du patron *"Acquisition des données"*.

tivement à l'aide des stéréotypes << *PatternClass* >>, << *PatternAttributes* >> et << *PatternOperation* >>.

Les classes *ControlUnit*, *Hazard*, *Measurement*, *AcquiredData* et *CalculatedData* constituent respectivement des instances des classes *Controller*, *Risk*, *DBMeasurement*, *SensorData* et *DerivedData* du patron.

Les attributs *ComputingPower* et *Label* associés à la classe *ControlUnit*, l'attribut *Type* associé à la classe *Hazard* et l'attribut *Error* associé à la classe *Measurement* représentent des attributs spécifiques du système Lateral Safe, alors que les autres attributs correspondent

à des attributs du patron. De même, toutes les opérations constituent des instances des opérations du patron.

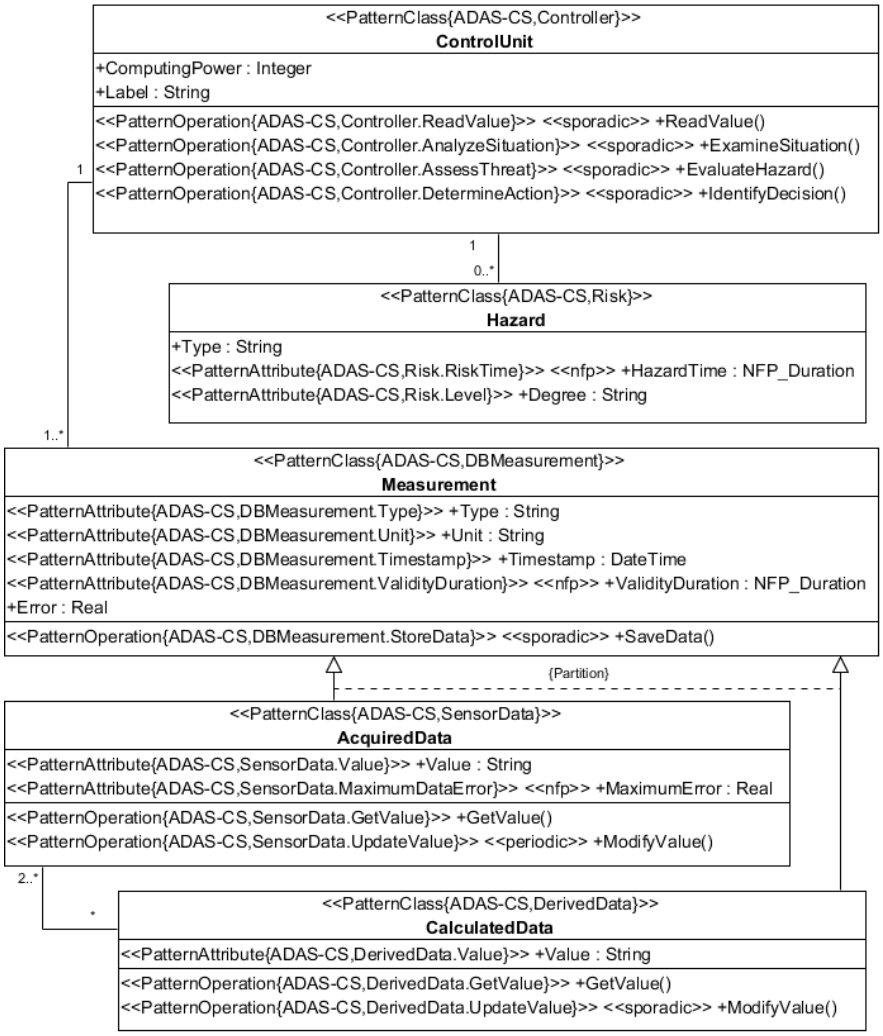


FIGURE 6.4 : Réutilisation du diagramme de classes du patron *"Contrôle de données"*.

De son côté, la figure 6.5 montre l'instanciation du diagramme de séquence du patron *"Contrôle de données"*. Les objets *Controller*, *DBMeasurement* et *Manager* du patron sont instanciés respectivement par *ControlUnit*, *Measurement* et *HMIManager*, qui sont identifiés par le stéréotype $<< PatternLifeline >>$ et ses propriétés *PatternName* et *Role*. Tous les messages du patron sont également instanciés dans le diagramme de séquence du système et sont stéréotypés $<< PatternMessage >>$.

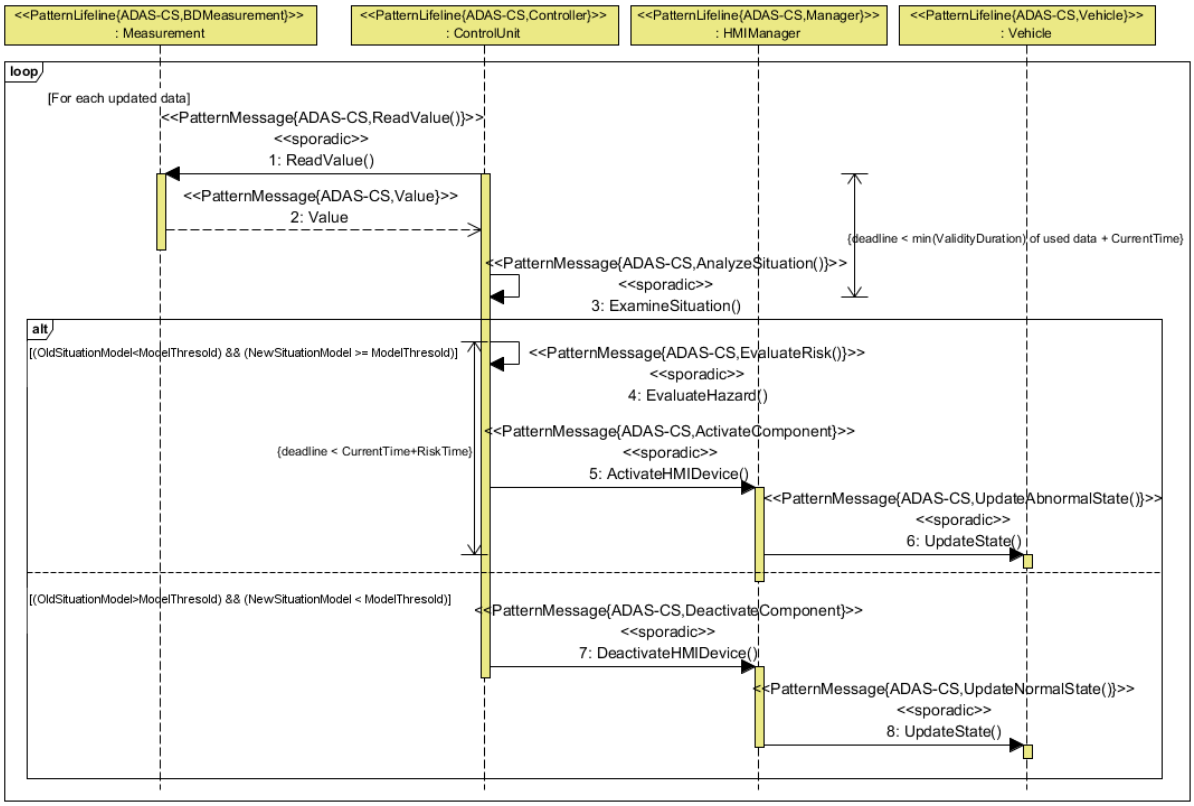


FIGURE 6.5 : Réutilisation du diagramme de séquence du patron *"Contrôle de données"*.

6.2.3.3 Réutilisation du patron *"Action"*

La figure 6.6 illustre la vue statique modélisant la fonctionnalité d'exécution des ordres de commande et/ou de déclenchement des alertes du système Lateral Safe en réutilisant le diagramme de classes du patron *"Action"*. La réutilisation de ce diagramme débute par le choix de variantes, puis l'adaptation des éléments selon les besoins. En effet, les classes *AutomaticActuator* et *AutomaticAction* sont supprimées lors de la modélisation du système vu qu'elles ne déclenchent pas l'activation des actionneurs pour l'exécution des ordres de commande. Cependant, les classes *Vehicle*, *Manager*, *HMIElement*, *WarningSignal*, *WarningSignalType*, *Driver* et *DriverAction* du patron sont respectivement instanciées par les classes *Vehicle*, *HMIManager*, *HMIDevice*, *AlarmSignal*, *AlarmModality*, *Driver* et *DriverAction*, qui sont identifiées à travers le stéréotype << *PatternClass* >> et ses propriétés *PatternName* et *Role*. Les classes *BeepSound* et *LightSymbol* représentent des variantes de la classe AlarmModality. Ainsi, elles sont stéréotypées << *PatternClass* >>. De même, tous les attributs et

toutes les opérations sont instanciés lors de la modélisation du diagramme de classes du système. Ils sont identifiés respectivement à l'aide des stéréotypes << *PatternAttribute* >> et << *PatternOperation* >>. En sus, les attributs *Direction* et *Volume* de la classe *BeepSound* et les attributs *Color*, *Flashing* et *Size* de la classe *LightSymbol* constituent des attributs spécifiques du système.

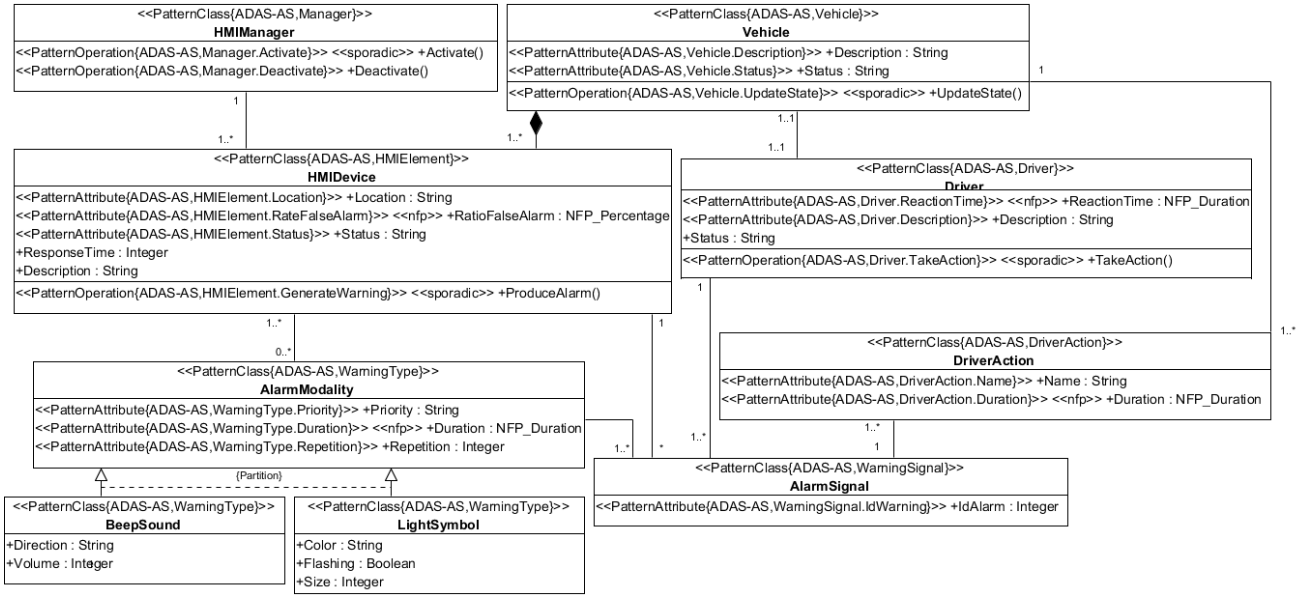


FIGURE 6.6 : Réutilisation du diagramme de classes du patron *"Action"*.

De son côté, la figure 6.7 montre la modélisation de la vue dynamique représentant la fonctionnalité d'exécution des ordres de commande et/ou de déclenchement des alertes du système Lateral Safe en réutilisant le diagramme de séquence du patron *"Action"*. D'une part, les objets *Manager*, *HMIElement* et *Driver* du patron sont respectivement instanciés par les objets *HMIManager*, *HMIDevice* et *Driver*. D'autre part, les messages *GenerateWarning()*, *TakeAction()* et *SendInformation()* sont respectivement instanciés par les messages *ProduceAlarm()*, *TakeAction()* et *ConveyInformation()*, alors que les messages *ExecuteAction()*, *Deactivate()* et le fragment combiné auquel ces messages sont associés sont supprimés dans le modèle du système du fait que ce système ne se base que sur le déclenchement des alertes pour éviter les risques.

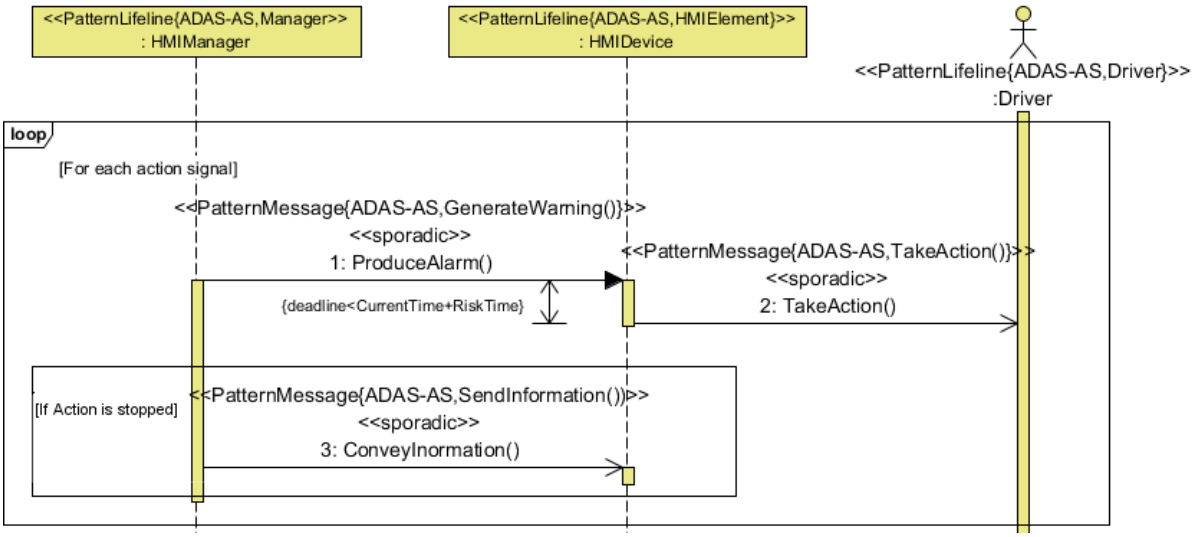


FIGURE 6.7 : Réutilisation du diagramme de séquence du patron *"Action"*.

6.3 Mise en œuvre des patrons de conception proposés

Pour automatiser la réutilisation des patrons, nous présentons, dans cette section, l'outil que nous avons développé pour la modélisation et la réutilisation des patrons, appelé *ADAS-Patterns* (Advanced Driver Assistance Systems Patterns). Nous avons implémenté cet outil vu que les outils de modélisation existants ne sont pas open-source (*e.g.,* Visual Paradigm [1]).

L'outil *ADAS-Patterns* a été développé sur la plateforme Eclipse [2], plus précisément en utilisant la bibliothèque Java Swing.

6.3.1 Eclipse

Eclipse permet de développer des Environnements de Développement de Logiciel (Integrated Development Environment, IDE) pour de nombreux langages : C, C++ et Java. Cette plateforme, qui a fait ses preuves, est fournie aux développeurs pour réaliser des développements informatiques en se basant sur des éditeurs, des constructeurs, des débogueurs et des outils permettant de gérer le développement. Eclipse n'est pas un logiciel monolithique, mais

1. http ://www.visual-paradigm.com/
2. http ://www.eclipse.org/platform/.

il utilise énormément le concept de modules, appelés plugins, dans son architecture. Il offre aussi un environnement flexible de conception d'IHM (Interface Homme Machine) en Swing.

Java Swing est une bibliothèque graphique pour le langage java. Il offre la possibilité de créer des interfaces graphiques en se basant sur les composants Swing (bouton, menu, zone de texte, etc.). En outre, grâce à cette bibliothèque, il est possible de dessiner des formes géométriques (ovale, rectangle, etc.) en se basant sur la classe java.awt.Graphics et les méthodes de Graphics (drawArc, drawOval, etc.).

6.3.2 Présentation de l'outil ADAS-Patterns

L'outil *ADAS-Patterns* permet d'appliquer les patrons proposés afin de faciliter la modélisation des fonctionnalités des systèmes d'aide à la conduite. En effet, cet outil assiste le concepteur d'applications dans le choix du patron approprié (*i.e., "Acquisition des données"*, *"Contrôle de données"* ou *"Action"*) et de l'adapter à un système d'aide à la conduite particulier.

Afin de réaliser ces trois patrons, nous avons implémenté plusieurs méthodes, parmi lesquelles, nous citons :

- La méthode permettant de réduire le modèle du patron en choisissant les variantes désirées. En effet, cette méthode permet d'afficher la liste des éléments optionnels et d'offrir la possibilité de choisir les variantes désirées comme le montre la figure 6.8. Tous les éléments dépendant de la variante à enlever sont aussi automatiquement supprimés du modèle. Cette méthode permet de contrôler l'étape de réduction d'un modèle d'un patron. En effet, elle ne permet pas de supprimer des éléments fondamentaux, stéréotypés << *mandatory* >>.
- La méthode permettant de renommer les éléments des modèles du patron selon les besoins de l'application cible. Cette méthode offre la possibilité de renommer les éléments (*e.g.,* classes, attributs, opérations, objets et messages) en se basant sur une comparaison linguistique des noms des termes à l'aide du dictionnaire du domaine que nous avons proposé (cf. étape 4 de la section 5.3.1 du chapitre 5), comme le montre

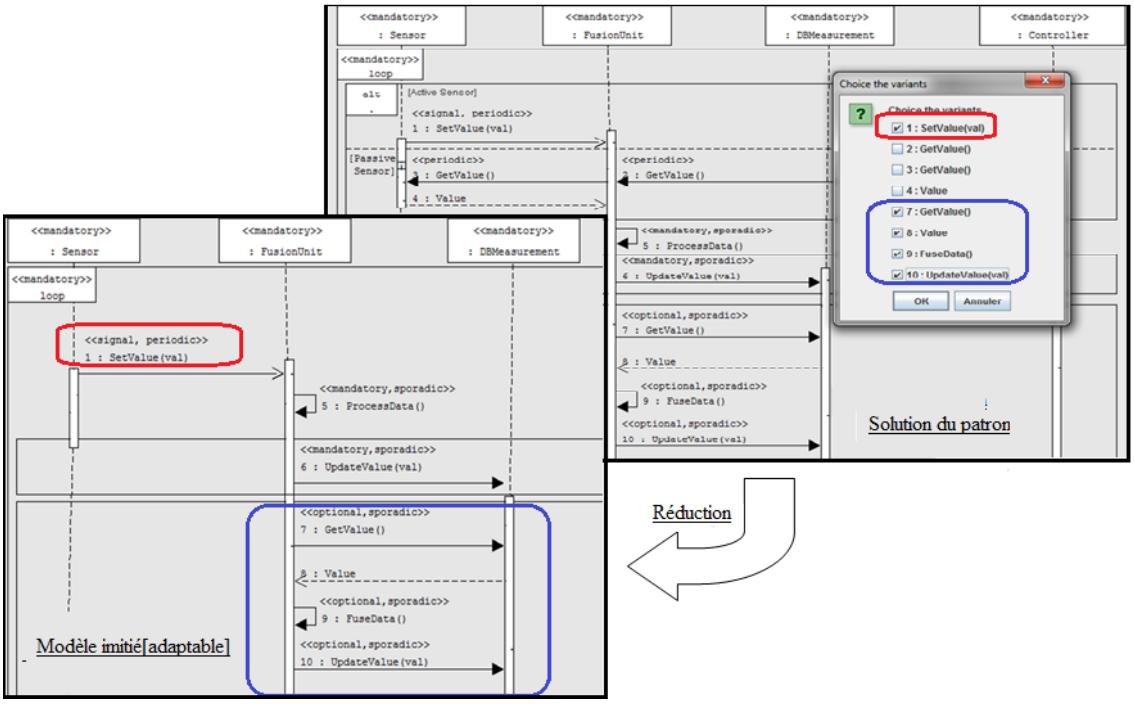


FIGURE 6.8 : Choix des variantes dans le diagramme de séquence du patron *"Acquisition de données"*.

la figure 6.9. En effet, cette méthode permet de vérifier la relation linguistique entre le nom de l'élément à renommer et le nom saisi par le concepteur d'applications. Si le nom saisi constitue un synonyme, un équivalent ou un variant, alors le nom de cet élément est remplacé par le nom saisi et il est identifié par le stéréotype correspondant (*i.e.*, $<< PatternCalss\{PatternName, Role\} >>$, $<< PatternAttribute\{PatternName, ClassName.Role\} >>$, $<< PatternOperation\{PatternName, ClassName.Role\} >>$, $<< PatternLifeline\{PatternName, Role\} >>$ et $<< PatternMessage\{PatternName, Role\} >>$). Par exemple, si la classe *Risk* du diagramme de classes du patron *"Acquisition des données"* est renommée par *Hazard* qui constitue un synonyme du concept risque, alors cette classe est stéréotypée automatiquement $<< PatternClass\{ADAS-DA, Risk\} >>$ (Figure 6.10).

- La méthode permettant d'ajouter des éléments au modèle d'un patron selon les besoins de l'application à modéliser. En effet, cette méthode offre la possibilité d'ajouter des éléments spécifiques à l'application cible en vérifiant le nom de l'élément à ajouter.

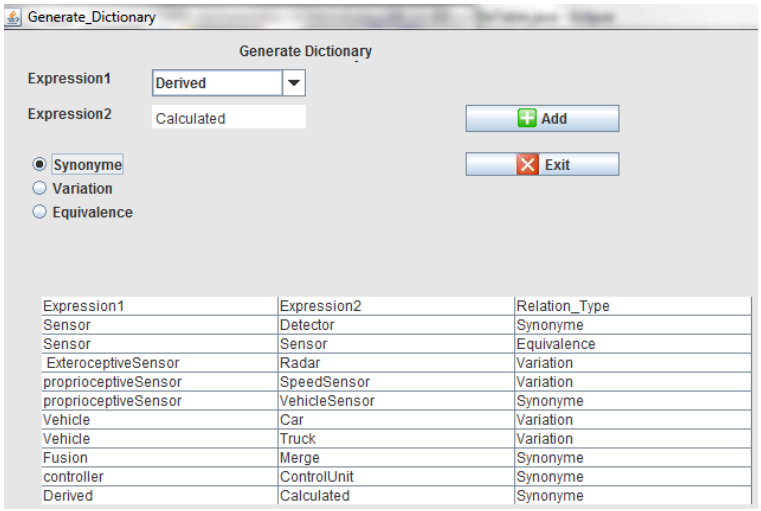


FIGURE 6.9 : Dictionnaire du domaine.

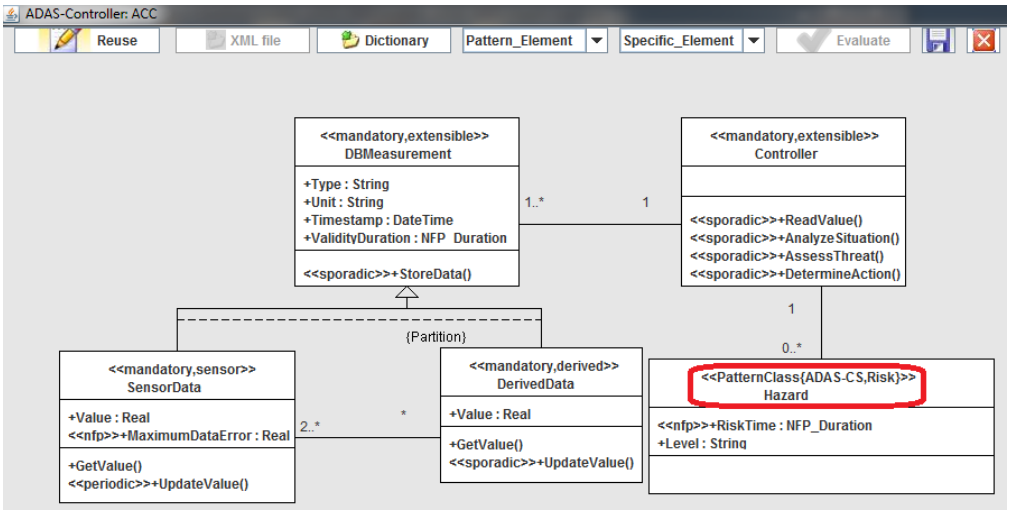


FIGURE 6.10 : Renommage de la classe *Risk* du patron *"Contrôle de données"*.

Dans cette situation, deux cas peuvent se présenter. Dans le premier cas, si l'élément à ajouter porte un nom qui représente un synonyme, un équivalent ou une variation d'un nom d'un élément du patron, alors une nouvelle instance de cet élément est créée tout en respectant certaines contraintes. Dans ce cas, tous les éléments en relation avec cette instance sont automatiquement créés. Par exemple, si une classe d'un modèle d'un patron a une multiplicité 1, alors une occurrence de cette classe ne peut participer qu'une seule fois à l'association. Par contre, si l'élément à ajouter porte le même nom que cette classe, alors un message d'erreur sera affiché. Dans le deuxième cas, si l'élément à ajouter porte un nom distinct du nom d'un élément du patron et n'est une

variation d'aucun élément de ce patron, alors l'élément est ajouté au modèle en tant qu'élément spécifique (Figure 6.11).

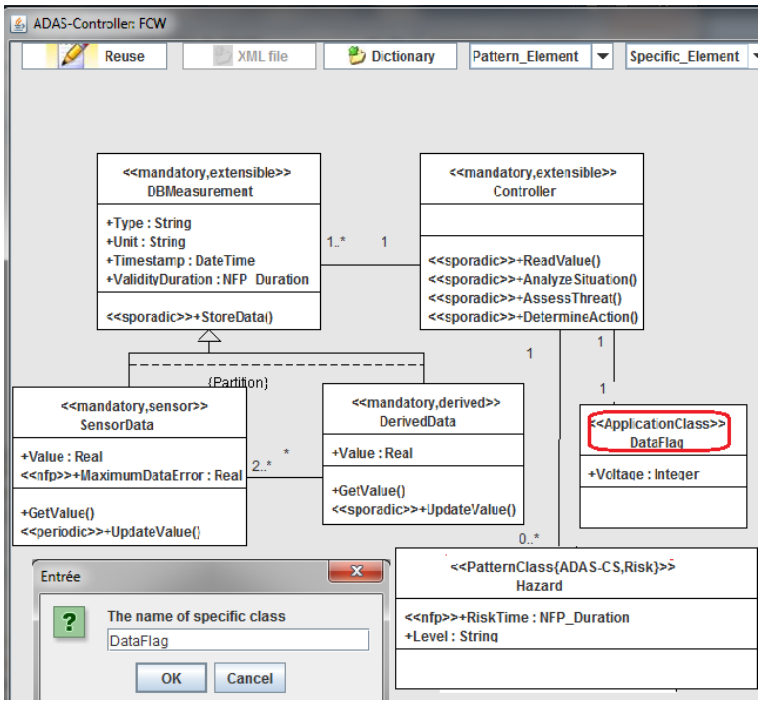


FIGURE 6.11 : Adjonction d'une classe spécifique au patron *"Contrôle de données"*.

Une fois que le patron choisi est adapté pour modéliser un système d'aide à la conduite particulier, l'outil *ADAS-Patterns* permet de générer le fichier XML correspondant à ce système (Figures 6.12 et 6.13). La figure 6.12 représente une partie du fichier XML du système ACC modélisé par la réutilisation du diagramme de classe du patron *"Acquisition des données"*. En effet, cette figure montre l'instanciation de la classe *Sensor* du patron et l'adjonction de la classe *Lane* en tant que classe spécifique. Nous présentons également dans la figure 6.13 une partie du fichier XML de ce système modélisé en réutilisant le diagramme de séquence du patron *"Acquisition des données"*. Cette figure montre l'instanciation des objets *Sensor*, *FusionUnit*, *DBMeasurement* et *Controller* du patron ainsi que l'instanciation des messages *SetValue()* et *ProcessData()*.

L'outil *ADAS-Patterns* permet d'évaluer les patrons proposés en se basant sur un ensemble de métriques de réutilisation qui sont calculées à partir des modèles des systèmes d'aide à la conduite obtenus par instanciation de patrons. Ces métriques permettent de cal-

```
<?xml version="1.0" encoding="UTF-8" ?>
- <PatternInstance name="ACC">
  - <classes>
    - <class name="Sensor" sterotype="<<PatternClass{ADAS-DA,Sensor}>>">
      - <attributs>
          <attribut name="Position" sterotype="<<PatternAttribute{ADAS-DA,Sensor_Location}>>"/>
          <attribut name="Accuracy" sterotype="<<PatternAttribute{ADAS-DA,Precision}>><<nfp>>"/>
          <attribut name="Periodicity" sterotype="<<PatternAttribute{ADAS-DA,Periodicity}>><<nfp>>"/>
          <attribut name="Freq" sterotype="<<PatternAttribute{ADAS-DA,Freq}>><<nfp>>"/>
          <attribut name="Type" sterotype="<<PatternAttribute{ADAS-DA,Type}>>"/>
        </attributs>
      - <operations>
          <operation name="Send()" sterotype="<<PatternOperation{ADAS-DA,Transmit()}>><<periodic>>"/>
        </operations>
      </class>

      .
      .
      .
    - <class name="Lane" sterotype="<<ApplicationClass>>">
      - <attributs>
          <attribut name="ID" />
          <attribut name="Type" />
          <attribut name="Width" />
        </attributs>
        <operations />
      </class>
    </classes>
</PatternInstance>
```

FIGURE 6.12 : Instanciation du diagramme de classes du patron *"Acquisition des données"*.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <PatternInstance name="ACC">
  - <classes>
    - <class name="Sensor" sterotype="<<PatternClass{ADAS-DA,Sensor}>>">
      - <attributs>
          <attribut name="Position" sterotype="<<PatternAttribute{ADAS-DA,Sensor_Location}>>"/>
          <attribut name="Accuracy" sterotype="<<PatternAttribute{ADAS-DA,Precision}>><<nfp>>"/>
          <attribut name="Periodicity" sterotype="<<PatternAttribute{ADAS-DA,Periodicity}>><<nfp>>"/>
          <attribut name="Freq" sterotype="<<PatternAttribute{ADAS-DA,Freq}>><<nfp>>"/>
          <attribut name="Type" sterotype="<<PatternAttribute{ADAS-DA,Type}>>"/>
        </attributs>
      - <operations>
          <operation name="Send()" sterotype="<<PatternOperation{ADAS-DA,Transmit()}>><<periodic>>"/>
        </operations>
      </class>

      .
      .
      .
    - <class name="Lane" sterotype="<<ApplicationClass>>">
      - <attributs>
          <attribut name="ID" />
          <attribut name="Type" />
          <attribut name="Width" />
        </attributs>
        <operations />
      </class>
    </classes>
</PatternInstance>
```

FIGURE 6.13 : Instanciation du diagramme de séquence du patron *"Acquisition des données"*.

culer le niveau de réutilisation des éléments du diagramme de classes (classes, attributs et opérations) et des éléments du diagramme de séquence (objets et messages) appartenant

aux patrons. Par conséquent, elles permettent de vérifier si la majorité de ces éléments de ces systèmes sont instanciés à partir des patrons ou non. Les figures 6.14 et 6.15 montrent respectivement des exemples de graphes générés automatiquement qui permettent d'illustrer les résultats de calcul des métriques de réutilisation des éléments du diagramme de classes et les éléments du diagramme de séquence lors de la réutilisation du patron *"Acquisition des données"* pour modéliser l'application ACC. Ces métriques font l'objet du chapitre suivant.

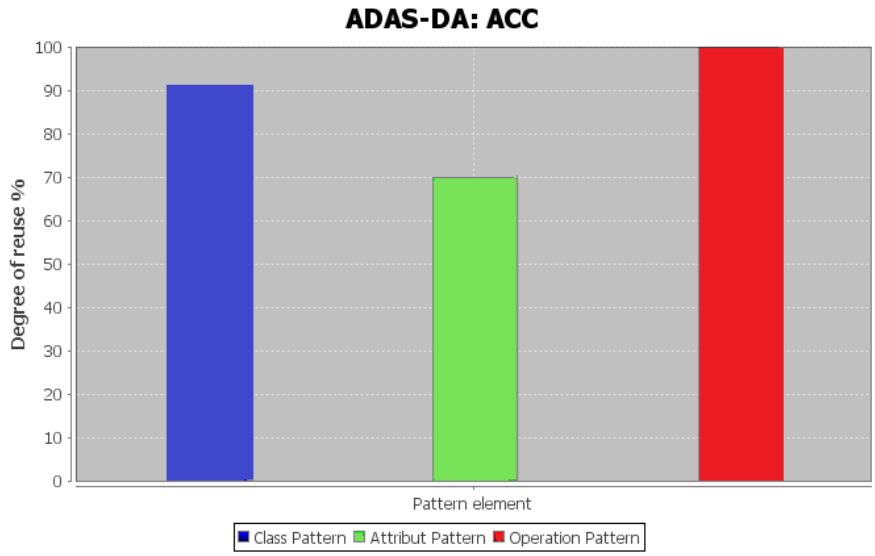


FIGURE 6.14 : Calcul des métriques de réutilisation après la réutilisation du diagramme de classes du patron *"Acquisition des données"*.

6.4 Conclusion

Dans ce chapitre, nous avons décrit un processus de réutilisation qui permet de guider un concepteur d'applications pour adapter correctement un patron aux besoins du système à modéliser. Ce processus est automatisé par le développement d'un prototype qui permet de spécifier et de réutiliser les patrons proposés afin de faciliter la modélisation des systèmes d'aide à la conduite et de gagner au niveau du temps de développement.

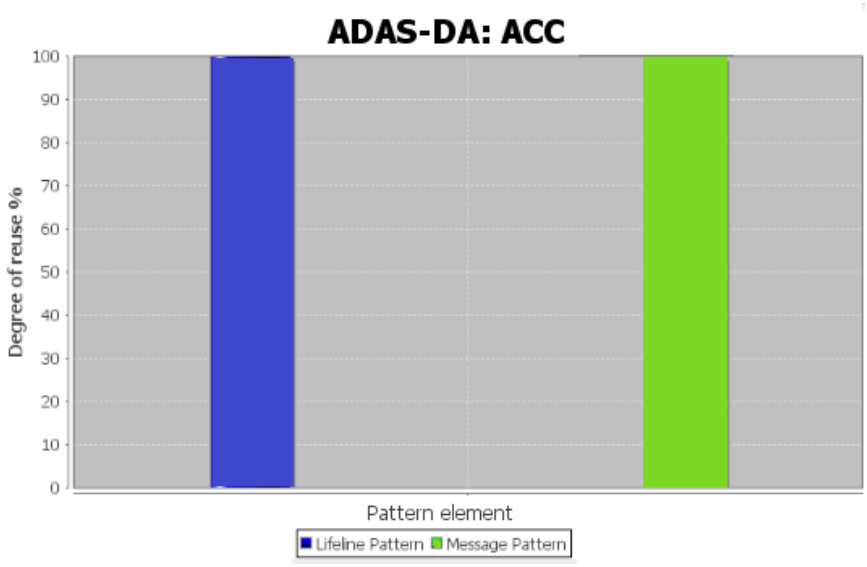


FIGURE 6.15 : Calcul des métriques de réutilisation après la réutilisation du diagramme de séquence du patron *"Acquisition des données"*.

Ce prototype peut bénéficier d'améliorations, particulièrement en ce qui concerne l'intégration des instances des patrons pour créer un modèle complet d'un système cible. Cette intégration n'est pas traité dans nos travaux.

Afin de valider les patrons proposés, nous procédons, dans le chapitre suivant, à une évaluation de ces patrons reposant sur des métriques.

Chapitre 7

Évaluation des patrons de conception proposés

7.1 Introduction

Afin d'évaluer l'efficacité des patrons de conception, il est important de vérifier si les modèles de patrons couvrent bien les exigences identifiées du domaine de l'aide à la conduite. Il est nécessaire aussi de vérifier que ces modèles représentent convenablement les concepts du domaine. C'est pourquoi, nous proposons dans ce chapitre de définir de nouvelles métriques pour mesurer la qualité des modèles de patrons pour la réutilisation, c'est-à-dire de vérifier la qualité de ces modèles en termes de représentation de la majorité des concepts du domaine. Nous proposons également d'adapter certaines métriques existantes pour mesurer le degré de réutilisation au niveau des diagrammes de classes et des diagrammes de séquence. Ces métriques sont ensuite appliquées pour évaluer la qualité des patrons que nous avons proposés.

7.2 Métriques d'évaluation

La réutilisabilité et la réutilisation sont deux aspects différents dans le développement orienté objet des systèmes d'information. La réutilisabilité est la possibilité qu'un composant puisse être réutilisé, alors que la réutilisation est l'utilisation des composants existants pour construire de nouveaux systèmes plutôt que de commencer à partir de zéro. Nous proposons alors deux types de métriques pour évaluer la qualité des modèles de patrons : métriques de réutilisabilité et métriques de réutilisation. Les métriques permettant d'évaluer les diagrammes de classes des patrons ont fait l'objet de la publication [Rekhis et al., 2013b].

7.2.1 Métriques de réutilisabilité

Les métriques de réutilisabilité estiment la possibilité qu'un composant soit réutilisable et permettent d'identifier la qualité d'un composant pour la réutilisation. Néanmoins, ces métriques ne permettent pas de mesurer le degré de réutilisation d'un composant dans une application particulière. Dans ce sens, différents travaux sont basés sur la définition des métriques de réutilisabilité pour les patrons. Parmi ces travaux, nous citons les travaux de Chidamber et Kemerer [Chidamber and Kemerer, 1994] qui ont défini les métriques *Depth of Inheritance Tree (DIT) of a class*, *Number of Children (NOC)* et *Coupling Between Object classes (CBO)*, etc. Ces métriques sont utilisées par Bhatia *et al.* [Bhatia and Mann, 2008] pour définir une formule pour la réutilisabilité basée sur le principe que les métriques *NOC* et *DIT* ont un effet positif sur la réutilisabilité d'une classe, alors que la métrique *CBO* en a un impact négatif. En outre, Gill et *al.* [Gill and Sikka, 2011] ont proposé les métriques *Breadth of Inheritance Tree (BIT)*, *Method Reuse Per Inheritance Relation (MRPIR)*, *Attribute Reuse Per Inheritance Relation (ARPIR)*, *Generality of Class (GC)* et *Reuse Probability (RP)* pour mesurer la réutilisabilité des classes en se basant sur les relations d'héritage.

Toutes ces métriques permettent d'estimer la probabilité de réutilisation d'un patron et d'évaluer sa qualité de conception, à savoir si la valeur de la métrique CBO augmente, la réutilisabilité diminue, et par conséquent, il devient difficile de modifier le système d'information.

Néanmoins, ces métriques ne permettent pas de mesurer la qualité d'un patron en termes d'encapsulation des spécificités du domaine de l'aide à la conduite. Pour pallier cette insuffisance, nous définissons de nouvelles métriques pour mesurer la réutilisabilité des éléments du diagramme de classes (classes, attributs et opérations) et des éléments du diagramme de séquence (objets et messages) des patrons de conception. L'objectif de ces métriques est de vérifier si nous retrouvons ou non les éléments des patrons dans des modèles d'applications modélisés sans réutilisation des patrons. Ces métriques sont calculées à partir de deux modélisations d'une même application. La première est modélisée sans réutilisation de patrons, alors que la deuxième est modélisée par la réutilisation de patrons. Les valeurs de ces métriques varient de 0 à 1. Si les valeurs de ces métriques s'approchent de 1, alors nous pouvons en conclure que le patron représente convenablement le domaine de l'aide à la conduite. Par contre, si les valeurs s'approchent de 0, cela signifie que les patrons ne représentent pas une solution adéquate pour le domaine.

7.2.1.1 Métrique de réutilisabilité pour le diagramme de classes

Métrique 1 : Réutilisabilité de Classes (RC)

La métrique RC est définie comme le ratio entre le nombre de Classes Identifiées du Patron (CIP) dans le diagramme de classes d'une application modélisée sans réutilisation du patron et le nombre de Classes Réutilisées du Patron (CRP) pour la même application modélisée par la réutilisation du patron, tel que c'est formalisé par l'équation 7.1.

Considérons un diagramme de classes comportant n classes : C_1, C_2,..., C_n.

$$RC = \frac{\sum_{i=1}^{n} CIP(C_i)}{\sum_{i=1}^{n} CRP(C_i)} \tag{7.1}$$

où,

$$CIP(C_i) = \begin{cases} 1 & \text{si } \textit{la classe identifiée représente une classe du patron} \\ 0 & \text{sinon.} \end{cases}$$

$$CRP(C_i) = \begin{cases} 1 & \text{si } \textit{la classe est réutilisée à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

Métrique 2 : Réutilisabilité d'Attributs (RA)

La métrique RA est définie comme le ratio entre le nombre d'Attributs Identifiés du Patron (AIP) dans le diagramme de classes d'une application modélisée sans réutilisation du patron et le nombre d'Attributs Réutilisées du Patron (ARP) pour la même application modélisée par la réutilisation du patron, comme c'est formalisé par l'équation 7.2.

Considérons un diagramme de classes comportant n classes : C_1, C_2,..., C_n et m_i attributs a_1, a_2,..., a_{m_i} pour chaque classe C_i.

$$RA = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m_i} AIP(a_{ij})}{\sum_{i=1}^{n}\sum_{j=1}^{m_i} ARP(a_{ij})} \tag{7.2}$$

où,

$$AIP(a_{ij}) = \begin{cases} 1 & \text{si } \textit{l'attribut identifiée représente un attribut d'une classe du patron} \\ 0 & \text{sinon.} \end{cases}$$

$$ARP(a_{ij}) = \begin{cases} 1 & \text{si } \textit{l'attribut est réutilisée à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

Métrique 3 : Réutilisabilité d'Opérations (RO)

La métrique RO est définie comme le ratio entre le nombre d'Opérations Identifiées du Patron (OIP) dans le diagramme de classes d'une application modélisée sans réutilisation du

patron et le nombre d'Opérations Réutilisées du Patron (ORP) pour la même application modélisée par la réutilisation du patron, tel que c'est formalisé par l'équation 7.3.

Considérons un diagramme de classes comportant n classes : C_1, C_2,..., C_n et k_i opérations op_1, op_2,..., op_{k_i} pour chaque classe C_i.

$$OR = \frac{\sum_{i=1}^{n}\sum_{q=1}^{k_i} OIP(op_{iq})}{\sum_{i=1}^{n}\sum_{q=1}^{k_i} ORP(op_{iq})} \tag{7.3}$$

où,

$$OIP(op_{iq}) = \begin{cases} 1 & \text{si } \textit{l'opération identifiée représente une opération d'une classe du patron} \\ 0 & \text{sinon.} \end{cases}$$

$$ORP(op_{iq}) = \begin{cases} 1 & \text{si } \textit{l'opération est réutilisée à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

7.2.1.2 Métrique de réutilisabilité pour le diagramme de séquence

Métrique 4 : Réutilisabilité d'OBjets (ROB)

La métrique ROB est définie comme le ratio entre le nombre d'OBjets Identifiées du Patron (OBIP) dans le diagramme de séquence d'une application modélisée sans réutilisation du patron et le nombre d'OBjets Réutilisées du Patron (OBRP) pour la même application modélisée par la réutilisation du patron, tel que c'est formalisé par l'équation 7.4.

Considérons un diagramme de séquence comportant n objets : O_1, O_2,..., O_n.

$$ROB = \frac{\sum_{i=1}^{n} OBIP(O_i)}{\sum_{i=1}^{n} OBRP(O_i)} \tag{7.4}$$

où,

$$OBIP(O_i) = \begin{cases} 1 & \text{si } l'objet\ identifiée\ représente\ un\ objet\ du\ patron \\ 0 & \text{sinon.} \end{cases}$$

$$OBRP(O_i) = \begin{cases} 1 & \text{si } l'objet\ est\ réutilisée\ à\ partir\ du\ patron \\ 0 & \text{sinon.} \end{cases}$$

Métrique 5 : Réutilisabilité de Messages (RM)

La métrique RM est définie comme le ratio entre le nombre de Messages Identifiées du Patron (MIP) dans le diagramme de séquence d'une application modélisée sans réutilisation du patron et le nombre de Messages Réutilisées du Patron (MRP) pour la même application modélisée par la réutilisation du patron, tel que c'est formalisé par l'équation 7.5.

Considérons un diagramme de séquence comportant m objets : M_1, M_2,..., M_m.

$$RM = \frac{\sum_{i=1}^{n} MIP(M_i)}{\sum_{i=1}^{m} MRP(MO_i)} \tag{7.5}$$

où,

$$MIP(M_i) = \begin{cases} 1 & \text{si } le\ message\ identifiée\ représente\ un\ message\ du\ patron \\ 0 & \text{sinon.} \end{cases}$$

$$MRP(M_i) = \begin{cases} 1 & \text{si } \textit{le message est réutilisée à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

7.2.2 Métriques de réutilisation

Certains travaux ont défini des métriques de réutilisation qui permettent de mesurer le nombre de lignes de code réutilisées par rapport au nombre total de lignes de code pour un logiciel particulier. Parmi ces travaux, nous citons ceux de Frakes and Terry [Frakes and Terry, 1996] où les auteurs ont défini les métriques : *Reuse Level (RL)* et *Reuse Frequency (RF)* qui sont mesurées pour différentes granularité (*e.g.,* lignes de code, fonctions et projets) et pour différents codes sources (*e.g.,* C, C++ et Java). De même, Aggarwal *et al.* ont défini dans [Aggarwal et al., 2005] deux métriques pour mesurer le degré de réutilisation des classes et des fonctions dans un logiciel orienté objet reposant sur la programmation générique sous forme de *template*. La première métrique, nommée *Class Template Factor (CTF)*, est définie comme le ratio entre le nombre de classes utilisant les templates et le nombre total de classes dans un code source. La deuxième métrique, nommée *Function Template Factor (FTF)*, est définie comme le ratio entre le nombre de fonctions utilisant les *templates* et le nombre total de fonctions.

Ces métriques de réutilisation se basent essentiellement sur la mesure du degré de réutilisation des codes sources utilisés pendant les dernières phases du cycle de développement d'un logiciel, mais elles ne considèrent pas les premières phases telles que l'analyse et la conception. En effet, l'analyse et la conception constituent des phases essentielles pour développer des logiciels de qualité, du fait qu'elles ont un impact sur le coût et le temps de développement. D'une part, nous proposons, face à ce constat d'adapter les métriques définies par Aggarwal *et al.* (*i.e.,* les métriques *Class Template Factor (CTF)* et *Function Template Factor (FTF)*) pour mesurer le niveau de réutilisation des classes et des opérations dans le diagramme de classes d'un patron. D'autre part, nous proposons de définir une autre métrique pour mesurer le degré de réutilisation des attributs vu qu'une classe possède deux parties importantes : les opérations, qui représentent les tâches exécutées par cette classe, et les attributs, qui repré-

sentent les propriétés de cette classe. Dans le même contexte, nous proposons de définir deux autres métriques pour mesurer le niveau de réutilisation des objets et des messages dans un diagramme de séquence d'un patron. En effet, la solution d'un patron est représentée à la fois par un diagramme de classes, qui modélise la vue statique et par un diagramme de séquence, qui modélise la vue dynamique. L'objectif de ces métriques est de vérifier si la majorité des éléments des applications sont instanciés à partir des patrons ou non. Les valeurs de ces métriques varient de 0 à 1. Si la valeur s'approche de 1, alors nous pouvons constater que la majorité des éléments du patron sont réutilisés et que le concepteur n'a besoin d'ajouter qu'un petit nombre d'éléments spécifiques à l'application.

7.2.2.1 Métrique de réutilisation pour le diagramme de classes

Métrique 1 : Degré de Réutilisation de Classes (DRC)

La métrique DRC est définie comme le ratio entre le nombre de Classes Réutilisées (CR) à partir d'un patron et le nombre total de classes constituant le diagramme de classes d'une application modélisée par réutilisation de ce patron, comme le montre l'équation 7.6.

Considérons un diagramme de classes correspondant à une instance du patron et comportant n classes C_1, C_2, ..., C_n.

$$DRC = \frac{\sum_{i=1}^{n} CR(C_i)}{n} \tag{7.6}$$

où,

$$CR(C_i) = \begin{cases} 1 & \text{si } \textit{la classe est réutilisée à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

Métrique 1 : Degré de Réutilisation d'Opérations (DRO)

La métrique DRO est définie comme le ratio entre le nombre d'Opérations Réutilisées (OR)

à partir d'un patron et le nombre total des opérations dans le diagramme de classes d'une application modélisée par réutilisation de ce patron, comme le montre l'équation 7.7.

Considérons un diagramme de classes correspondant à une instance du patron et comportant n classes C_1, C_2, ..., C_n et k_i opérations op_1, op_2,..., op_{k_i} pour chaque classe C_i.

$$DRO = \frac{\sum_{i=1}^{n}\sum_{q=1}^{k_i} OR(op_{iq})}{\sum_{i=1}^{n} k_i} \tag{7.7}$$

où,

$$OR(op_{iq}) = \begin{cases} 1 & \text{si } \textit{l'opération est réutilisée à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

Métrique 1 : Degré de Réutilisation d'Attributs (DRA)

La métrique DRA est définie comme le ratio entre le nombre des Attributs Réutilisés (AR) à partir d'un patron et le nombre total d'attributs dans le diagramme de classes d'une application modélisée par réutilisation de ce patron, comme le montre l'équation 7.8.

Considérons un diagramme de classes correspondant à une instance du patron et comportant n classes C_1, C_2, ..., C_n et m_i attributs a_1, a_2,..., a_{m_i} pour chaque classe C_i.

$$DRA = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m_i} AR(a_{ij})}{\sum_{i=1}^{n} m_i} \tag{7.8}$$

où,

$$AR(a_{ij}) = \begin{cases} 1 & \text{si } \textit{l'attribut est réutilisé à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

7.2.2.2 Métrique de réutilisation pour le diagramme de séquence

Métrique 4 : Degré de Réutilisation d'OBjets (DROB)

La métrique DROB est définie comme le ratio entre le nombre d'OBjets Réutilisés (OBR) à partir d'un patron et le nombre total des objets dans le diagramme de séquence d'une application modélisée par réutilisation de ce patron, comme le montre l'équation 7.9.

Considérons un diagramme de séquence correspondant à une instance du patron et comportant n objets O_1, O_2, ..., O_n.

$$DROB = \frac{\sum_{i=1}^{n} OBR(O_i)}{n} \tag{7.9}$$

où,

$$OBR(O_i) = \begin{cases} 1 & \text{si } \textit{l'objet est réutilisée à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

Métrique 4 : Degré de Réutilisation de Messages (DRM)

La métrique DRM est définie comme le ratio entre le nombre de Messages Réutilisés (MR) à partir d'un patron et le nombre total des messages dans le diagramme de séquence d'une application modélisée par réutilisation de ce patron, comme le montre l'équation 7.10.

Considérons un diagramme de séquence correspondant à une instance du patron et comportant m messages M_1, M_2, ..., M_m.

$$DRM = \frac{\sum_{i=1}^{m} MR(M_i)}{m} \tag{7.10}$$

où,

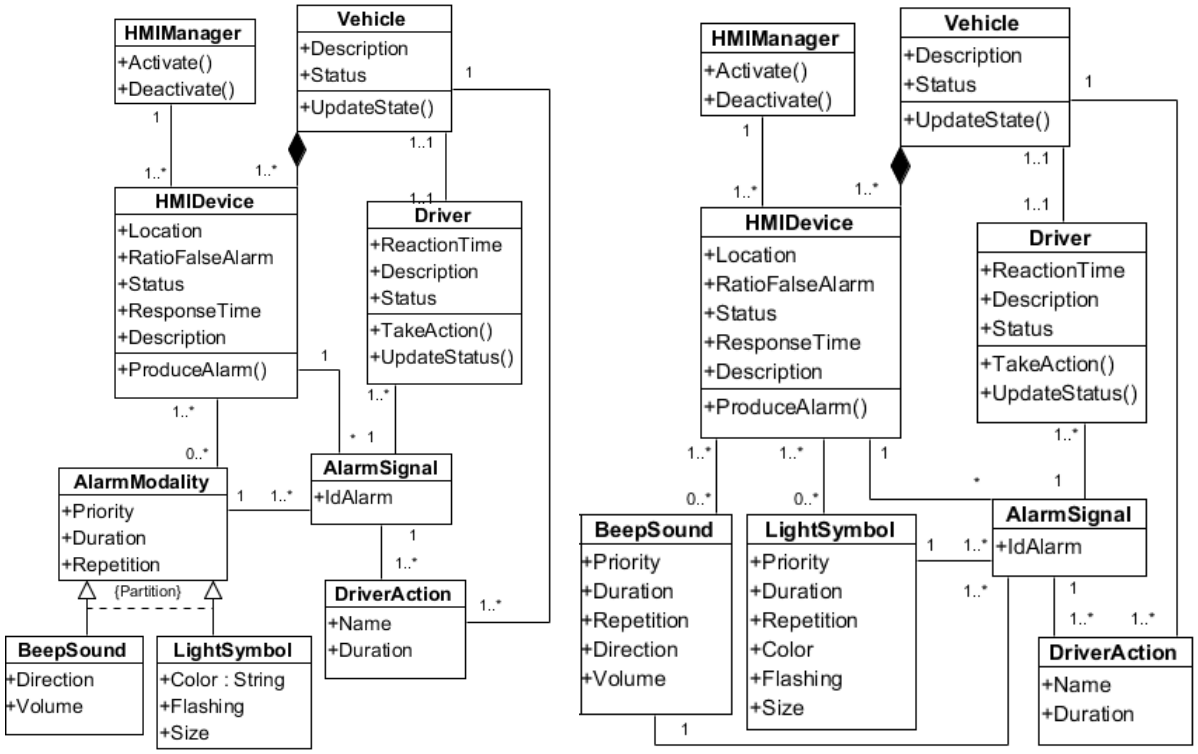
$$MR(M_i) = \begin{cases} 1 & \text{si } \textit{lemessage est réutilisée à partir du patron} \\ 0 & \text{sinon.} \end{cases}$$

7.3 Évaluation des patrons de conception

Dans cette section, nous nous sommes intéressés à l'évaluation des patrons que nous avons proposés dans le chapitre 5 en utilisant les métriques décrites ci-dessus. Pour le faire, nous calculons ces métriques pour six applications d'aide à la conduite qui ont été modélisées par deux enseignants spécialistes en conception UML, sans réutilisation des patrons. Ces applications sont aussi modélisées par réutilisation des patrons par d'autres enseignants. Une fois que les diagrammes de classes et de séquence modélisés par les enseignants ont été réalisés, nous identifions les fragments (*i.e.,* de classes et de séquence) relatifs à chaque patron de conception d'aide à la conduite TR en appliquant une étape de décomposition comme c'est décrit dans le chapitre 5. La modélisation de chaque diagramme d'un système peut être effectuée de différentes manières, et par conséquent, le résultat du calcul des métriques peut être différent d'une modélisation à une autre. En effet, si des éléments jouent le même rôle dans l'application, alors le concepteur possède deux possibilités de modélisation : il peut utiliser la relation d'héritage ou non. Ces deux possibilités sont correctes, mais le calcul des métriques peut donner des résultats différents. Pour éliminer cette ambiguïté et avoir les mêmes résultats quel que soit le modèle, nous proposons de considérer les classes spécialisées en tant que classes réutilisées à partir du patron et de ne pas compter la classe générique. Dans le même contexte, nous proposons de calculer, pour chaque classe spécialisée, la somme de ses propres attributs (respectivement opérations) et des attributs (respectivement opérations) hérités de la classe générique. Par exemple, nous considérons, dans la figure 7.1, deux modèles correspondant à une instanciation du patron *"Action"* pour la même application, élaborés de deux manières différentes. La figure 7.1(a) représente le fragment qui comporte deux relations d'héritage, alors que la figure 7.1(b) représente le même fragment sans utilisation de la relation d'héritage. Dans la figure 7.1(a), nous considérons que les classes *HMIManager*, *HMIDevice*, *AlarmSignal*, *AlarmModality*, *BeepSound*, *LightSymbol*, *Driver*, *DriverAction* et *Vehicle* sont

instanciées à partir du patron *"Action"*. Dans ce cas, le résultat pour la métrique DRC est égal à $\frac{7}{9}$. Cependant, nous considérons dans la figure 7.1(b) que les classes *HMIManager*, *BeepSound*, *LightSymbol*, *AlarmSignal*, *Driver*, *DriverAction* et *Vehicle* sont instanciées à partir du patron *"Action"*. Dans ce cas, le résultat pour la métrique DRC est égal à $\frac{7}{7}$. Étant donné que les bips sonores et les symboles lumineux sont deux types d'alertes, nous proposons de les considérer en tant que deux classes instanciées à partir du patron dans le modèle présenté dans la figure 7.1(b) pour obtenir le même résultat de la métrique DRC quel que soit le modèle.

Concernant les attributs (respectivement les opérations), la valeur de la métrique DRA est égale à $\frac{16}{24}$, du fait que les attributs *Priority*, *Duration* et *Repetition* sont calculés deux fois pour avoir les mêmes résultats de calcul des métriques.



(a) Modèle avec héritage

(b) Modèle sans héritage

FIGURE 7.1 : Deux modèles différents de la même application.

7.3.1 Évaluation du patron *"Acquisition des données"*

Nous présentons dans le tableau 7.1 les résultats du calcul des métriques de réutilisabilité et des métriques de réutilisation obtenus pour le diagramme de classes du patron *"Acquisition des données"*. Rappelons que ce patron est réutilisé pour modéliser six applications d'aide à la conduite : FCW (Frontal Collision Warning), Saspence, Compose, Safelane, RDCW (Road Departure Crash Warning) et BLA (Blind Spot Assist). Une description de ces applications modélisées avec et sans réutiliser les patrons est détaillée en Annexe B.

Les valeurs des métriques de réutilisabilité obtenues montrent que la majorité des classes, attributs et opérations du patron *"Acquisition des données"* qui sont réutilisés, sont identifiés. Par exemple, les valeurs des métriques de réutilisabilité obtenues pour l'application *Safelane* indiquent que, lors de la modélisation de cette application sans réutiliser le patron, 90% de classes du patron sont identifiées, 82% des attributs sont identifiés et 60% des opérations sont aussi identifiées. Ceci nous a permis de conclure que le diagramme de classes du patron *"Acquisition des données"* représente convenablement les concepts et les exigences de la fonctionnalité d'acquisition des données.

Concernant les métriques de réutilisation, elles ont des valeurs qui indiquent que la majorité des classes, des attributs et des opérations des modèles d'applications sont instanciés à partir du patron. Par exemple, les valeurs des métriques de réutilisation obtenues pour l'application Saspence montrent que 90% des classes (DRC = 0.9), 74% des attributs (DRA = 0.74) et 93% des opérations (DRO = 0.93) sont instanciés à partir du diagramme de classes du patron *"Acquisition des données"*. Il y a même des cas (applications FCW et BLA) où toutes les classes représentent des instances des classes du patron (DRC = 1), c'est-à-dire qu'aucune classe spécifique n'est ajoutée lors de la modélisation de l'application. Par conséquent, nous constatons un bon niveau de réutilisation des éléments du diagramme de classes du patron *"Acquisition des données"* pendant la modélisation des systèmes d'aide à la conduite.

Par ailleurs, nous présentons dans le tableau 7.2 les résultats du calcul des métriques de réutilisabilité et des métriques de réutilisation obtenus pour le diagramme de séquence du patron *"Acquisition des données"*. Ce tableau montre que les mesures de réutilisation sont

très élevées : dans la majorité des systèmes, tous les objets (DROB = 1) et les messages (DRM = 1) du patron *"Acquisition des données"* correspondent à des instances des éléments du diagramme de séquence de ce patron. Ainsi, nous déduisons, pour tous les fragments des systèmes d'aide à la conduite relatifs au patron *"Acquisition des données"*, que la majorité des éléments (objets et messages) modélisés sont réutilisés à partir de ce patron. Cela signifie que les éléments spécifiques ajoutés à ces systèmes sont mineurs par rapport aux éléments réutilisés dans la majorité des cas.

De plus, ce tableau montre que tous les objets réutilisés sont identifiés (ROB = 1) dans toutes les applications modélisées sans réutilisation du patron *"Acquisition des données"*. Il permet également de montrer que, dans la plupart des cas (*e.g.,* FCW, Saspence, BLA, Safelane et RDCW), la moitié des messages réutilisés du patron *"Acquisition des données"* sont identifiés, vu que la majorité des valeurs sont proches de $\frac{1}{2}$. Ceci signifie que le diagramme de séquence de ce patron représente convenablement les concepts de la fonctionnalité d'acquisition et leur comportement.

Pour conclure, les valeurs des métriques de réutilisabilité obtenues pour les diagrammes de classes et de séquence montrent que le patron *"Acquisition des données"* a une bonne aptitude à être réutilisé pour la modélisation de la fonctionnalité d'acquisition des données.

Tableau 7.1 : Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de classes du patron *"Acquisition des données"*.

		FCW	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
Métriques de réutilisation	**DRC**	0.9	0.9	0.92	1	0.91	1
	DRA	0.68	0.74	0.58	0.68	0.67	0.7
	DRO	1	0.93	0.88	0.93	0.94	0.86
Métriques de réutilisabilité		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	RC	0.89	0.89	0.9	0.78	0.9	0.83
	RA	0.64	0.75	0.78	0.59	0.82	0.57
	RO	0.43	0.75	0.53	0.54	0.6	0.56

Tableau 7.2 : Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de séquence du patron *"Acquisition des données"*.

Métriques de réutilisation		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	DROB	1	0.8	1	1	1	1
	DRM	1	0.93	1	0.9	1	0.86
Métriques de réutilisabilité		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	ROB	1	1	1	1	1	1
	RM	0.5	0.62	0.46	0.67	0.5	0.54

7.3.2 Évaluation du patron *"Contrôle de données"*

Les valeurs obtenues pour les métriques de réutilisation et de réutilisabilité pour le diagramme de classes et le diagramme de séquence du patron *"Contrôle de données"* sont représentées respectivement dans le tableau 7.3 et le tableau 7.4. Ces tableaux montrent que dans la majorité des applications, toutes les classes (DRC = 1) et tous les objets (DROB = 1) sont réutilisés à partir du patron *"Contrôle de données"*. Dans ce cas, les concepteurs n'ont besoin que d'ajouter quelques attributs et/ou opérations et quelques messages spécifiques aux applications modélisées. Ces tableaux montrent également que la majorité des éléments (attributs, opérations et messages) modélisés sont réutilisés à partir de ce patron. Par exemple, les valeurs obtenues pour l'application *Safelane* (c.f. tableau 7.3) montrent que 100% des classes (DRC = 1), 70% des attributs (DRO = 0.7) et 90% des opérations (DRO = 0.9) sont réutilisés à partir du patron *"Contrôle de données"*. Nous pouvons donc conclure que le degré de réutilisation des éléments de ce patron est élevé.

Par ailleurs, nous constatons, d'après les tableaux 7.3 et 7.4, que la majorité des éléments (classes, attributs, objets et messages) qui sont réutilisés, sont identifiés. Par exemple, les valeurs des métriques de réutilisabilité obtenues, dans le tableau 7.3, pour l'application Compose montrent que 80% des classes (RC = 0.80) du patron sont identifiées, 63% des attributs (RA = 0.63) sont identifiés et 56% des opérations (RO = 0.56) sont identifiées. De même, les valeurs des métriques de réutilisabilité obtenues, dans le tableau 7.4, pour l'application Compose montrent que 100% des objets (ROB = 1) du patron sont identifiés et 87% des messages

(RM = 0.87) du patron sont identifiés. Ainsi, ces valeurs montrent que le patron *"Contrôle de données"* représente convenablement les concepts de la fonctionnalité de contrôle et de traitement des données.

Tableau 7.3 : Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de classes du patron *"Contrôle de données"*.

Métriques de réutilisation		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	DRC	0.83	1	1	1	1	1
	DRA	0.75	0.75	0.69	0.82	0.7	0.64
	DRO	0.9	1	1	0.9	0.9	0.82
Métriques de réutilisabilité		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	RC	0.8	1	0.8	0.6	0.6	0.6
	RA	0.71	0.67	0.63	0.44	0.67	0.56
	RO	0.56	0.56	0.56	0.56	0.44	0.44

Tableau 7.4 : Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de séquence du patron *"Contrôle de données"*.

Métriques de réutilisation		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	DROB	0.8	0.8	1	1	1	1
	DRM	0.8	0.8	1	0.89	0.89	0.89
Métriques de réutilisabilité		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	ROB	1	0.75	1	0.75	1	0.75
	RM	0.5	0.75	0.87	0.75	0.75	0.75

7.3.3 Évaluation du patron *"Action"*

Les valeurs obtenues pour les métriques de réutilisation et de réutilisabilté pour le patron *"Action"* sont représentées dans le tableau 7.5 (pour le diagramme de classes) et le tableau 7.6 (pour le diagramme de séquence). Ces tableaux montrent que toutes les valeurs des métriques de réutilisation (DRC, DRA, DRO, DROB et DRM), calculées pour chaque application, sont supérieures ou égales à $\frac{1}{2}$. Ceci signifie que la réutilisation des éléments de ce patron est

intéressante pour modéliser la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes.

Par ailleurs, les tableaux 7.5 et 7.6 montrent que les valeurs obtenues pour les métriques de réutilisabilité RC, RO, ROB et RM sont supérieures à $\frac{1}{2}$. Ceci prouve que les classes, les attributs, les objets et les messages qui sont réutilisés du patron, sont identifiés. Néanmoins, ces tableaux montrent que, dans certaines applications, les valeurs obtenues pour la réutilisabilité des attributs sont inférieures à $\frac{1}{2}$ (par exemple 44% des attributs réutilisées sont identifiées pour l'application FCW et 42% des attributs réutilisées sont identifiées pour l'application Saspence). Ces valeurs faibles s'expliquent par le fait que les diagrammes de classes modélisés par certains enseignants représentent généralement la plupart des opérations à réutiliser mais, moins d'attributs.

Tableau 7.5 : Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de classes du patron *"Action"*.

		FCW	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
Métriques de réutilisation	**DRC**	1	1	1	1	1	1
	DRA	0.57	0.66	0.71	0.65	0.65	0.66
	DRO	0.83	0.71	0.67	0.7	0.67	0.63
Métriques de réutilisabilité		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	RC	0.63	0.76	0.78	0.71	0.78	0.78
	RA	0.44	0.42	0.47	0.46	0.71	0.63
	RO	0.8	0.8	1	0.8	0.67	0.8

Tableau 7.6 : Valeurs des métriques de réutilisation et de réutilisabilité calculées pour le diagramme de séquence du patron *"Action"*.

		FCW	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
Métriques de réutilisation	**DROB**	1	1	1	1	1	1
	DRM	1	1	0.71	0.75	0.83	1
Métriques de réutilisabilité		**FCW**	**Saspence**	**Compose**	**BLA**	**Safelane**	**RDCW**
	ROB	1	1	1	1	0.75	1
	RM	0.67	0.67	0.8	0.67	0.75	0.67

7.4 Conclusion

Dans ce chapitre, nous avons évalué la qualité des patrons proposés pour la modélisation des systèmes d'aide à la conduite reposant sur l'utilisation de bases de données temps réel. Cette évaluation se base sur les mesures calculées pour deux types de métriques dans six applications d'aide à la conduite. Le premier type correspond aux métriques de réutilisabilité. Les valeurs de ces métriques indiquent que la majorité des éléments des patrons sont retrouvés dans les applications modélisées sans réutilisation de ces patrons. Ainsi, elles prouvent que les patrons proposés représentent correctement les concepts et les exigences du domaine de l'aide à la conduite. Le deuxième type concerne les métriques de réutilisation qui consistent à quantifier le degré de réutilisation des éléments des différents patrons dans les applications d'aide à la conduite. Les valeurs élevées de ces métriques montrent que la majorité des éléments sont instanciés à partir des patrons proposés.

Conclusion générale

Bilan

Avec l'augmentation du nombre d'accidents, beaucoup de constructeurs automobiles trouvent un intérêt évident à augmenter les fonctionnalités des véhicules par des systèmes d'aide à la conduite en termes de sécurité. C'est dans ce cadre que s'inscrivent les travaux présentés dans ce mémoire. Pour cela, nous avons commencé par un état de l'art sur les systèmes de sécurité routière. Nous avons particulièrement présenté une étude des systèmes d'aide à la conduite qui existent dans la littérature et leurs caractéristiques, d'une part, et des différentes approches pour la tolérance aux fautes des capteurs, d'autre part. Cette étude nous a amené à constater que les systèmes d'aide à la conduite traitent un grand volume de données qui doivent être mises à jour régulièrement, mais ces systèmes ne permettent pas de stocker ces données. Ceci rend la gestion d'un grand nombre de données plus difficile, ce qui peut engendrer des conséquences graves lorsque certaines données sont perdues. Face à ce constat, nous avons proposé, dans ce mémoire, de faire évoluer l'architecture existante des systèmes d'aide à la conduite en intégrant une base de données temps réel permettant de gérer efficacement les contraintes temporelles relatives à un grand volume de données. Cette nouvelle architecture vise à améliorer la tolérance aux fautes affectant les capteurs en se basant sur la méthode d'estimation *"Imputation par la valeur précédente"*, ce qui permet de donner des résultats précis avec un temps de calcul minimal. De plus, l'intégration de la base de données permet de réduire le temps d'exécution des transactions vu que la gestion des transactions

est basée sur la QdD. Pour mettre en œuvre cette première contribution, nous avons effectué des simulations qui ont confirmé l'apport de cette contribution.

La gestion d'un grand volume de données et leurs contraintes temporelles rend la conception des systèmes d'aide à la conduite reposant sur l'utilisation d'une base de données plus difficile. De plus, l'étude de l'état de l'art des systèmes d'aide à la conduite nous a amené à constater que les travaux existants portent généralement sur la gestion de la fiabilité, les méthodes de fusion et le développement des systèmes, mais ne portent pas sur leur conception. Pour ces raisons, nous avons proposé de mettre en place un cadre méthodologique, pour les systèmes d'aide à la conduite reposant sur l'utilisation d'une base de données temps réel, basé sur la réutilisation des patrons. En effet, l'étude des différents types de composants réutilisables nous a permis d'opter pour une démarche d'ingénierie basée sur la réutilisation de ce type de composant. Ce cadre méthodologique a proposé, d'une part, de modéliser des patrons de conception et, d'autre part, de réutiliser ceux-ci lors du développement de nouveaux systèmes d'aide à la conduite. Nous avons pour cela étudié les principaux travaux qui portent sur la spécification, la modélisation et la réutilisation des patrons de conception, d'une part, et sur la modélisation des systèmes TR (STR) à travers les patrons, d'autre part. Cette étude nous a permis de soulever certaines insuffisances parmi lesquelles nous citons :

- Certains patrons dédiés au domaine TR permettent d'exprimer uniquement les exigences non fonctionnelles relatives aux mécanismes propres à ce domaine (*e.g.,* tolérance aux fautes et ordonnancement des tâches), et d'autres permettent d'exprimer les exigences fonctionnelles et non fonctionnelles du domaine TR. Les patrons qui expriment uniquement les exigences non fonctionnelles ne permettent pas de décrire la structure des STR (*i.e.,* les participants, les attributs et les opérations) et leur comportement. Alors que les patrons qui expriment les exigences fonctionnelles et non fonctionnelles sont abstraits, ce qui rend leur instanciation pour modéliser un domaine TR particulier complexe. Le concepteur doit ainsi spécifier les exigences du domaine considéré qui ne figurent pas dans la solution des patrons. De plus, les patrons dédiés au domaine TR ne modélisent pas les aspects des bases de données temps réel (*e.g.,* la gestion des transactions en se basant sur la QdD).

- Le critère de la traçabilité de l'instanciation des patrons n'est exprimé que partiellement. En effet, les extensions d'UML n'identifient pas tous les éléments qui jouent un rôle dans les patrons, ce qui engendre des difficultés pour vérifier si des éléments appartenant aux modèles d'applications sont instanciés à partir des patrons ou non.

Pour élaborer des solutions aux problèmes recensés, nous avons apporté quatre principales contributions.

La première contribution consiste à définir un profil UML, appelé UML-RTDB2, une extension du profil UML-RTDB [Idoudi et al., 2008b] qui exprime les aspects des bases de données temps réel. Le profil que nous avons proposé se distingue par l'introduction (i) des stéréotypes permettant d'exprimer la variabilité des patrons, (ii) des stéréotypes permettant de représenter explicitement les éléments relatifs à chaque instance de patrons et (iii) des stéréotypes permettant d'exprimer les propriétés non fonctionnelles. De plus, nous avons défini des contraintes exprimées en OCL pour vérifier la cohérence entre les éléments de la vue statique des patrons, d'une part, et la cohérence entre les éléments de la vue dynamique des patrons, d'autre part.

La deuxième contribution est la proposition de trois patrons pour la modélisation des systèmes d'aide à la conduite reposant sur l'utilisation de bases de données temps réel. Le premier, nommé *"Acquisition des données"*, concerne la fonctionnalité d'acquisition des données temps réel. Le deuxième, nommé *"Contrôle de données"*, sert à analyser les données stockées dans la base pour évaluer la situation et prendre les actions appropriées. Alors que le dernier, nommé *"Action"*, permet de modéliser la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes. La création de ces patrons se base sur un processus qui définit les étapes à suivre pour créer les modèles structurels et les modèles dynamiques. Ce processus permet de décrire les étapes à suivre pour déterminer les fonctionnalités et les exigences du domaine de l'aide à la conduite, d'une part, et de définir les règles d'unification pour générer les modèles de la vue statique et les modèles de la vue dynamique à partir de la comparaison d'un ensemble de systèmes du domaine, d'autre part. En effet, nous adoptons les règles d'unification proposées dans [Rekhis et al., 2010a] pour la construction

d'un diagramme de classes du patron et les règles que nous avons proposées dans [Marouane et al., 2013] pour la construction d'un diagramme de séquence du patron.

La troisième contribution concerne la réutilisation des patrons proposés, qui consiste à générer des modèles d'un système cible par instanciation et adaptation de ces patrons aux exigences du système. Pour cela, nous avons développé un outil, appelé *ADAS-Patterns*, qui permet d'automatiser la réutilisation des patrons.

La quatrième contribution consiste à évaluer les patrons proposés en se basant sur deux catégories de métriques : les métriques de réutilisation et les métriques de réutilisabilité dont certaines sont proposées par nous-mêmes. Les valeurs des métriques de réutilisation que nous avons obtenues pour six études de cas indiquent un degré élevé de réutilisation des patrons proposés vu que la majorité des classes, des attributs, des opérations, des objets et des messages des modèles d'applications sont instanciés à partir des patrons. De même, les valeurs des métriques de réutilisabilité obtenues montrent que nous avons retrouvé la majorité des classes, des attributs, des opérations, des objets et des messages appartenant aux différents patrons proposés, dans les diagrammes de classes et de séquence relatifs aux systèmes d'aide à la conduite modélisés sans la réutilisation des patrons. Par conséquent, nous avons constaté que les patrons que nous avons proposés représentent convenablement le domaine de l'aide à la conduite.

Perspectives de recherche

Nos travaux concernant la modélisation et la réutilisation des patrons de conception peuvent être améliorés à plusieurs niveaux, tant à court terme qu'à long terme.

À court terme, il s'agit d'améliorer le processus de réutilisation par la prise en compte de l'intégration des instances des patrons pour créer un modèle complet d'un système cible. Nous comptons aussi définir les moyens pour automatiser le processus de création des patrons. Il semble également intéressant d'évaluer les contraintes exprimées en OCL sur les modèles des patrons pour vérifier si ces derniers respectent les contraintes associées à chaque stéréotype.

À long terme, il serait intéressant de :

- proposer un outil de génération automatique de code à l'aide de patrons de conception qui prend en compte les variantes d'implémentation, vu qu'un même patron peut avoir un grand nombre d'implémentations différentes ;
- définir une ontologie spécifique au domaine de l'aide à la conduite pour combler les lacunes d'utilisation d'un dictionnaire et limiter l'intervention du concepteur pour spécifier les relations linguistiques entre les termes ;
- intégrer des patrons proposés dans une démarche d'ingénierie des systèmes d'aide à la conduite à base de patrons. Un système de patrons de conception se doit, en effet, d'être le plus complet possible pour qu'il fournisse une bonne aide à la conception et au développement des systèmes. Il convient donc de définir de nouveaux patrons de conception de type processus étendant notre système pour avoir un système de patrons plus complet. Ce système de patrons contiendra des solutions génériques à des problèmes de conception, d'une part, et des solutions génériques de guidage de l'ingénierie des systèmes, d'autre part.

Publications

Conférences Internationales

- Hela Marouane, Saoussen Rekhis, Rafik Bouaziz, Claude Duvallet, and Bruno Sadeg, **Definition and reuse of analysis patterns for real time applications**, Proceedings of the 2nd International Conferences on Pervasive Patterns and Applications (Patterns'2010), pages 84-91, Lisbon, Portugal, November 21-26, 2010.
- Hela Marouane, Achraf Makni, Rafik Bouaziz, Claude Duvallet, and Bruno Sadeg, **A Real-Time Design Pattern for Advanced Driver Assistance Systems**, Proceedings of the 17^{th} Conference on Pattern Languages of Programs (EuroPLoP'2012), pages C6 :1–C6 :11, Irsee Monastery, Bavaria, Germany, July 11-15, 2012.
- Hela Marouane, Achraf Makni, Claude Duvallet, Bruno Sadeg, and Rafik Bouaziz, **A Real-Time Design Pattern for Actuators in Advanced Driver Assistance Systems**, In the Eighth International Conference on Software Engineering Advances (ICSEA), pages 152-161, Venice, Italy, 2013.
- Saoussen Rekhis, Hela Marouane, Rafik Bouaziz, Claude Duvallet, and Bruno Sadeg, **Metrics for Measuring Quality of Real-time Design Patterns**, In the Eighth International Conference on Software Engineering Advances (ICSEA), pages 162-168, Venice, Italy, 2013.

Articles soumis

- Hela Marouane, Claude Duvallet, Bruno Sadeg, Achraf Makni, and Rafik Bouaziz, **A Real-Time Database System in Advanced Driver Assistance Systems**, soumis à la revue IEEE Transactions on Vehicular Technology, impact factor 2.063.
- Hela Marouane, Achraf Makni, Rafik Bouaziz, Claude Duvallet, and Bruno Sadeg, **Define and Reuse of a Design Pattern for Advanced Driver Assistance Systems**, soumis à la revue Software and Systems Modeling, impact factor 1.25
- Hela Marouane, Achraf Makni, Claude Duvallet, Rafik Bouaziz, and Bruno Sadeg, **Specification and Instantiation of a Design Pattern for Controller in ADAS**, soumis à la conférence Australian Software Engineering, classée B.

Références

Abdella, M., Marwala, T., 2005. Treatment of Missing Data Using Neural Networks and Genetic Algorithms. In : IEEE International Joint Conference on Neural Networks. Montreal, QC, Canada, pp. 598–603.

Abdo, A., Saijai, J., Damlakhi, W., Ding, S. X., 2009. An Observer-Based Fault Detection Approach for Vehicle Lateral Dynamics Control System. In : International Federation of Automatic Control (IFAC). pp. 1409–1414.

Adelberg, B. S., García-Molina, H., Kao, B., 1995. Applying Update Streams in a Soft Real-Time Database System. In : ACM SIGMOD International Conference on Management of Data. ACM, New York, NY, USA, pp. 245–256.

Aggarwal, K. K., Singh, Y., Kaur, A., Malhotra, R., 2005. Software Reuse Metrics for Object-Oriented Systems. In : Proceedings of the third ACIS International Conference on Software Engineering Research, Management and Applications (SERA'05). IEEE computer society, pp. 48–54.

Alexander, C., 1979. The Timeless Way of Building. Center for Environmental Structure. Oxford University Press.

Amditis, A., Bimpas, M., Thomaidis, G., Tsogas, M., Netto, M., Mammar, S., Beutner, A., Möhler, N., Wirthgen, T., Zipser, S., Etemad, A., Lio, M. D., Cicilloni, R., 2010. A Situation-Adaptive Lane-Keeping Support System : Overview of the SAFELANE Approach. IEEE Intelligent Transportation Systems Society 11 (3), 617–629.

Amditis, A., Floudas, N., Kaiser-Dieckhoff, U., Hackbarth, T., den Broek, B. V., Miglietta, M., Danielson, L., Gemou, M., Bekiaris, E., 2008. Integrated Vehicle's Lateral Safety : The LATERAL SAFE Experience. IEEE Intelligent Transportation Systems Society 2 (1), 15–26.

Amirijoo, M., Hansson, J., Son, S. H., 2006. Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations. IEEE Transactions on Computers 55 (3), 304–319.

Andler, S. F., Hansson, J., Eriksson, J., Mellin, J., Berndtsson, M., Eftring, B., 1996. DeeDS Towards a Distributed and Active Real-Time Database System. SIGMOD Rec. 25 (1), 38–51.

Anwar, S., 2010. Fault Detection. V. Kordic, Intech Publishing, Vienna, Austria, Ch. Isolation, and Control of Drive By Wire Systems, pp. 231–255.

Apvrille, L., Courtiat, J.-P., Lohr, C., Saqui-Sannes, P. D., 2004. TURTLE : A Real-Time UML Profile Supported by a Formal Validation Toolkit. IEEE Transactions on Software Engineering 30, 473–487.

Apvrille, L., Saqui-Sannes, P. D., Pacalet, R., Apvrille, A., 2006. Un environnement de conception de systèmes distribués basé sur UML. Annales des Télécommunications 61 (11), 1347–1368.

Armoush, A., 2010. Design Patterns for Safety-Critical Embedded Systems. Ph.D. thesis, Naturwissenschaften der RWTH Aachen University.

Arnaud, N., 2008. Fiabiliser la réutilisation des patrons par une approche orientée complétude, variabilité et généricité des spécifications. Ph.D. thesis, Université Joseph Fourier-Grenoble I.

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C., 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing 1 (1), 11–33.

Barbier, F., 2002. Business Component-Based Software Engineering. Springer US.

Barbier, F., Cauvet, C., Oussalah, M., Bennasri, S., Souveyet, C., 2004. Key Concepts and Reuse Techniques in the Information System Engineering. L'Objet 10 (1), 11–35.

Barbier, F., Cauvet, C., Oussalah, M., Rieu, D., Bennasri, S., Souveyet, C., 2002. Composants dans l'ingénierie des systèmes d'information : concepts clés et techniques de réutilisation. In : Information Interaction Intelligence - Actes des deuxièmes assises nationales du GdR I3. Nancy, France, pp. 95–117.

Bass, L., John, B. E., Juristo, N., Sanchez-Segura, M., 2004. Usability-Supporting Architectural Patterns. In : 26^{th} International Conference on Software Engineering (ICSE'04). IEEE, pp. 716–717.

Bayley, I., Zhu, H., 2012. A Formal Language for the Expression of Pattern Compositions. International Journal on Advances in Software 4, 354–366.

Beck, K., Cunningham, W., 1987. Using Pattern Languages for Object Oriented Programs. In : OOPSLA-87 workshop on the Specification and Design for Object-Oriented Programming.

Bekiaris, E., Spadoni, A., Nikolaou, S., 2009. SAFERIDER Project : New Safety and Comfort in Powered Two Wheelers. In : 2^{nd} conference on Human System Interactions. IEE press, Piscataway, NJ, USA, pp. 597–599.

BenAbdallah, H., Bouassida, N., Gargouri, F., Hamadou, A. B., 2004. A UML Based Framework Design Method. Journal of Object Technology 3 (8), 97–120.

Berrut, C., Front-Conte, A., 2001. Patterns Retrieval System : A First Attempt. In : Proceedings of the 5^{th} International Conference on Applications of Natural Language to Information Systems-Revised Papers. Springer-Verlag, London, UK, pp. 352–363.

Bertolazzi, E., Biral, F., Lio, M. D., Saroldi, A., Tango, F., 2010. Supporting Drivers In Keeping Safe Speed and Safe Distance : The SASPENCE Subproject Within the European Framework Programme 6 Integrating Project PReVENT. IEEE Transactions on Intelligent Transportation Systems 11 (3), 525–538.

Bhatia, P. K., Mann, R., 2008. An Approach to Measure Software Reusability of OO Design. In : Proceedings of 2nd National Conference on Challenges & Opportunities in Information Technology (COIT-2008) RIMT-IET. Mandi GobindgarhA, pp. 26–30.

Bhojan, R. J., Vivekanandan, K., Ganesan, S., 2014. Mobile Test Automation Framework for Automotive HMI. International Journal of Advanced Research in Computer and Communication Engineering 3 (1), 5316–5320.

Biral, F., Lot, R., Sartori, R., Borin, A., Roessler, B., 2010a. An Intelligent Frontal Collision Warning System for Motorcycles. In : Bicycle and Motorcycle Dynamics 2010. Symposium on the Dynamics and Control of Single Track Vehicles. Delft, The Netherlands, pp. 1–12.

Biral, F., Mauro, D. L., Lot, R., Sartori, R., 2010b. An Intelligent Curve Warning System for Powered Two Wheel Vehicles. European Transport Research Review 2 (3), 147–156.

Blanke, M., Staroswiecki, M., Wu, N. E., 2001. Concepts and Methods in Fault-Tolerant Control. In : American Control Conference (ACC). pp. 2606–2620.

Boskovic, J., Mehra, R., 2004. Failure Detection, Identification and Reconfiguration System for a Redundant Actuator Assembly. In : $5th$ IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS. pp. 411–416.

Bottoni, P., Guerra, E., Lara, J., 2009. Formal Foundation for Pattern-Based Modelling. In : 12^{th} International Conference on Fundamental Approaches to Software Engineering, Held as Part of the Joint European Conferences on Theory and Practice of Software. Springer Berlin Heidelberg, York, UK, pp. 278–293.

Boyles, S., 2011. A Comparison of Interpolation Methods for Missing Traffic Volume Data. In : 90^{th} Annual Meeting of Transportation Research Board. Washington, DC.

Cauvet, C., Rieu, D., Front-Conte, A., Ramadour, P., 2001. Réutilisation en ingénierie des systèmes d'information. In : Ingénierie des systèmes d'information. Hermès, pp. 115–147.

Chidamber, S. R., Kemerer, C. F., 1994. A Metrics Suite for Object Oriented Design. IEEE Transaction on Software Engineering 20 (6), 476–493.

Clauß, M., Jena, I., 2001. Modeling Variability with UML. In : In GCSE 2001 Young Researchers Workshop. Springer Verlag, Erfurt, Germany.

Coad, P., North, D., Mayfield, M., 1996. Object Models : Strategies, Patterns, and Applications. Yourdon Press Computing Series. Prentice Hall.

Conte, A., Fredj, M., Giraudin, J.-P., Rieu, D., 2001. P-sigma : un formalisme pour une représentation unifiée de patrons. In : INFORSID. Genève, pp. 67–86.

Coplien, J. O., 1992. Advanced C++ Programming Styles and Idioms. Addison-Wesley Publishing Co., Boston, MA, USA.

Courtiat, J.-P., Santos, C. A. S., Lohr, C., Outtaj, B., 2000. Experience With RT-LOTOS, A Temporal Extension of the LOTOS Formal Description Technique. Computer Communications 23 (1), 1104–1123.

Czarnecki, K., Eisenecker, U. W., 2000. Generative Programming : Methods, Tools, and Applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Darms, M., Foelster, F., Schmidt, J., Froehlich, D., Eckert, A., 2010. Data Fusion Strategies in Advanced Driver Assistance Systems. SAE International Journal of Passenger Cars - Mechanical Systems 3 (2), 176–182.

Dashtban, B., 2012. Scientific Workflow Patterns. Master's thesis, University of Manchester.

Delight, B., 2013. Optimizing the Execution Time of SQLite on an ABB Robot Controller. Master's thesis, Mälardalen University.

Dong, J., Sheng, Y., Zhang, K., 2007. Visualizing Design Patterns in Their Applications and Compositions. In : IEEE Transactions on Software Engineering. IEEE Press, Piscataway, NJ, USA, pp. 433–453.

Eden, A. H., 2000. Precise Specification of Design Patterns and Tool Support in Their Application. Ph.D. thesis, Department of Computer Science, Tel Aviv University.

Etienne, J.-P., Bouzefrane, S., 2006. Vers une approche par composants pour la modélisation d'applications temps réel. In : Confèrence Francophone de Modélisation et Simulation. Rabat, pp. 1–10.

Evans, L., 2004. Airbag Benefits, Airbag Costs. Society of Automotive Engineers, Warrendale, Penn.

Falkenhainer, B., Forbus, K. D., Gentner, D., 1989. The Structure-Mapping Engine : Algorithm and Examples. Artificial Intelligence 41, 1–63.

Fernandez, E. B., Yuan, X., 2010. An Analysis Pattern for Invoice Processing. In : 16^{th} Conference on Pattern Languages of Programs. ACM, New York, NY, USA, pp. 1–10.

Fowler, M., 1997. Analysis Patterns : Reusable Object Models. Addison-Wesley.
URL `http://books.google.tn/books?id=4V8pZmpwmBYC`

Frakes, W., Terry, C., 1996. Software Reuse : Metrics and Models. ACM Comput. Surv. 28 (2), 415–435.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns : Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

General Motors Corporation, 2008. Nouvelle génération d'appuie-tête actifs : objectif sécurité. `http://planer-motorshow.gmeuropearchive.info/shows/insignia/downloads/opel/fr/pdf/FR_03_Insignia.pdf`.

Gerdes, M., Jahr, R., Ungerer, T., 2013. ParMERASA Pattern Catalogue : Timing Predictable Parallel Design Patterns. Tech. rep., Department of Computer Science at the University of Augsburg, technical Report.

Ghahroudi, M. R., Sabzevari, R., 2009. Multisensor Data Fusion Strategies for Advanced Driver Assistance Systems. In : Sensor Data Fusion, I-Tech Education and Publishing KG. pp. 141–166.

Gherbi, A., Khendek, F., 2006. UML Profiles for Real-Time Systems and their Applications. Journal of Object Technology 5, 149–169.

Gietelink, O., 2007. Design and Validation of Advanced Driver Assistance Systems. Ph.D. thesis, Delft University of Technology.

Gill, N. S., Sikka, S., 2011. Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD. International Journal on Computer Science and Engineering (IJCSE) 3 (6), 2300–2309.

Goodwill, J., 2003. Jakarta Struts par la pratique. Eyrolles, Boston, MA, USA.

Götz, M., 2009. Modeling Workflow Patterns Through a Control-Flow Perspective Using BPMN and the BPM Modeler Bizagi. Ph.D. thesis, Institute of Applied Informatics and Formal Description Methods University Karlsruh.

Gurp, J. V., Bosch, J., Svahnberg, M., 2001. On the Notion of Variability in Software Product Lines. In : Working IEEE/IFIP Conference on Software Architecture. WICSA'01. IEEE Computer Society, Washington, DC, USA, pp. 45–54.

Gzara, L., Rieu, D., Tollenaere, M., 2000. Patterns Approach to Product Information Systems Engineering. Requirements Engineering 5 (3), 157–179.

Ha, H.-Y., Yang, Y., Fleites, F. C., Chen, S.-C., 2013. Correlation-Based Feature Analysis And Multi-Modality Fusion Framework for Multimedia Semantic Retrieval. In : IEEE International Conference on Multimedia and Expo (ICME). San Jose, CA, pp. 1–6.

Habibinia, D., Ebadpour, M., Sharifian, M., Sarem, Y. N., Khosrovjerdi, M., 2013. Mixed H_2/H_∞ Approach to Detect Fault in Parallel Hybrid Electric Vehicle. International Journal on Technical and Physical Problems of Engineering (IJTPE) 5 (3), 126–132.

Hameed, H. A., El-Bakry, H. M., Sultan, T., 2012. An Efficient Simulator for Replication in Distributed Real-Time Database Systems. International Journal of Computational Linguistics and Natural Language Processing 1 (4), 123–131.

Hastie, T., Tibshirani, R., Friedman, J., 2009. The Elements of Statistical Learning : Data Mining, Inference, and Prediction. Springer-Verlag.

Heineman, G. T., Councill, W. T. (Eds.), 2001. Component-Based Software Engineering : Putting the Pieces Together. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Idoudi, N., Duvallet, C., Bouaziz, R., Sadeg, B., Gargouri, F., 2008a. Structural Model of Real-Time Database : An Illustration. In : 11^{th} IEEE International Symposium on Object-oriented Real-time distributed Computing. IEEE Computer Society, Orlando, United State, pp. 58–65.

Idoudi, N., Duvallet, C., Sadeg, B., Bouaziz, R., Gargouri, F., 2009. How to Model a Real-Time Database? In : 12^{th} IEEE International Symposium on Object-oriented Real-time distributed Computing (IEEE ISORC'2009). Tokyo, Japan, pp. 321–325.

Idoudi, N., Louati, N., Duvallet, C., Bouaziz, R., Sadeg, B., faiez Gargouri, 2008b. A Framework to Model Real-Time Databases. International Journal of Computing and Information Sciences 6 (1), 19–28.

Isermann, R., Schwarz, R., Stolzl, S., 2002. Fault-Tolerant Drive-By-Wire Systems. Control Systems Magazine, IEEE 22, 64–81.

Jiang, N., 2007. A Data Imputation Model in Sensor Databases. In : Third International Conference High Performance Computing and Communications HPCC. Houston, USA, pp. 86–96.

John, I., Dörr, J., 2003. Elicitation of Requirements From User Documentation. In : 9^{th} International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ'03). Klagenfurt/Velden, Austria, pp. 3–12.

Johnson, R. E., 1997. Frameworks = (Components + Patterns). Communications ACM 40 (10), 39–42.
URL `http://doi.acm.org/10.1145/262793.262799`

Kim, S., Son, S. H., Stankovic, J. A., 2002. Performance Evaluation on a Real-Time Database. In : In IEEE Real-Time Technology and Applications Symposium. IEEE, pp. 253–265.

Kouroshfar, E., Shahir, H. Y., Ramsin, R., 2009. Process Patterns for Component-Based Software Development. In : 12^{th} International Symposium on Component-Based Software Engineering (CBSE'09). Springer Berlin Heidelberg, East Stroudsburg, PA, USA, pp. 54–68.

Labibes, K., Papp, Z., Thean, A., Lemmen, P., Dorrepaal, M., Leneman, F., 2003. An Integrated Design and Validation Environment for Intelligent Vehicle Safety systems (IVSS). In : 10^{th} World Congress and exhibition on ITS, Proceedings on CD-ROM. Madrid, Spain.

Langr, J., 1999. Essential Java Style : Patterns for Implementation. Prentice Hall PTR.

Lauffenburger, J.-P., 2002. Contribution à la surveillance temps-réel du système conducteur-véhicule-environnement : élaboration d'un système intelligent d'aide à la conduite. Ph.D. thesis, Université de Haute Alsace.

LeBlanc, D., Sayer, J., Winkler, C., Ervin, R., Bogard, S., Devonshire, J., Mefford, M., Hagan, M., Bareket, Z., Goodsell, R., Gordon, T., 2006. Road Departure Crash Warning System Field Operational Test : Methodology and Results. Tech. rep., U.S. Department of Transportation, University of Michigan Transportation Research Institute, Visteon Corporation AssistWare Technologies.

Leneman, F., Verburg, D., De Hair-Buijssen, S., 2008. Prescan, Testing and Developing Active Safety Applications Through Simulation. In : Tagung Aktive Sicherheit. Technical University of Munich, Munchen, Germany, pp. 1–8.

Li, Y., Parker, L. E., 2008. A Spatial-Temporal Imputation Technique for Classification with Missing Data in a Wireless Sensor Network. In : International Conference on Intelligent Robots and Systems. Nice, pp. 3272–3279.

Lindström, J., 2003. Optimistic Concurrency Control Methods for Real-Time Database Systems. Ph.D. thesis, Helsinki Finland University.

Lombardi, F., Park, N., Al-Hashimi, M. A., Pu, H. H., 2001. Modeling The Dependability of N-Modular Redundancy on Demand Under Malicious Agreement. In : Pacific Rim International Symposium on Dependable Computing. IEEE Computer Society, pp. 68–75.

Loo, K. N., Lee, S. P., Chiew, T. K., 2012. UML Extension for Defining the Interaction Variants of Design Patterns. Journal of IEEE software 29 (5), 64–72.

Lopez-Otero, P., Docio-Fernandez, L., Garcia-Mateo, C., 2014. Introducing a Framework for the Evaluation of Music Detection Tools. In : 9^{th} Language Resources and Evaluation Conference. San Jose, CA, pp. 568–572.

Mapelsden, D., Hosking, J., Grundy, J., 2002. Design Pattern Modelling and Instantiation Using DPML. In : Fortieth International Conference on Tools Pacific : Objects for Internet, Mobile and Embedded Applications. CRPIT'02. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 3–11.

Marouane, H., Makni, A., Bouaziz, R., Duvallet, C., Sadeg, B., 2012. A Real-Time Design Pattern for Advanced Driver Assistance Systems. In : 17^{th} European conference on Pattern Languages of Programs (EuroPLoP). Hillside, Kloster Irsee, Germany, pp. C6 :1–C6 :11.

Marouane, H., Makni, A., Duvallet, C., Sadeg, B., Bouaziz, R., 2013. A Real-Time Design Pattern for Actuators in Advanced Driver Assistance Systems. In : The Eighth International Conference on Software Engineering Advances (ICSEA). Xpert Publishing Services, Venice, Italy, pp. 152–161.

Marouane, H., Rekhis, S., Bouaziz, R., Duvallet, C., Sadeg, B., 2010. Definition and Reuse of Analysis Patterns for Real-Time Applications. In : International Conferences on Pervasive Patterns and Applications (Patterns'2010). Lisbon, Portugal, pp. 84–91.

McGregor, J. D., 2004. Software Product Lines. Journal of Object Technology 3, 65–74.

Mobileye, 2007. Lane Departure Warning. `http://www.prevenireaccidente.ro/brosuri/LaneDeparture.pdf`.

Muller, H. E., Linn, B., 1998. Seat Belt Pretensioners. Society of Automotive Engineers.

Nellborn, C., 1999. Business and Systems Development : Opportunities for an Integrated Way-of-Working. In : Perspectives on Business Modelling : understanding and changing organisations. Springer Berlin Heidelberg, Berlin, pp. 197–216.

Németh, B., Gaspar, P., Bokor, J., Sename, O., Dugard, L., 2012. Fault-Tolerant Control Design for Trajectory Tracking in Driver Assistance Systems. 8^{th} International Federation of Automatic Control Symposium on Fault Detection, Supervision and Safety of Technical Processes 8, 186–191.

OMG, 1999. OMG Unified Modeling Language Specification.

OMG, 2004. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms.

OMG, 2005a. UML Profile for Schedulability, Performance, and Time Specification, Version 1.1.
URL `http://www.omg.org/cgi-bin/doc?formal/2005-01-02`

OMG, 2005b. UML2.0 Superstructure, formal/ 05-07-04.

OMG, 2007. Unified Modeling Language (UML) Infrastructure, v2.1.2, formal/2007-11-04.

OMG, 2009. Unified Modeling Language : Superstructure Version 2.2, formal 2009-02-02.

OMG, 2011. A UML Profile for MARTE.
URL `http://www.omg.org/spec/MARTE/1.1/`

OMG, 2012. OMG Object Constraint Language (OCL). Available from : http ://www.omg.org/spec/OCL/2.3.1/.

ONISR, janvier 2014. Bilan de l'accidentalité routière en 2013.

Oudghiri, M., Octobre 2008. Commande multi-modèles tolérante aux défauts : Application au contrôle de la dynamique d'un véhicule automobile. Ph.D. thesis, Université de Picardie Jules Verne.

Ozsoyoglu, G., Snodgrass, R. T., 1995. Temporal and Real-Time Databases : A Survey. IEEE Transactions on Knowledge and Data Engineering 7, 513–532.

Pekka, A., Jari, R., 2013. Patterns for Light-Weight Fault Tolerance and Decoupled Design in Distributed Control Systems. In : VikingPLoP Conference. Eloranta, Veli-Pekka ; Koskinen, Johannes ; Leppänen, Marko, pp. 1–18.

Piao, J., McDonald, M., 2008. Advanced Driver Assistance Systems from Autonomous to Cooperative Approach. Transport Reviews 28 (5), 659–684.

Port, D., 1998. Derivation of Domain Specific Design Patterns. USC Center for software engineering.

Prestl, W., Sauer, T., Steinle, J., Tschernoster, O., 2000. The BMW Active Cruise Control ACC. SAE transactions 109 (7), 119–125.

Quet, P.-F., Salman, M., 2007. Model-Based Sensor Fault Detection and Isolation for X-By-Wire Vehicles Using a Fuzzy Logic System with Fixed Membership Functions. In : American Control Conference. pp. 2314–2319.

Ramamritham, K., 1993. Real-Time Database. International Journal of Distributed and Parallel Databases 1 (2), 199–226.

Ramamritham, K., Son, S. H., Dipippo, L. C., 2004. Real-Time Databases and Data Services. Real-Time Systems 28 (2-3), 179–215.

Raphael, E., Kiefer, R., Reisman, P., Hayon, G., 2011. Development of a Camera-Based Forward Collision Alert System. SAE International Journal of Passenger Cars-Mechanical 4 (1), 467–478.

Reinhartz-Berger, I., Sturm, A., 2009. Utilizing Domain Models for Application Design and Validation. Information and Software Technology 51 (8), 1275–1289.

Rekhis, S., Bouassida, N., Bouaziz, R., Duvallet, C., Sadeg, B., 2013a. Domain Engineering : Product Lines, Languages, and Conceptual Models. Springer Berlin Heidelberg, Ch. Modeling Real-Time Design Patterns with the UML-RTDP Profile, pp. 59–82.

Rekhis, S., Bouassida, N., Duvallet, C., Bouaziz, R., Sadeg, B., 2010a. A Process to Derive Domain-Specific Patterns : Application to the Real-Time Domain. In : 14^{th} East European Conference on Advances in Databases and Information Systems (ADBIS). Novi Sad, Serbia, pp. 475–489.

Rekhis, S., Bouassida, N., Duvallet, C., Bouaziz, R., Sadeg, B., 2010b. Modeling Real-Time Applications with Reusable Design Patterns. International Journal of Advanced Science and Technology (IJAST) 22, 71–86.

Rekhis, S., Bouassida, N., Duvallet, C., Bouaziz, R., Sadeg, B., 2010c. A UML-Profile for Domain Specific Patterns : Application to Real-Time. In : DE@CAISE'10 : the Domain Engineering workshop of the 22^{nd} International Conference on Advanced Information Systems Engineering (CAiSE'10). Hammamet, Tunisia, pp. 32–46.

Rekhis, S., Marouane, H., Bouaziz, R., Duvallet, C., Sadeg, B., 2013b. Metrics for Measuring Quality of Real-Time Design Patterns. In : The Eighth International Conference on Software Engineering Advances (ICSEA). Xpert Publishing Services, Venice, Italy, pp. 162–168.

Renault, 2008. Renault conçoit des véhicules homogènes en matière de sécurité passive et active. `http://s1.e-monsite.com/2009/03/11/1311877016466-dp-atelier-securite-pdf.pdf`, communiqué de presse.

Rieu, D., 1999. Ingénierie des systèmes d'information. Ph.D. thesis, Institut national polytechnique de Grenoble, mémoire d'habilitation à diriger les recherches.

Rieu, D., Giraudin, J. P., Saint-Marcel, C., 1999a. Génie Objet-analyse et conception de l'évolution d'objets. M. Oussalah, Hermès, Ch. Réutilisation et patrons d'ingénierie.

Rieu, D., Giruadin, J. P., Siant-Marcel, C., Front-Conte, A., 1999b. Des opérations et des relations pour les patrons de conception. In : INFORSID. La Garde, France, pp. 259–277.

Roberts, D., Johnson, R., 1996. Evolving Frameworks : A Pattern Language for Developing Object-Oriented Frameworks. In : Third Conference on Pattern Languages and Programming. Addison-Wesley, Allerton Park, Illinois.

Rolland, C., 2005. L'ingénierie des méthodes : une visite guidée. e-TI - la revue électronique des technologies de l'information 1.
URL `http://www.revue-eti.netdocument.php?id=726`

Salameh, N., 2011. Conception d'un système d'alerte embarqué basé sur les communications entre véhicules. Ph.D. thesis, Institut National des Sciences Appliquées de Rouen.

Sangha, M. S., Yu, D., Gomm, J. B., 2011. Sensor Fault Diagnosis for Automotive Engines With Real Data Evaluation. International Journal of Engineering, Science and Technology 3 (8), 13–25.

Sangha, M. S., Yu, D. L., Gomm, J. B., 2012. Sensor Fault Detection, Isolation, Accommodation and Unknown Fault Detection in Automotive Engine Using AI. International Journal of Engineering, Science and Technology 4 (3), 53–65.

Sarika, S., Sasipraba, T., Selvamani, K., 2013. A Real Time Design Pattern Using MappReduce Startegy Visit For Distributed Service Oriented Strategy. International Journal on Advances in Computing and Communication Technologies 1 (1), 36–39.

Schmidt, D. C., Fayad, M., Johnson, R. E., 1996. Software Patterns. Communications ACM 39 (10), 37–39.

Selic, B., 1998. Using UML for Modeling Complex Real-Time Systems. In : ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems. LCTES'98. Springer-Verlag, London, UK, UK, pp. 250–260.

Selic, B., Gullekson, G., Ward, P. T., 1994. Real-Time Object-Oriented Modeling. John Wiley and Sons, New-York.

Semmak, F., 1998. Réutilisation de composants de domaine dans la conception de systèmes d'information. Ph.D. thesis, Université Paris I.

Sename, O., Tudon-Martınez, J., Fergani, S., 2013. LPV Methods for Fault-Tolerant Vehicle Dynamic Control. In : Conference on Control and Fault-Tolerant Systems (SysTol).

Shuai, W., Jian, L., Shuming, J., Zhiqiang, W., Jianfeng, Z., Shijie, X., 2012. Study of Real-Time Data Collection and Release with In-vehicle System. In : 2nd International Conference on Computer Application and System Modeling. Atlantis Press, Paris, France, pp. 1472–1475.

Smith, B. L., Scherer, W. T., Conklin, J. H., 2003. Exploring Imputation Techniques for Missing Data in Transportation Management Systems. Transportation Research Record : Journal of the Transportation Research Board 1836, 132–142.

Spanoudakis, G., Constantopoulos, P., 1996. Elaborating Analogies From Conceptual Models. Internation Journal of Intelligent Systems 1 (11).

Stankovic, J. A., Ramamritham, K., 1988. Hard Real-Time System : a Tutorial. In : IEEE Computer Society Press.

Staron, M., Kuzniarz, L., 2008. Guidelines for Creating "Good" Stereotypes. In : Nordic Workshop on Model Driven Engineering. pp. 1–17.

Staroswiecki, M., Gehin, A.-L., 2001. From Control to Supervision. Annual Reviews in Control 25, 1–11.

Steffen, T., Michail, K., Dixon, R., Zolotas, A., Goodall, R. M., 2009. Optimal Passive Fault Tolerant Control of a Hight Redundancy Actuator. In : 7^{th} International Fedaration Automatic Control Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS). pp. 1234–1239.

Szyperski, C., 2002. Component Software : Beyond Object-Oriented Programming, 2nd Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Tabbache, B., Benbouzid, M. H., Kheloui, A., Bourgeot, J. M., 2013. Virtual-Sensor-Based Maximum-Likelihood Voting Approach for Fault-Tolerant Control of Electric Vehicle Powertrains. IEEE Transactions On Vehicular Technology 62 (3), 1075–1083.

Tass, 2012. Prescan Help.

URL `https://www.tassinternational.com`

TimesTen Team, C., 1999. In-memory Data Management for Consumer Transactions the Timesten Approach. In : ACM SIGMOD International Conference on Management of Data. ACM, New York, NY, USA, pp. 528–529.

Tong, S., Yang, G., Zhang, W., 2011. Observer-Based Fault-Tolerant Control Against Sensor Failures For Fuzzy Systems With Time Delays. International Journal of Applied Mathematics and Computer Science 21 (4), 617–627.

Valeo, 2008. L'automobile et la sécurité. `http://www.valeo.com/medias/upload/2012/12/3008/l-automobile-et-la-securite.pdf`.

Varrier, S., Vivas-Lopez, C., de Jesus Lozoya-Santos, J., Tudon-Martınez, J., Koenig, D., John-Jairo, Martinez-Molina, Morales-Menéndez, R., 2013. Applicative Fault Tolerant Control for Semi-Active Suspension System : Preliminary Results. In : European Control Conference (ECC). pp. 3803–3808.

Viehweider, A., Nam, K., Fujimoto, H., Hori, Y., 2012. A Fault Detection and Isolation Scheme for Lateral Vehicle Dynamics of EVs using a Quantitative Parity Space Approach. In : 38^{th} Annual Conference on IEEE Industrial Electronics Society. pp. 4630–4636.

Vuori, M., Virtanen, H., Koskinen, J., Katara, M., 2011. Safety Process Patterns in the Context of IEC 61508-3. Tech. rep., Tampere University of Technology, Tampere.

Yuan, Y., 1997. SMOS : A Memory-Resident Object Store. Ph.D. thesis, University of Rhode.

Zhanga, Y., Jiang, J., 2008. Bibliographical Review on Reconfigurable Fault-Tolerant Control Systems. Annual Reviews in Control 32 (2), 229–252.

Ziadi, T., Jézéquel, J.-M., Fondement, F., 2003. Product Line Derivation With UML. In : Software Variability Management Workshop, Univ. of Groningen Departement of Mathematics and Computing Science. Rennes, France.

Zou, J.-X., Xu, H.-B., 2009. Design and Reliability Analysis of Emergency Trip System With Triple Modular Redundancy. In : International Conference on Circuits and Systems. pp. 1006–1009.

Annexe A

Relations linguistiques et règles d'unification

Tableau A.1 : Relations linguistiques pour la comparaison des noms de classes, attributs et opérations [BenAbdallah et al., 2004].

Élément	**Relation**
Classes	• N_equiv($C(Ai)$ [1], ..., $C(An)$) : indique que les noms de classes sont identiques ou synonymes. • N_var($C(Ai)$, ..., $C(An)$) : indique que les noms de classes sont des variations d'un concept général. • N_dist($C(Ai)$, ..., $C(An)$) : indique qu'aucune relation n'existe entre les noms de classes.

Attributs	• Att_equiv($C(Ai), ..., C(An)$) : indique que les classes ont des noms d'attributs identiques ou synonymes de même type. • Att_int($C(Ai), ..., C(An)$) : indique que les classes ont des attributs en commun. • Att_conf($C(Ai), ..., C(An)$) : indique qu'il existe au moins un attribut d'une classe ayant un nom équivalent avec un attribut de chacune des autres classes, mais avec des types différents.
Opérations	• Op_equiv($C(Ai), ..., C(An)$) : indique que les classes ont des noms d'opérations identiques ou synonymes de même paramètres. • Op_int($C(Ai), ..., C(An)$) : indique que les classes ont des opérations communes.

Tableau A.2 : Relations linguistiques pour la comparaison des noms des objets et des messages [BenAbdallah et al., 2004].

Élément	**Relation**
Objet	• N_equiv($O(Ai)^2, ..., O(An)$) : indique que les noms des objets sont identiques ou synonymes. • N_dist($O(Ai), ..., O(An)$) : indique qu'aucune relation n'existe entre les noms des objets.
Message	• N_equiv($M(Ai)^3, ..., M(An)$) : indique que les noms des messages sont identiques ou synonymes. • N_dist($M(Ai), ..., M(An)$) : indique qu'aucune relation n'existe entre les noms des messages.

Tableau A.3 : Règles d'unification des diagrammes de classes.

Règle
RC1 : S'il existe un ensemble de classes $\{C(Ai), ..., C(An)\}$ qui ont des noms équivalents, des attributs équivalents et des opérations équivalentes, alors une classe représentative est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >> [Rekhis et al., 2010a].

RC2 : S'il existe un ensemble de classes équivalentes (N_equiv($C(Ai), ..., C(An)$) qui ont des attributs et/ou opérations qui ne sont pas communs à toutes les applications, mais qu'ils sont pertinents pour le domaine, alors ces attributs et/ou opérations sont ajoutés au patron en tant qu'élément optionnel stéréotypé << *optional* >>. La pertinence est déterminée à partir du ratio de couverture du domaine (Rdc) [BenAbdallah et al., 2004]. Ce ratio est calculé comme suit : $Rdc(E) = \frac{Nombre\ d'occurrences\ de\ l'élément\ E\ dans\ les\ fragments\ d'applications\ (Ai,...An)}{Nombre\ d'applications}$. Un élément est pertinent si $Rdc \geq \frac{1}{2}$.
RC3 : S'il existe des attributs en conflit (Att_conf($C_{Ai}, ..., C_{An}$)) ayant le même nom et des types compatibles (*e.g.,* réel et entier), alors il leur correspond dans le patron un attribut qui a le même nom et le type le plus général : *String* [Rekhis et al., 2010a].
RC4 : S'il existe un ensemble de classes $\{C(Ai), ..., C(An)\}$ qui ont des noms distincts, des attributs et/ou des opérations communs et jouant le même rôle qu'un concept de domaine, alors une classe dont le nom correspond au rôle joué par les classes $\{C(Ai), ..., C(An)\}$ est ajoutée au patron en tant que classe fondamentale stéréotypée << *mandatory* >>. Les attributs et/ou les opérations qui ne sont pas communs à toutes les classes sont ajoutés comme éléments optionnels, lorsqu'ils sont pertinents pour le domaine [Rekhis et al., 2010a].
RC5 : Si deux ou plusieurs classes des applications sont ajoutées au patron, alors toutes les relations de ces classes (association, héritage, composition, etc.) seront transférées dans le patron [Rekhis et al., 2010a]. Nous associons à chaque relation un stéréotype << *mandatory* >> ou << *optional* >>. Une relation est stéréotypée << *mandatory* >> si les deux classes sont fondamentales, alors qu'elle est stéréotypée << *optional* >> si les deux classes sont optionnelles ou si l'une des deux classes est optionnelle.

RC6 : S'il existe un ensemble de classes {C(Ai),..., C(An)} qui partagent des attributs et/ou des opérations (Att_int (C(Ai),...,C(An)) et/ou Op_int (C(Ai),...,C(An))) et ayant des noms qui représentent une variation d'un concept de domaine, alors une classe fondamentale ayant les attributs et/ou les opérations communs est ajoutée au patron.
RC7 : S'il existe un ensemble de classes {C(Ai),..., C(An)} qui ne sont pas communs à toutes les classes (Ai,...,An) mais qui sont pertinents pour le domaine ($Rdc \geq \frac{1}{2}$), alors elles sont ajoutées au patron en tant que classes optionnelles.

Annexe B

Modélisation des applications d'aide à la conduite

Étude de cas $N°1$: Système de prévention de collision frontale

Description du système

Le principe de fonctionnement du système d'alertes de risque de collision frontale, ou FCW (Frontal Collision Warning) s'appuie sur une surveillance de la route à l'aide de capteurs extéroceptifs (radars), qui sont installés à l'avant du véhicule. Ces capteurs permettent de détecter des obstacles mobiles (motos, automobiles, etc.) et de déterminer la distance entre le véhicule et ces obstacles. Si le système détecte que le véhicule s'approche trop près de l'obstacle (distance inférieure à un seuil fixé), alors il fournit des alertes sonores et lumineuses. Mais, lorsque le système détecte que le risque de collision augmente et que le conducteur ne réagit pas à temps, il freine alors automatiquement afin d'éviter la collision. FCW a été évalué par NHTSA (National Highway Traffic Safety Administration) dans un véhicule Volvo S80.

FCW utilise un ensemble de capteurs actifs embarqués dans le véhicule :

- Des radars situés à l'avant du véhicule permettent de détecter des obstacles mobiles à une distance pouvant aller de 1 à 100 mètres. Ils disposent d'une résolution de 0.5 mètres et d'une précision de +/- 5%.
- Des capteurs proprioceptifs (capteur de vitesse de lacet, accéléromètre, capteur de positionnement et capteur de frein) permettent d'observer l'état du véhicule (vitesse de lacet, accélération, vitesse longitudinale, etc.).

Ces capteurs acquièrent périodiquement les données relatives aux obstacles détectés (position, vitesse, identifiant, type, etc.) et au véhicule (vitesse, accélération, etc.). Ces données sont ensuite transmises avec une fréquence élevée vers l'unité de fusion qui a pour rôle de les traiter et de fusionner celles qui proviennent de différents capteurs pour augmenter la précision et la confiance des données acquises et avoir l'état de chaque élément observé (en mouvement, en arrêt, etc.).

Le contrôleur, doté d'une grande capacité mémoire et d'un temps de réponse rapide, reçoit les données acquises et analyse la situation, y compris le calcul du temps restant avant collision (Time To Collision TTC). Le calcul de TTC est effectué en fonction des données acquises par les capteurs (distance entre le véhicule et l'obstacle détecté). Chaque donnée acquise ou calculée est caractérisée par un type, une durée de validité, une date de mise à jour et une unité. Les données acquises sont aussi caractérisées par un taux d'erreur maximale. Ce contrôleur permet de déterminer la présence des situations à risque (risque de collision) *i.e.,* lorsque le contrôleur détecte que la valeur de TTC est inférieure à une valeur de TTC seuil, il évalue le degré du risque et décide de l'action à mener pour cette situation.

Le contrôleur envoie les actions à un gestionnaire d'interfaces homme-machine pour activer les composants de l'interface homme-machine appropriés. Ces composants permettent ensuite de générer des alertes (a) visuelles par le biais d'un dispositif d'affichage à tête haute (HUD) composé de plusieurs LEDs lumineuses et avec différentes couleurs et (b) sonores à l'aide d'un haut-parleur installé derrière le tableau de bord. Chaque type d'alerte est défini par un niveau (alerte de niveau 1, alerte de niveau 2 et alerte de niveau 3), un taux de répétition et une durée. Ces alertes sont transmises au conducteur afin qu'il puisse prendre des actions correctives à temps *i.e.,* le temps de réaction du conducteur est inférieur à TTC. La vitesse

du véhicule sera ajustée afin d'éviter une collision suite aux actions prises par le conducteur. Un indicateur de données contrôle et affiche l'état du système à chaque que fois qu'une action est exécutée.

Modélisation du système sans réutilisation de patrons

La figure B.1 illustre le diagramme de classes du système FCW modélisé sans réutilisation des différents patrons que nous avons proposés.

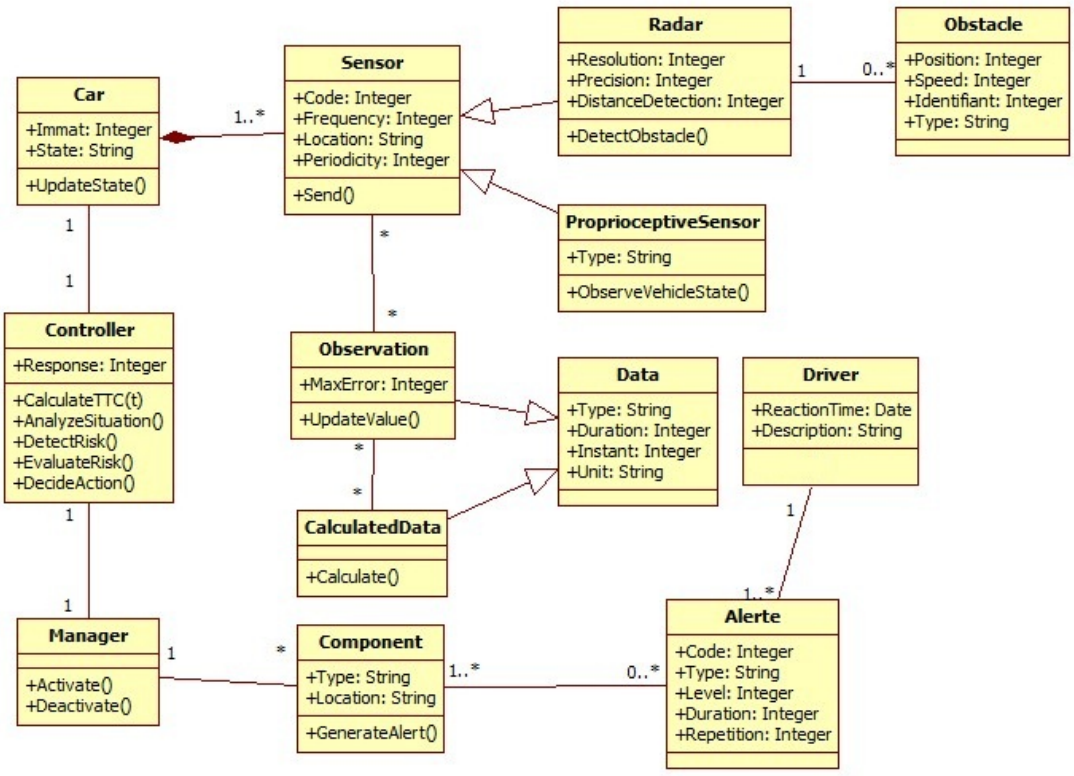


FIGURE B.1 : Modélisation du diagramme de classes du système FCW sans réutilisation de patrons.

La figure B.2 présente le diagramme de séquence du système FCW modélisé sans réutilisation des différents patrons que nous avons proposés.

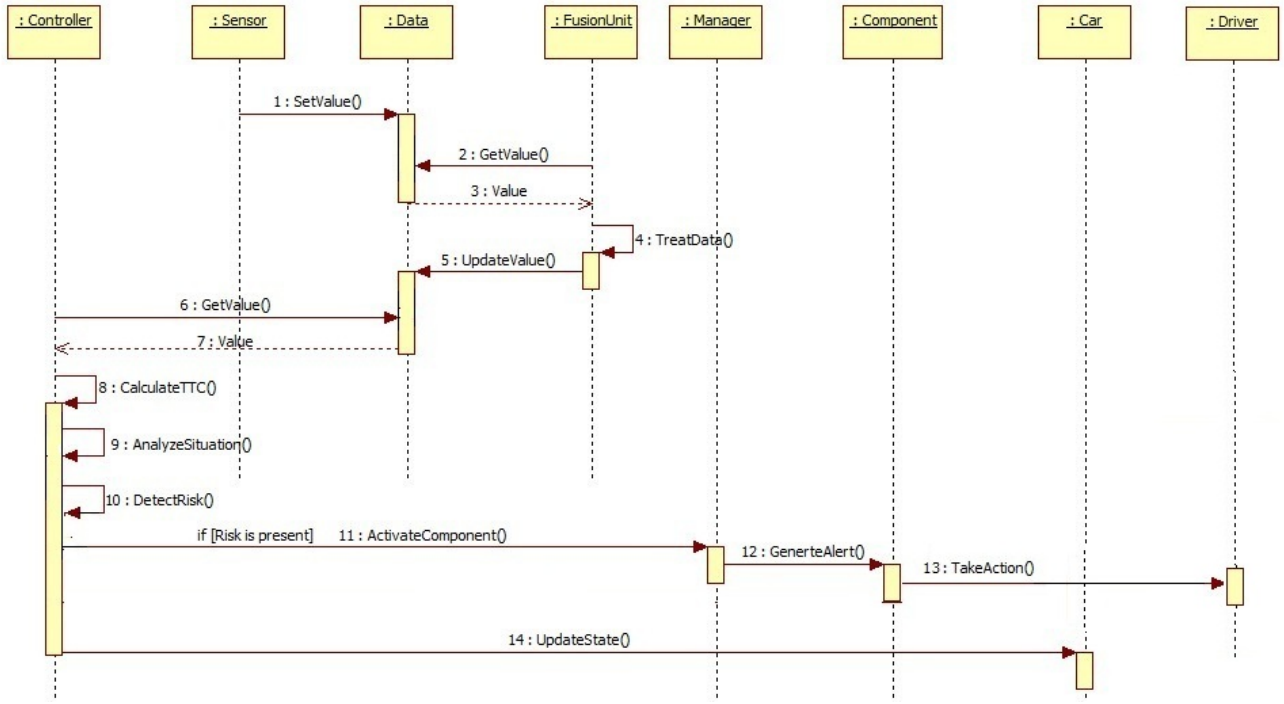


FIGURE B.2 : Modélisation du diagramme de séquence du système FCW sans réutilisation de patrons.

Modélisation du système par réutilisation de patrons

Modélisation des diagrammes de classes

La figure B.3 montre le diagramme de classes de la fonctionnalité d'acquisition des données du système FCW modélisé par réutilisation de la vue statique du patron *"Acquisition des données"*. La figure B.4 montre le diagramme de classes de la fonctionnalité de contrôle de données du système FCW modélisé par réutilisation de la vue statique du patron *"Contrôle de données"*.

La figure B.5 montre le diagramme de classes de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes du système FCW modélisé par réutilisation de la vue statique du patron *"Action"*.

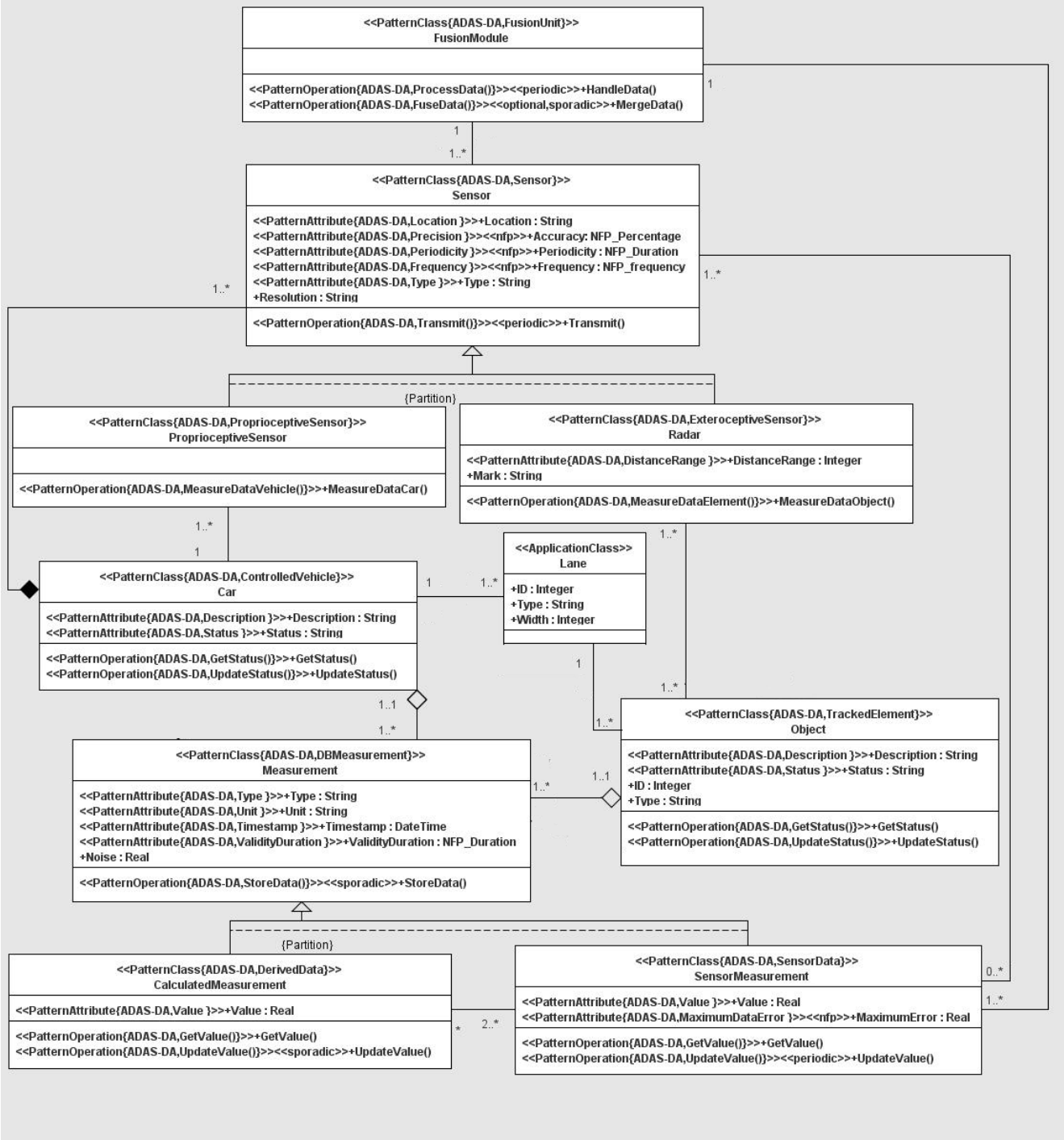


FIGURE B.3 : Modélisation du diagramme de classes du système FCW par réutilisation de la vue statique du patron *"Acquisition des données"*.

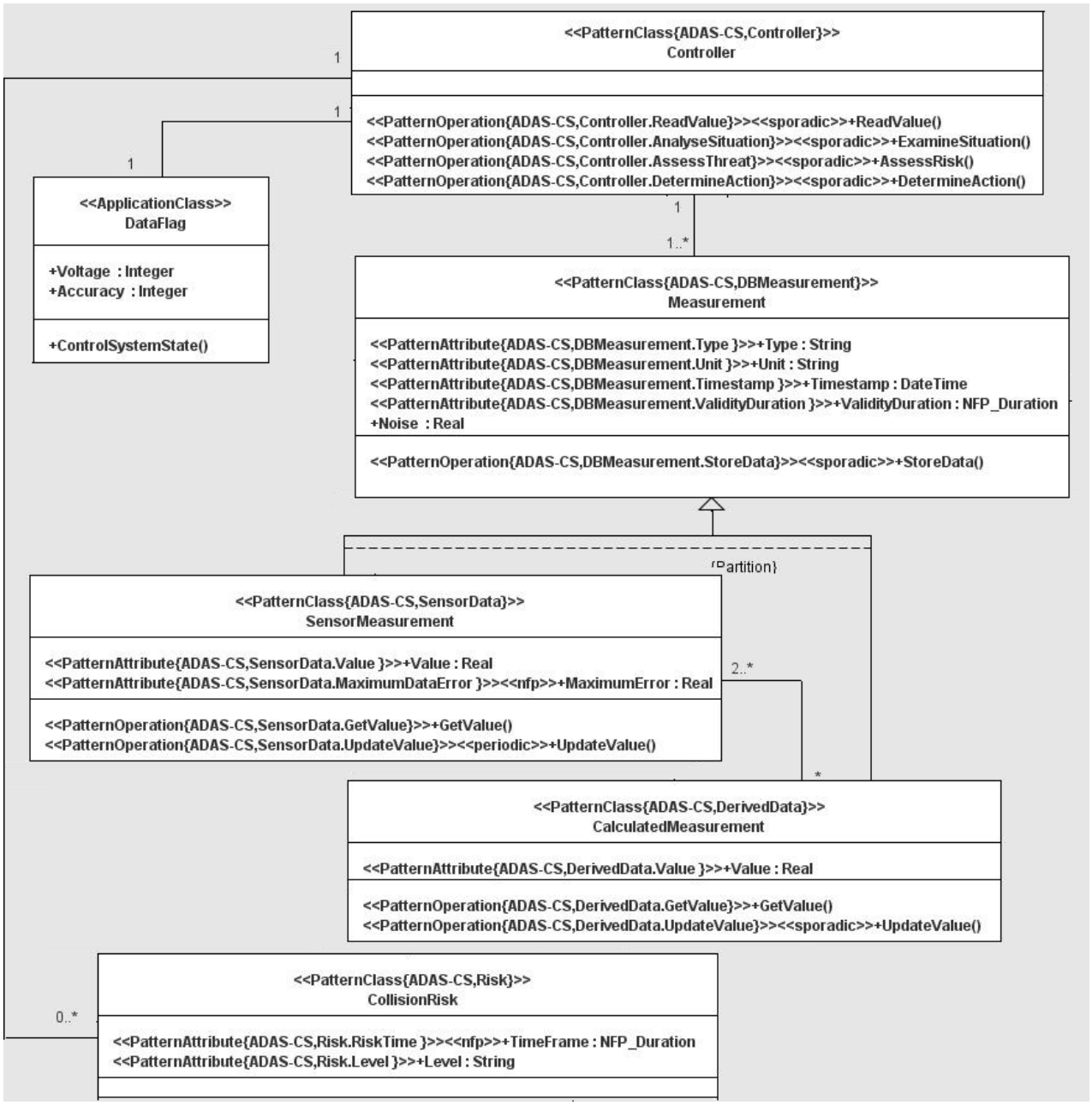


FIGURE B.4 : Modélisation du diagramme de classes du système FCW par réutilisation de la vue statique du patron *"Contrôle de données"*.

Modélisation des diagrammes de séquence

La figure B.6 montre la modélisation du diagramme de séquence de la fonctionnalité d'acquisition des données par réutilisation de la vue dynamique du patron *"Acquisition des données"*.

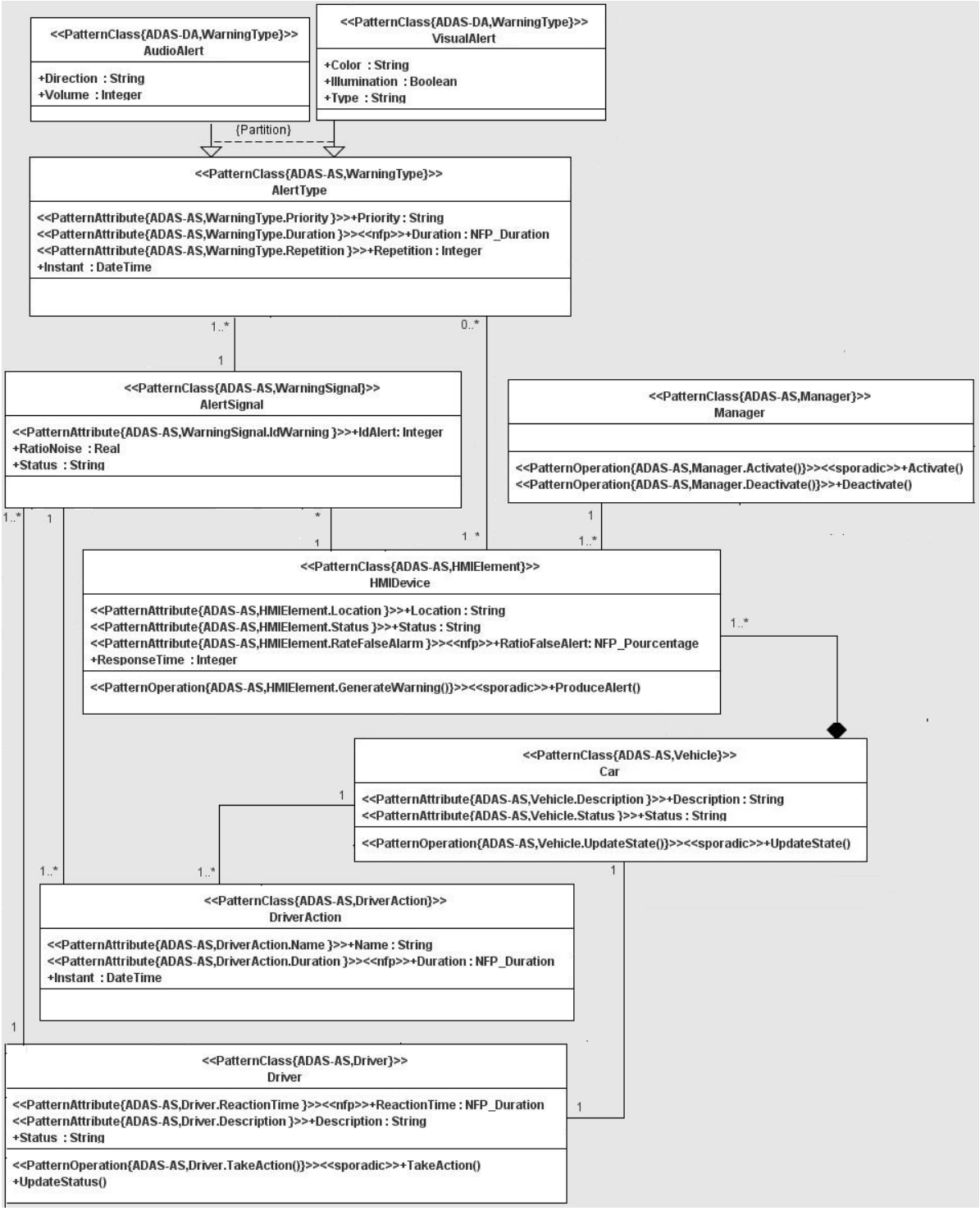


FIGURE B.5 : Modélisation du diagramme de classes du système FCW par réutilisation de la vue statique du patron *"Action"*.

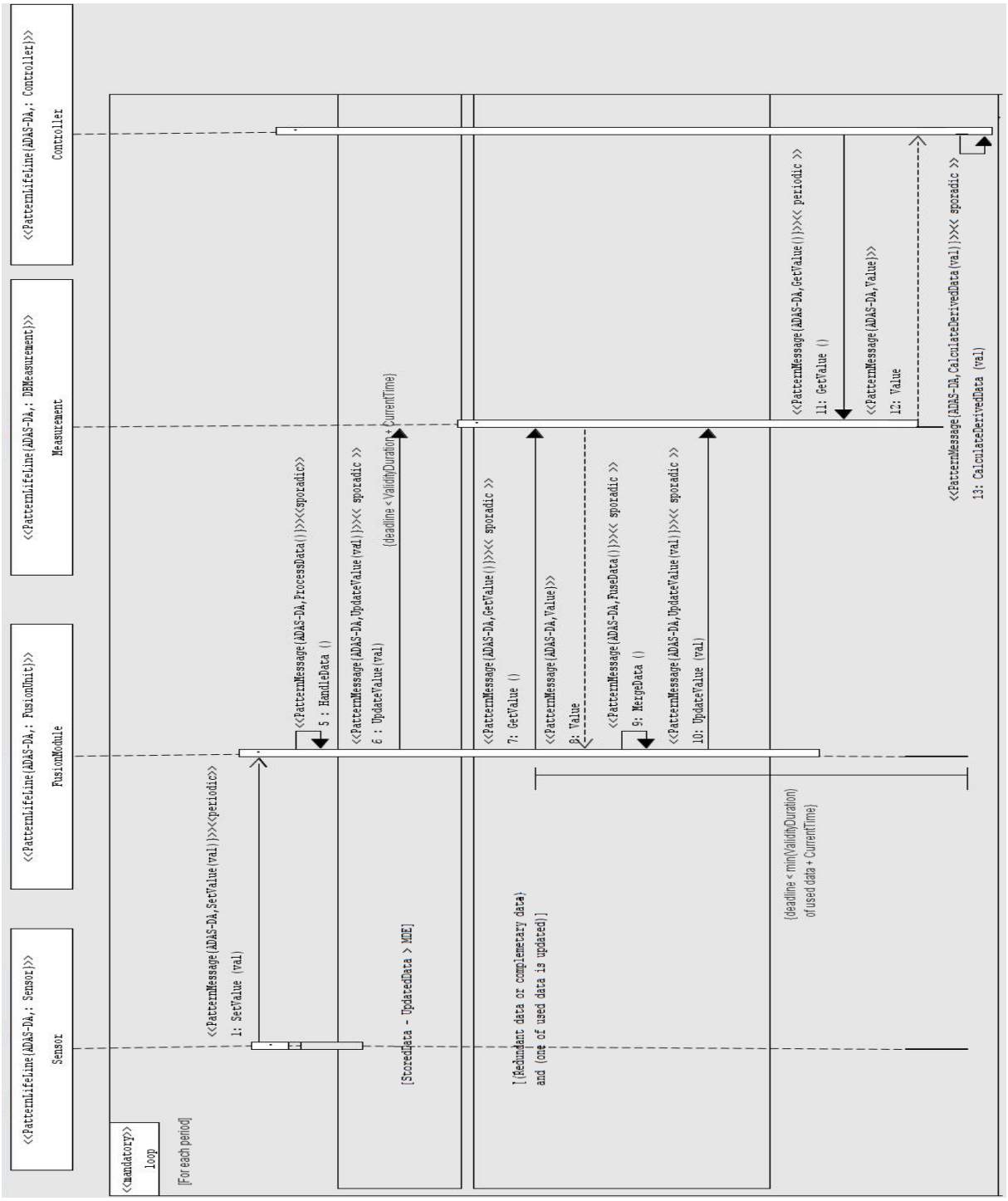


FIGURE B.6 : Modélisation du diagramme de séquence du système FCW par réutilisation de la vue dynamique du patron *"Acquisition des données"*.

La figure B.7 présente la modélisation du diagramme de séquence de la fonctionnalité de contrôle et de traitement des données par réutilisation de la vue dynamique du patron *"Contrôle de données"*.

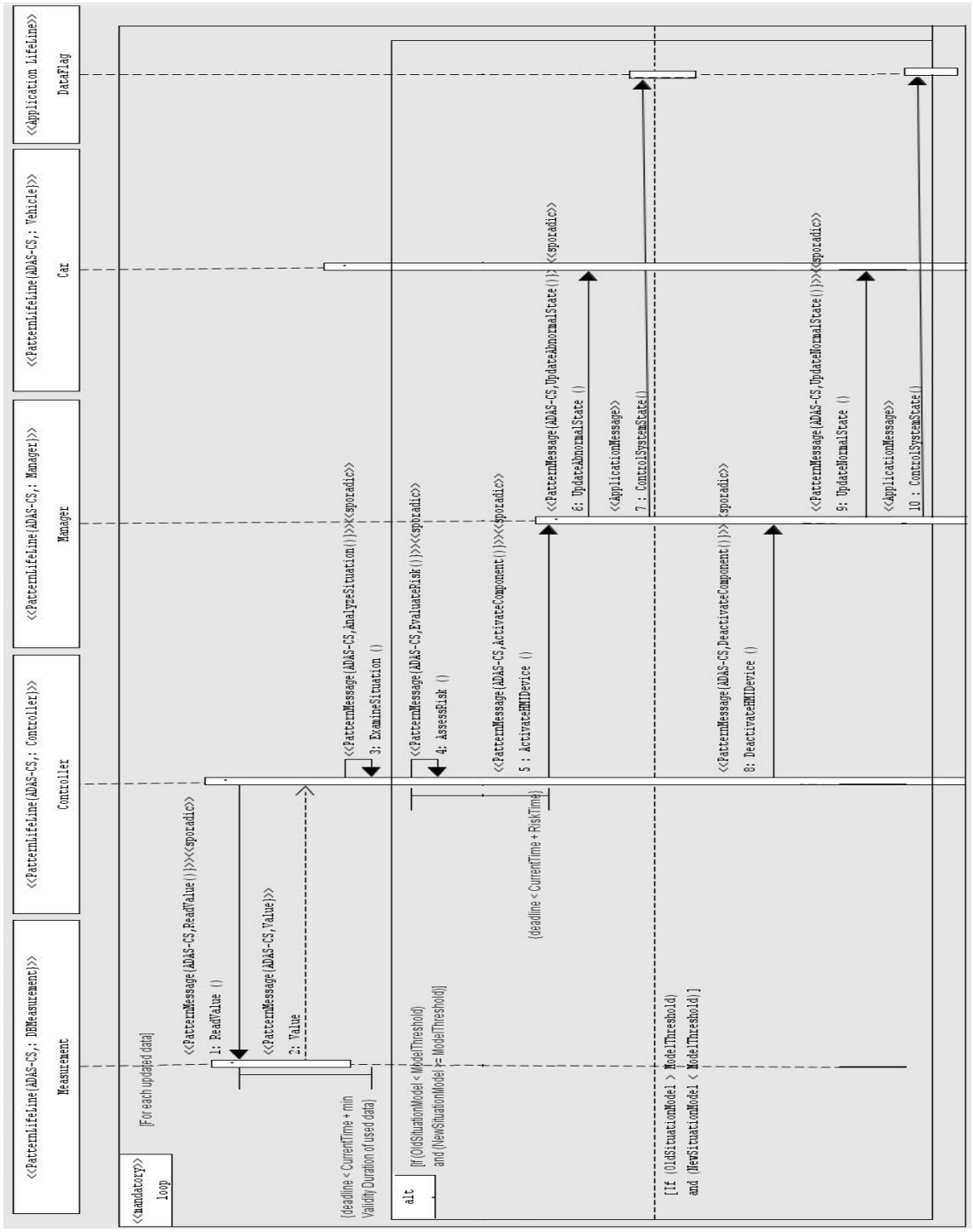


FIGURE B.7 : Modélisation du diagramme de séquence du système FCW par réutilisation de la vue dynamique du patron *"Contrôle de données"*.

La figure B.8 présente la modélisation du diagramme de séquence de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes par réutilisation de la vue dynamique du patron *"Action"*.

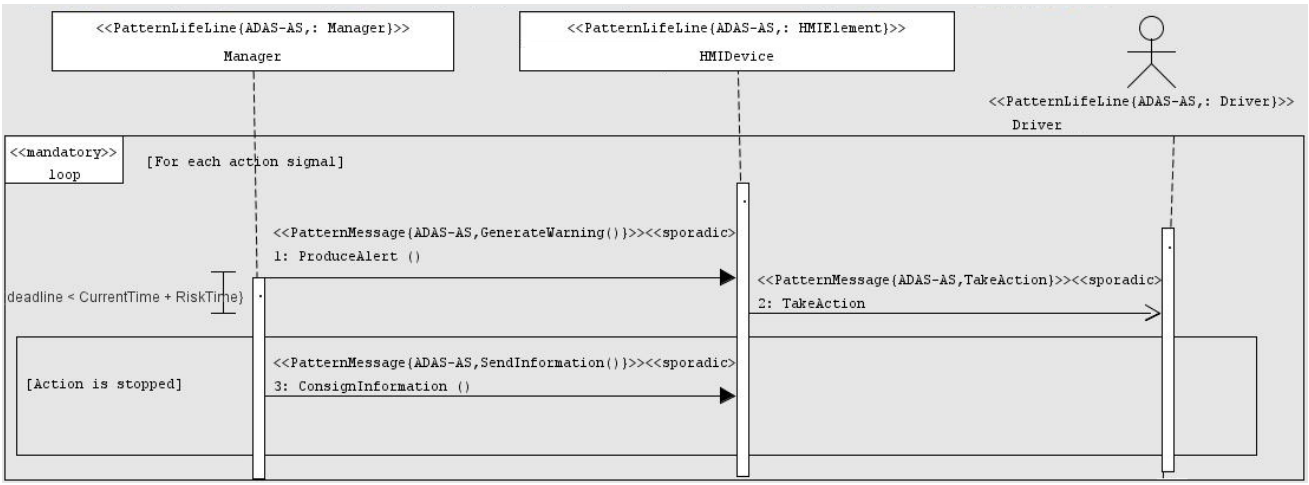


FIGURE B.8 : Modélisation du diagramme de séquence du système FCW par réutilisation de la vue dynamique du patron *"Action"*.

Étude de cas $N°2$: Système SASPENCE

Description du système

Le sous-projet SASPENCE (SAfe SPEed and Safe DistaNCE) a pour objectif principal le développement d'un système capable d'aider le conducteur à maintenir la vitesse maximale autorisée et la distance de sécurité selon une condition de conduite donnée (*i.e.,* type de la route, géométrie de la route, situation du trafic et conditions météorologiques). Ce système consiste à éviter les risques d'accidents liés à une vitesse excessive et une inter-distance inappropriée à une situation particulière.

Le module de décision du système consiste à estimer une trajectoire optimale et à calculer une manœuvre de référence (*i.e.,* calculer la vitesse maximale et la distance de sécurité) en se basant sur la fusion des données relatives à l'état du véhicule et son environnement (géométrie de la route, obstacles, etc.). Si le système de décision détecte une situation critique en comparant la manœuvre de référence avec la manœuvre actuelle, alors il génère des alertes à destination du conducteur (alertes visuelles, sonores ou vibrations de la pédale d'accélération).

SASPENCE utilise un ensemble de capteurs embarqués dans le véhicule : (a) des capteurs extéroceptifs (caméra CCD, DGPS [1] et radars) permettant d'observer l'état de l'environnement extérieur du véhicule (route, obstacle, etc.) et (b) des capteurs proprioceptifs permettant d'observer l'état et la dynamique du véhicule (vitesse de lacet, accélération, vitesse longitudinale, etc.). Les capteurs extéroceptifs sont les suivants :

- Une caméra CCD, située près du rétroviseur, permet d'observer la route et déterminer son état et ses paramètres (largeur de voie, angle de la courbure, position des marqueurs au sol, etc.) avec une bonne précision. Ce capteur permet de mesurer les données toutes les 20 ms.
- DGPS permet de déterminer périodiquement et avec une bonne précision la géométrie de la route ainsi que la position du véhicule.
- Des radars permettent de détecter des objets situés à l'avant du véhicule à une distance d'environ 120 à 150 mètres. Ces capteurs disposent d'une fréquence de transmission de 77 Ghz. Ils permettent de mesurer les données toutes les 10 ms.

Les données acquises, qui sont transmises régulièrement vers une unité de fusion à travers le bus CAN (avec une fréquence élevée de transmission), doivent être mises à jour régulièrement. Elles sont traitées et fusionnées par l'unité de fusion pour disposer d'un état précis de chaque élément observé (le véhicule et la route, y compris les obstacles). Elles sont ensuite utilisées par une unité de décision permettant d'analyser la situation pour surveiller correctement le véhicule et la route, pour détecter les obstacles, y compris les obstacles mobiles (motos, piétons, bus, etc.). Le principe de fonctionnement de l'unité de contrôle se base sur les deux fonctions suivantes :

- une fonction pour maintenir une distance de sécurité entre le véhicule et celui qui le précède afin d'éviter une collision due à un freinage brusque. Cette fonction dépend de certains facteurs, tels que le temps de réaction du conducteur, la distance de freinage et la surface de la route.
- une fonction pour adapter la vitesse du véhicule en fonction de la vitesse maximale autorisée.

1. Differential Global Positioning System

Le contrôleur, qui surveille le véhicule et la route, calcule une manœuvre de référence optimale en se basant sur les informations acquises par les capteurs. Si le contrôleur détecte la présence d'une situation à risque (la distance entre le véhicule et celui qui précède est inférieure à un seuil ou la vitesse du véhicule dépasse la vitesse autorisée), alors il prend la décision convenable à cette situation. Chaque donnée acquise ou calculée est caractérisée par un type, une durée de validité, une date de mise à jour, une unité et le taux de bruit. Chaque donnée acquise est caractérisée par un taux d'erreur maximal qui précise si la donnée doit être mise à jour ou non.

Après avoir déterminé la présence des situations à risque, le contrôleur évalue la gravité du risque (faible, moyen ou critique) et il décide de l'action à mener pour cette situation précédant le risque (Time To Collision $<$ 2s).

Le contrôleur envoie les actions à un gestionnaire pour activer les différents composants de l'interface Homme-Machine. Ces composants permettent de générer des alertes (a) visuelles représentées sous forme d'icônes avec différentes couleurs suivant le degré de gravité, à travers un dispositif d'affichage intégré au tableau de bord, (b) sonores, en utilisant les haut-parleurs et/ou (c) haptiques (des vibrations de la pédale d'accélérateur). Chaque type d'alerte est défini par un niveau, un taux de répétition et une durée. Ces alertes sont transmises au conducteur afin qu'il prenne des actions correctives à temps, c'est à dire qu'il doit réagir le plus tôt possible pour éviter le risque en ajustant la vitesse du véhicule.

Modélisation du système sans réutilisation de patrons

La modélisation du diagramme de classes et du diagramme de séquence sans réutilisation de patrons est illustrée respectivement par les figures B.9 et B.10.

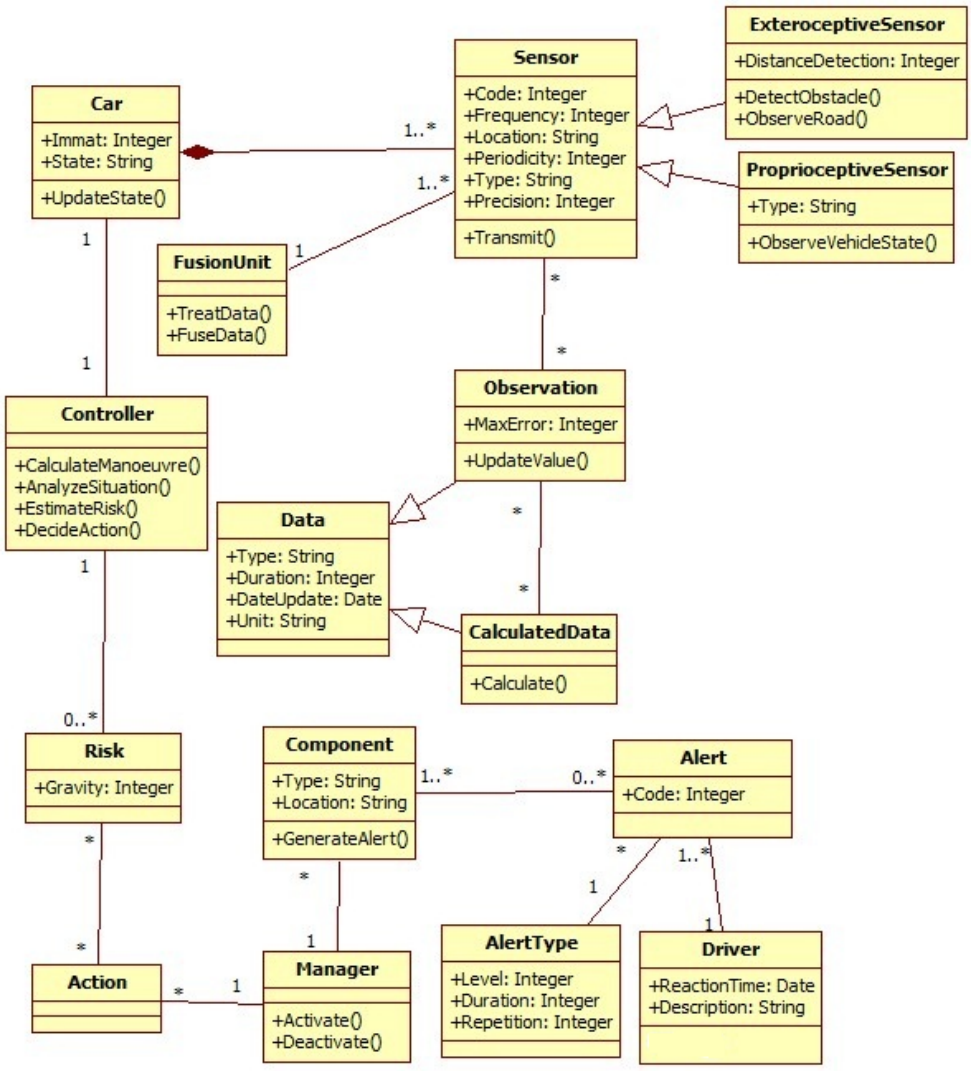


FIGURE B.9 : Modélisation du diagramme de classes du système SASPENCE sans réutilisation de patrons.

Modélisation du système avec réutilisation des patrons

Modélisation des diagrammes de classes

Le diagramme de classes de la fonctionnalité d'acquisition des données modélisé par réutilisation de la vue statique du patron *"Acquisition des données"* est présenté dans la figure B.11.

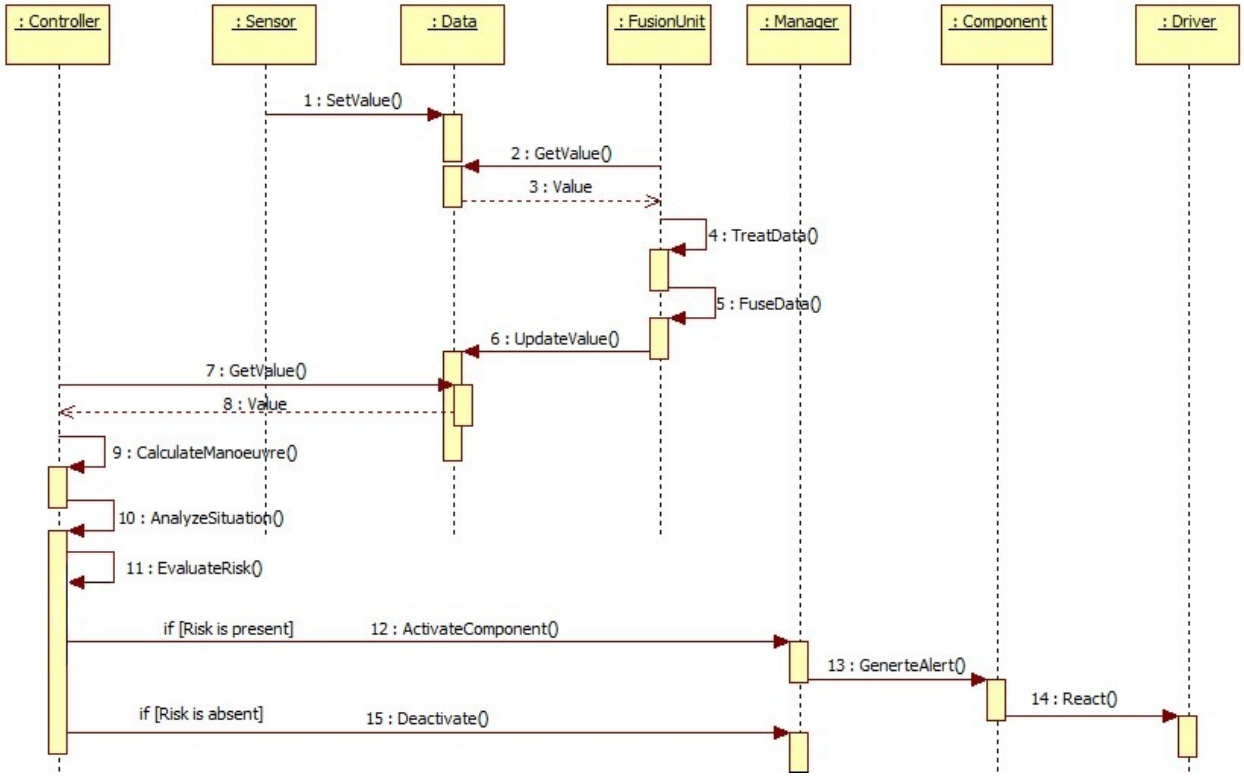


FIGURE B.10 : Modélisation du diagramme de séquence du système SASPENCE sans réutilisation de patrons.

Le diagramme de classes de la fonctionnalité de contrôle et de traitement de données modélisé par réutilisation de la vue statique du patron *"Contrôle de données"* est présenté dans la figure B.12.

Le diagramme de classes de la fonctionnalité d'exécution des commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue statique du patron *"Action"* est présenté dans la figure B.13.

Modélisation des diagrammes de séquence

Le diagramme de séquence de la fonctionnalité d'acquisition des données modélisé par réutilisation de la vue dynamique du patron *"Acquisition des données"* est présenté dans la figure B.14.

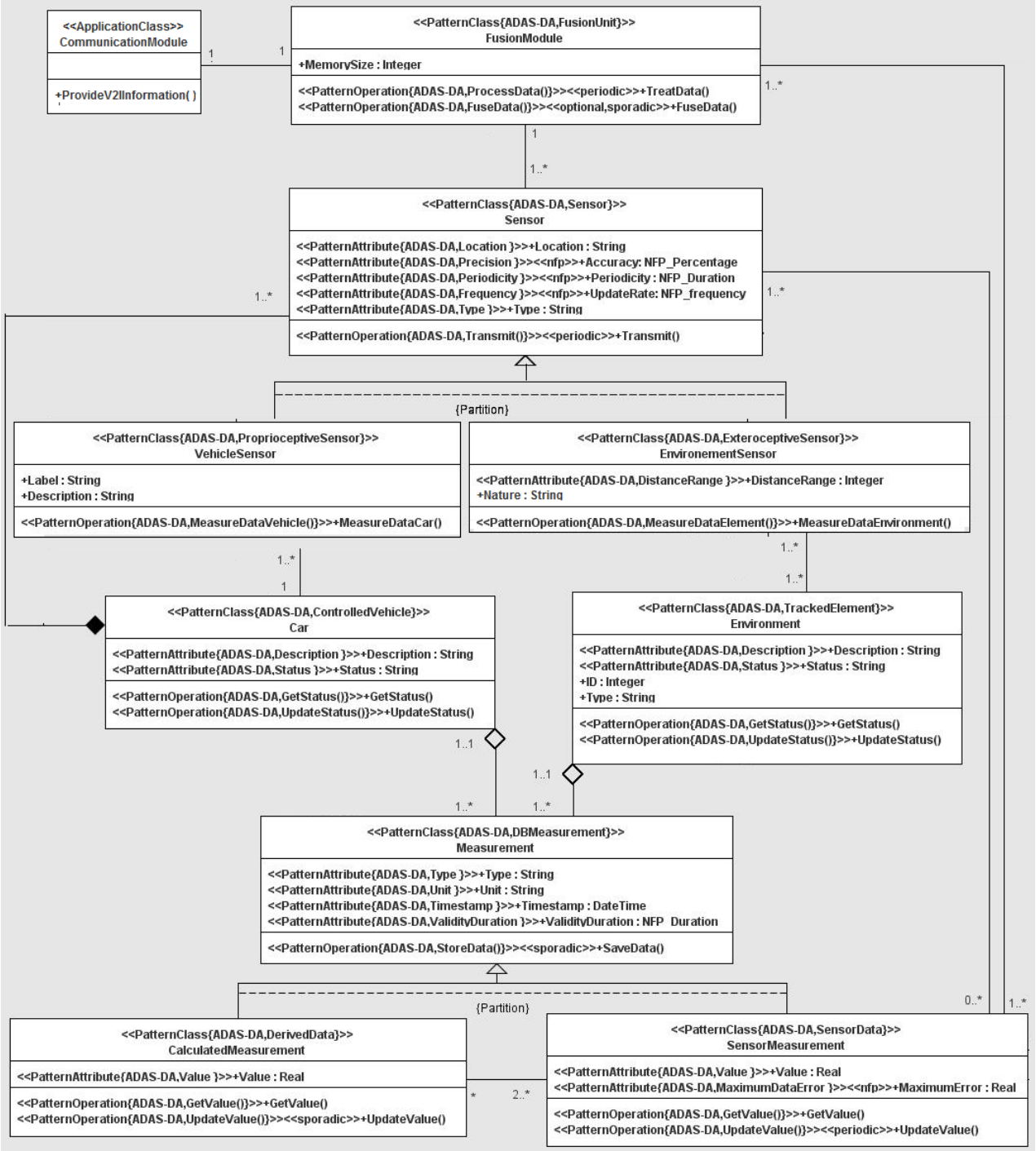


FIGURE B.11 : Modélisation du diagramme de classes du système SASPENCE par réutilisation de la vue statique du patron *"Acquisition des données"*.

Le diagramme de séquence de la fonctionnalité de contrôle et de traitement de données modélisé par réutilisation de la vue dynamique du patron *"Contrôle de données"* est présenté dans la figure B.15.

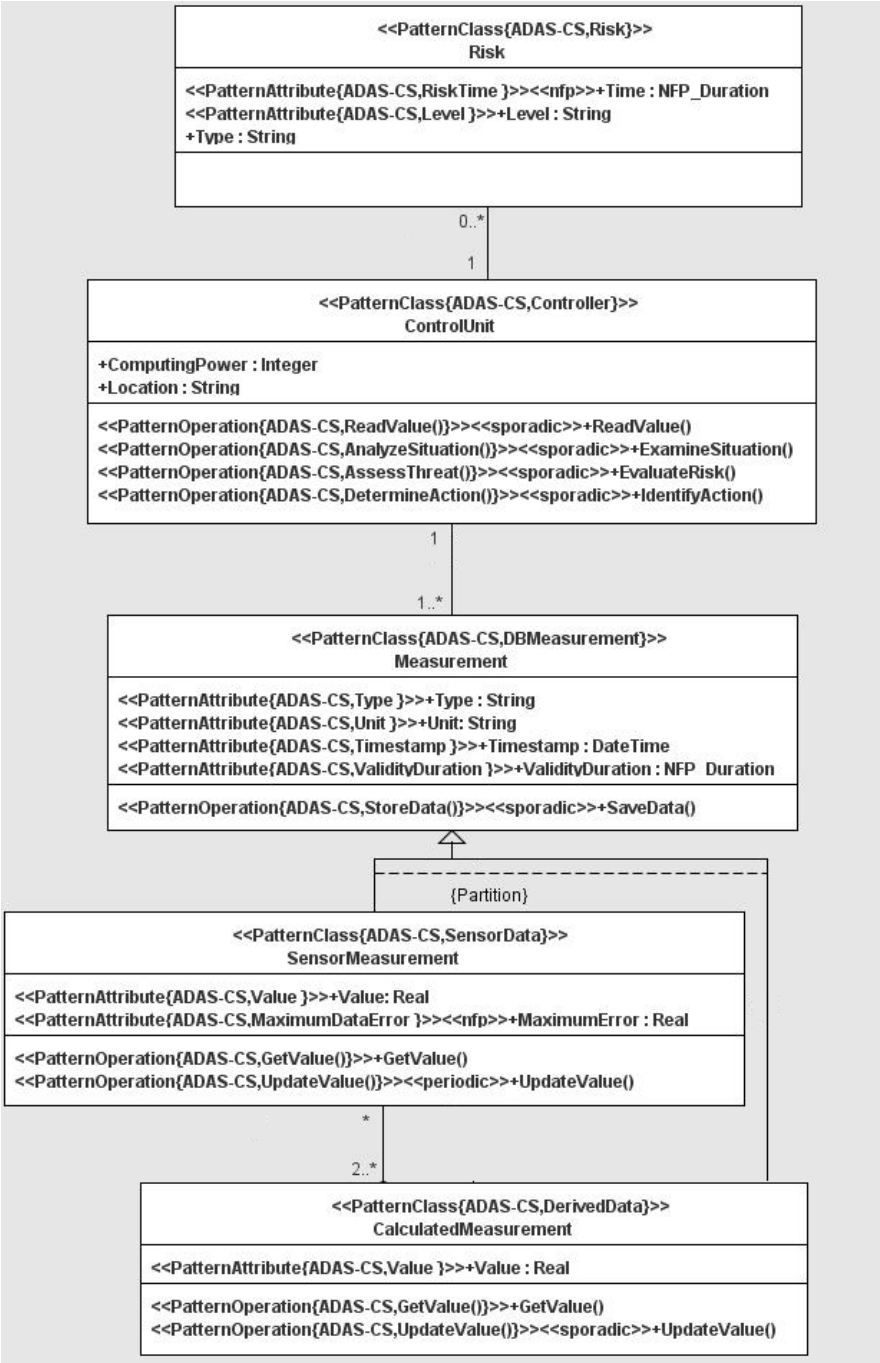


FIGURE B.12 : Modélisation du diagramme de classes du système SASPENCE par réutilisation de la vue statique du patron *"Contrôle de données"*.

Le diagramme de séquence de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue dynamique du patron *"Action"* est présenté dans la figure B.16.

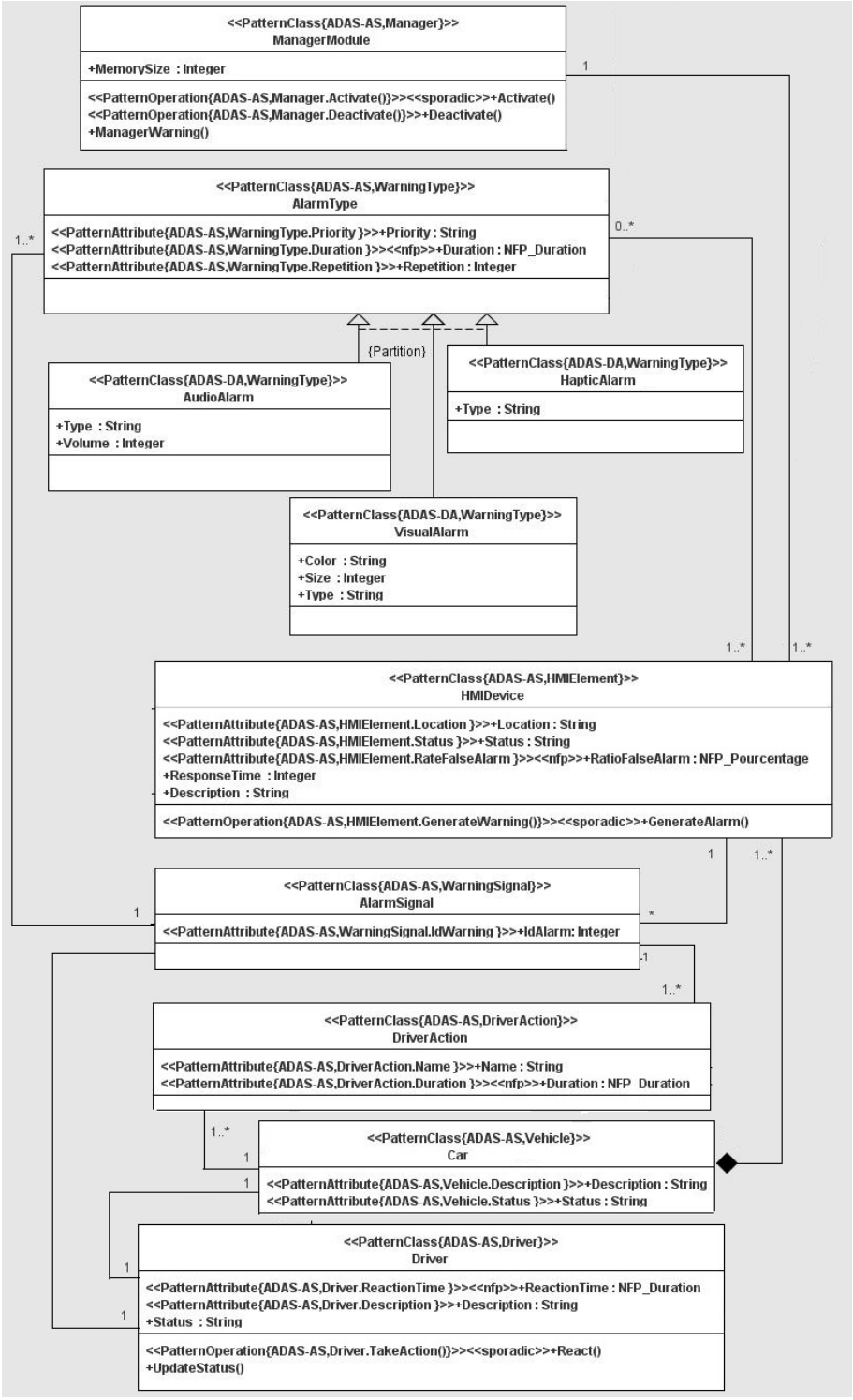


FIGURE B.13 : Modélisation du diagramme de classes du système SASPENCE par réutilisation de la vue statique du patron *"Action"*.

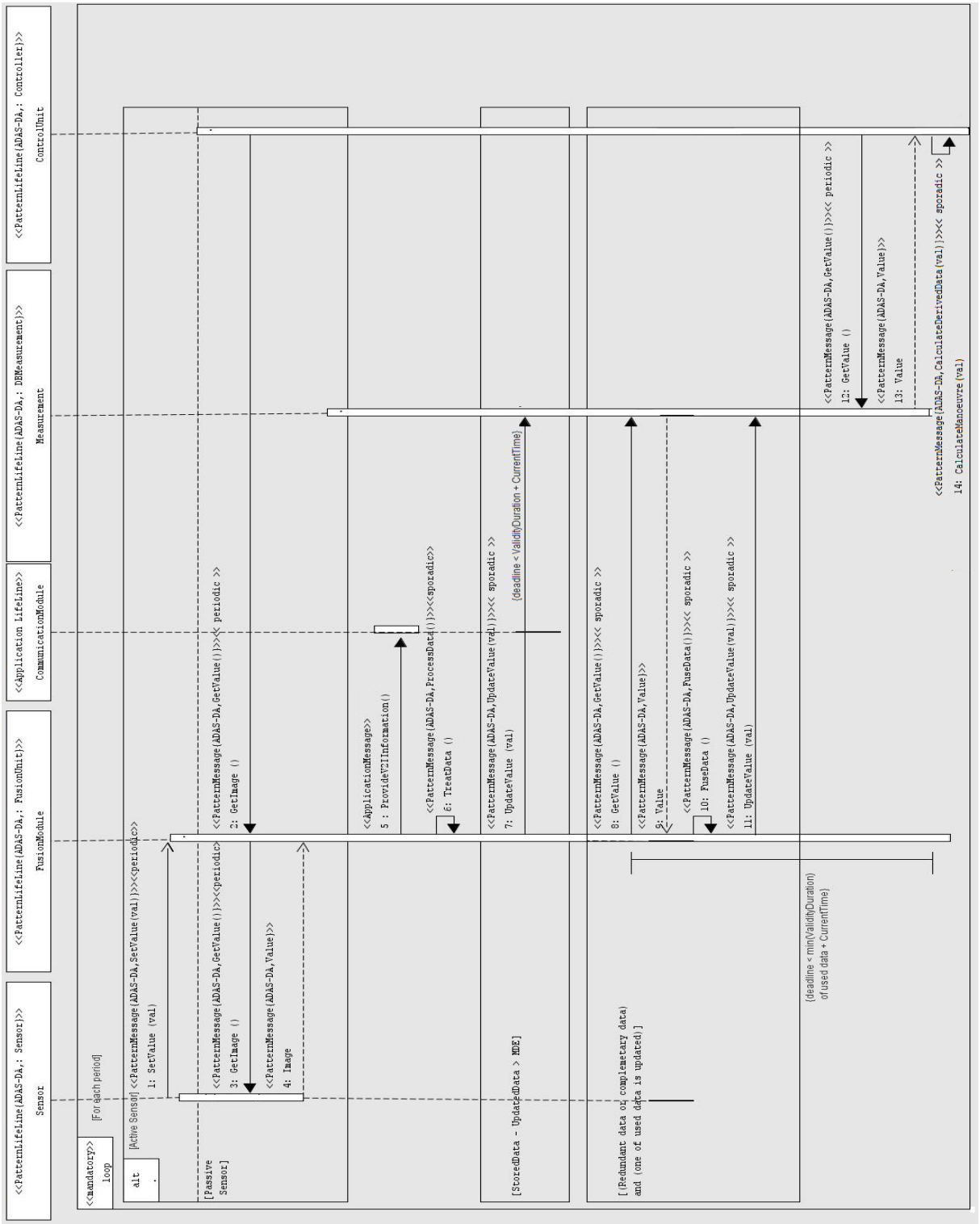


FIGURE B.14 : Modélisation du diagramme de séquence du système SASPENCE par réutilisation de la vue dynamique du patron *"Acquisition des données"*.

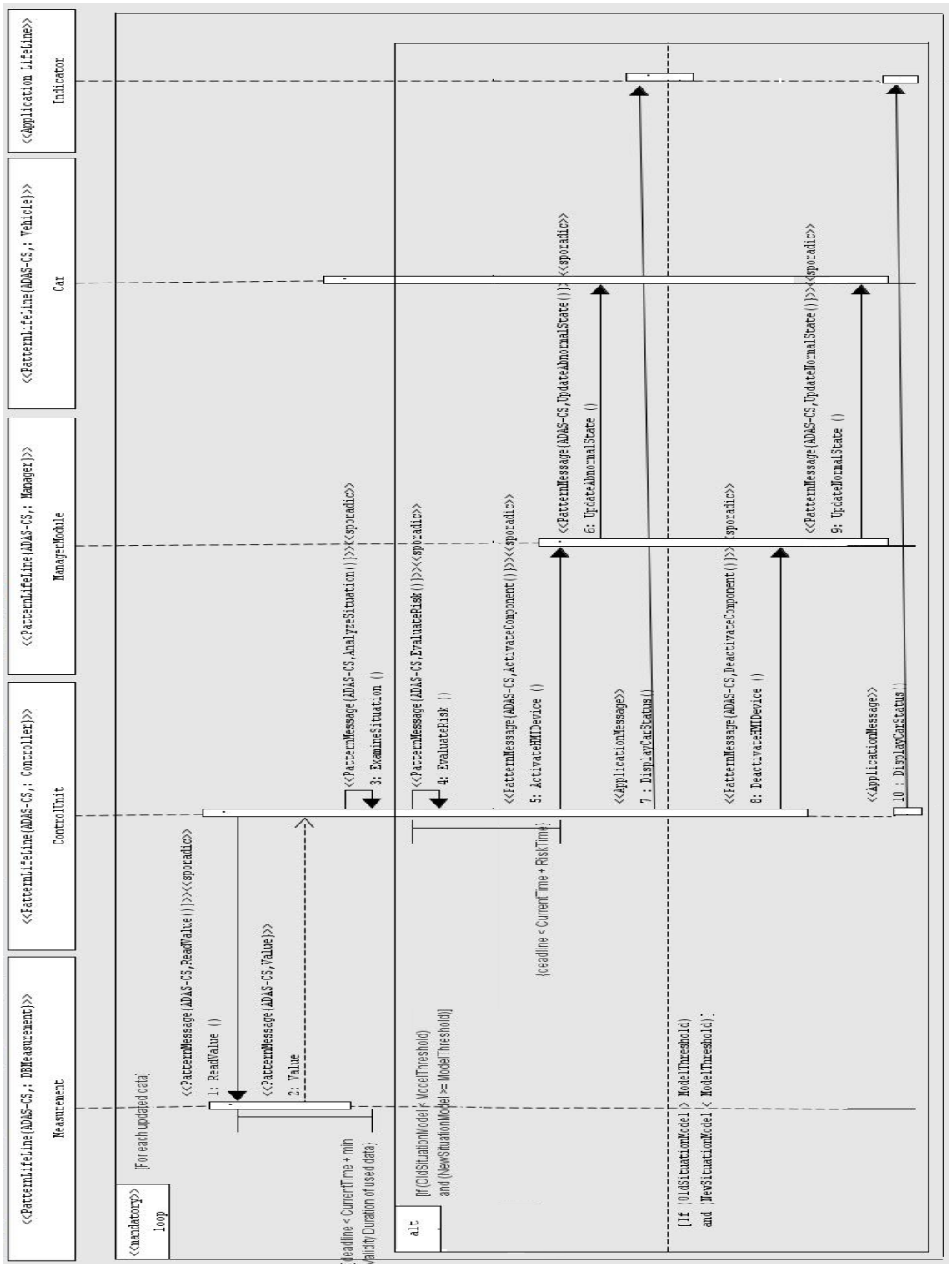


FIGURE B.15 : Modélisation du diagramme de séquence du système SASPENCE par réutilisation de la vue dynamique du patron *"Contrôle de données"*.

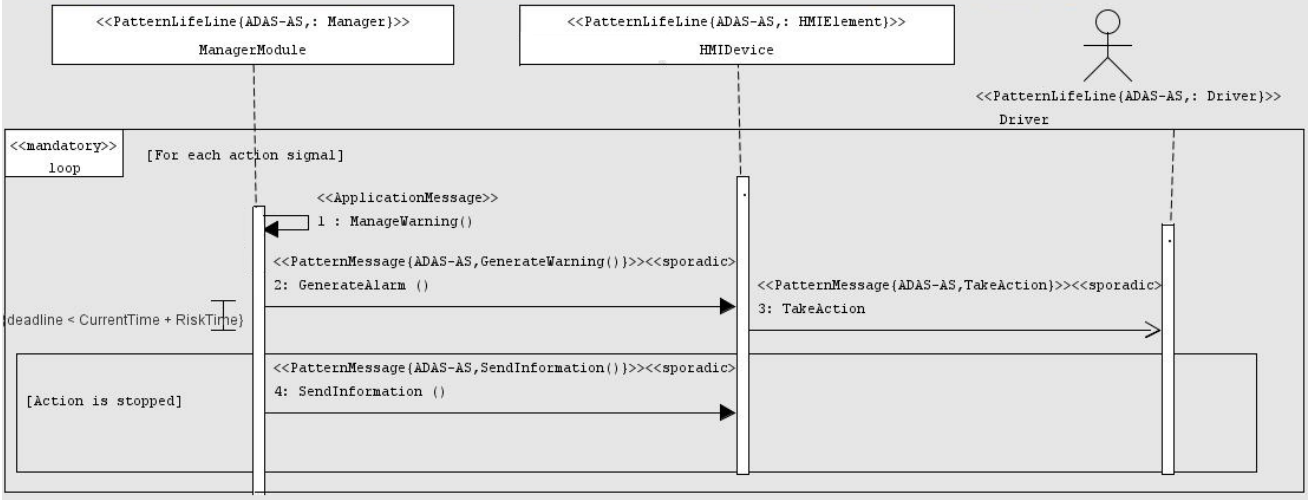


FIGURE B.16 : Modélisation du diagramme de séquence du système SASPENCE par réutilisation de la vue dynamique du patron *"Action"*.

Étude de cas $N°3$: Système COMPOSE

Description du système

Le sous-projet COMPOSE a été réalisé au sein du projet PReVENT. Il avait pour but de développer un système d'aide à la conduite permettant d'éviter les risques de collision.

Ce système a été mis en œuvre dans les camions VOLVO. Le principe de son fonctionnement se base sur la surveillance de la route (autoroutes et routes nationales) et la détection des obstacles mobiles (automobiles, motos et piétons) à l'aide de capteurs extéroceptifs (*i.e.,* permettant d'observer l'état de l'environnement extérieur du véhicule).

COMPOSE utilise un ensemble de capteurs embarqués dans le camion :

- Une caméra infrarouge, placée à l'avant du camion, permet de surveiller la route et de détecter la présence d'obstacles. La caméra dispose d'un champ de vision de 48°x37° (largeur x hauteur) et une résolution de 384x288 pixels. La quantité de données produites par la détection infrarouge est de l'ordre de quelques Ko/s.

- Un laser mesure les distances entre 0,3 et 80 mètres. Le capteur peut être utilisé avec différentes fréquences de balayage produisant différentes résolutions angulaires. Le laser est intégré dans le coin gauche du camion avec un champ de vision horizontal de 215° pour suivre le véhicule qui précède.
- Des capteurs proprioceptifs qui permettent d'observer l'état du camion (vitesse longitudinale, vitesse de rotation et angle).

Les capteurs transmettent les données sensorielles périodiquement à travers le bus CAN. Ces données doivent être mises à jour régulièrement, et sont utilisées pour calculer des paramètres afin de connaître l'état précis de chaque élément observé (le camion et les obstacles détectés). Chaque donnée sensorielle ou calculée est caractérisée par un type, une durée de validité, une date de mise à jour et une unité. Pour les données sensorielles, il faut aussi préciser le taux d'erreur maximale qui indique si la donnée doit être mise à jour ou non.

Le contrôleur reçoit les données acquises afin de surveiller correctement le véhicule et les obstacles détectés. Il permet (i) d'analyser la situation, (ii) d'évaluer la situation (faible, moyen ou grave) et (iii) de déterminer l'action à mener pour cette situation avant qu'une collision ne se produise (Time To Collision TTC). Si le contrôleur détermine un risque de collision, il calcule le paramètre de freinage requis pour éviter la collision. Le contrôleur envoie les actions à un gestionnaire d'actions permettant d'activer les composants responsable à exécuter l'action. En effet, ce gestionnaire peut activer (i) des actionneurs automatiques (freins) qui permettent d'exécuter des commandes sur les composants du véhicule (le freinage automatique doit s'effectuer avant que le risque ne soit avéré) et/ou (ii) des composants de l'interface Homme-Machine situés au niveau du tableau de bord. Ces composants permettent de générer des alertes visuelles pour avertir le conducteur. Chaque type d'alerte est caractérisée par une durée, une priorité et un taux de répétition.

Les alertes visuelles sont représentées sous formes de pictogrammes de différentes couleurs selon le degré de gravité de la situation. Elles sont fournies au conducteur afin de prendre des actions correctives à temps (il a besoin d'un temps de réaction réagir qui dépend de son état : expérience, fatigue et somnolence).

Les actionneurs automatiques et les composants de l'interface Homme-Machine sont activés lorsque le système détecte un risque de collision. Le camion s'arrête automatiquement afin d'éviter la collision à la suite des actions prises, soit par l'actionneur automatique ou par le conducteur.

Modélisation du système sans réutilisation de patrons

La figure B.17 et la figure B.18 présentent respectivement le diagramme de classes et le diagramme de séquence du système COMPOSE modélisés sans réutilisation des patrons proposés.

Modélisation du système avec réutilisation de patrons

Modélisation des diagrammes de classes

Le diagramme de classes de la fonctionnalité d'acquisition des données modélisé par réutilisation de la vue statique du patron *"Acquisition des données"* est présenté dans la figure B.19.

Le diagramme de classes de la fonctionnalité de contrôle et de traitement des données modélisé par réutilisation de la vue statique du patron *"Contrôle de données"* est présenté dans la figure B.20.

Le diagramme de classes de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue statique du patron *"Action"* est présenté dans la figure B.21.

Modélisation des diagrammes de séquence

Le diagramme de séquence de la fonctionnalité d'acquisition des données modélisé par réutilisation de la vue dynamique du patron *"Acquisition des données"* est présenté dans la figure B.22.

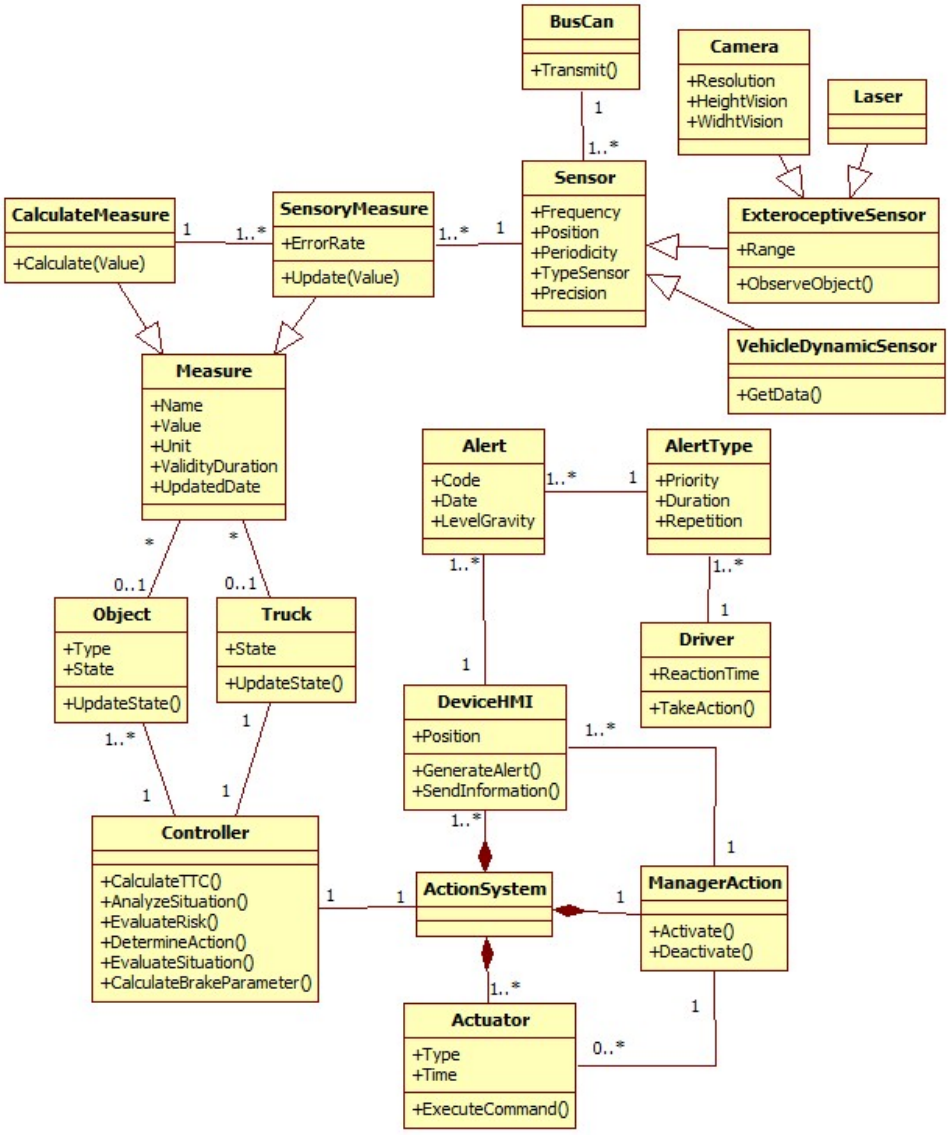


FIGURE B.17 : Modélisation du diagramme de classes du système COMPOSE sans réutilisation de patrons.

Le diagramme de séquence de la fonctionnalité de contrôle et de traitement des données modélisé par réutilisation de la vue dynamique du patron *"Contrôle de données"* est présenté dans la figure B.23.

Le diagramme de séquence de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue dynamique du patron *"Action"* est présenté dans la figure B.24.

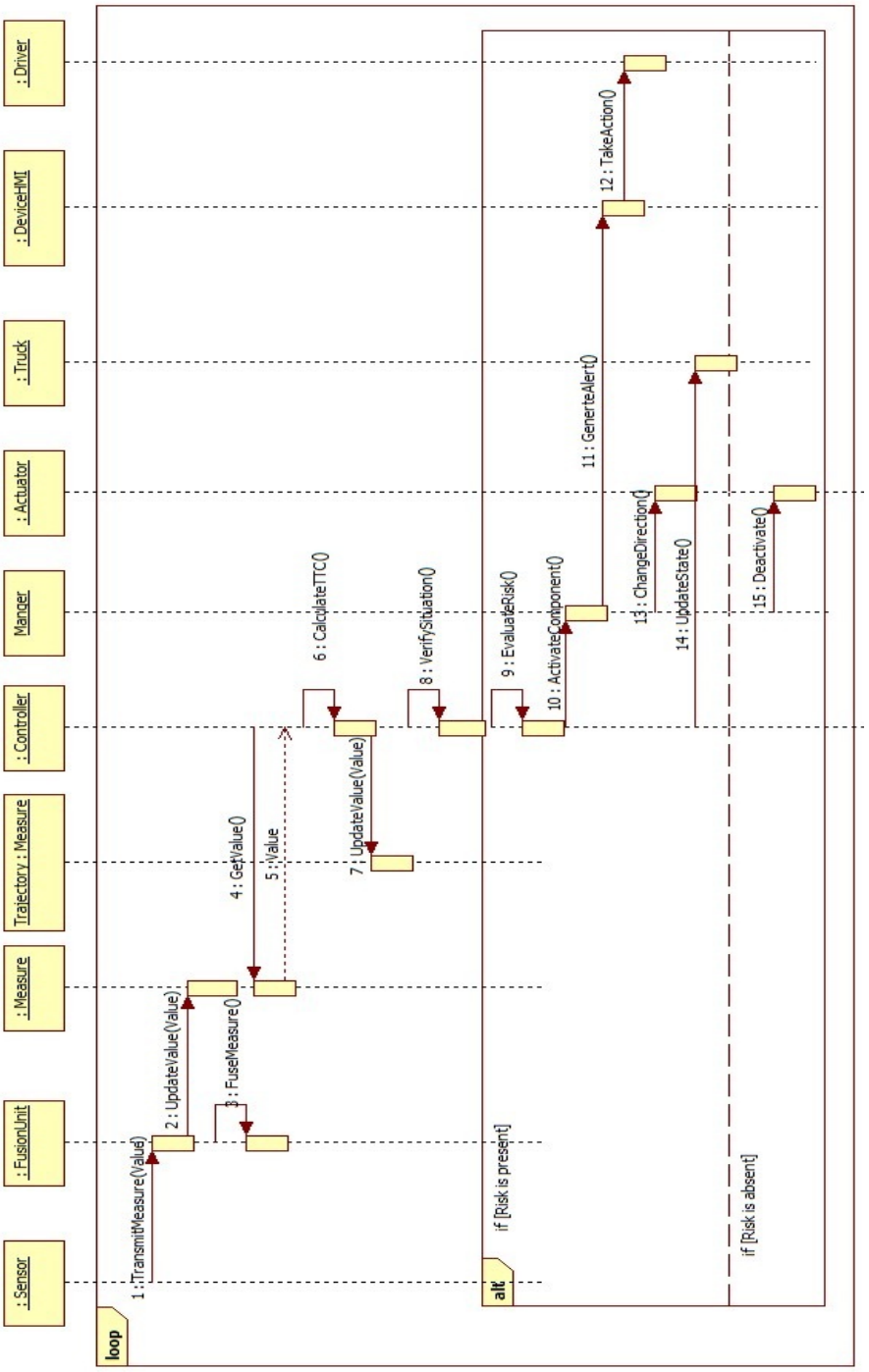


FIGURE B.18 : Modélisation du diagramme de séquence du système COMPOSE sans réutilisation de patrons.

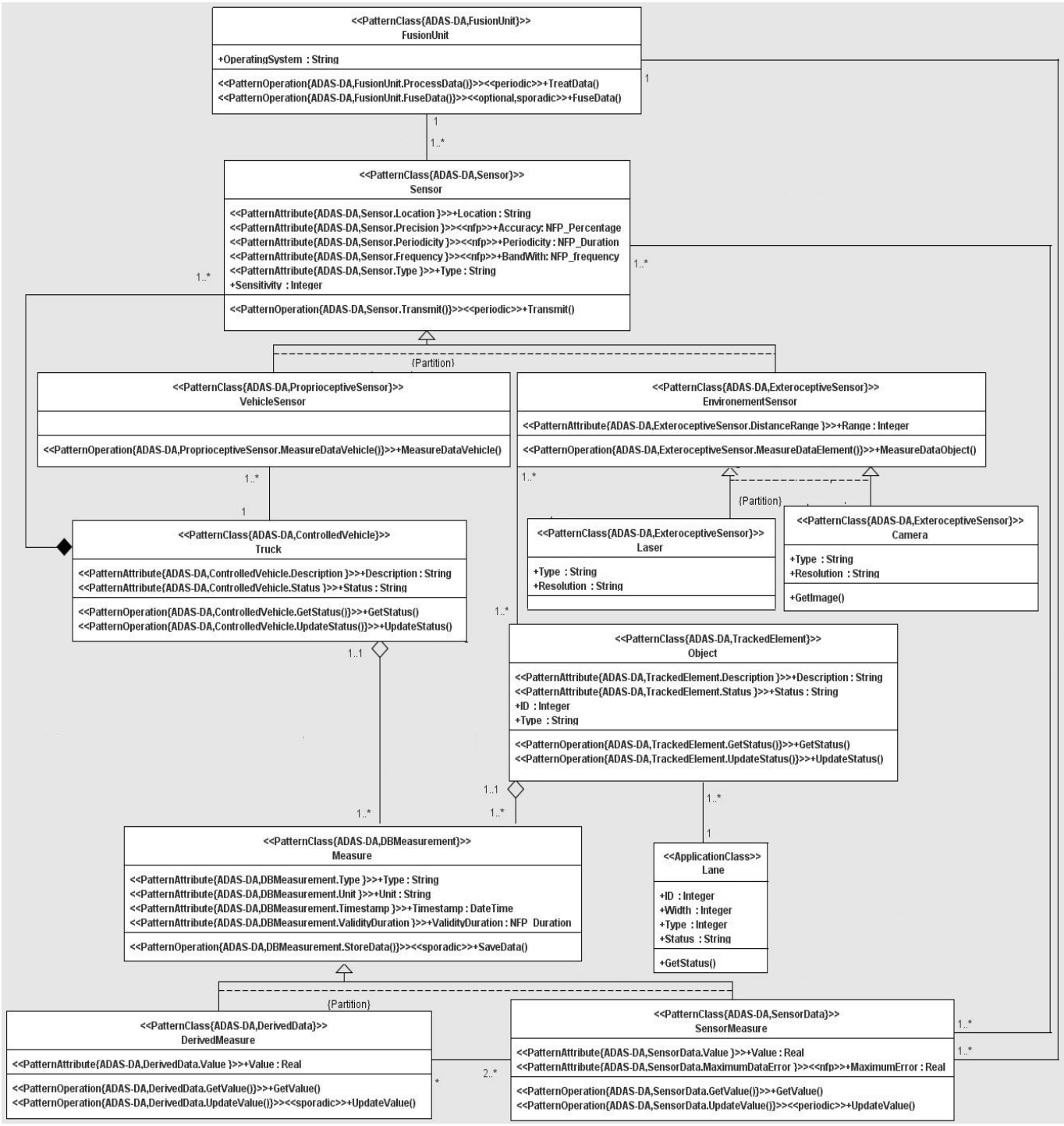


FIGURE B.19 : Modélisation du diagramme de classes du système COMPOSE par réutilisation de la vue statique du patron *"Acquisition des données"*.

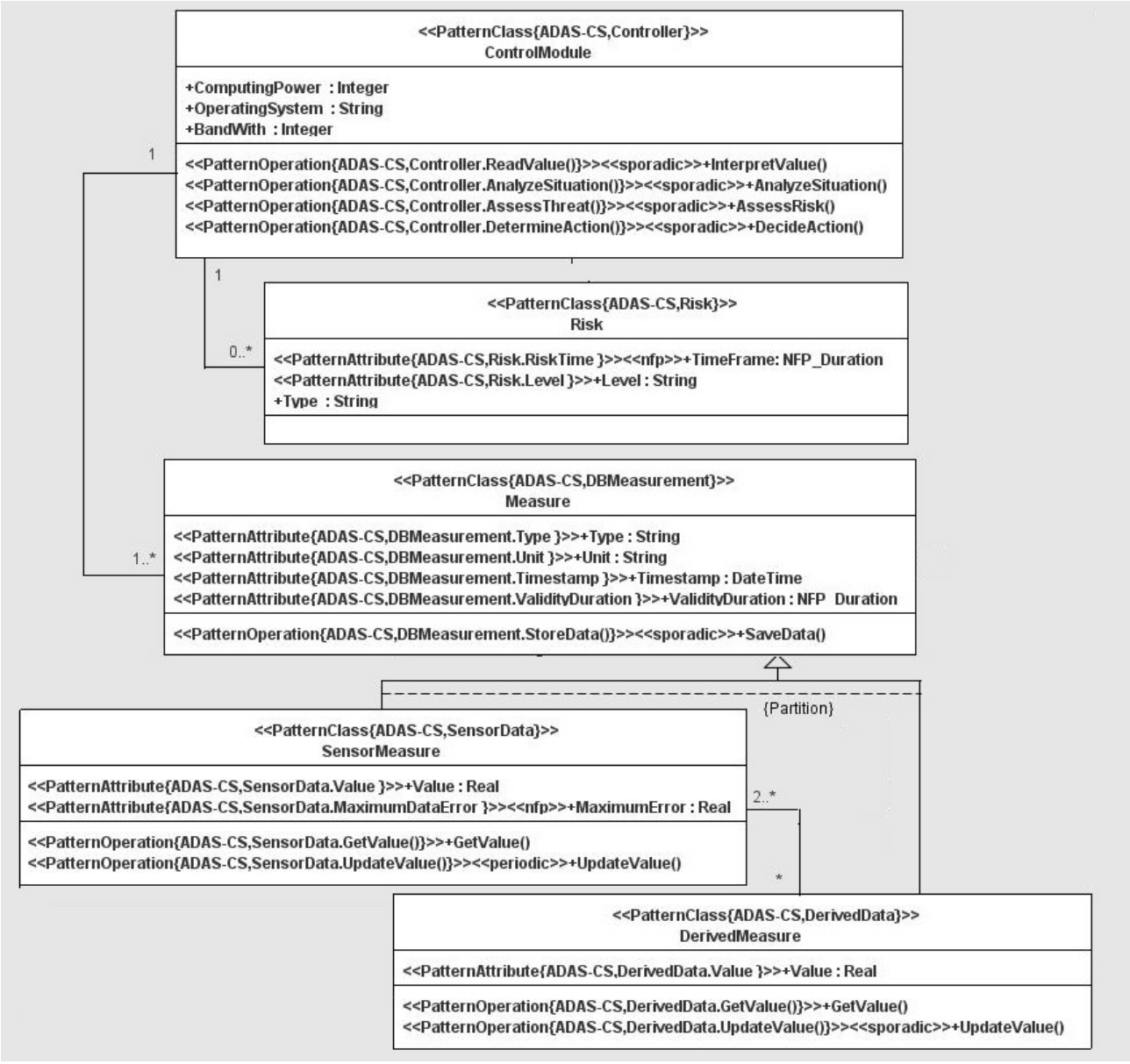


FIGURE B.20 : Modélisation du diagramme de classes du système COMPOSE par réutilisation de la vue statique du patron *"Contrôle de données"*.

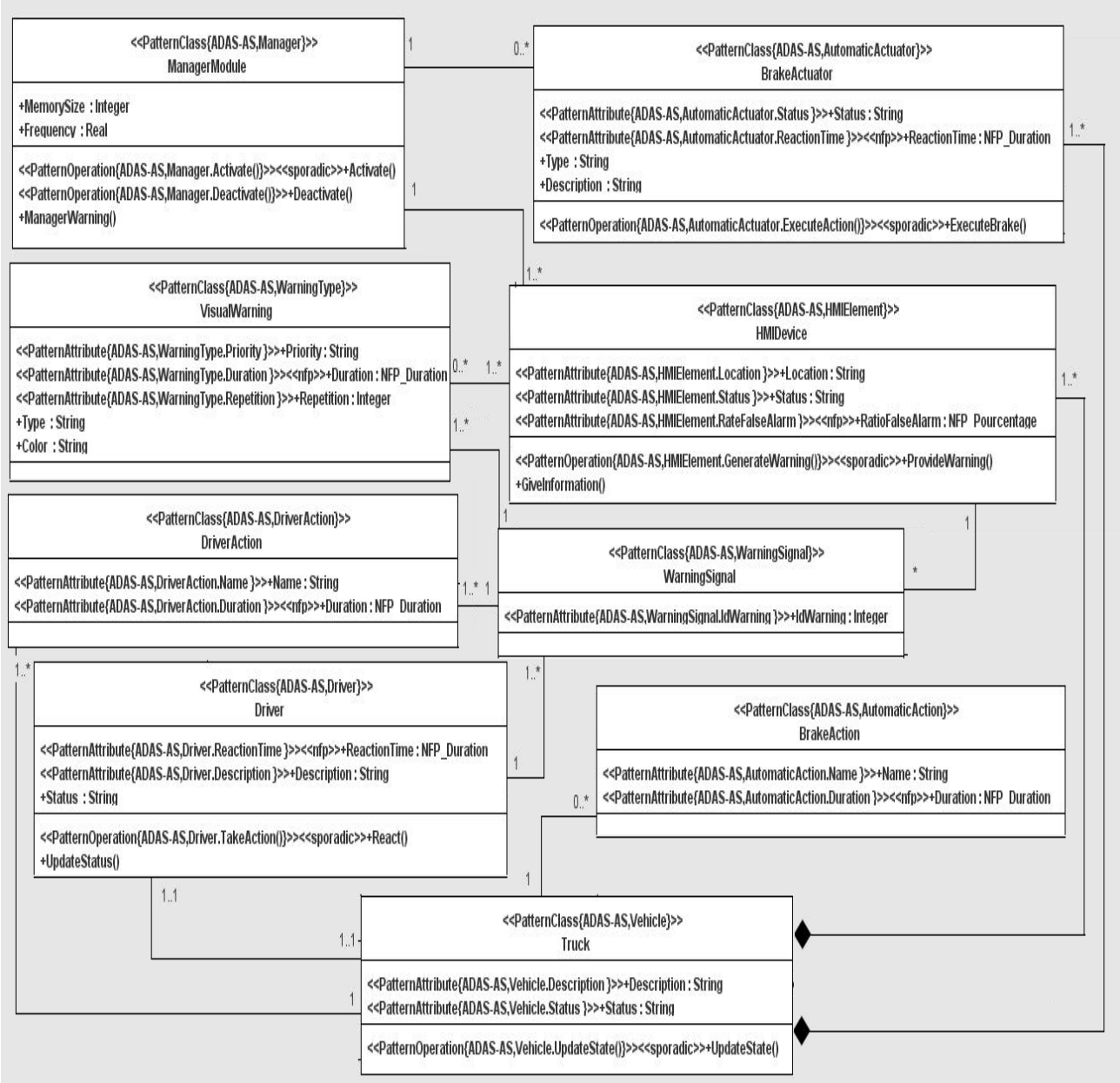


FIGURE B.21 : Modélisation du diagramme de classes du système COMPOSE par réutilisation de la vue statique du patron *"Action"*.

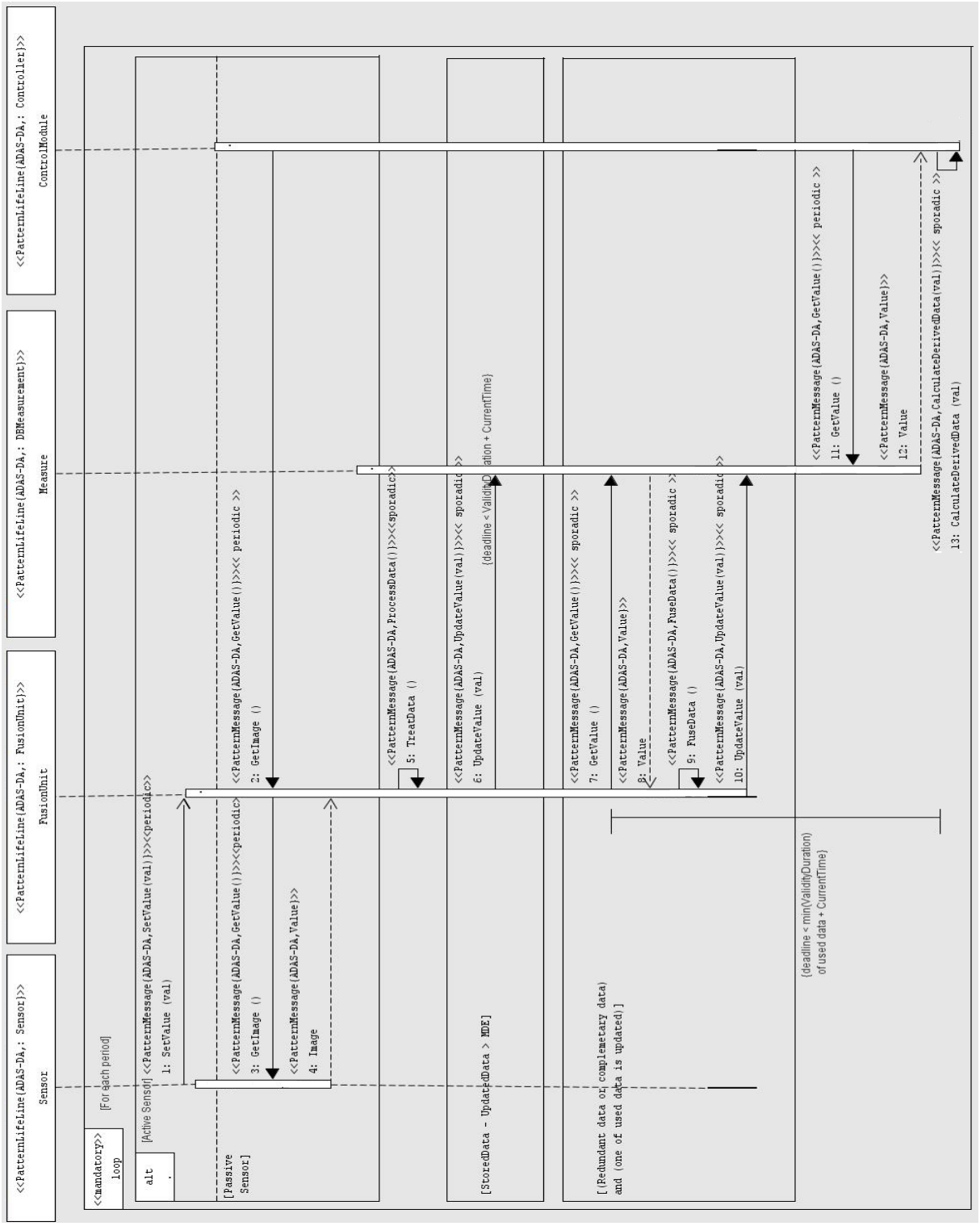


FIGURE B.22 : Modélisation du diagramme de séquence du système COMPOSE par réutilisation de la vue dynamique du patron *"Acquisition des données"*.

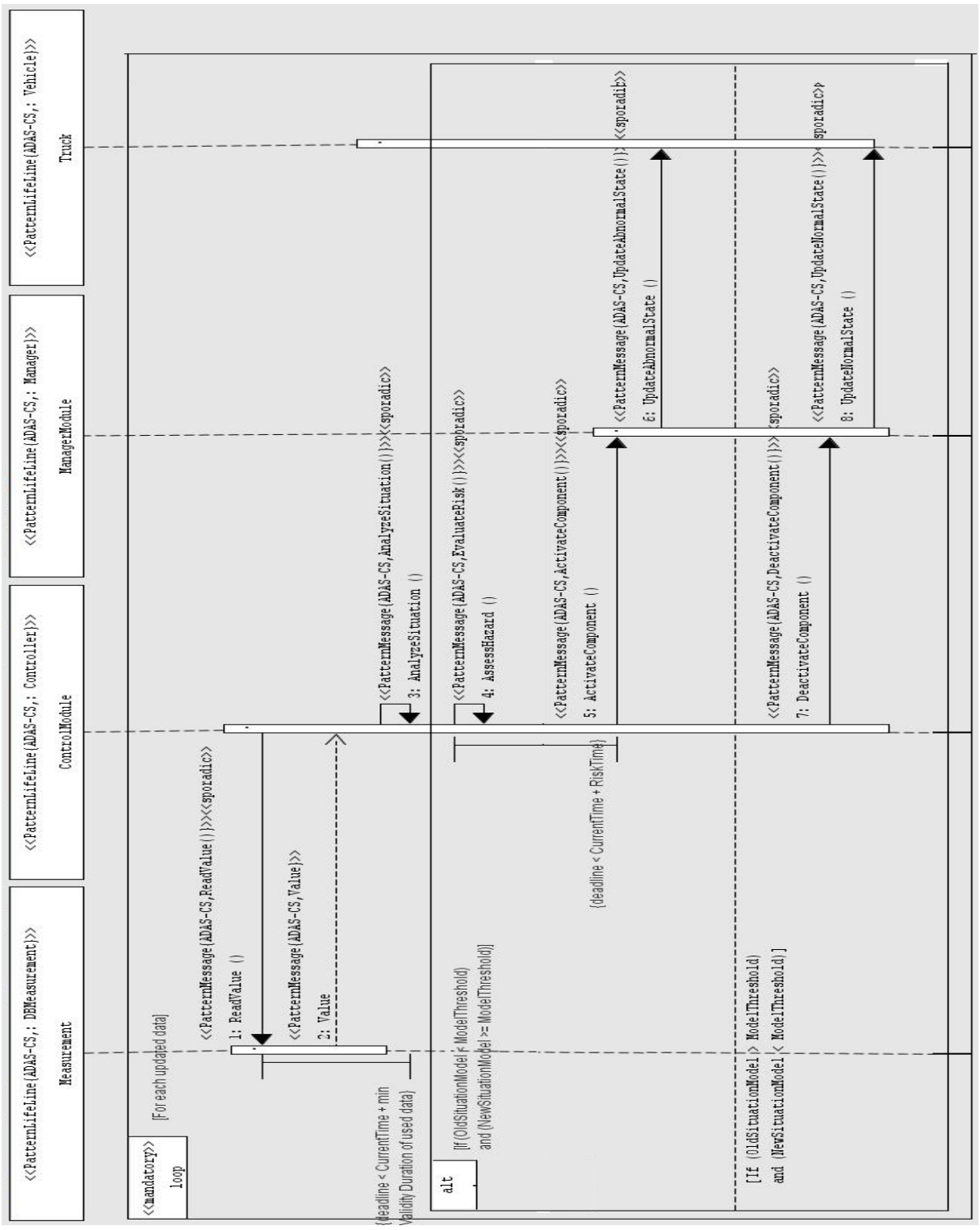


FIGURE B.23 : Modélisation du diagramme de séquence du système COMPOSE par réutilisation de la vue dynamique du patron *"Contrôle de données"*.

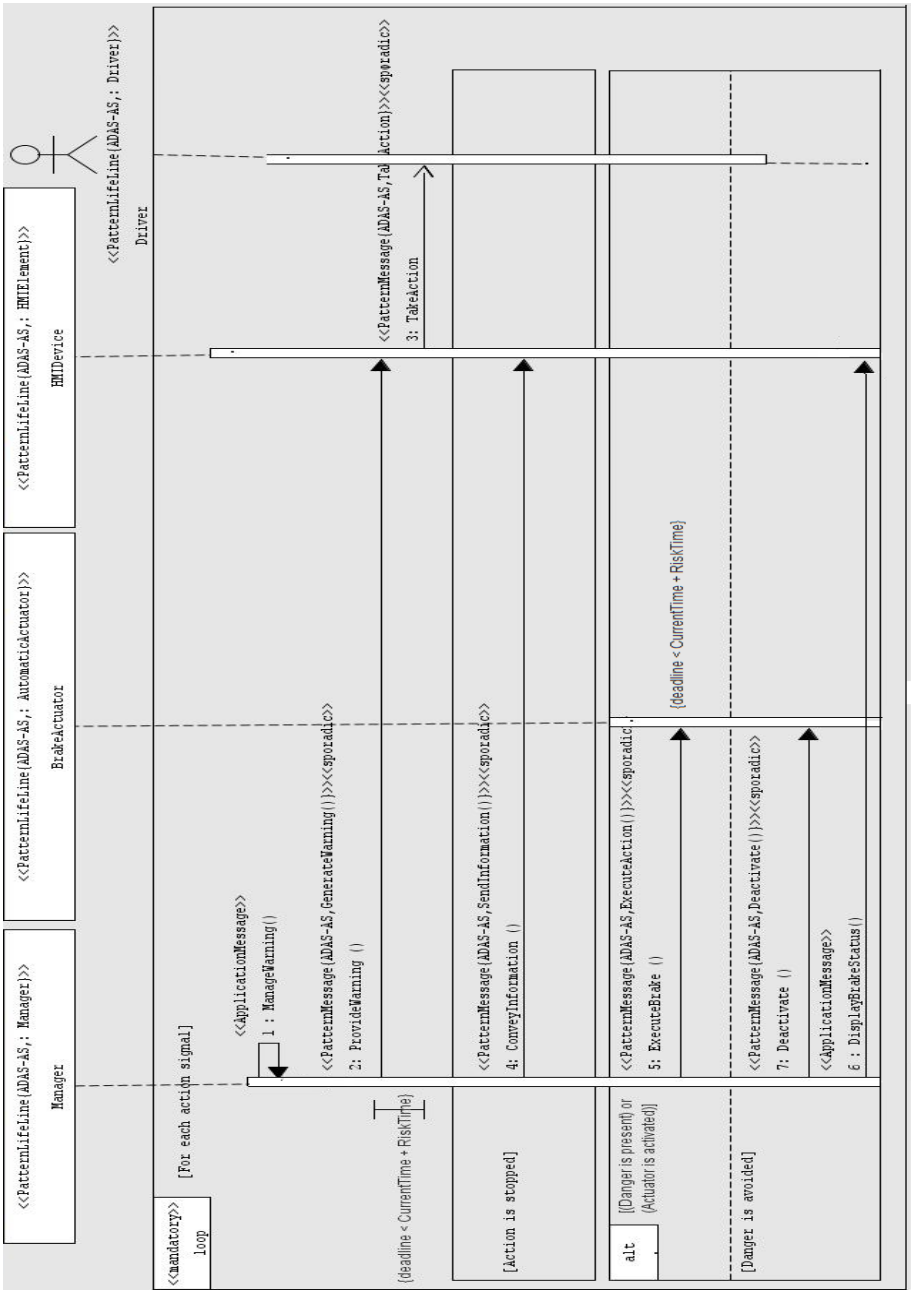


FIGURE B.24 : Modélisation du diagramme de séquence du système COMPOSE par réutilisation de la vue dynamique du patron *"Action"*.

Étude de cas $N°4$: Système d'assistance de détection d'angles morts

Description du système

Le système de détection d'angles morts (BLA : Blind Spot Assist system) chez VOLVO permet d'avertir le conducteur de la présence d'un véhicule dans l'angle mort, une zone hors du champ de vision du conducteur sur le côté de la voiture. Lorsque le clignotant est activé, le système informe le conducteur par un signal visuel qu'il n'est pas indiqué de changer la voie à ce moment-là au de vu la présence d'un véhicule situé quelques mètres derrière la voiture, sur une file voisine. Ce système peut avoir de nombreuses fausses alertes causées par les véhicules stationnés ou les véhicules de petites tailles comme les bicyclettes. Ces fausses alertes se produisent lorsque le capteur détecte un véhicule qui n'existe pas. Ce système utilise des capteurs embarqués dans la voiture :

- Des capteurs proprioceptifs actifs permettent de mesurer l'état et la dynamique relatifs au véhicule avec une bonne précision et une haute fréquence.
- Des caméras numériques passives sont situées dans chaque rétroviseur extérieur latéral. Elles permettent d'acquérir des images pour détecter d'autres véhicules (camions, bicyclettes, etc.) roulant sur les côtés de la voiture. Elles sont caractérisées par un champ de vision de 120°, une résolution de 101376 pixels, une distance de détection couvrant 3 mètres sur les côtés et une dimension de 41 x 28 x 35 mm. Elles acquièrent des images avec une bonne précision et une haute fréquence. De plus, elles doivent détecter d'une manière fiable les véhicules de petites tailles afin de réduire le nombre de fausses alarmes.

Chaque capteur transmet périodiquement les mesures vers une unité de fusion qui permet de traiter ces mesures et d'extraire des informations sur les véhicules détectés sur les côtés du véhicule équipé du système. Chaque mesure est mise à jour et enregistrée dans une base de données lorsque la différence entre la valeur stockée et la valeur mise à jour est supérieure à une erreur maximale associée à chaque mesure.

Le contrôleur analyse la situation de conduite et calcule la distance entre la voiture et le véhicule détecté en latéral en utilisant les mesures acquises. De plus, il utilise ces mesures pour régler le seuil des alertes. Chaque mesure est caractérisée par une valeur, une unité, un instant de mise à jour et une durée de validité.

Une fois que le contrôleur détecte que la distance est supérieure ou égale à la distance de sécurité, il évalue le degré de gravité du risque, détermine l'action appropriée à la situation et l'envoie à un gestionnaire d'interface homme-machine. Ce dernier active les indicateurs à LED de dimension 16 mm placés dans les rétroviseurs extérieurs pour que le conducteur puisse prendre les actions correctives à temps. Ces indicateurs affichent des témoins de différents couleurs éclairés selon le niveau de gravité du risque. En effet, le système peut signaler deux niveaux d'alertes : une alerte imminente ou une alerte de précaution. Ces alertes sont caractérisées par un instant d'émission, une durée, un taux de répétition et une priorité. Une fois que le conducteur exécute l'action, la voiture rétablit son état et un indicateur affiche que le signal a été généré et que le conducteur a pris l'action corrective.

Modélisation du système sans réutilisation de patrons

La figure B.25 et la figure B.26 présentent respectivement le diagramme de classes et le diagramme de séquence du système BLA modélisés sans réutilisation des patrons proposés.

Modélisation du système avec réutilisation de patrons

Modélisation des diagrammes de classes

Le diagramme de classes de la fonctionnalité d'acquisition des données modélisé par réutilisation de la vue statique du patron *"Acquisition des données"* est présenté dans la figure B.27.

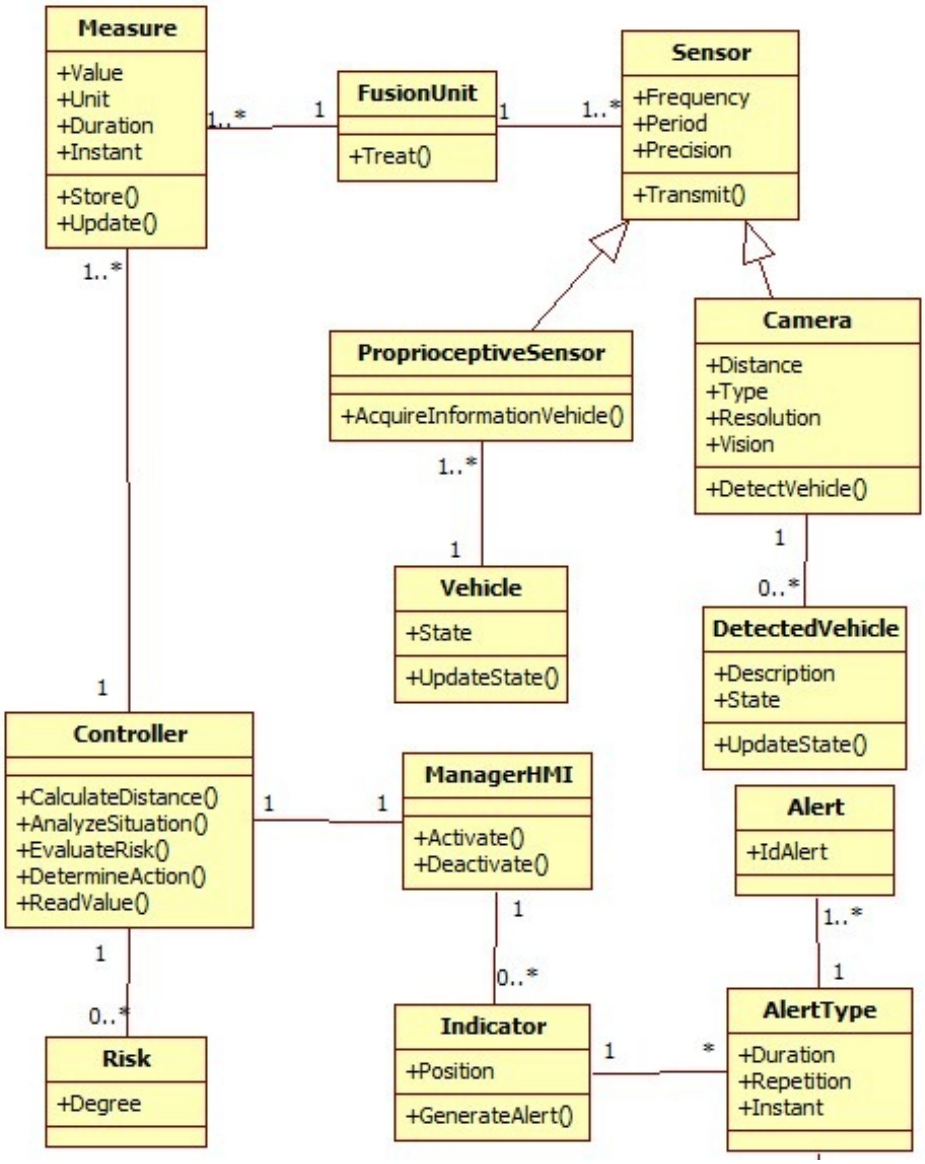


FIGURE B.25 : Modélisation du diagramme de classes du système BLA sans réutilisation de patrons.

Le diagramme de classes de la fonctionnalité de contrôle et de traitement des données modélisé par réutilisation de la vue statique du patron *"Contrôle de données"* est présenté dans la figure B.28.

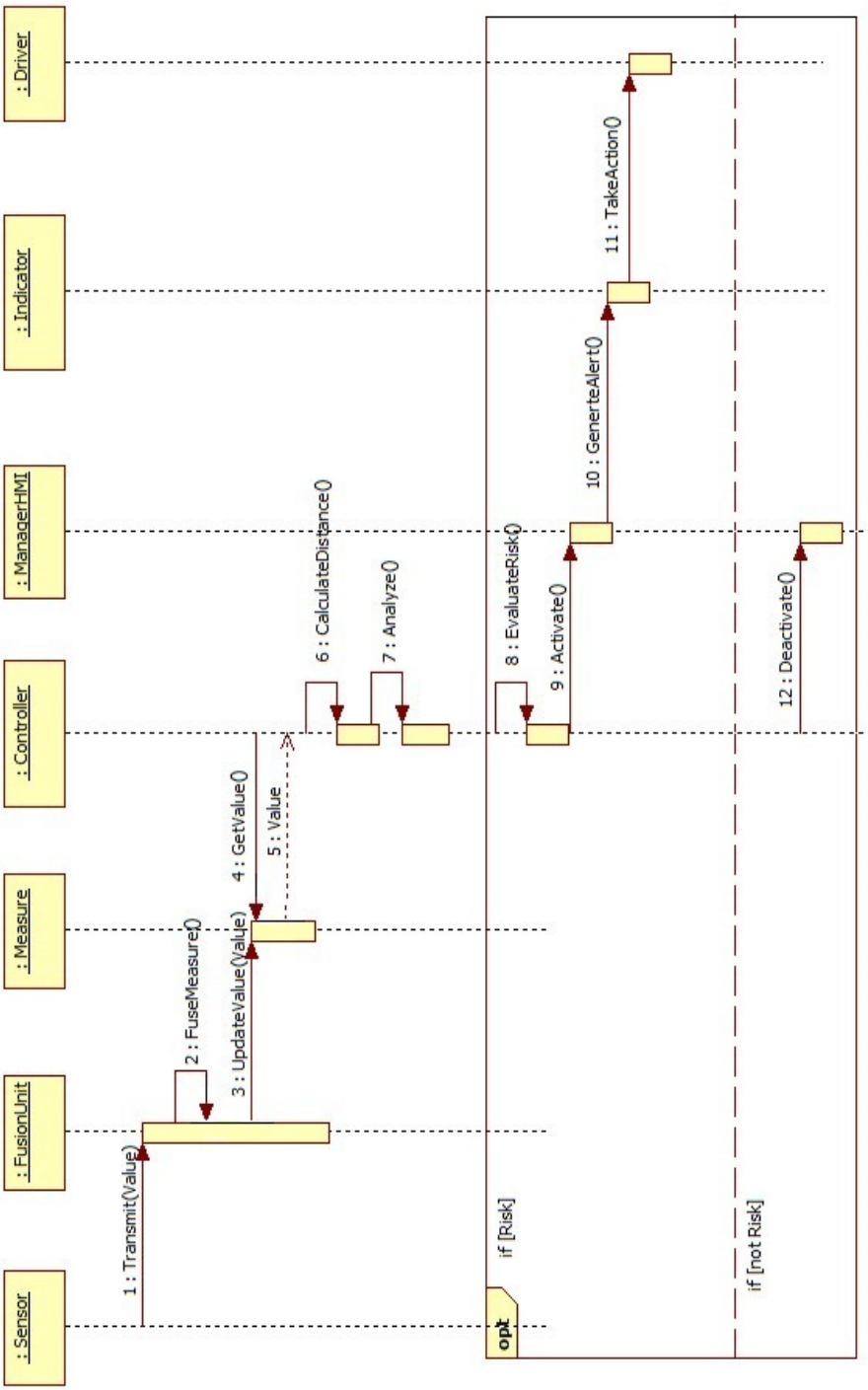


FIGURE B.26 : Modélisation du diagramme de séquence du système BLA sans réutilisation de patrons.

Le diagramme de classes de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue statique du patron *"Action"* est présenté dans la figure B.29.

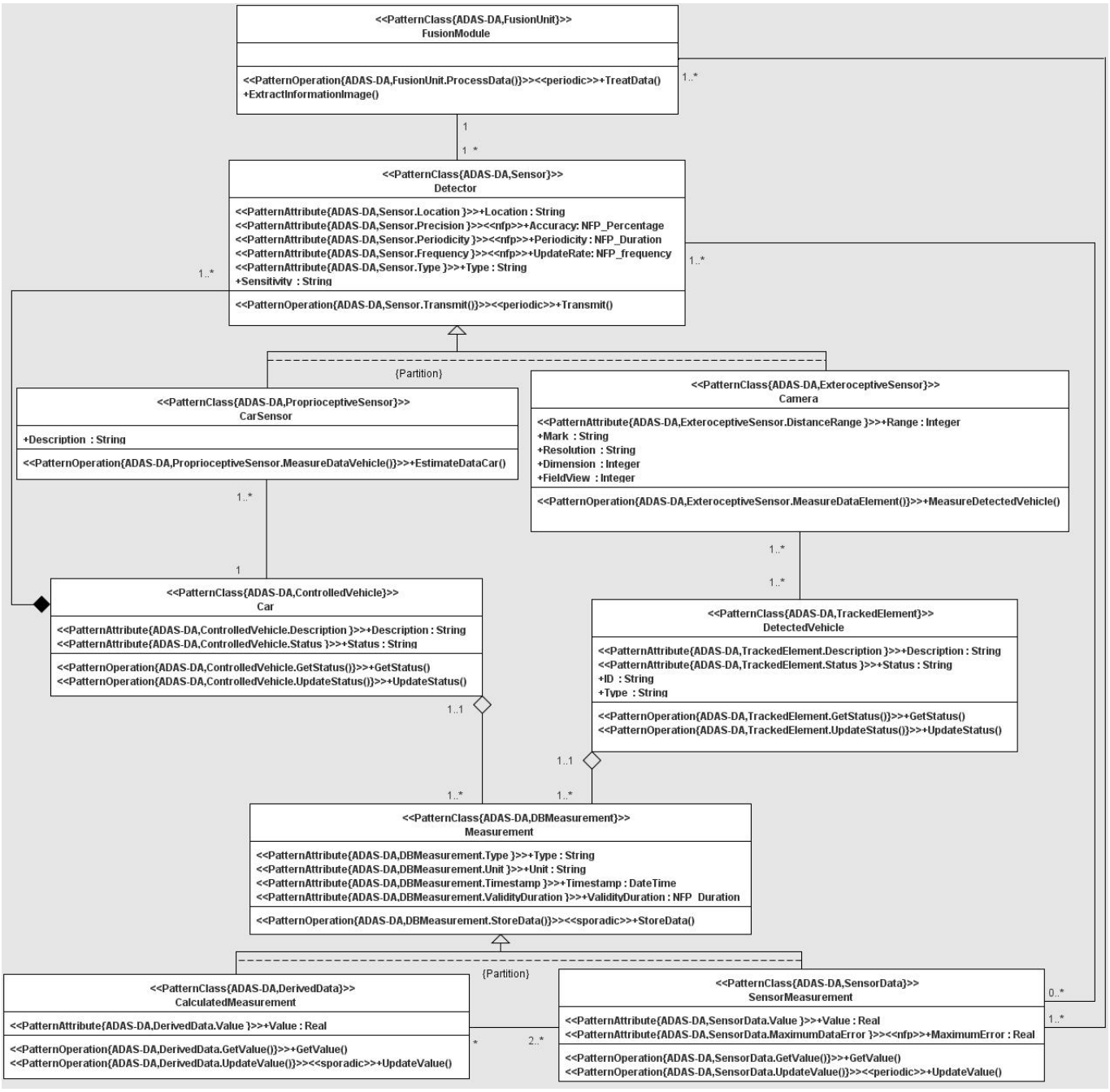


FIGURE B.27 : Modélisation du diagramme de classes du système BLA par réutilisation de la vue statique du patron *"Acquisition des données"*.

Modélisation des diagrammes de séquence

Le diagramme de séquence de la fonctionnalité d'acquisition des données modélisé par réuti-

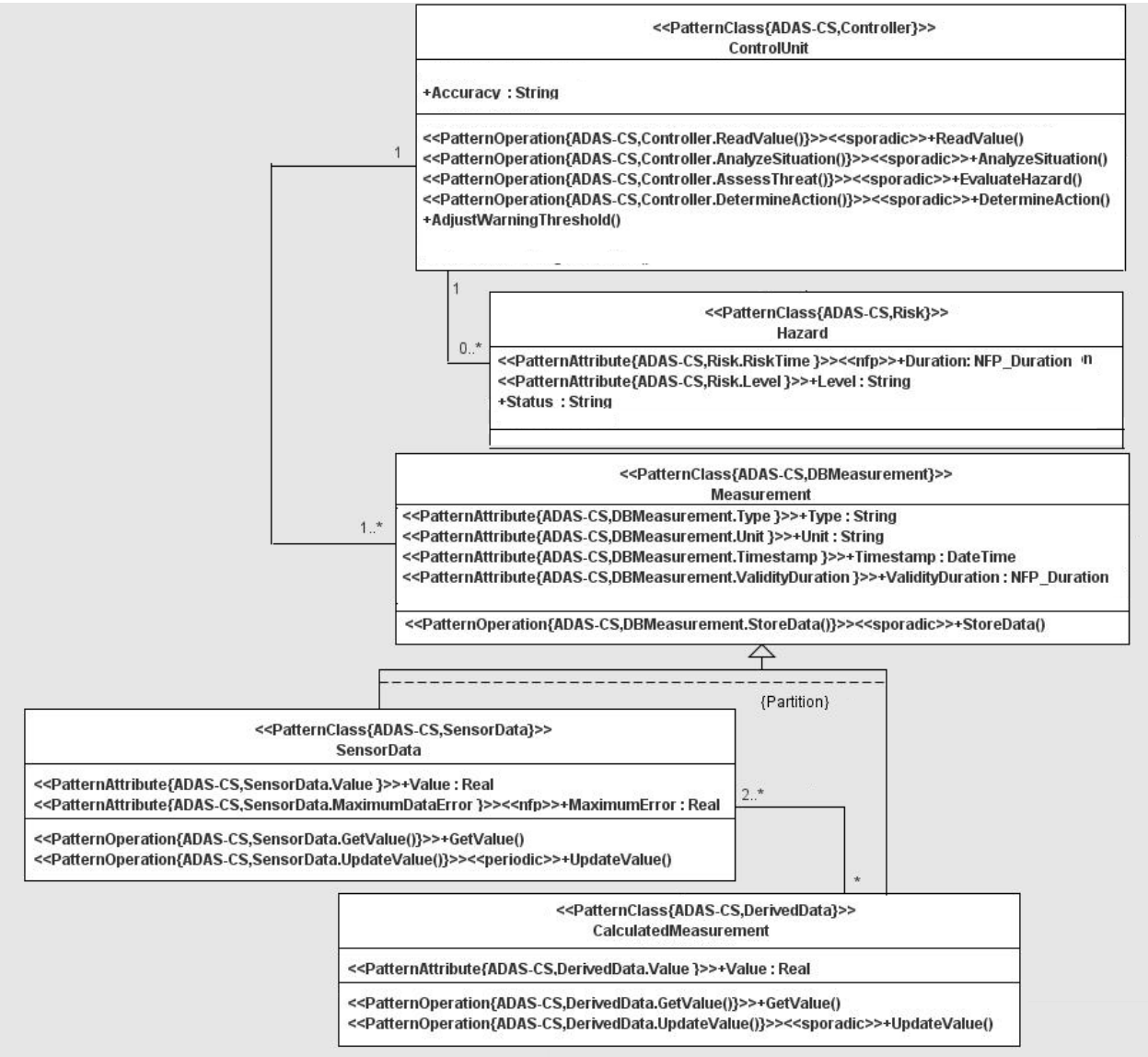


FIGURE B.28 : Modélisation du diagramme de classes du système BLA par réutilisation de la vue statique du patron *"Contrôle de données"*.

lisation de la vue dynamique du patron *"Acquisition des données"* est présenté dans la figure B.30.

Le diagramme de séquence de la fonctionnalité de contrôle et de traitement des données modélisé par réutilisation de la vue dynamique du patron *"Contrôle de données"* est présenté dans la figure B.31.

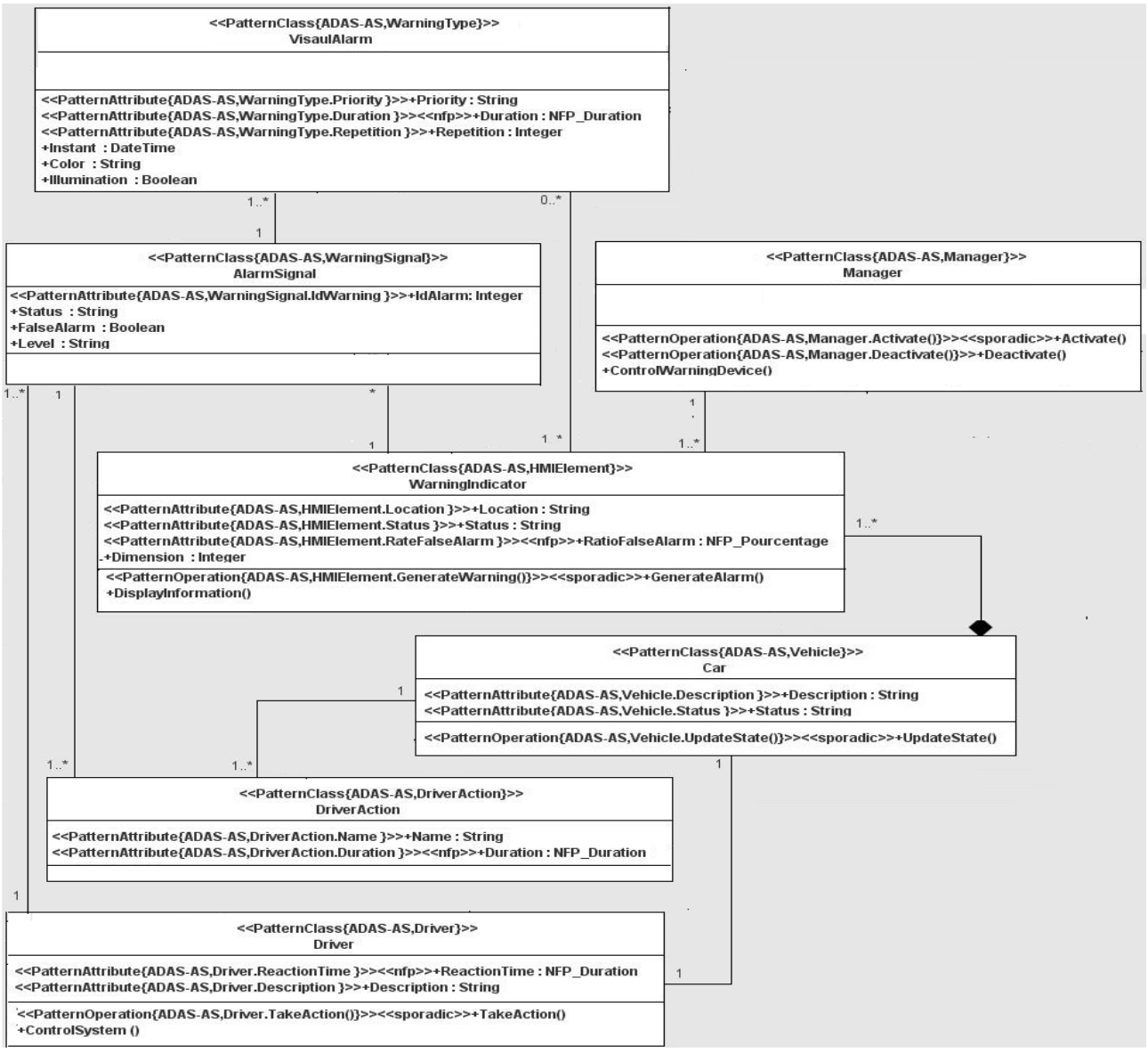


FIGURE B.29 : Modélisation du diagramme de classes du système BLA par réutilisation de la vue statique du patron *"Action"*.

Le diagramme de séquence de la fonctionnalité d'exécution des commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue dynamique du patron *"Action"* est présenté dans la figure B.32.

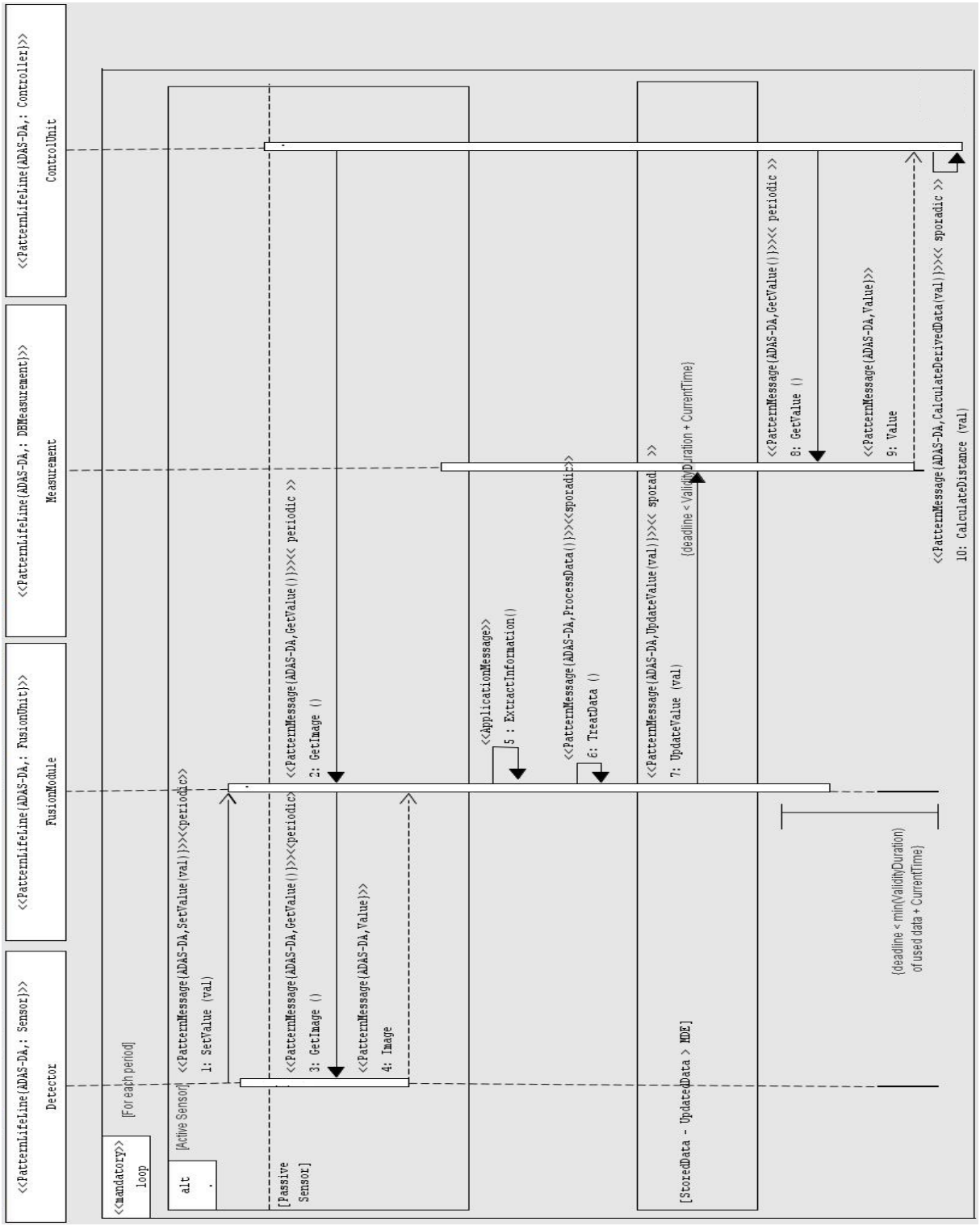


FIGURE B.30 : Modélisation du diagramme de séquence du système BLA par réutilisation de la vue dynamique du patron *"Acquisition des données"*.

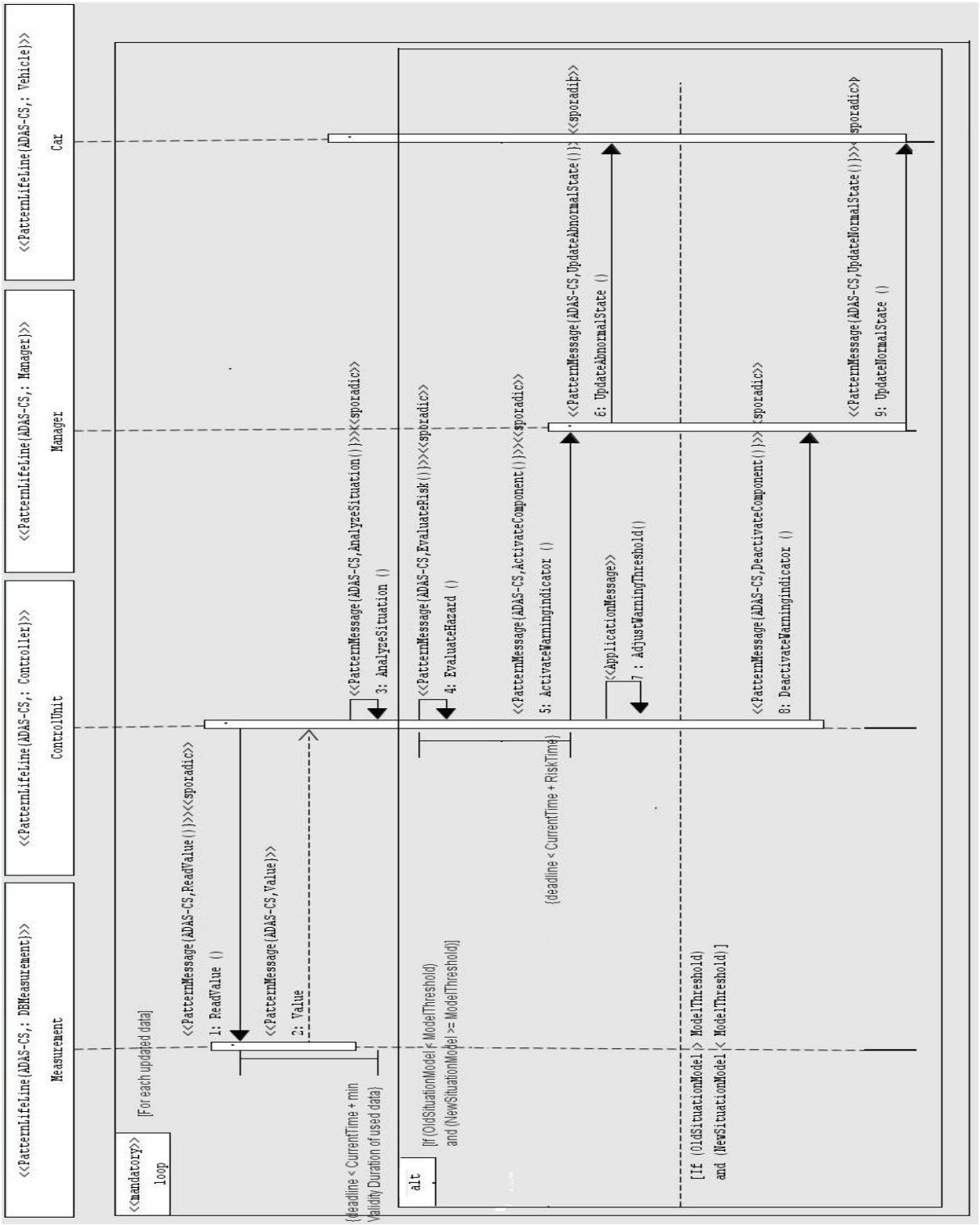


FIGURE B.31 : Modélisation du diagramme de séquence du système BLA par réutilisation de la vue dynamique du patron *"Contrôle de données"*.

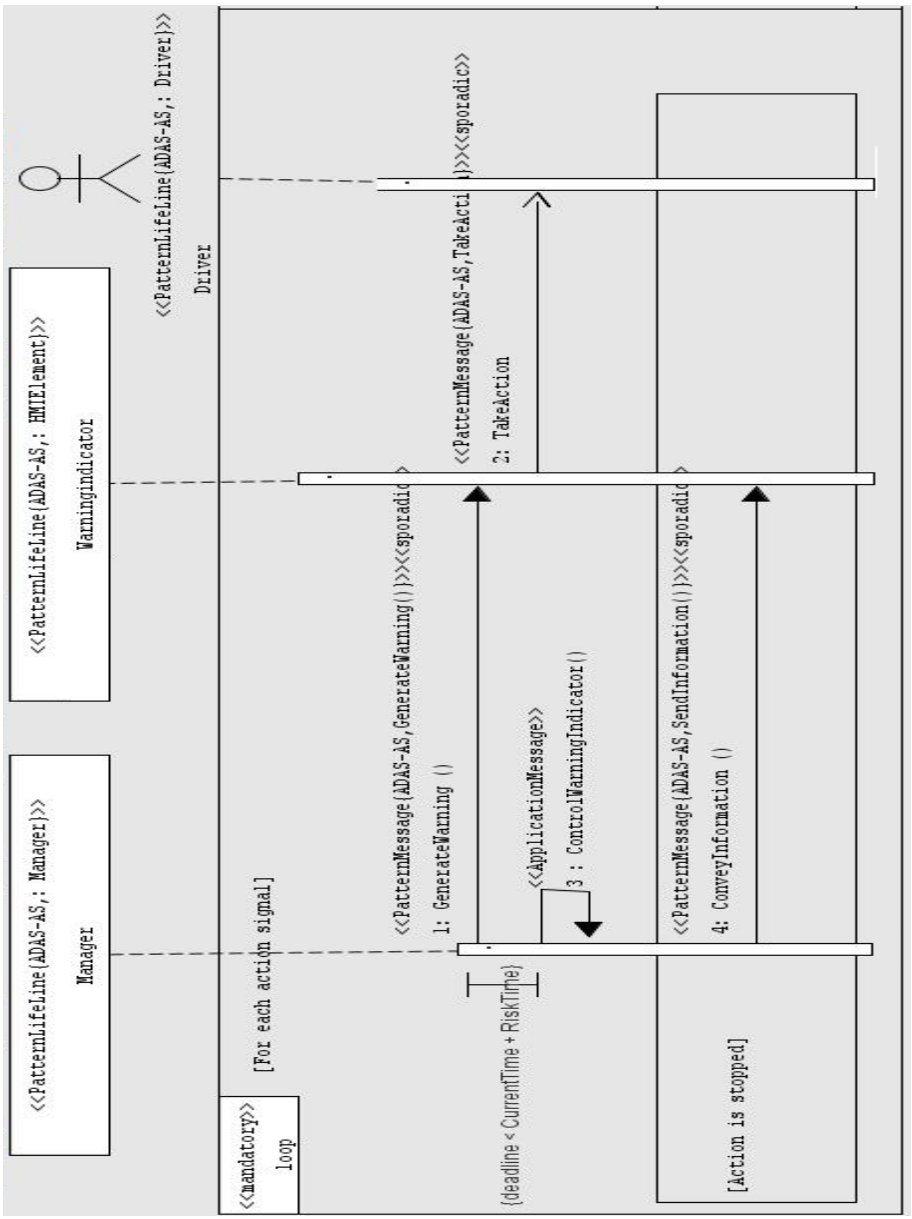


FIGURE B.32 : Modélisation du diagramme de séquence du système BLA par réutilisation de la vue dynamique du patron *"Action"*.

Étude de cas $N°5$: Système SAFELANE

Description du système

Le sous-projet SAFELANE fait partie du projet européen PReVENT. SAFELANE a pour objectif le développement d'un système qui permet d'assister le conducteur en cas de sortie de

voie involontaire. Ce système est principalement conçu pour fonctionner sur des autoroutes, des routes nationales et des routes départementales dans des conditions défavorables.

SAFELANE est composé d'un système de capteurs extéroceptifs et proprioceptifs, d'actionneurs et d'un composant de décision. Le développement de SAFELANE est basé sur la fusion des données provenant du système de vision, des cartes digitales et des capteurs actifs (*i.e.,* radar et laser) afin de donner un état précis de l'environnement.

SAFELANE propose une technologie qui dispose d'un ensemble de capteurs installés dans le véhicule :

- Une caméra de type CMOS passive située dans le rétroviseur permet d'acquérir des images pour scruter la route et de déterminer son état et ses paramètres (position des marqueurs au sol, largeur de la route, nombre de voies, etc.). Cette caméra est caractérisée par : un champ de vision de 60°, une résolution de 640x480 pixels, une plage de niveaux de gris de 100 dB et une distance de détection couvrant 5 à 50 mètres.
- Un capteur électronique actif fournit des informations sur les segments de route se trouvant devant le véhicule. Ces informations sont extraites d'une base de données cartographique.
- Des capteurs proprioceptifs actifs permettent d'observer l'état du véhicule. SAFELANE utilise deux types de capteurs proprioceptifs : (a) un capteur de vitesse permettant de mesurer la vitesse longitudinale du véhicule avec une précision de 2% par rapport à la vitesse actuelle et (b) un capteur de vitesse de lacet permettant de mesurer le déplacement latéral du véhicule (la vitesse de lacet, la vitesse latérale, l'accélération latérale et l'angle du volant) avec un taux de précision de 1-2 milliradian/s et une période de mise à jour de 10 ms.

Les données acquises (sensorielles) sont mises à jour et transmises périodiquement à travers le bus CAN à une unité de fusion, qui permet de traiter ces données et de fusionner les données redondantes afin de reconstruire un état précis de chaque élément observé (véhicule et route). Ensuite, toutes les données sont sauvegardées dans une base de données. Chaque donnée est caractérisée par un type, une durée de validité, une date de mise à jour, un taux d'erreur et une unité. Ces données sont par la suite transmises au contrôleur pour analyser

la situation, détecter la voie et estimer la trajectoire la plus probable. Le contrôleur permet de déterminer l'existence de situations à risque, d'évaluer le risque et de décider de l'action à mener pour chaque situation. Chaque risque est caractérisé par un niveau (faible, moyen ou grave) et un temps dans lequel il se produit. Le contrôleur calcule la valeur requise pour la modification de la direction ou le freinage. Chaque donnée calculée est caractérisée par un type, une durée de validité, une date de mise à jour et une unité.

Lorsqu'une situation à risque est détectée, le contrôleur envoie les actions à un gestionnaire qui est responsable de gérer ces actions et d'activer l'actionneur et/ou le composant de l'interface Homme-Machine. L'actionneur automatique (composant de direction assistée) permet d'exécuter des commandes sur les composants du véhicule (le temps de réaction doit permettre d'éviter le risque). Les dispositifs de l'interface Homme-Machine, situés au niveau du tableau de bord, permettent de générer des alertes visuelles et sonores pour avertir le conducteur, et sont caractérisés par un taux de fausses alertes. Chaque type d'alerte est caractérisée par une durée, une priorité et un taux de répétition.

Les alertes visuelles peuvent être des symboles et du texte avec différentes couleurs et tailles. Les alertes sonores sont en fonction du changement de voie (droite ou gauche). Un son est produit dans le haut parleur de gauche ou de droite selon la direction à prendre. Ces alertes sont transmises au conducteur afin qu'il puisse intervenir à temps. Une fois que l'action est exécutée, l'état du véhicule est rétabli.

Modélisation du système sans réutilisation de patrons

La figure B.33 et la figure B.34 présentent respectivement le diagramme de classes et le diagramme de séquence du système SAFELANE modélisés sans réutilisation des patrons proposés.

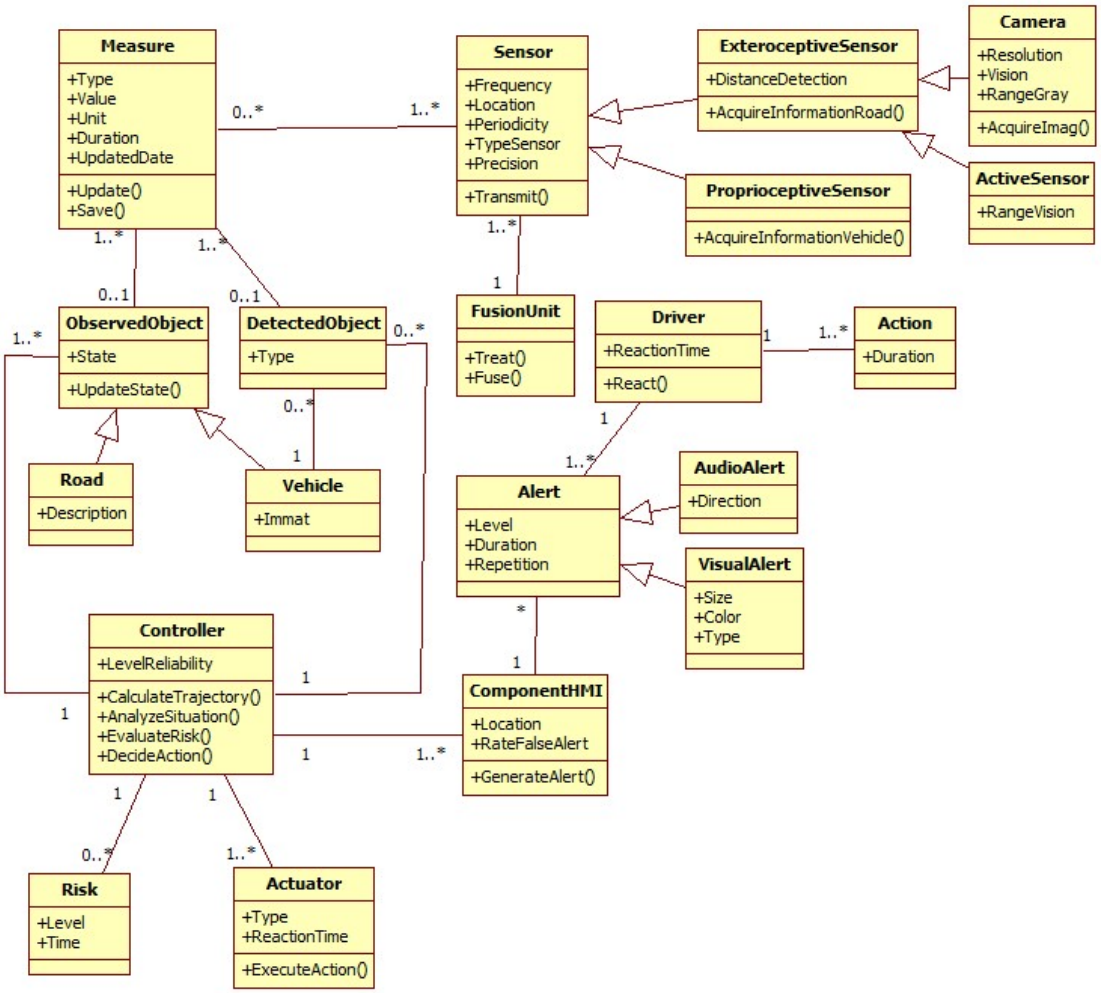


FIGURE B.33 : Modélisation du diagramme de classes du système SAFELANE sans réutilisation de patrons.

Modélisation du système avec réutilisation de patrons

Modélisation des diagrammes de classes

Le diagramme de classes de la fonctionnalité d'acquisition des données modélisé par réutilisation de la vue statique du patron *"Acquisition des données"* est présenté dans la figure B.35.

Le diagramme de classes de la fonctionnalité de contrôle et de traitement des données modélisé par réutilisation de la vue statique du patron *"Contrôle de données"* est présenté dans la figure B.36.

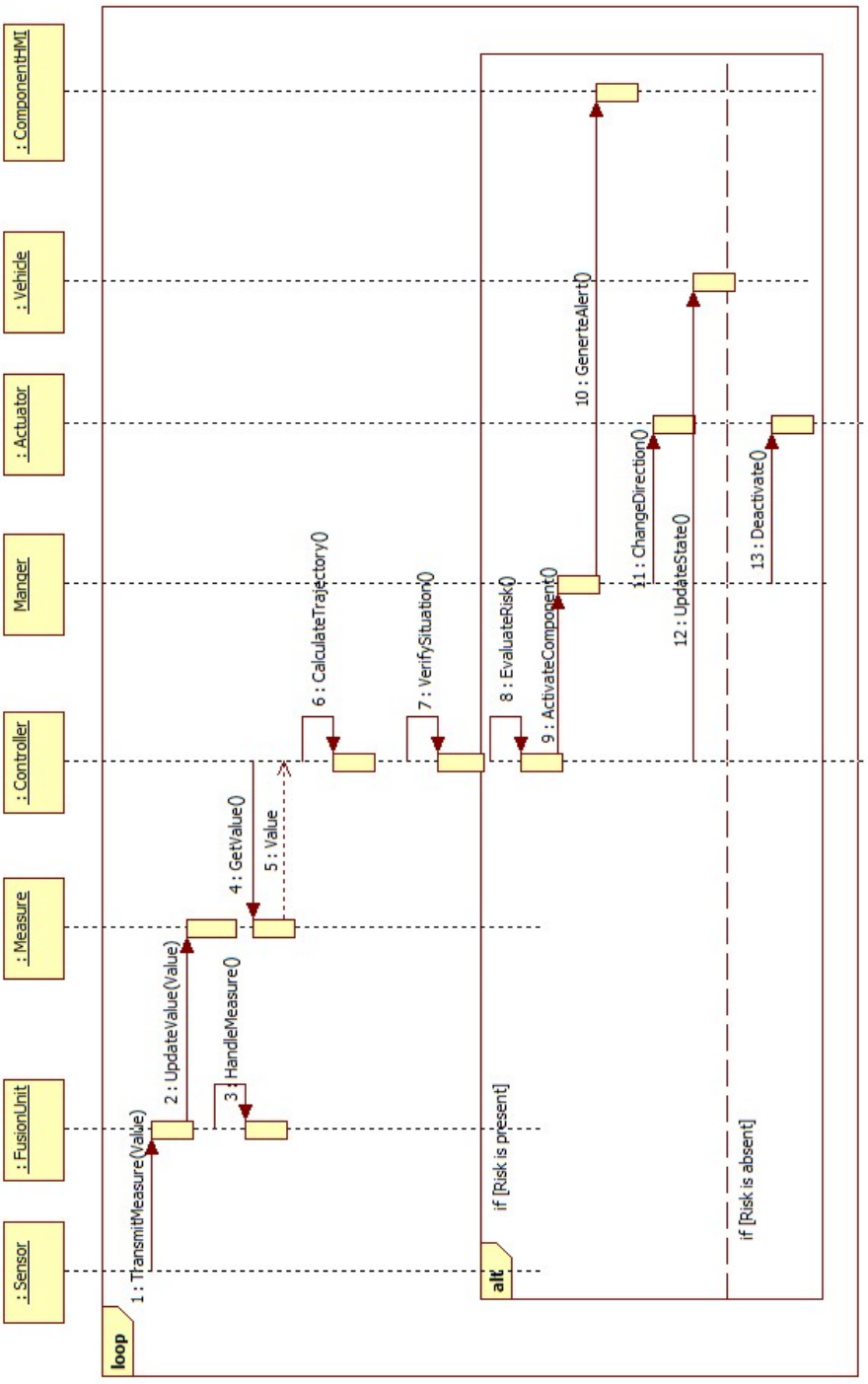


FIGURE B.34 : Modélisation du diagramme de séquence du système SAFELANE sans réutilisation des patron.

Le diagramme de classes de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue statique du patron *"Action"* est présenté dans la figure B.37.

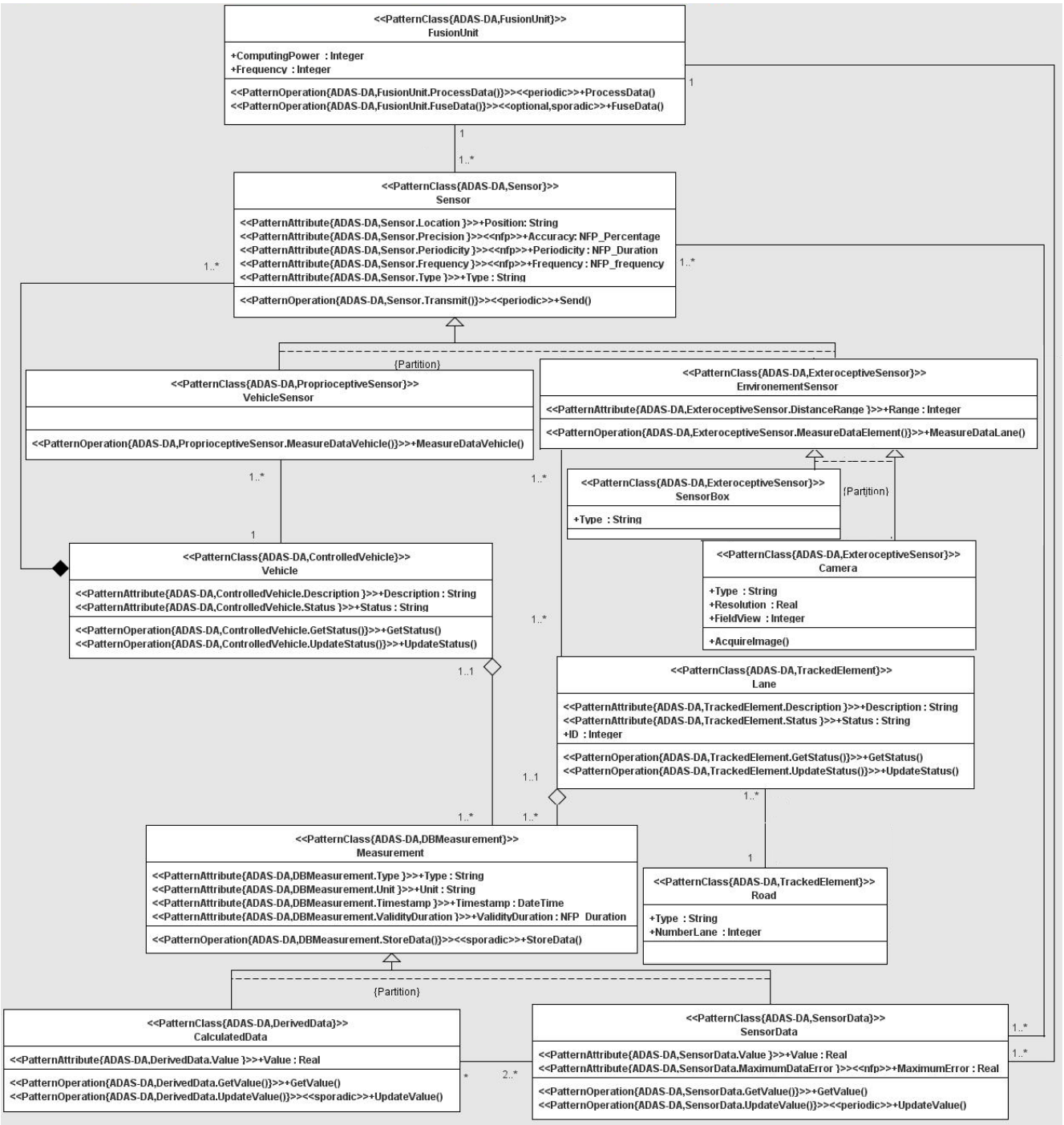


FIGURE B.35 : Modélisation du diagramme de classes du système SAFELANE par réutilisation de la vue statique du patron *"Acquisition des données"*.

Modélisation des diagrammes de séquence

Le diagramme de séquence de la fonctionnalité d'acquisition des données modélisé par réuti-

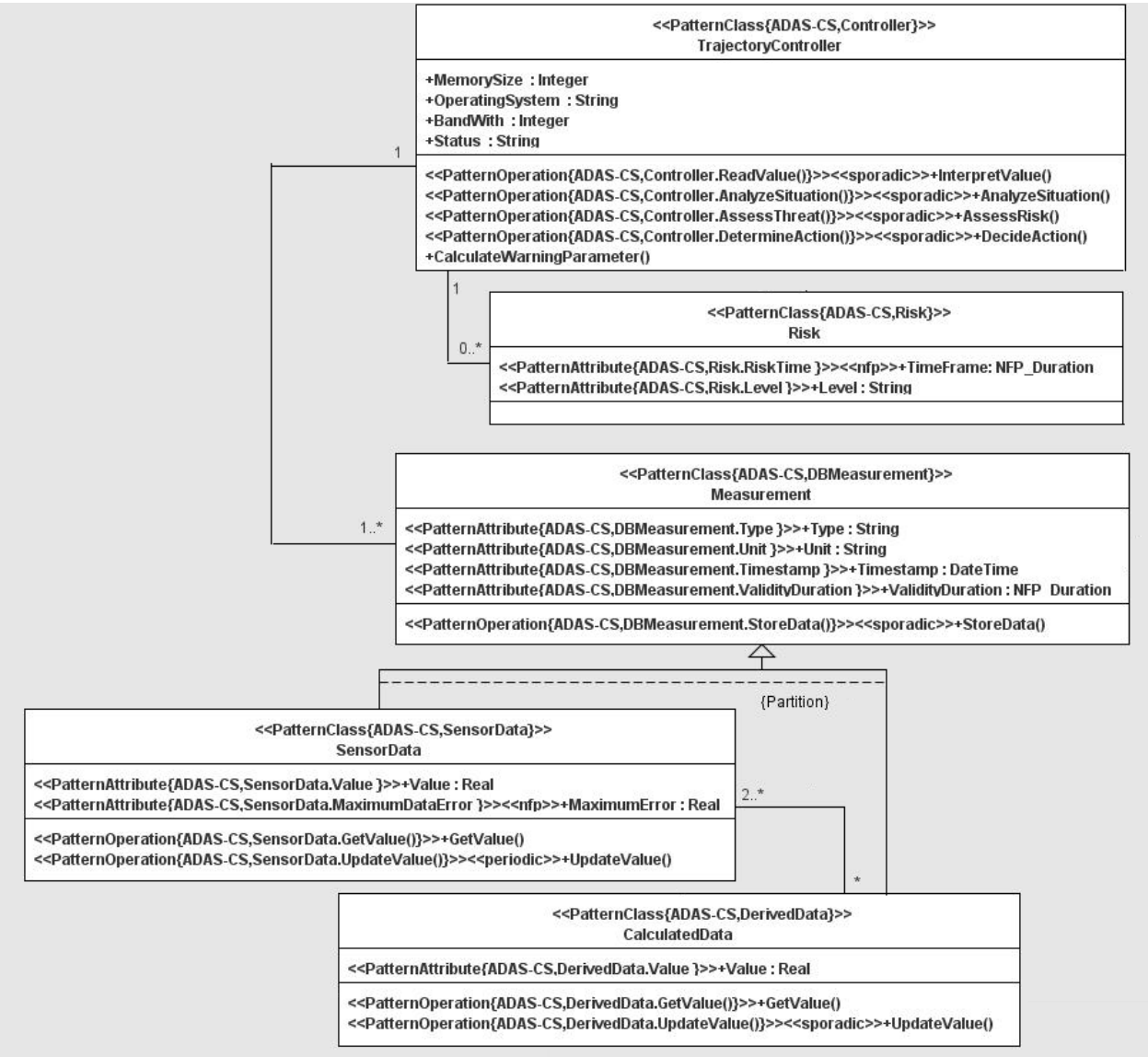


FIGURE B.36 : Modélisation du diagramme de classes du système SAFELANE par réutilisation de la vue statique du patron *"Contrôle de données"*.

lisation de la vue dynamique du patron *"Acquisition des données"* est présenté dans la figure B.38.

Le diagramme de séquence de la fonctionnalité de contrôle et de traitement des données modélisé par réutilisation de la vue dynamique du patron *"Contrôle de données"* est présenté dans la figure B.39.

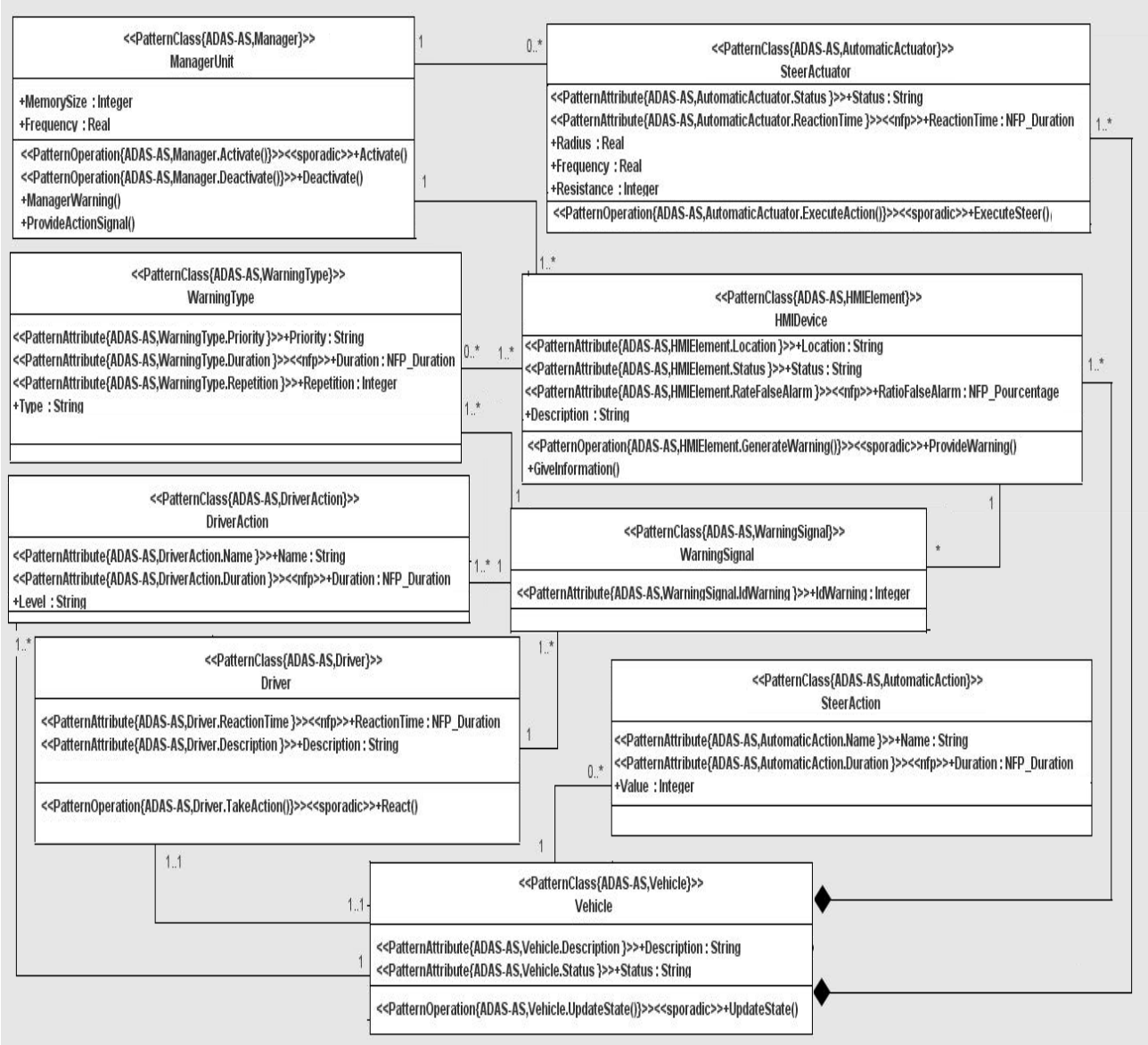


FIGURE B.37 : Modélisation du diagramme de classes du système SAFELANE par réutilisation de la vue statique du patron *"Action"*.

Le diagramme de séquence de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue dynamique du patron *"Action"* est présenté dans la figure B.40.

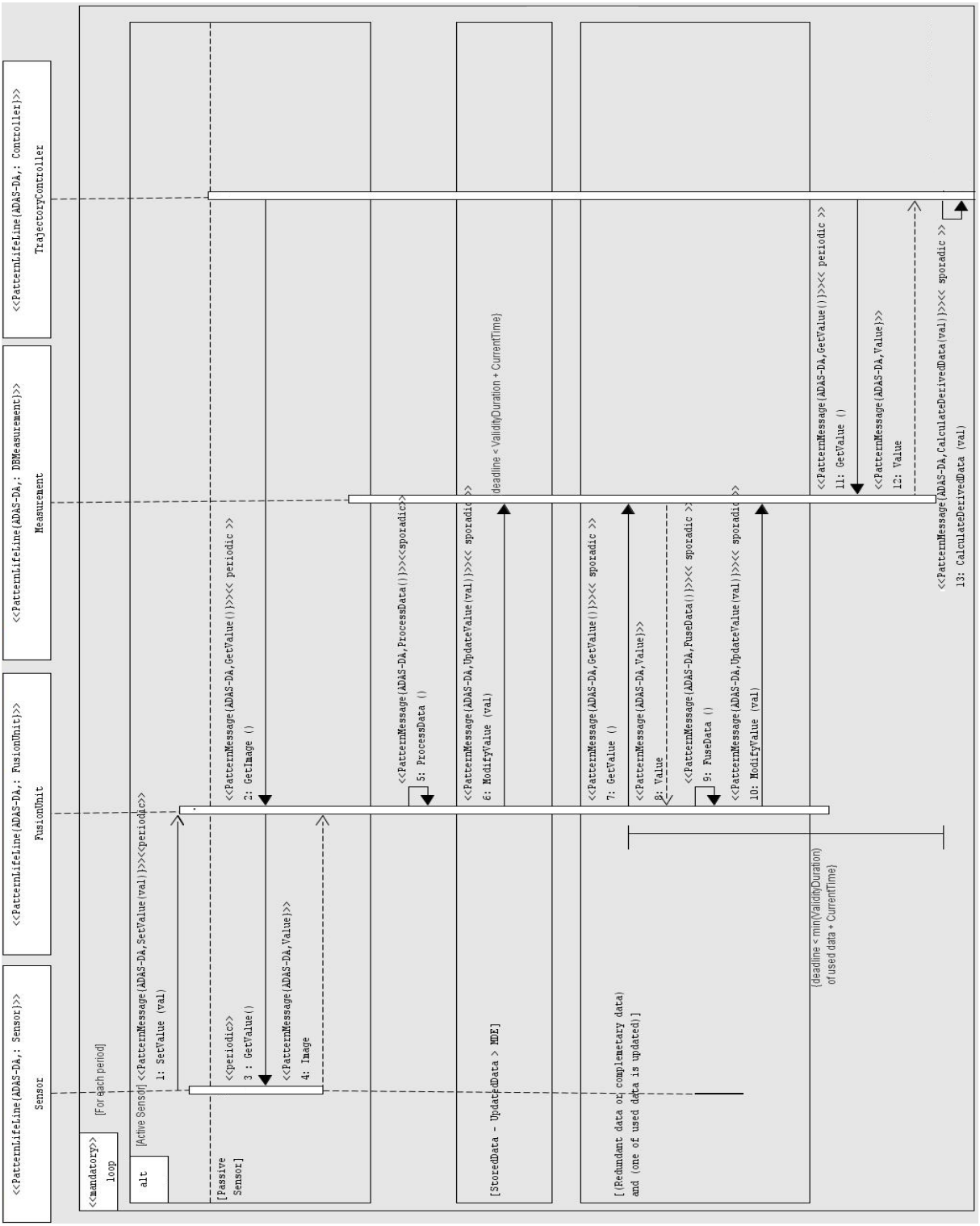


FIGURE B.38 : Modélisation du diagramme de séquence du système SAFELANE par réutilisation de la vue dynamique du patron *"Acquisition des données"*.

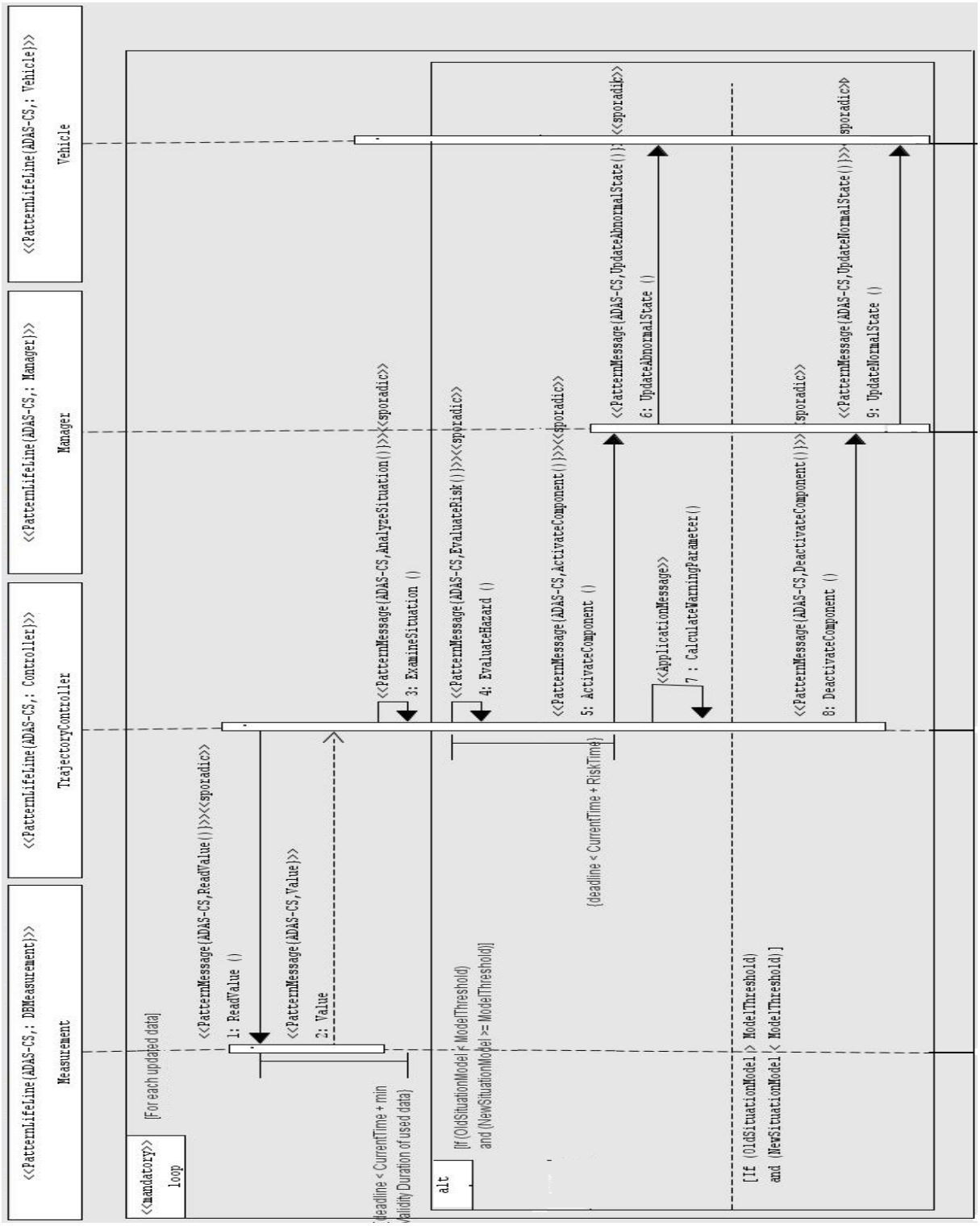


FIGURE B.39 : Modélisation du diagramme de séquence du système SAFELANE par réutilisation de la vue dynamique du patron *"Contrôle de données"*.

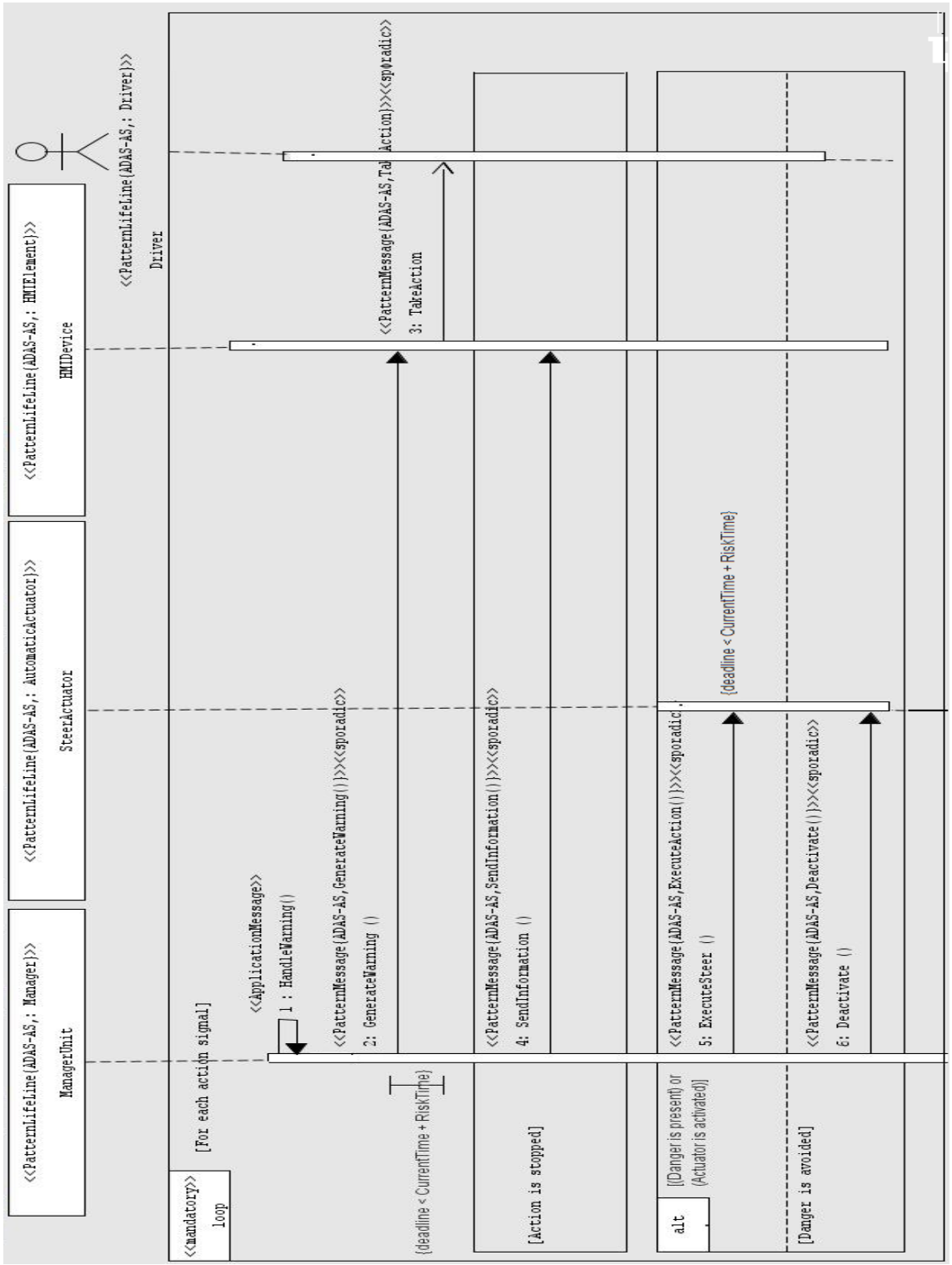


FIGURE B.40 : Modélisation du diagramme de séquence du système SAFELANE par réutilisation de la vue dynamique du patron *"Action"*.

Étude de cas $N°6$: Système RDCW

Description du système

Le système d'alerte pour prévenir les sorties de route RDCWS (Road Departure Crash Warning System) a été développé dans le cadre du projet RDCW FOT (Field Operation test). Ce système fournit au conducteur une assistance longitudinale en délivrant un avertissement sur la vitesse maximale autorisée à l'approche d'un virage. De plus, il fournit une assistance latérale en avertissant le conducteur si un déplacement latéral excessif survient.

RDCWS utilise un ensemble de capteurs embarqués dans le véhicule :

- Une caméra CCD située sur le pare-brise, permet de prendre des images sur la route et de déterminer son état et ses paramètres (largeur de voie, angle de la courbure, position des marqueurs au sol, etc.) avec une bonne précision. Cette caméra offre une distance de détection jusqu'à 70 mètres. Elle permet de mesurer les données toutes les 40 ms.
- Un capteur de positionnement absolu permet de déterminer avec une bonne précision et une fréquence de 1 Hz la géométrie de la route ainsi que la position du véhicule.
- Deux radars à l'avant du véhicule (longitudinaux) permettent de détecter des objets à une distance d'environ 30 m à 60 m, et offrent une fréquence de transmission de 77 Ghz.
- Deux radars latéraux sur les cotés (droite et à gauche) permettent de mesurer la distance latérale entre le véhicule et des objets situés à proximité de celui-ci. Ils détectent les objets avec une précision de +/- 20 cm. Ils disposent d'une fréquence de transmission de 24 Ghz.
- Des capteurs proprioceptifs permettent d'observer l'état du véhicule (vitesse de lacet, accélération, position, etc.) avec une bonne précision.

Ces capteurs transmettent périodiquement les mesures acquises vers une unité de fusion à travers un bus CAN. Cette unité permet de traiter les mesures et de fusionner les données redondantes et complémentaires pour avoir un état précis de toute la situation de conduite.

Ensuite, ces mesures sont sauvegardées dans une base lorsque la différence entre la valeur acquise et la valeur stockée est supérieure à une erreur maximale associée à la mesure. Chaque mesure est caractérisée par un type, une durée, un instant, une valeur et une unité.

Un module de contrôle (Situation Awareness Module SAM) estime la manœuvre actuelle en se basant sur les mesures acquises pour analyser la situation. De plus, ce module utilise ces mesures pour régler les seuils d'avertissement, afin d'éviter les fausses alertes. Lorsque SAM détecte (i) une sortie involontaire de voie sans activation du clignotant, (ii) un risque de collision avec un obstacle et/ou (iii) un passage par un virage avec une vitesse excessive, il évalue le degré du risque ; c'est-à-dire si le risque est faible, moyen ou critique et décide de l'action à mener pour cette situation précédant le risque. Le module de contrôle est caractérisé par une grande capacité de calcul et une haute précision.

Le module de contrôle envoie les actions à un processeur d'alertes pour activer le composant de l'interface Homme-Machine approprié. Il utilise une mémoire de 256 Mo et un système d'exploitation QNX. En effet, ces composants permettent de générer (i) des alertes visuelles (représentées sous forme d'icônes avec différentes couleurs) affichées dans l'écran LCD du tableau de bord, (ii) des alertes sonores (générées soit par le haut parleur gauche, soit le droit selon la direction de sortie de voie, soit sous forme d'un message vocal) et/ou (iii) des alertes haptiques (des vibrations du siège du conducteur ou une légère torsion sur le volant), ainsi que par l'affichage de messages d'information (*e.g.,* état du système). Chaque type d'alerte est défini par un degré, un taux de répétition, une priorité, un instant et une durée pour réduire le nombre de fausses alertes. Ces alertes sont transmises au conducteur afin qu'il prenne des actions correctives à temps (le temps de réaction du conducteur doit être inférieur à 2 s). Suite aux actions prises par le conducteur, l'état du véhicule sera corrigé afin d'éviter la collision.

Modélisation du système sans réutilisation de patrons

La figure B.41 et la figure B.42 présentent respectivement le diagramme de classes et le diagramme de séquence du système RDCW modélisés sans réutilisation des patrons proposés.

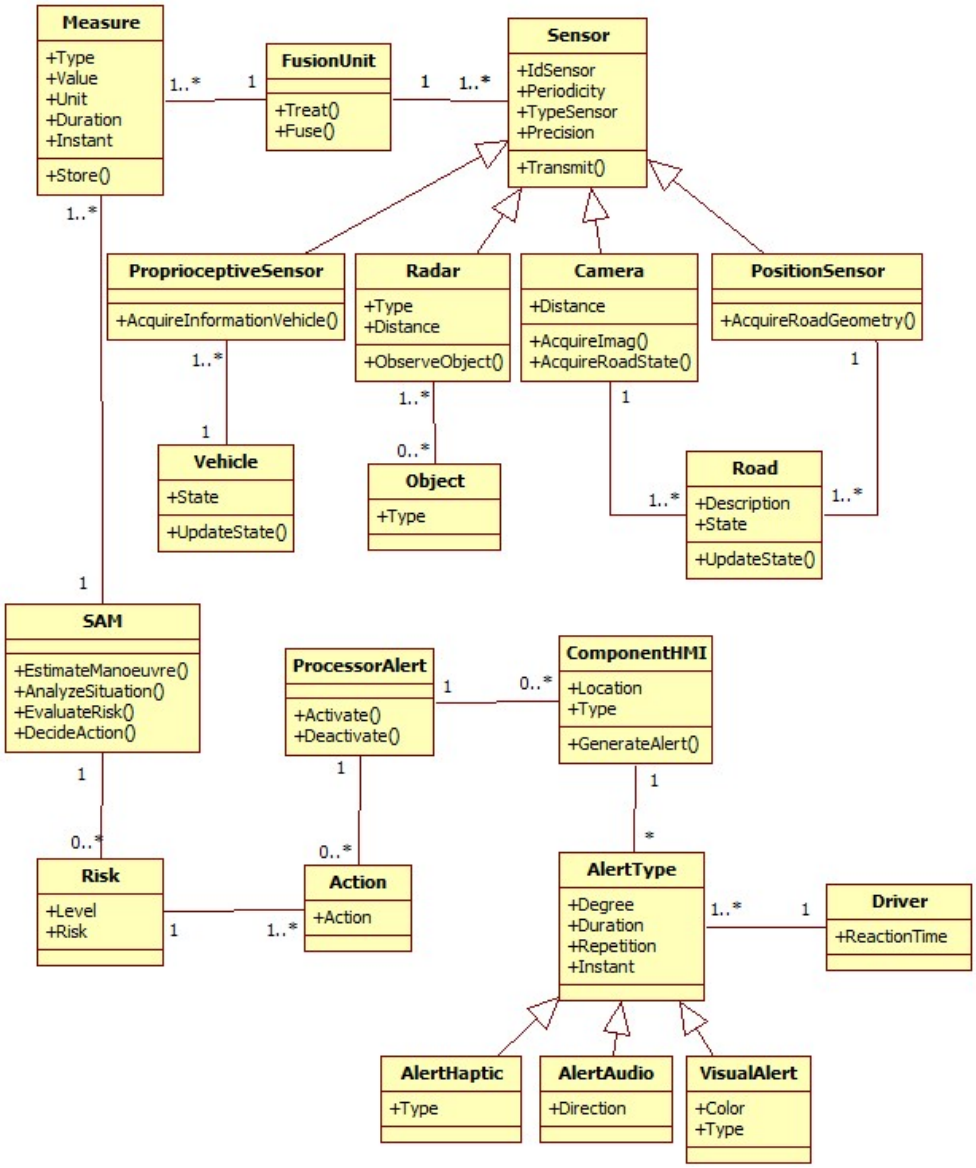


FIGURE B.41 : Modélisation du diagramme de classes du système RDCW sans réutilisation de patrons.

Modélisation du système avec réutilisation de patrons

Modélisation des diagrammes de classes

Le diagramme de classes de la fonctionnalité d'acquisition des données modélisé par réutilisation de la vue statique du patron *"Acquisition des données"* est présenté dans la figure B.43.

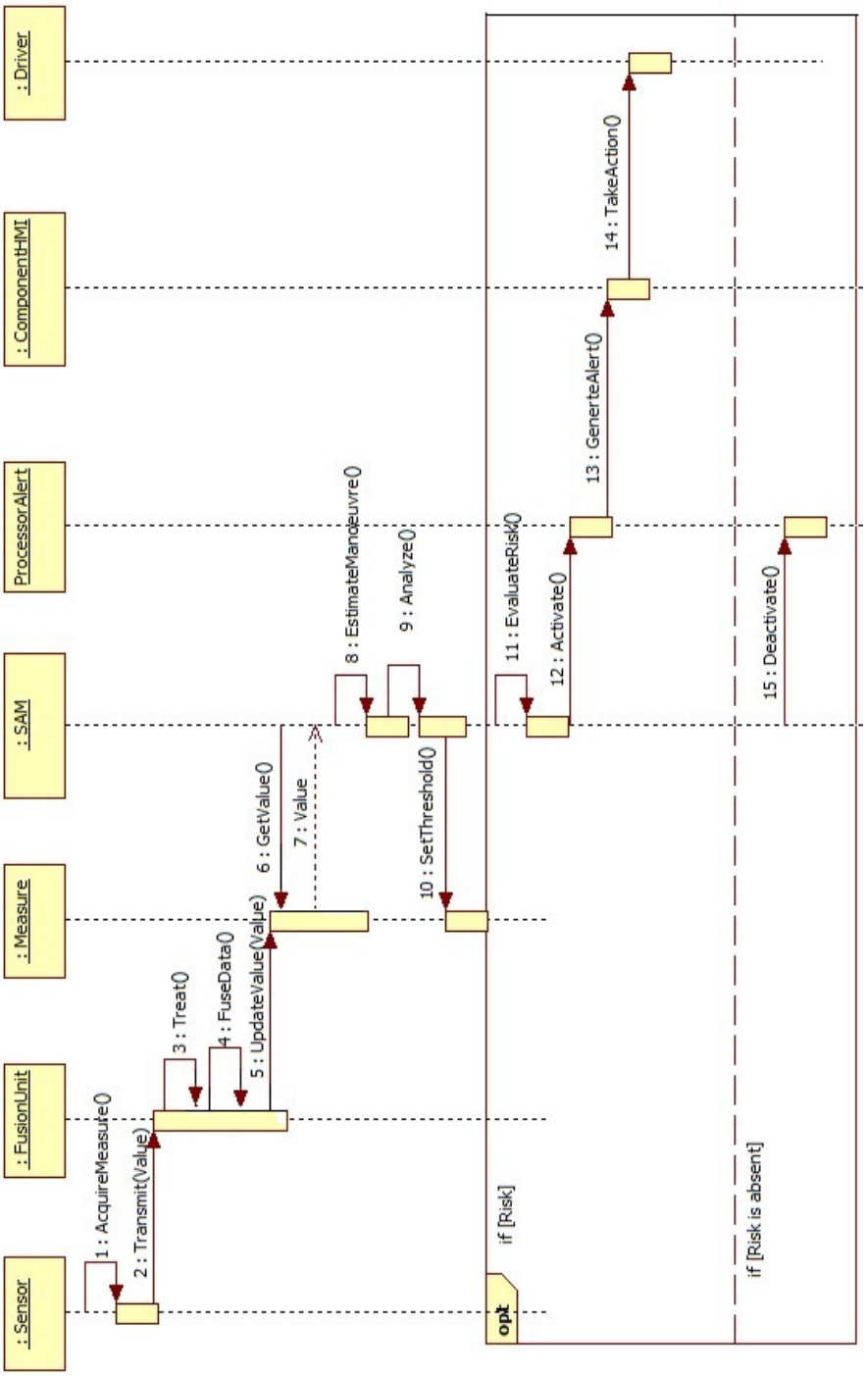


FIGURE B.42 : Modélisation du diagramme de séquence du système RDCW sans réutilisation de patrons.

Le diagramme de classes de la fonctionnalité de contrôle et de traitement des données modélisé par réutilisation de la vue statique du patron *"Contrôle de données"* est présenté dans la figure B.44.

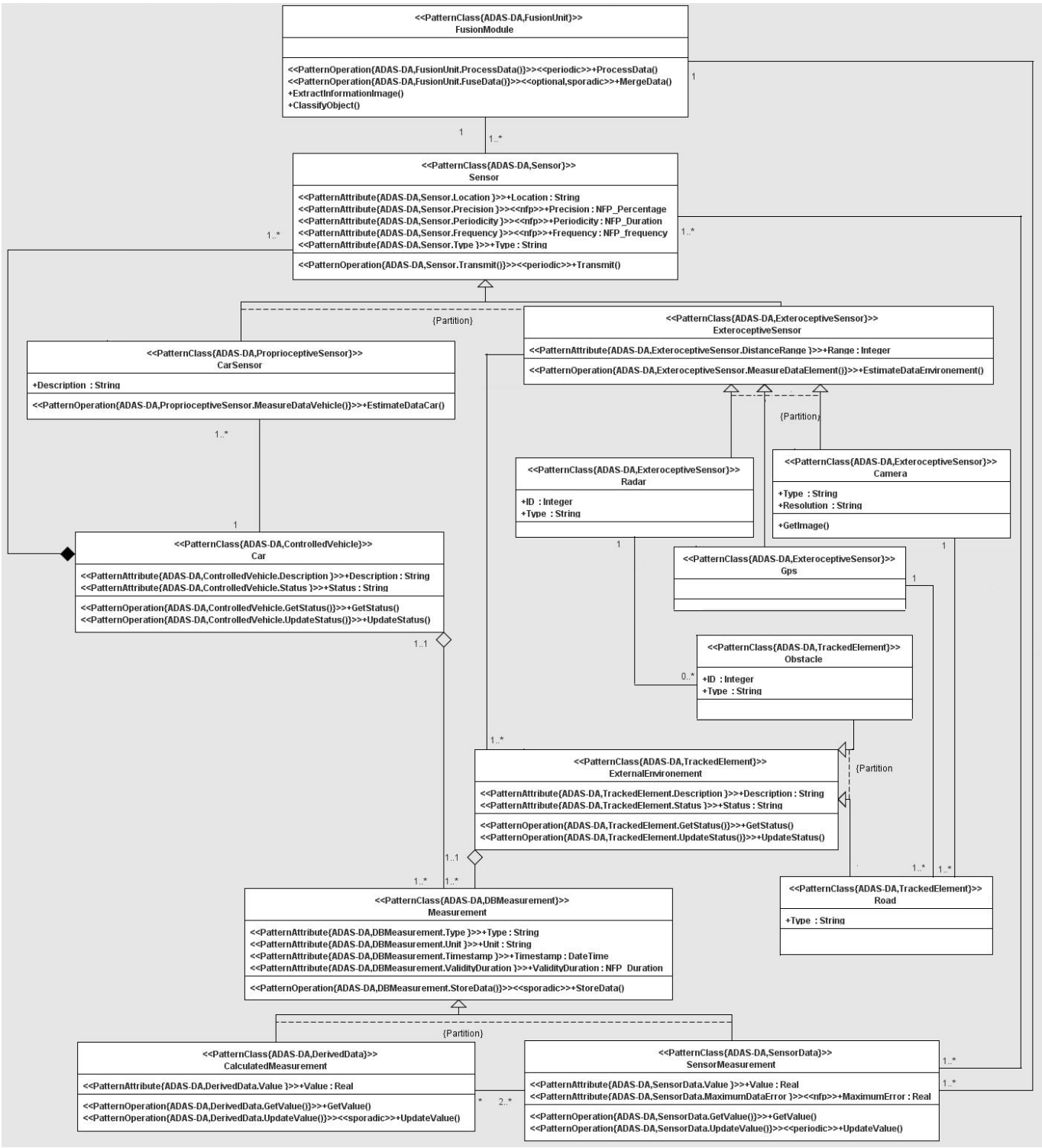


FIGURE B.43 : Modélisation du diagrammes de classes du système RDCW par réutilisation de la vue statique du patron *"Acquisition des données"*.

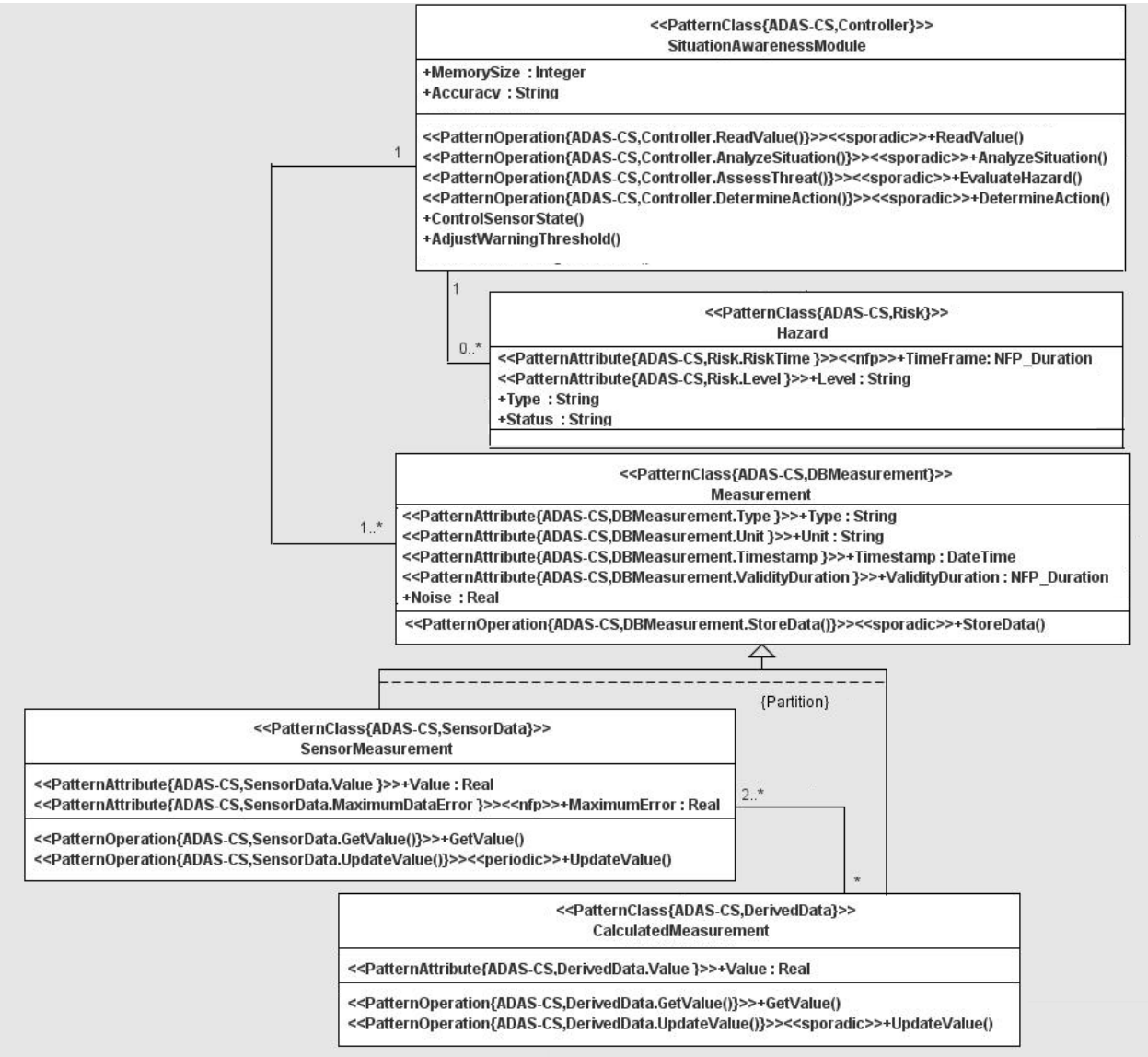


FIGURE B.44 : Modélisation du diagramme de classes du système RDCW par réutilisation de la vue statique du patron *"Contrôle de données"*.

Le diagramme de classes de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue statique du patron *"Action"* est présenté dans la figure B.45.

Modélisation des diagrammes de séquence

Le diagramme de séquence de la fonctionnalité d'acquisition des données modélisé par réuti-

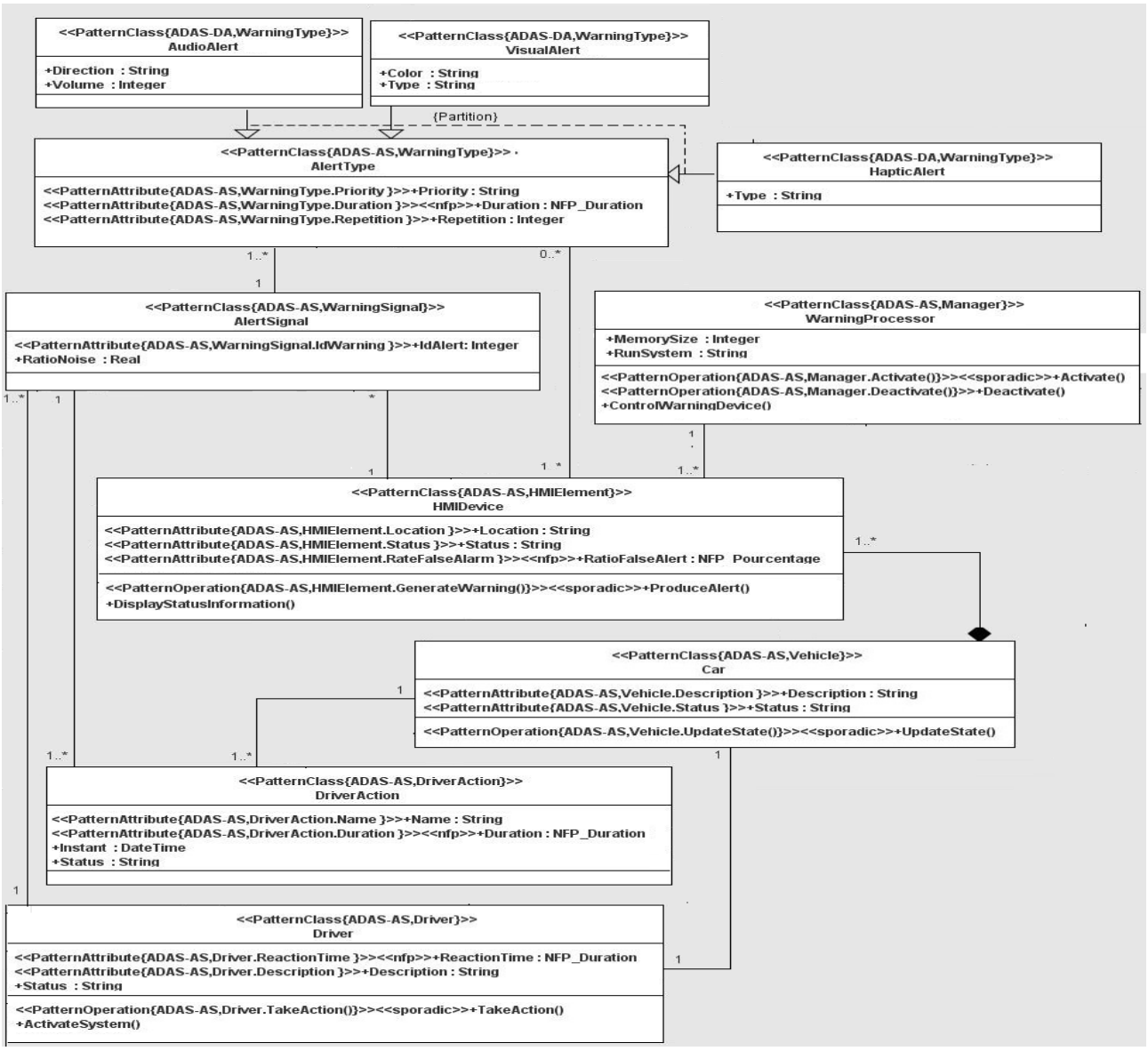


FIGURE B.45 : Modélisation du diagramme de classes du système RDCW par réutilisation de la vue statique du patron *"Action"*.

lisation de la vue dynamique du patron *"Acquisition des données"* est présenté dans la figure B.46.

Le diagramme de séquence de la fonctionnalité de contrôle et de traitement des données modélisé par réutilisation de la vue dynamique du patron *"Contrôle de données"* est présenté dans la figure B.47.

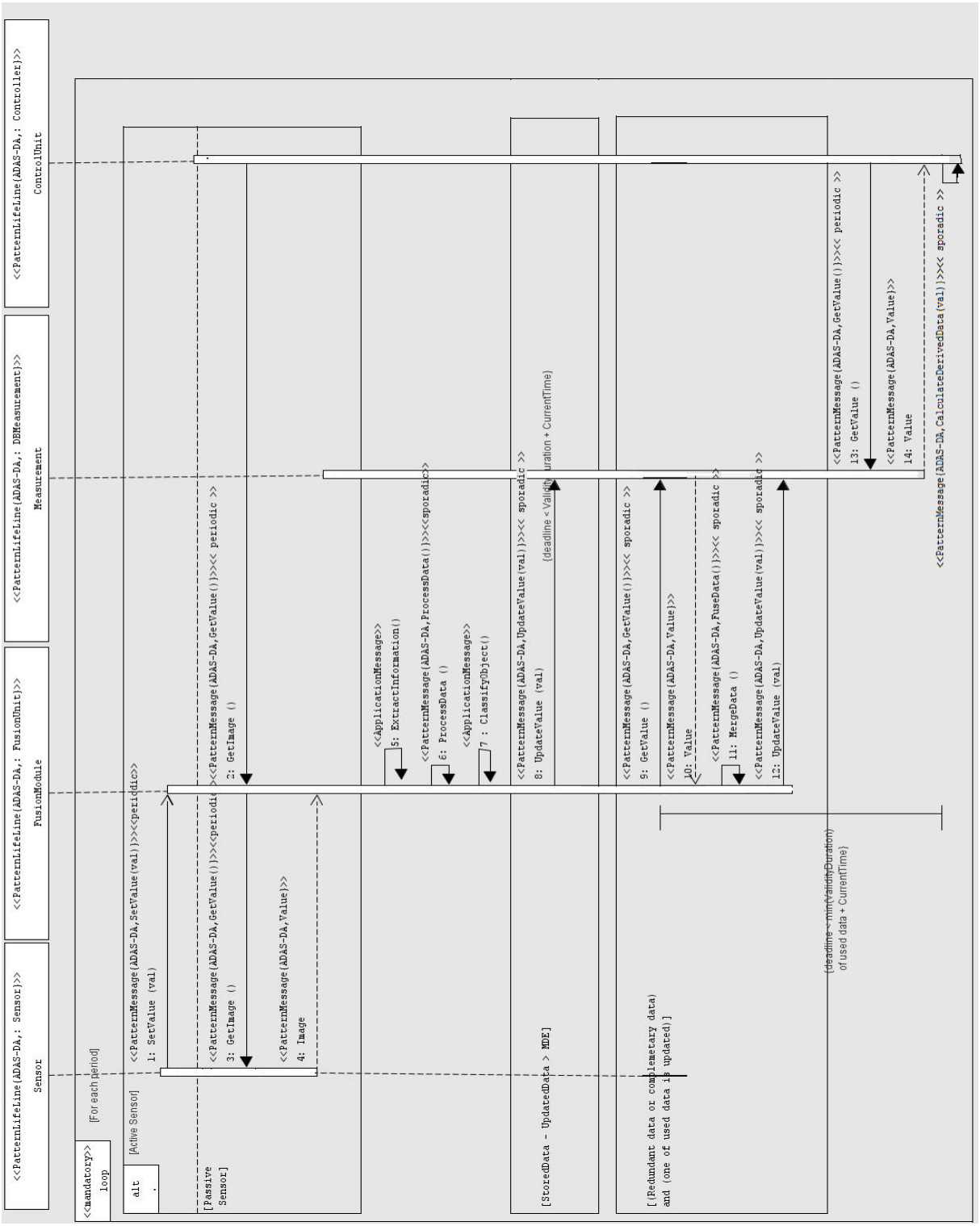


FIGURE B.46 : Modélisation du diagramme de séquence du système RDCW par réutilisation de la vue dynamique du patron *"Acquisition des données"*.

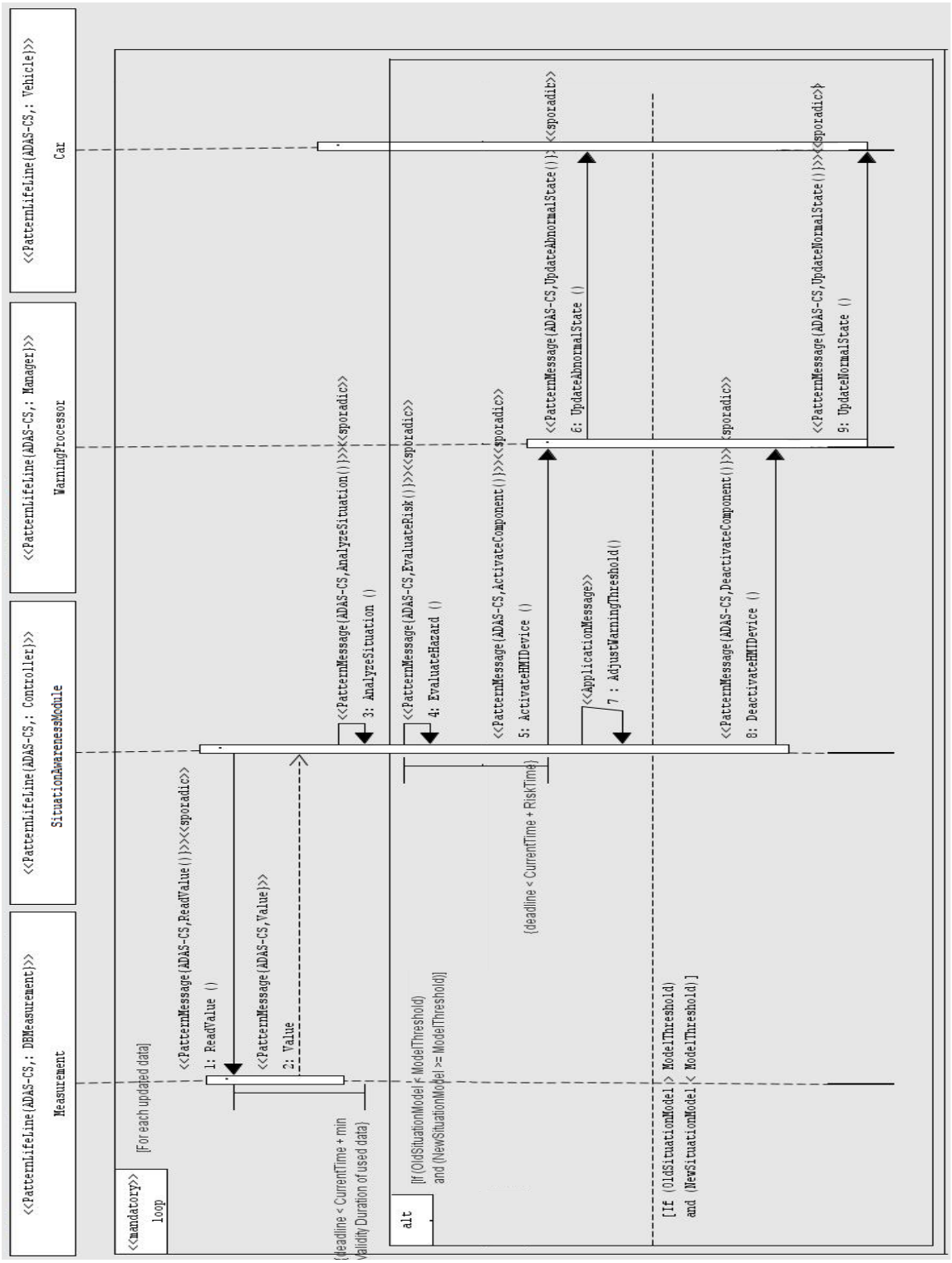


FIGURE B.47 : Modélisation du diagramme de séquence du système RDCW par réutilisation de la vue dynamique du patron *"Contrôle de données"*.

Le diagramme de séquence de la fonctionnalité d'exécution des ordres de commandes et/ou de déclenchement des alertes modélisé par réutilisation de la vue dynamique du patron *"Action"* est présenté dans la figure B.48.

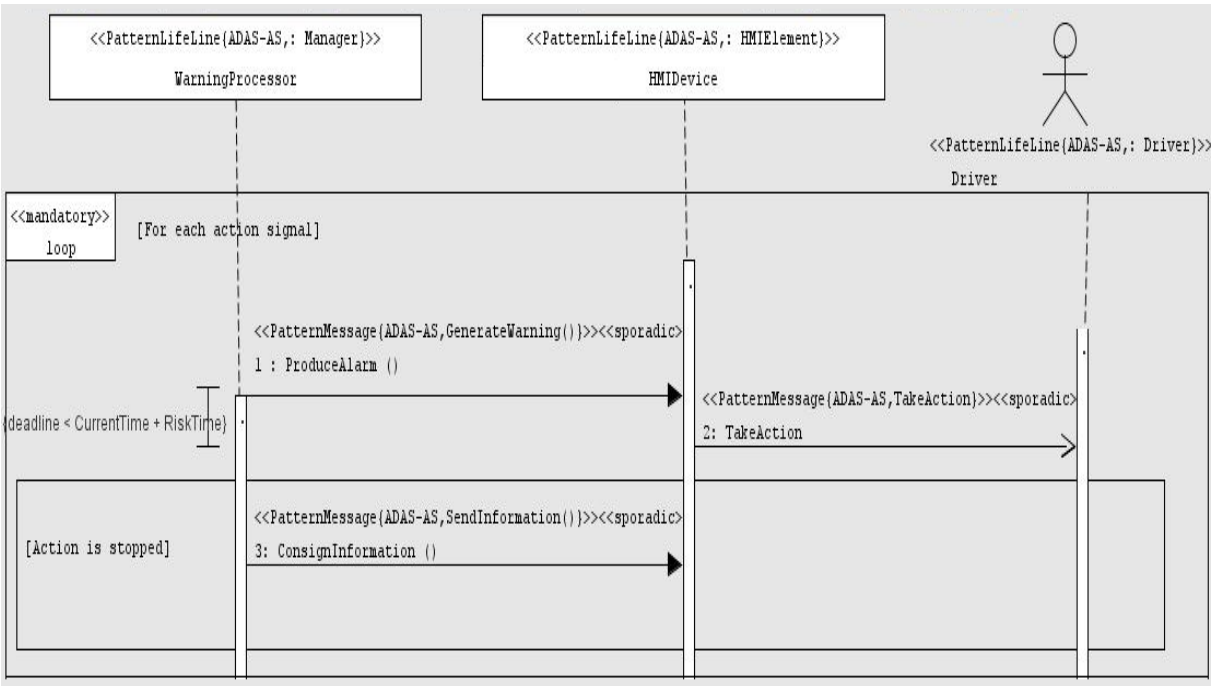


FIGURE B.48 : Modélisation du diagramme de séquence du système RDCW par réutilisation de la vue dynamique du patron *"Action"*.

Résumé : Les systèmes d'aide à la conduite gèrent un grand volume de données qui doivent être mises à jour régulièrement. Cependant, ces systèmes ne permettent, ni de les stocker, ni de les gérer d'une manière efficace. Pour ces raisons, nous proposons l'intégration d'un système de bases de données temps réel (TR) dans les systèmes d'aide à la conduite. Cela permet d'améliorer la tolérance aux fautes, de réduire le nombre de transactions et de réduire leur temps de réponse. La gestion d'un grand volume de données et leurs contraintes TR rend ces systèmes plus complexes, ce qui rend leur modélisation plus difficile. Pour remédier à cette complexité, nous avons proposé trois patrons de conception en nous basant sur un processus de création de patrons. Ce processus permet de définir les étapes à suivre pour déterminer les fonctionnalités et les exigences du domaine d'aide à la conduite, d'une part, et de définir les règles d'unification pour générer les diagrammes UML de classes et de séquence, d'autre part. Pour représenter ces patrons, nous avons proposé le profil UML-RTDB2, pour tenir compte : (i) de l'expression de la variabilité des patrons, (ii) de la représentation des contraintes TR et des aspects non fonctionnels et (iii) des éléments instanciés à partir des patrons lors de la modélisation d'une application cible. Une fois les patrons créés, ils peuvent être réutilisés par les concepteurs pour modéliser des systèmes spécifiques. Pour cela, nous avons proposé un processus de réutilisation pour guider les concepteurs d'applications lors de la réutilisation des solutions de patrons. Enfin, nous avons procédé à l'évaluation de ces patrons en utilisant deux catégories de métriques.

Mots-clés : Systèmes d'aide à la conduite, patrons de conception TR, profil UML, réutilisation des patrons.

Abstract : Advanced Driver Assistance Systems (ADAS) manage an important volume of data that must be updated regularly. However, ADAS don't store, nor manage efficiently these data. For these reasons, we propose to integrate a real-time (RT) database system into ADAS. The integration of the RT database system allows improving the fault tolerance, reducing the number of transactions and minimizing their response time. The management of a lot of data makes these systems complex, thus, their design is highly difficult. To tackle this problem, we have proposed three patterns based on the pattern development process. This process allows defining the steps to follow in order to determine the functionalities and the requirements of the driver assistance domain on one hand, and defining the unification rules for the generation of the UML class and sequence diagrams, on the other hand. In order to represent these patterns, we have proposed UML-RTDB2 profile, which allows (i) expressing the variability of patterns, (ii) representing the real time constraints and the non functional properties and (iii) identifying the role played by each pattern element in a pattern instance. Once the proposed patterns are created, they can be reused by designers to model a specific application. For this reason, we have proposed a process to assist the applications designers when instantiating the patterns solutions. Finally, we have evaluated these patterns based on two categories of metrics.

Keywords : ADAS, RT design patterns, UML profile, reuse of patterns.