UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

FACULTÉ DES SCIENCES DHAR EL MAHRAZ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH

THÈSE de DOCTORAT en COTUTELLE

Pour obtenir le grade de

DOCTEUR EN INFORMATIQUE

UNIVERSITÉ SIDI MOHAMMED BEN ABDALLAH

Faculté des Sciences Dhar El Mahraz -Fès
Centre d'études doctorales "Sciences et Technologies"
Formation Doctorale : STIC
Laboratoire : Informatique, Imagerie et Analyse Numérique (LIIAN)

&

UNIVERSITÉ PARIS 1 PANTHEON-SORBONNE

Ecole Doctorale de Management
Centre de Recherche en Informatique (CRI)

PAR

LAMIAE DOUNAS

MONITORING ET DIAGNOSTIC DES EXIGENCES DES SYSTÈMES AUTO-ADAPTATIFS

Soutenue le 27 avril 2019, devant le jury composé de :

Rachid EL KOUCH, Professeur à l'Institut National des Postes et Télécoms de Rabat - Président du jury.

Camille SALINESI, Professeur à l'Université Paris1 Panthéon-Sorbonne - Directeur de thèse.

Omar EL BEQQALI, Professeur à la Faculté des Sciences Dhar El Mahraz de Fès - Directeur de thèse.

Ali IBRIZ, Professeur à l'Ecole Supérieure de Technologie de Fès - Rapporteur.

Mostafa BELLAFKIH, Professeur à l'Institut National des Postes et Télécoms de Rabat - Rapporteur.

Mourad OUSSALAH, Professeur à l'Université de Nantes - Rapporteur.

Raul MAZO Pena, Professeur à l'Université Paris1 Panthéon-Sorbonne - Examinateur.

RESUME

Les systèmes auto-adaptatifs sont capables d'assurer leurs fonctionnements d'une façon autonome et de changer leurs comportements en réponse aux variations de leurs environnements d'exécution. Le mécanisme d'auto-adaptation consiste en une boucle de rétroaction par laquelle les sorties du système sont surveillées et une action de compensation est prise si le système ne remplit pas sa fonction. Cette capacité d'auto-adaptation donne lieu à une grande autonomie et variété. Cependant, plus l'autonomie et la capacité d'adaptation sont grandes, plus le système devient complexe. Au cours de la dernière décennie, les capacités auto- * de ces systèmes se sont améliorées. Cependant, le corpus des connaissances existant est loin d'être suffisant pour répondre au changement de l'environnement d'exécution et à l'évolution dynamique des exigences. D'autant plus qu'il est toujours difficile de faire accepter cette technologie qui nécessite de la confiance - ce qui est difficile à cerner et qui est rapidement éliminée si certains types de ces systèmes échouent. Le monitoring de la satisfaction des exigences à l'exécution et idéalement, le diagnostic des non-conformités aux exigences sont les clés pour rendre ces systèmes plus robustes aux changements inattendus et les faire évoluer avec une grande capacité d'autonomie.

Cette thèse propose un cadre logiciel de monitoring d'exigences dédié aux systèmes auto-adaptatifs. Le cadre logiciel définit une méthodologie de spécification d'exigences pour simplifier la tâche de la spécification et assurer la cohérence des exigences spécifiées. Cette méthodologie définit la variabilité de l'environnement d'exécution, la variabilité des stratégies d'adaptation du système cible en termes de buts et d'opérations, ainsi que les interrelations entre eux à l'aide des claims et des soft-dépendances. En temps réel, le fonctionnement du monitoring est assuré par quatre processus : un monitoring du système cible et de son environnement d'exécution, une analyse de la satisfaction des exigences en cas de changement, un diagnostic des exigences en cas de non-conformités détectées et enfin une planification des configurations optimales pour réconcilier le système avec ses buts.

Un prototype a été développé pour concrétiser le raisonnement du monitoring et mener à bien une évaluation empirique de notre approche. Cette évaluation a porté sur deux cas d'études. Le résultat a démontré l'expressivité de l'approche et sa performance en termes de temps d'exécution et de passage à l'échelle.

Mots-clés : systèmes auto-adaptatifs, monitoring des exigences, modélisation orientée buts, boucles de rétroaction, programmation par contraintes, évolution, diagnostic des défaillances, intelligence artificielle.

ABSTRACT

Self-adaptive systems are built on the premise that they can monitor their environments, detect the need for change, and apply the required actions to ensure their proper functioning without or with minimal human intervention. The self-* capabilities, which are the key features of self-adaptation, have been improved over the last decade. However, the existing body of knowledge is far from being sufficient to meet the dynamic evolution of requirements and the inherent uncertainty of the operating environment. Monitoring the satisfaction of the requirements at runtime, and ideally diagnosing the cause of requirements denial is the key toward making these systems more robust to unexpected changes and allow them to evolve with a great capacity for autonomy.

This thesis proposes RMAS, a framework for monitoring and diagnosing the requirements at runtime. RMAS defines a methodology to facilitate the requirements specification and ensure the consistency of the specified requirements based on goal oriented modeling, variability analysis, and constraint programming. This methodology establishes the variability of the execution environment, the variability of the target system's adaptation strategies in terms of goals and operations, and the interrelationships between them using claims and soft-dependencies. At runtime, RMAS defines four mechanisms: a monitoring of the target system and its operating environment, an analysis of requirements satisfaction in case of change, a diagnosis of the requirements whenever a deviation is detected, and finally a planning of optimal configurations to reconcile the system with its goals.

A prototype has been developed and tested using two cases studies. The result of the evaluation indicates a good expressiveness of the specification and a good performance regarding the execution time and the scalability.

Keywords: self-adaptive systems, requirements monitoring, goal modeling, feedback loops, constraints programming, evolution, and model-based fault diagnosis, artificial intelligence.

Je dédie cette thèse à la mémoire de mon cher PAPA, à ma chère MAMAN, et à mon adorable fils OMAR.

REMERCIEMENTS

Une thèse se construit sur des rencontres et des échanges. Je voudrais remercier ici toutes les personnes qui ont contribué directement ou indirectement à l'aboutissement de mes travaux de thèse.

Je souhaite remercier avant tout mes directeurs de thèse : Monsieur El Beqqali Omar, professeur à la Faculté des Sciences Dhar El Mehraz de Fès, et Monsieur Salinesi Camille, directeur du Centre de Recherche en Informatique (CRI) et professeur à l'Université Paris 1 Panthéon-Sorbonne, pour leur implication dans la direction de ces travaux de recherche, pour leurs conseils scientifiques et leurs qualités humaines.

Réviser une thèse n'est pas une tâche facile, je remercie vivement les membres du jury pour leurs remarques et leurs conseils qui m'ont permis d'améliorer ce manuscrit. Je remercie Monsieur El Kouch Rachid, Professeur à l'Institut National des Postes et Télécommunications (INPT) d'avoir accepté de présider le jury de la soutenance. Monsieur Bellafkih Mostafa, Professeur à l'Institut National des Postes et Télécommunications (INPT) – d'avoir accepté de rapporter cette thèse et pour son accueil chaleureux et ses conseils tout au long des deux années que j'ai passé à Rabat. Monsieur Ibriz Ali, Professeur habilité à l'Ecole Supérieur de Technologie (EST) de Fès, d'avoir accepté de rapporter cette thèse. Monsieur Oussalah Mourad, Professeur à l'Université de Nantes, d'avoir accepté de rapporter cette thèse et d'avoir fait le déplacement depuis Nantes pour participer à ce jury. Et enfin, Monsieur Mazo Raul, Professeur à l'université Paris 1 Panthéon-Sorbonne, d'avoir fait le déplacement depuis Paris pour participer à ce jurer mais aussi pour ses conseils avisés et son écoute tout au long de mes séjours de recherche à Paris.

J'exprime ma gratitude au Centre National de Recherche Scientifique et Technique (CNRST) qui a financé cette thèse à ses débuts.

Je remercie tous les membres du laboratoire de recherche LIIAN, à la faculté des sciences Dhar El Mehraz de Fès.

J'ai également été bien accueillie par le CRI, qui a été un lieu d'échange et de collaboration avec de nombreux chercheurs et professeurs : Madame Carine Souveyet, Madame Rebecca Deneckere, Monsieur Daniel Diaz, Madame Irina Rychkov, et spécialement Madame Bénédicte Le Grand et Madame Manuele Kirsch Pinheiro. Je vous remercie tous pour votre accueil.

Je tiens à remercier le personnel administratif de la Faculté des Sciences Dhar El Mehraz et du CRI, qui m'ont accompagné dans toutes les démarches administratives avec gentillesse et réactivité.

Je remercie également les doctorants du monde entier que j'ai pu fréquenter durant la thèse – Maroc, France, Tunisie, Algérie, Roumanie, Colombie, Brésil et Russie. Il est difficile d'en faire une liste

exhaustive. Je remercie spécialement elena Kushnareva, danny Munera, flavia Caia, david Beserra pour leur joie de vivre, juan Carlos Muñoz et luisa Rincon pour l'aide et l'assistance qu'ils m'ont apportés concernant l'outil VariaMos, yosra Lassoued, fabrice Boissier et ali Jaffal d'être toujours à l'écoute.

Mes remerciements vont aussi à ma famille et mes amis qui ont plus particulièrement assuré le soutien affectif à ce travail.

Je garde pour la fin un remerciement particulier pour mon mari, Mouhcine. Je te remercie pour ta patience et ton soutien dans les périodes les plus difficiles de la thèse. Je n'aurais pas pu avancer sans ton encouragement et ton amour.

SOMMAIRE

Introduction 15

Problématique 16

Questions de recherche 17

Méthodologie de recherche : 'Design Science' 18

Contributions 19

Organisation de la thèse 20

Partie 1 : Etat de l'Art 21

1. Concepts de base et Terminologie 23

1.1. Monitoring des exigences à l'exécution 23

1.2. Ligne de produits logiciels dynamique (LdPD) 26

1.3. Modélisation Orientée But 28

1.4. Instrumentation 31

1.5. MAPE-K loop 31

1.6. Evolution 32

1.7. Conclusion du chapitre 33

2. Approches du monitoring des exigences - Revue Systématique 35

2.1 Taxonomie du monitoring des exigences à l'exécution 36

2.1.1 Spécification des exigences 39

2.1.1.1 Type de propriétés 39

2.1.1.2 Méthodes de spécification 40

2.1.2 But du monitoring 43

2.1.3 Approche 43

2.1.3.1 Implémentation du monitoring 43

2.1.3.2 Emplacement du code du monitoring 47

2.1.3.3 Déclencheur 47

2.1.3.4 Support d'incertitude 47

2.1.4 Instrumentation : couches d'abstraction du système 48

2.1.5 Mode de monitoring 50

2.2 Résultats d'analyse des études primaires 50

2.3 Discussion 63

2.4 Validité de la recherche 66

2.5 Conclusion du chapitre 67

Partie 2 : Approche Proposée 68

3. Présentation de l'approche 70
3.1 Exemple illustratif 70
3.1.1 Présentation de l'exemple 70
3.1.2 Exemples des exigences à surveiller 71
3.2 Approche générale : RMAS 72
3.2.1 Spécification des exigences du monitoring 74
3.2.1.1 Modélisation des exigences 74
3.2.1.2 Support d'incertitude 78
3.2.1.3 Formalisation de la spécification 79
3.2.2 Monitoring des exigences en ligne 84
3.2.2.1 Manipulation du M@RT 85
3.2.2.2 Analyse des exigences 85
3.2.2.3 Diagnostic des causes des non-conformités 86
3.2.2.4 Planification des solutions optimales 89
3.3 Conclusion du chapitre 90

4. Mise en œuvre 93
4.1 Architecture 93
4.2 Fonctionnalités 95
4.3 Conclusion du chapitre 104

Partie 3 : Evaluation empirique 106

5. Etude de cas 1 : Système e-learning adaptatif 'INSPIREus' 108
5.1 Méthodologie d'expérimentation 108
5.1.1 Collecte des données 108
5.1.2 Traitement des données 109
5.2 Résultats 110
5.3 Discussion 116
5.4 Validité de recherche 116
5.5 Conclusion du chapitre 117

6. Etude de cas 2 : Contrôleur adaptatif d'intersection à feux de circulation 119
6.1 Notions fondamentales concernant le trafic aux carrefours à feux 120
6.2 La régularisation du trafic 121
6.3 Description générale du système du monitoring pour un contrôleur de feux adaptatif 123

6.4 Spécification des exigences du monitoring par REFAS 126
6.5 Validation expérimentale 132
6.6 Etude comparative 138
6.7 Conclusion du chapitre 138
Conclusion générale et perspectives 139
Références bibliographiques 142
Annexe 154

LISTE DES FIGURES

Figure 1 : Produit unique pour tous Vs produits individuels (Pohl, Böckle, & van Der Linden, 2005) 27
Figure 2: Modèle de buts montrant le AND/OR raffinement des buts 29
Figure 3 : le principe de la boucle MAPE-K 32
Figure 4 :Taxonomie du monitoring des exigences à l'exécution avec 5 dimensions (la spécification des exigences, l'approche de mise en œuvre, le but du monitoring, le mode d'exécution du monitoring et l'instrumentation). 38
Figure 5: les couches d'abstraction (Faber et al., 2016). 49
Figure 6 : distribution des études primaires selon l'outil supportant l'automatisation d'analyse des exigences 52
Figure 7 : Distribution des études primaires selon le domaine d'application 53
Figure 8 : Corrélation entre l'implémentation du monitoring et le domaine d'application 54
Figure 9 : Corrélation entre les langages de spécification et domaine d'application 55
Figure 10 : Corrélation entre but de monitoring et les propriétés sujettes au monitoring 57
Figure 11 : Corrélation entre but de monitoring et les méthodes de spécification des exigences 58
Figure 12 : corrélation entre la méthode de spécification des exigences et l'implémentation du monitoring 59
Figure 13 : Corrélation entre but de monitoring et le mode de monitoring 60
Figure 14 : Corrélation entre les méthodes de spécification et le support d'incertitude 60
Figure 15 : Corrélation entre l'implémentation du monitoring et le support d'incertitude 61
Figure 16 : Corrélation entre les couches d'abstraction et les technologies d'instrumentation 62
Figure 17 : aperçu général de l'approche du monitoring des exigences en ligne 73
Figure 18 : la vue de variabilité appliquée à INSPIREus 75
Figure 19 : la vue des soft goals appliquée à INSPIREus 76
Figure 20 : la vue de contexte appliquée à INSPIREus 77
Figure 21 : La vue satisficing soft-goals appliquée à INSPIREus 78
Figure 22 : architecture logique du prototype RMAS 94
Figure 23 : aperçu des relations entre les fonctionnalités offertes par le prototype RMAS 95
Figure 24 : interface d'initialisation de l'attribut de mappage entre un concept REFAS et les champs des fichiers de journalisation (étapes de 1 à 7 en rouge) 96
Figure 25 : les phases d'analyse des données. (Dounas et al., 2019) 109
Figure 26 : illustration de la précision pour chaque apprenant groupée par style d'apprentissage. (Dounas et al., 2019) 112
Figure 27 : Distribution du niveau d'accomplissement des exigences Exig3, et Exig4 par apprenant (code des couleurs : 'nuances du vert' Classe A ; 'jaune' Classe B ; 'orange' Classe C ; and 'rouge' Classe D). (Dounas et al., 2019) 113
Figure 28 : Distribution du niveau d'accomplissement des exigences Exig1, Exig2 par apprenant (code des couleurs : 'nuances du vert' Classe A ; 'jaune' Classe B ; 'orange' Classe C ; and 'rouge' Classe D. (Dounas et al., 2019) 114

Figure 29 : Distribution de la qualité des configurations par apprenant (code des couleurs : 'nuances du vert' Classe A; 'jaune' Classe B; 'orange' Classe C; and 'rouge' Classe D. (Dounas et al., 2019) *115*
Figure 30 : distribution des OPERs sources de violation de l'exigence Exig1 par apprenant (Dounas et al., 2019) *115*
Figure 31 : fonctionnement général d'un monitoring des exigences pour un système de contrôle de feux de circulation *124*
Figure 32 : Mouvements typiques des véhicules et des piétons dans une intersection à quatre branches aux deux sens et un virage à gauche protégé *124*
Figure 33 : Schéma de phasage et de déplacement pour l'intersection de deux voies à sens unique *126*
Figure 34 : la variabilité pour le contrôleur de carrefour à feux *127*
Figure 35 : la vue soft goals pour le contrôleur de carrefour à feux *128*
Figure 36 : la vue de contexte du trafic *129*
Figure 37: débit de saturation, temps effectif du vert, et temps perdu dans un mouvement *130*
Figure 38 : la vue satisficing soft goals pour le contrôleur de carrefour à feux *132*
Figure 39: Modèle d'exigences à 10 niveaux *134*
Figure 40 : comparaison du temps de diagnostic de RMAS et les approches de Wang et al. [Ref45], Ali et al. [Ref50], et Dalpiaz et al. [Ref46] face à des modèles d'exigences de tailles différentes. *138*

Figure A. 1 : Nombre d'études par année de publication *167*
Figure A. 2 : distribution de la qualité des études primaires dans le temps *168*
Figure A. 3 : notation des études primaires selon les six questions de qualité (CASP) *168*
Figure A. 4 : Distribution des études primaires selon les types de propriétés. *169*
Figure A. 5 : Distribution des études primaires selon les méthodes de modélisation *169*
Figure A. 6 : Fréquence des scores de qualité obtenus selon les méthodes de spécification *170*
Figure A. 7 : Distribution des études primaires selon le but de monitoring *171*
Figure A. 8 : Distribution des études primaires selon l'implémentation du monitoring *171*
Figure A. 9 : Distribution des études primaires selon le déclencheur de monitoring *172*
Figure A. 10 : Distribution des études primaires selon leur support à l'incertitude *172*
Figure A. 11 : Distribution des études primaires selon les couches d'abstraction *173*
Figure A. 12 : Distribution des études primaires selon le mode de monitoring *173*

LISTE DES ABRÉVIATIONS, DES SIGLES ET DES ACRONYMES

AADL Architecture Analysis and Design Language

AHP Procédure hiérarchique d'analyse (en anglais, Analytic Hierarchy Process).

BDI Modèle "Croyance-Désir-Intention » (en anglais, Belief–Desire–Intention)

BMC Brzozowski et Mc Cluskey « Méthode d'élimination »

CFG Grammaire non contextuelle (en anglais, Contesterez Grammars)

CIM Modèle métier indépendant (en anglais, Computation Independant Model)

CTL Logique arborescente (en anglais, Computation Tree Logic)

CSL Logique temporelle pour les chaînes de Markov à temps continu (en anglais, Continuous Stochastic Logic**)**

CTMC Chaine de Marcov à temps continue (en anglais, Continuous-Time Markov Chain)

DAS Système dynamique adaptatif (en anglais, Dynamic Adaptive System)

DTMC Chaine de Marcov à temps discret (en anglais, Discrete-Time Markov Chain)

ERE Expression régulière étendue (en anglais, Extended Regular Expressions).

FSA Automate fini (en anglais, Finite State Automaton)

FSM Automate fini (en anglais, Finite State Machines)

HMM Chaine de Markov cachée (en anglais, Hiden Markov Model)

JMX Java Management Extensions

LTL Logique temporelle linéaire

LTS Système de transition étiqueté (en anglais, Labeled Tranistion System)

MDA Architecture dirigée par les modèles (en anglais, Model Driven Architecture)

MFOTL Logique métrique du premier ordre (en anglais, Metric First-Order Temporal Logic)

MOF Meta-Object Facility

NFA	Automate fini non déterministe (en anglais, Non-Deterministic Finite Automate)
NFR	Exigences non fonctionnelles (en anglais, Non-Functional Requirements)
OCL$_{TM}$	OCL with Temporal Message logic
OrBAC	MultiOrganization-Based Access Control
PCTL	Logique probabiliste arborescente (en anglais, Probabilistic Computational Tree Logic)
PMM	Property Meta-Model
PRISM	Probabilistic Symbolic Model Checker
pTLTL	Logique temporelle avec des opérateurs du temps passé (en anglais, Past Time LTL)
RELAX	Relax-ed requirements language
RETE	Algorithme performant de filtrage par motif
rPATL	Variante de PATL (en anglais**,** Probabilistic Alternating-Time Temporal Logic)
RTL	Logique en temps réel (en anglais, Real-Time Logic)
SAS	Système auto-adaptatif (en anglais, Self-Adaptive System)
SAT	Problème de satisfiabilité
SBA	Application orientée service (en anglais, Service Based Applications)
SBS	Systèmes orientés service (en anglais, System Based Service)
SLA	Accord de niveau de service (en anglais**,** Service Level Agreement)
SLAng	Specification Language for SLAs
SMT	Satisfiabilité modulo des théories
SMG	Jeu stochastique à plusieurs joueurs (en anglais, Stochastic Multiplayer Game)
SOA	Architecture orientée services (en anglais, Service-Oriented Architecture)
SOS	Systèmes des systèmes (en anglais, System Of Systems)
WSDL	Web Services Description Language

INTRODUCTION

L'évolution des technologies numériques a connu une accélération sans précédent ces dernières décennies. Aujourd'hui, la plupart des équipements à notre disposition sont bardés d'électronique, capables de communiquer entre eux et disposant d'une certaine autonomie faisant d'eux des entités à part entière. L'émergence de cet « internet des objets » a fait d'internet un outil puissant de partage de données, mais a engendré une grande distribution et hétérogénéité des systèmes.

La complexité des systèmes logiciels actuels et les croissants coûts de leurs maintenances ont remis en question toutes les pratiques de l'ingénierie logicielle utilisées jusqu'alors pour la conception jusqu'à la maintenance des systèmes, en particulier celles liées à la définition des exigences (GAO, 1992). Par conséquent, la communauté de génie logiciel s'est mise à chercher des méthodes innovatrices de développement en s'inspirant des domaines connexes notamment la biologie, la robotique et la théorie de contrôle. Cela a abouti à des systèmes auto-adaptatifs qui ont la capacité de produire une large variété de réponses pour satisfaire les exigences et optimiser leur état en temps réel face à un environnement d'exécution en constante évolution. Nous entendons par l'environnement d'exécution tout ce qui est observable par le système, tels que les entrées des utilisateurs et les dispositifs matériels externes.

Le développement des systèmes auto-adaptatifs est basé sur la prémisse qu'ils peuvent s'auto-configurer, s'auto-optimiser et s'auto-réparer, et réduire ainsi les coûts de maintenance. Pour ce faire, ces systèmes doivent être capables d'évaluer leur performance, détecter le besoin du changement et appliquer les actions requises à l'exécution avec une grande capacité d'autonomie (sans ou avec un minimum d'intervention humaine), et idéalement diagnostiquer des problèmes en se référant à des modèles du système lui-même, ainsi qu'à des modèles de l'environnement d'exécution. L'auto-adaptation est assurée par le principe de « feedback » (Salehie Tahvildari, 2009), l'un des concepts fondamentaux de la théorie de contrôle, qui consiste dans la triade : *surveiller* son propre comportement et l'environnement d'exécution, *analyser* les changements importants (causes internes, évènements externes… etc.), et *décider* comment agir face aux changements pour optimiser son fonctionnement. Cette capacité d'auto-adaptation donne lieu à l'autonomie et à la variété. Cependant, plus l'autonomie et la capacité d'adaptation sont grandes, plus le système devient complexe. Le défi majeur de la communauté de génie logiciel reste alors la conception de ces systèmes. En effet, pour répondre à toutes les situations auxquelles il pourrait être confronté, un système auto-adaptatif dépend fortement de la perception qu'il a de son environnement et des exigences qu'il doit satisfaire. En pratique, il est très difficile de prédire tous les scénarios en temps de conception et il est impossible de s'assurer que la couverture est complète. D'autre part, il n'est pas toujours intuitivement clair comment évaluer les fonctionnalités et les performances de ces systèmes, et la vérification de leur cohérence devient rapidement intraitable (Nuseibeh et al., 2000) (Laddaga, 1999). Il en résulte que le

comportement du système n'émerge clairement qu'au moment d'exécution, cela signifie que certains comportements peuvent être perçus comme imprévisibles par les utilisateurs. Ce manque d'intelligibilité peut amener les utilisateurs à perdre leur confiance en ces systèmes et cesser de les utiliser (Muir, 1994) (Sawyer et al., 2010).

Problématique

Un grand nombre des travaux de recherche dans le domaine des SASs se sont intéressés au monitoring des exigences. Fickas et Feather (1998) ont été les premiers à souligner le besoin de surveiller la satisfaction des exigences à l'exécution. Par la suite, plusieurs approches (Robinson, 2006) (Wang et al. 2009) (Souza & Mylopoulos, 2009) (Welsh & Sawyer, 2009) (Bencomo et al., 2010) ont été proposées pour vérifier si le comportement d'un système adaptatif contribue à la satisfaction des exigences ou s'écarte de celles-ci. Cependant, le monitoring des exigences étant une discipline en plein développement, il existe encore quelques défis (décrits ci-après) qui ont motivé la recherche présentée dans cette thèse :

- Diagnostic : Le premier défi est lié au diagnostic et à l'évaluation des SASs qui restent jusqu'à nos jours des problèmes majeurs de l'ingénierie logicielle. En effet, il n'y a pas une manière claire comment évaluer ces systèmes et bien que l'évaluation soit toujours possible, cela n'implique toujours pas un diagnostic des défaillances/défauts détectés et donc une élimination des causes de celles-ci. Selon Laddaga (2006), l'une des pratiques utilisées pour améliorer un système adaptatif est de répondre à une mauvaise performance détectée en appliquant au hasard des algorithmes ou des bouts de codes jusqu'à la détection des vraies sources de défaillances. En effet, une interface de diagnostic à différents niveaux de granularité et basée sur l'analyse de la satisfaction des exigences fait encore défaut (Robinson, 2010). Idéalement, une telle interface présenterait aux développeurs et aux utilisateurs finaux un diagnostic complet des causes immédiates jusqu'à la racine des non-conformités.

- Généricité : malgré l'intérêt porté au monitoring des exigences, les approches de monitoring existantes sont généralement étroitement liées à la plate-forme d'exécution, au type d'architecture, ou la technologie (par exemple les applications java) de mise en œuvre du système considéré, sans pour autant étudier la possibilité de généralisation. Par conséquent, les solutions proposées ne sont pas applicables sur d'autres systèmes ou problèmes similaires.

- Abstraction à la technologie : seuls quelques travaux de monitoring ont fait abstraction à la technologie, et se sont penchés sur l'analyse des interrelations entre les exigences indépendamment de choix de réalisation du système, ce qui est essentiel pour permettre l'évolution du monitoring et assure sa flexibilité face à un environnement en constant changement.

Ainsi, l'objectif de la présente recherche est alors de remédier à ces problèmes en proposant une approche systématique pour la conception d'un monitoring des exigences des SASs visant à évoluer

ces systèmes dans des situations indésirables et imprévisibles – représentées par des non-conformités aux exigences spécifiées.

Questions de recherche

Notre question de recherche est ainsi formulée par cet énoncé :

QR : Comment améliorer la conformité des systèmes auto-adaptatifs à leurs exigences et permettre leur évolution en temps réel d'une manière générique et automatique ?

Cette question centrale de recherche se compose des deux sous-questions suivantes :

QR1 : Comment spécifier formellement la conformité aux exigences et automatiser la génération d'un monitoring de ces exigences à partir de la spécification ?

L'importance des spécifications formelles dans l'ingénierie logicielle est largement reconnue. Elles permettent une description abstraite et précise du comportement d'un système et fournissent une base précise pour le développement des tests et leur automatisation. Cependant, il est très difficile de formuler manuellement ces spécifications devant la complexité des SASs tout en garantissant leur validité. En effet, leur complexité dépend de leur capacité d'adaptation. Plus la capacité d'adaptation est grande, plus le système à traiter est complexe et plus sa vérification devient rapidement intraitable du fait du grand nombre de scénarios à vérifier. Ainsi, pour garantir la validité de la spécification tout en maîtrisant la complexité du système cible, le modèle d'exigences devrait être représenté de manière suffisamment formelle pour être interprétées automatiquement mais pas trop formelle pour imposer une charge supplémentaire aux analystes et aux ingénieurs.

Pour ce faire, la formulation de la spécification dans le cadre de cette thèse est basée sur deux stratégies : (1) l'abstraction des composants communs et variables du système cible dans un modèle de haut niveau permettant de décrire les stratégies d'adaptations et de les connecter aux exigences (les exigences du système et les exigences du monitoring) spécifiées indépendamment des technologies adoptées lors du développement du système. Cela permet de gérer la complexité du système cible tout en raisonnant sur l'architecture logicielle du système, ce qui constituerait une base pour le diagnostic des non-conformités. (2) la génération automatiquement d'une spécification formelle à partir du modèle crée. Cette automatisation vise à faciliter la tâche des concepteurs/analystes qui ne maîtrisent généralement qu'un nombre limité de langages et technologies ; et à réduire ainsi les erreurs de spécification liées au facteur humain.

QR2 : Comment améliorer la conformité des systèmes auto-adaptatifs à leurs exigences à l'aide des informations du monitoring ?

La réponse à cette question vise à combler le besoin des outils de monitoring des exigences pour améliorer la conformité des SASs aux exigences en temps réel. En effet, La surveillance des exigences

n'est pas une finalité en soi, le but ultime de cette thèse est de permettre la détection et le diagnostic des non-conformités à différents niveaux de granularité. Ces informations seront exploitées en temps réel pour générer automatiquement des actions de correction. Dans le cas d'impossibilité de remédiation automatique, ces informations seront stockées dans une base de données, et pourront être utilisées par l'équipe d'ingénieur du système cible pour entamer plus tard un processus d'amélioration du modèle des exigences. Ainsi notre approche permet d'évoluer les systèmes auto-adaptatifs en permanence en fonction des besoins d'adaptation détectés par le monitoring.

Méthodologie de recherche : 'Design Science'

La recherche présentée dans cette thèse est axée sur le design. Elle vise à concevoir un monitoring des exigences afin d'assurer la conformité des systèmes auto-adaptatifs à leurs exigences en temps réel, d'une manière générique et automatique. Cette recherche suit la méthodologie de 'Design Science' - initialement introduite par March Simon (1995) comme une tentation de créer une science unifiant les pratiques de design qui produisent des 'choses' servant des objectifs humains ; elle a été progressivement formalisée par d'autres chercheurs en systèmes d'information (SI) Hevener et al. (2004), Peffers et al. (2007) pour transformer le problème de conception en une démarche scientifique permettant de (i) produire des contributions scientifiques théoriques rigoureuses et (ii) produire des artefacts (ex : solution logicielle, algorithmes, un langage ou des méthodologies) à partir des pratiques existantes pour la résolution d'un problème du monde réel où dont les solutions existantes ne sont pas efficaces.

Parmi les processus proposés pour mener à bien une recherche en science de design, nous avons suivi les processus de Peffers et al. (2007) à savoir, (1) identification du problème et motivation, (2) définition des objectifs de la solution, (3) conception et développement, (4) démonstration, (5) évaluation et (6) communication. À cet effet, les stratégies de recherche suivantes ont été mises en œuvre :

Tout d'abord, une revue de littérature ad hoc a été menée sur les systèmes auto-adaptatifs pour identifier les lacunes dans ce domaine et les besoins d'amélioration du point de vue de l'ingénierie des exigences.

Ensuite une revue systématique sur le monitoring des exigences à l'exécution a été effectuée pour analyser les pratiques dans ce domaine et les contributions dédiées aux systèmes auto-adaptatifs. Par la suite, les tendances dans ce domaine et les points d'amélioration ont été décelées.

Sur la lumière des résultats des deux précédentes étapes, une approche générique et entièrement automatisée de monitoring et de diagnostic des exigences a été proposée visant à améliorer la conformité des SASs à leurs exigences à l'exécution.

Enfin, l'approche a été appliquée sur deux cas d'études et une série d'expérimentations a été menée

pour tester la performance du prototype développé.

Contributions

La contribution associée à la thèse s'articule autour de :

- Une revue de littérature sur le monitoring des exigences à l'exécution visant à déceler les pratiques de ce domaine émergent.
- Une approche générique de monitoring des exigences, dédiée aux systèmes auto-adaptatifs. Elle est générique dans le sens où le monitoring des exigences supporte tout type de systèmes auto-adaptatifs, à condition que la spécification des exigences soit modélisée sous forme de ligne de produits logiciels dynamique et que les interfaces de communication avec le système soient fournies. Cela implique que toute action d'adaptation impliquant des tâches spécifiques au système cible devra être mise en œuvre par l'équipe de développement du système cible.
- Une approche automatique permettant d'améliorer la conformité des systèmes auto-adaptatifs aux exigences en ligne, partant du monitoring et la vérification des exigences jusqu'à la génération des remédiations aux défaillances détectées.
- Un cadre logiciel et outil prototype appelé RMAS (en anglais : Runtime Monitoring of self-Adaptive Systems) qui concrétise les approches susmentionnées. RMAS prend en entrée :

a. Un modèle de spécification des exigences sous forme de ligne de produits logiciels, représenté dans un format lisible par la machine.
b. Les informations sur l'exécution du système fournies par le système à travers les interfaces de communication.

En temps de conception, RMAS permet à l'équipe de développement l'annotation de la spécification pour lier les composant architecturaux à l'implémentation réelle du système cible, et la génération automatiquement d'un programme de contraintes à partir de la spécification. En temps réel, RMAS effectue les tâches suivantes :

- Pour chaque changement des valeurs des variables d'environnement ou du comportement du système, RMAS décide de l'activation ou la désactivation de la vérification de conformité.
- Dans le cas d'activation de la vérification, il résout automatiquement le système de contraintes à l'aide d'un solveur de contrainte.
- En cas d'impossibilité de solutions avec les valeurs des variables du contexte en cours. Un processus de diagnostic sera déclenché pour détecter les composants faisant défaut.
- Finalement, un processus de génération de remédiation aux non-conformités consiste à résoudre le système avec les données de contexte collectées et la planification des solutions sans les composants défaillants. Les solutions générées par le solveur peuvent être utilisées pour améliorer la satisfaction des exigences en temps réel ou plus tard dans un processus de maintenance.

- Une série d'expérimentations qui repose sur deux cas d'études visant a démontré l'expressivité de l'approche et la performance de l'outil développé en termes de temps d'exécution et passage à l'échelle.

Organisation de la thèse

Le manuscrit de la thèse est organisé en trois parties dont chacune comporte deux chapitres :

- La première partie présente un état de l'art sur le monitoring des exigences à l'exécution.

- Chapitre 1 introduit les concepts de base et les notations utilisés tout au long de la thèse.

- Chapitre 2 présente une revue systématique de la littérature sur le monitoring des exigences à l'exécution visant à fournir une vision unifiée de ce domaine émergent. D'une part, une taxonomie a été créée d'une manière itérative afin de mieux analyser les approches existantes et faciliter la comparaison entre eux. D'autre part, une analyse croisée des cinq dimensions de la taxonomie a été menée dans le but de ressortir les tendances dans ce domaine et les points d'améliorations possibles.

- La deuxième partie est dédiée à la description de l'approche proposée et son implémentation.

- Chapitre 3 présente notre approche du monitoring des exigences à l'exécution des SASs et un cadre logiciel RMAS qui se compose de 4 composants : (i) un monitoring des changements du système cible et son environnement d'exécution, (ii) un composant d'analyse des exigences, (iii) un composant de diagnostic des non-conformités aux exigences, et finalement (iv) un composant de planification des configurations optimales pour réconcilier le système avec ses buts.

- Chapitre 4 présente la mise en œuvre de l'approche, y compris les algorithmes qui automatisent le raisonnement du monitoring des exigences.

- La troisième partie présente l'évaluation de l'approche. Une série d'expérimentations a été menées en se basant sur deux cas d'études issues des domaines d'éducation à distance et du transport intelligent.

- Chapitre 5 décrit la première étude de cas liée à un système e-learning adaptatif appelé INSPIREus. L'étude vise à tester la faisabilité de l'approche et son expressivité.

- Chapitre 6 décrit la deuxième étude de cas liée à un contrôleur adaptatif d'intersection à feux de circulation. L'étude vise à évaluer la performance du prototype développé face à des spécifications de grandes tailles.

Enfin, une conclusion résume les contributions de ce travail et présente les perspectives de recherche envisagées.

Première Partie
État de l’art

Cette première partie vise à fournir les éléments de compréhension du concept du monitoring des exigences et à présenter les approches existantes dans ce domaine et leurs champs d’applications. Pour ce faire :

- Le premier chapitre définit les concepts de base utilisés tout au long de la thèse.
- Le deuxième chapitre présente une revue systématique (RS) sur le monitoring des exigences à l’exécution qui suit la méthode systématique proposée par Kitchenham et al. (2009). À cet objectif, nous proposons une taxonomie sur le monitoring des exigences à l’exécution qui est créée d’une manière itérative à partir de la lecture des études primaires. La taxonomie est ensuite utilisée pour classer et analyser ces études primaires. À la lumière des résultats de la revue systématique, nous nous intéressons à des recherches approfondies sur les pratiques d'ingénierie associées au monitoring des exigences à l’exécution pour les systèmes auto-adaptatifs.

Chapitre 1

1. CONCEPTS DE BASE ET TERMINOLOGIE

Ce chapitre vise à définir les concepts et les termes de base liés à l'auto-adaptation et aux exigences, notamment le monitoring des exigences, les systèmes auto-adaptatifs, les lignes de produits logiciels dynamiques, la modélisation orientée but, l'instrumentation, la boucle MAPE-K et l'évolution des systèmes.

1.1. Monitoring des exigences à l'exécution

La gestion des exigences est essentielle pour concevoir des systèmes logiciels conformes aux besoins des utilisateurs dans les temps et dans les budgets. Ainsi, une bonne spécification conduit à une meilleure implémentation alors qu'une mauvaise définition des exigences ou une mauvaise interprétation de celles-ci est à l'origine de la plupart des défaillances des systèmes logiciels (GAO, 1992) (Heitmeyer et al., 1996). Le comportement total des systèmes auto-adaptatifs étant émergent seulement en temps réel, les tests de vérification et de validation (V & V) pendant le cycle de développement bien que nécessaires pour réduire le risque de défaillances, resteront à jamais insuffisants pour garantir la fiabilité de ces systèmes et assurer leur conformité aux exigences. D'autant plus qu'ils sont parfois coûteux et délicats du fait de l'utilisation des plateformes matérielles dédiées onéreuses dans certains domaines tel que l'avionique. Ainsi, pour gagner la confiance à l'égard de ces systèmes, les techniques classiques d'assurance qualité doivent être repensées et étendues à l'exécution pour résoudre les problèmes émergents lorsque plus de détails sur l'architecture technique du système sont disponibles et plus d'informations peuvent être collectées en observant le système. Ainsi, en absence des techniques d'assurance qualité adaptées à ces systèmes, le monitoring des exigences à l'exécution, qui est l'objectif de cette thèse, peut être utilisé comme un moyen de vérification assurant une détection des non-conformités aux exigences (événement indésirable, déviation, problème inattendu, dysfonctionnement, erreur grave, etc.) échappées à la vérification et la validation pendant le cycle de développement, et une protection supplémentaire contre les principaux défis des systèmes auto-adaptatifs, notamment l'imprévisibilité qui découle de leur complexité et leur variabilité, les changements fréquents dans l'environnement d'exécution, ainsi que la transparence et la sûreté de leur fonctionnement.

Un système de monitoring à l'exécution a pour but de déterminer si l'exécution en cours d'un système préserve les propriétés définies dans une spécification. Il a été longtemps utilisé pour le profilage, l'analyse des performances, et le débogage, pour n'en citer que quelques-uns. Cependant, une limitation majeure de ces types de systèmes de monitoring est le manque de profondeur dans l'information qu'ils fournissent. En effet, avec les techniques de profilage et le débogage, il est difficile de tracer les différentes interactions entre les composants du système en cas de défaillance (Gu et al., 1994), d'autant plus que ces techniques ne permettent pas de représenter des concepts abstraits tels

que les exigences et les objectifs dans les modèles conceptuels qu'elles utilisent (Robinson, 2 010). Alternativement, le monitoring des exigences étend l'idée de l'analyse de performances et des événements en se basant sur une vue abstraite de l'exécution du système qui interprète les événements logiciels de bas niveau comme contribuant à la satisfaction ou à la violation des exigences spécifiées. Cela permet de vérifier en continue le comportement du système par rapport à ses exigences.

De manière générale, le monitoring des exigences à l'exécution est devenu une pratique courante que ce soit pour la vérification ou la validation (V &V) :

- La vérification, qui est « la confirmation par examen et apport de preuves tangibles que les exigences spécifiées ont été satisfaites » (Source : ISO/CEI 12207 : 1995), a pour but de déterminer si le logiciel a été construit correctement et qu'il fonctionne conformément aux exigences spécifiées. Les tests de vérification sont réalisés généralement avant le déploiement du système, cependant ces pratiques émanant des schémas linéaires classiques de **conception --> implémentation --> test** commencent à changer. Aujourd'hui, le développement logiciel nécessite de plus en plus de réactivités pour répondre aux changements fréquents des exigences et des contraintes de l'environnement externe. Cela a conduit à un cycle de développement incrémental qui tire profit des techniques de vérifications formelles des démarches agiles (Larman, 2004) telles que l'intégration continue, jeu de planning et les tests de recette. Les tests peuvent alors être étendus en temps réel, pour détecter des non-conformités qui entérinent le non-respect des exigences internes dues à l'implémentation du système ou aux changements dans l'environnement d'exécution. Lorsque la vérification porte sur un modèle du système et ne requière pas des exécutions du système réel, on parle de vérification statique. Ce type de vérification s'appuie généralement sur une analyse statique du code binaire, avec notamment la revue de code, estimation de complexité et les techniques de couverture du graphe de contrôle, ou encore sur une vérification formelle qui permet de s'assurer du bon comportement du système ou vérifier ses propriétés temporelles. En revanche, si la vérification nécessite l'exécution du système, on parle de *vérification en ligne*. Il s'agit de vérifier une propriété donnée avec des traces d'exécutions ou en exécutant simultanément le système avec un jeu des données d'entrées, et de comparer les données de sorties avec le résultat attendu. Une erreur est alors détectée lorsque les données de sorties diffèrent des données attendues. Finalement, bien que la vérification permet la détection des erreurs de fonctionnement, un zéro défaut n'implique en aucun cas que le système est correct. Une phase de validation est alors nécessaire pour valider le système.
- La validation, qui est « la confirmation par examen et apport de preuves tangibles que les exigences particulières pour un usage spécifique prévu sont satisfaisantes. » (Source : ISO/CEI 12207 : 1995), est plutôt orientée utilisation, il détermine si le système construit satisfait les exigences des utilisateurs finaux. Le déroulement des tests dans un environnement simulé avec des données simulées contribue à valider partiellement les exigences des utilisateurs. En effet, certaines défaillances peuvent échapper à ces tests de validation, en particulier pour les systèmes auto-adaptatifs qui sont en interaction continue

avec l'environnement externe. D'où le besoin de valider en continue le comportement du système contre les exigences utilisateurs, qui permet aux développeurs et aux utilisateurs d'avoir continuellement des retours d'informations sur la façon dont un système contribue à la satisfaction des exigences des utilisateurs. Ces informations peuvent être exploitées pour engager une démarche d'amélioration continue du système. La validation à l'exécution (y compris le monitoring des exigences à l'exécution) peut devenir alors un catalyseur essentiel pour que les méthodes modernes de développement atteignent leur plein potentiel (Robinson, 2010).
Finalement, des exemples des cas réels des outils de monitoring des exigences sont GLIMPSE, Orchids, ANNA CSS et JRTM :

GLIMPSE[1] est un outil de monitoring d'infrastructure qui a été conçu pour répondre aux exigences d'adaptation et pour la prise en compte du contexte des systèmes distribués modernes opérant dans un environnement en constante évolution. Glimpse est basé sur le modèle de communication publication/abonnement. Il adopte une approche basée sur les modèles (MDA), où les propriétés à surveiller sont conformes au méta-modèle eCore. Ces propriétés sont définies par les consommateurs dans leur propre langue et sont ensuite traduites dans un langage d'événements complexe, comme Drools Fusion2, à l'aide des transformations standard pilotées par des modèles. L'instrumentation de Glimpse recueille des informations sur l'infrastructure et les services du système à l'aide des sondes, qui sont réalisées par injection du code dans le système cible ou en utilisant des proxys. Ces informations représentent les événements primitifs qui sont traités par un CEP pour déduire des événements complexes correspondant aux demandes des consommateurs. Plusieurs moteurs de règles peuvent être utilisés pour cette tâche notamment Drools Fusion ou RuleML[3].
BEEPBEEP[4] est un outil de monitoring pour les applications Web AJAX. Il permet de vérifier la conformité de leur communication avec les contrats d'utilisation des services Web (leurs interfaces) exprimées dans LTL-FO+, une extension du premier ordre de la logique temporelle linéaire LTL. La spécification des exigences définit des propriétés de sûreté liées au classement des messages, aux valeurs de données dans un ou plusieurs messages ou à une combinaison des deux. L'utilisation de BeepBeep empêche ainsi une application d'envoyer des messages erronés ou des appels de méthode, économisant ainsi la bande passante et le temps de traitement du serveur.

Orchids[5] est un outil de détection d'intrusion en temps réel, capable de rechercher dans un flux d'événements des modèles d'attaque, qui sont spécifiés dans une variante de la logique temporelle linéaire. La spécification est compilée en automates non déterministes pour une correspondance rapide

[1] une première version de Glimpse est disponible à : http://labsewiki.isti.cnr.it/labse/tools/cpmm/public/main
[2] http://www.jboss.org/drools/drools-fusion.html
[3] http://ruleml.org.
[4] http://beepbeep.sourceforge.net
[5] http://www.lsv.fr/Software/orchids/

des modèles. L'outil traite des problèmes majeurs tels que la corrélation d'événements, l'optimisation des requêtes, la désynchronisation et précision des horloges.

ANNA CSS (Anna Consistency Checking System) est outil de vérification du comportement des programmes en Ada, en temps réel. L'outil prend en entrée une spécification formelle en langage Anna (Annotated Ada), qui une extension du langage Ada conçue pour répondre au besoin d'augmenter Ada avec des annotations de sorte que des méthodes formelles bien établies de vérification et de diagnostic puissent être appliquées aux programmes Ada, et la transforme en un programme Ada instrumenté avec des fonctions de vérification et des appels à celles-ci. Cette version du programme Ada/Anna est considérée comme un programme d'« autovérification » Ada. Ce programme appelle le code de vérification de cohérence aux points du programme où la spécification peut être violée. L'outil fournit des messages de diagnostic lorsqu'un comportement incohérent est détecté. Le message inclut l'annotation violée et le texte source ada dont l'exécution ne respecte pas les annotations.

JRTM : (Java Runtime Timing Constraint Monitor) est un outil de monitoring de la conformité des systèmes Java distribués à des contraintes temporelles spécifiées en logique temporelle RTL (logique temps réel). Ces propriétés sont exprimées sous forme des assertions Java qui définissent les relations entre les différents événements du système cible pendant son exécution. Des messages d'occurrence de ces événements sont signalés par le système cible au moyen de déclencheurs des événements qui sont insérés dans le programme cible par les programmeurs java. L'outil conserve ces messages dans une file d'attente triée, à un moment approprié, il vérifie le message d'événement en tête contre les contraintes qui y sont associées à l'aide de graphe de contraintes. Si une violation de la spécification est détectée, les utilisateurs sont avertis et des exceptions prédéfinies peuvent être exécutées pour appeler des actions de récupération.

1.2. Ligne de produits logiciels dynamique (LdPD)

Selon le Software Engineering Institute[6], une ligne de produits logiciels (LdP) est :

"A set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way".

[6] www. sei.cmu.edu/productlines

Un LdP représente alors un ensemble de systèmes décrits par une structure (ex : un modèle de caractéristique) incluant deux parties principales :

- **Points communs** : sont les artefacts et les exigences communs à tous les produits de la ligne de produits.

- **Points de variation** : sont tous les aspects communs à certains produits de la gamme de produits, mais pas à tous. La modélisation et la gestion de ces variantes sont devenues alors les moteurs de l'ingénierie de la ligne de produits logiciels.

Le but est de passer d'une production d'un système unique à une production d'une gamme de produits via une compréhension intégrée de l'ensemble de ces produits permettant une personnalisation de masse, une réutilisation à grande échelle et une réduction des coûts et des délais de commercialisation.

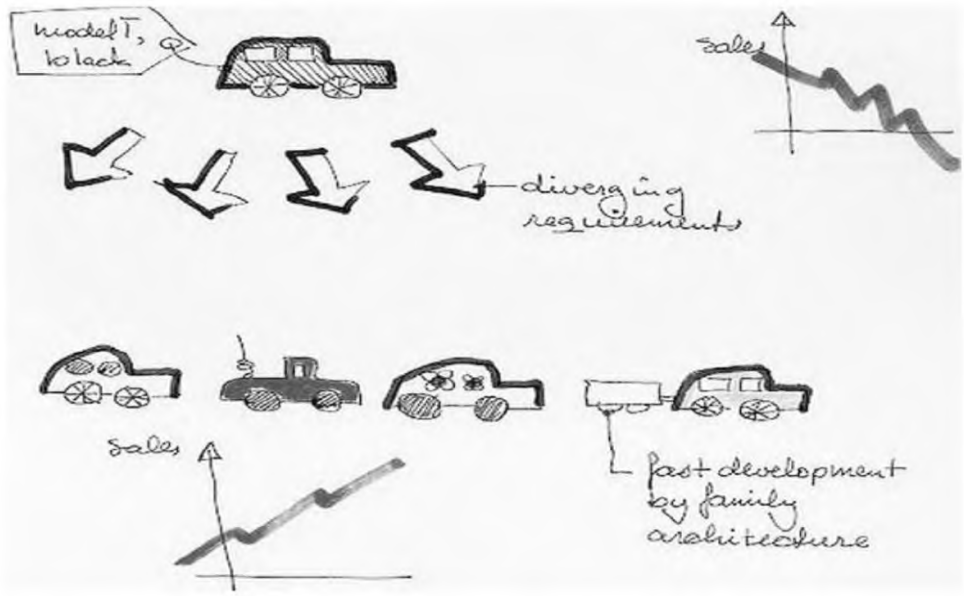


Figure 1 : Produit unique pour tous Vs produits individuels (Pohl, Böckle, & van Der Linden, 2005)

Un exemple de la personnalisation de masse est donné par la Figure 1. En effet, les quatre voitures au bas de la Figure 1 sont les différentes voitures qui répondent aux besoins de différents clients, chacune étant produite en grand nombre.

Traditionnellement, l'ingénierie des lignes de produits logiciels prend en charge la sélection des variations présentes dans un produit spécifique au cours du cycle de développement - de la conception à l'initialisation du système.

Cependant, compte tenu du besoin d'une plus grande capacité d'adaptation pour face à l'incertitude des environnements actuels, il est devenu indispensable de considérer ces variations au moment de l'exécution. D'où l'apparition des lignes de produits logiciels dynamiques (LdPD), qui répondent à ce besoin.

Alors que l'ingénierie des lignes de produits logiciels se concentre sur une compréhension du marché qui conduit l'analyse de la variabilité dans une LdP, une LdPD est conçue pour s'adapter aux variations des exigences individuelles et des contraintes de ressources en retardant la sélection des variantes au moment de l'exécution. Cette adaptation peut être réalisée par une intervention humaine, ou automatiquement comme c'est le cas des systèmes auto-adaptatifs.

1.3. Modélisation Orientée But

Les méthodes de modélisation traditionnelles, comme UML, conviennent bien pour décrire comment le système futur doit être construit mais ne permettent pas d'explorer des questions comme "Pourquoi le logiciel est nécessaire" et "Pourquoi une alternative de conception est justifiée". Cela conduit souvent à des investissements dans des systèmes qui ne sont pas correctement justifiés. En considérant que ces méthodes sont trop opérationnelles et mal adaptées pour décrire le problème à résoudre, la communauté de l'ingénierie des exigences s'est intéressée à la question de « pourquoi » des systèmes. Elle a ainsi reconnu que les buts tiennent un rôle de premier plan pour la validation de la complétude et de la cohérence des exigences qui leur sont associées. Une telle reconnaissance a conduit à une série de recherches sur la modélisation des buts, la spécification des buts et le raisonnement basé sur les buts à des fins multiples telles que l'élaboration des exigences, le monitoring et la vérification ou la gestion des conflits. Ainsi deux méthodes de modélisation orientées but fut leur apparition : KAOS (Knowledge Acquisition in autOmated Specification) de Van Lamsweerde (1991) et I * de Yu (1995) qui sont devenues des références dans le domaine de l'ingénierie des exigences.

KAOS : L'intérêt de KAOS est qu'il propose un langage d'acquisition du modèle d'exigences permettant de formaliser plusieurs types d'exigences fonctionnelles et non-fonctionnelles incluant la performance, la sûreté de fonctionnement et la sécurité d'une façon naturelle et précise ; il propose également un formalisme pour automatiser les preuves formelles, et le raisonnement autour des buts. En termes de méthodologie, KAOS a trois composants :

- Un modèle conceptuel « méta-modèle » pour l'acquisition et la structuration du modèle d'exigences, avec un langage d'acquisition associé. Le méta-modèle définit les concepts utilisés dans KAOS, tels que les buts, les entités, les agents, les contraintes, les relations et le type d'associations autorisées entre les concepts. Par exemple, un agent peut être associé à un objectif via un lien de responsabilité.
- Un ensemble de stratégies d'acquisition pour l'élaboration du modèle d'exigences. Ces stratégies définissent les étapes à suivre pour acquérir le modèle d'exigences à partir du système composite, qui sont des instances du méta-modèle conceptuel.
- Un assistant d'acquisition qui fournit un support automatisé pour guider le processus d'acquisition en fonction de ces stratégies.

KAOS définit un but comme « un objectif non opérationnel à atteindre par le système composite » (Dardenne et al., 1993). Le terme « non opérationnel » signifie que le but ne peut pas être atteint par des objets et des actions disponibles pour un agent. Ces buts sont spécifiés en utilisant un ET / OU raffinement des buts en sous-buts buts en se basant sur les exigences qui spécifient de quelle façon un but devrait être atteint. Le processus de raffinement est illustré graphiquement par la Figure 2. Le but A représente le but ultime du système cible, qui est ET raffiné en trois buts : le but B, le but C et le but D. Le but B est ensuite OU raffiné en deux buts, à savoir le but E et le but F.

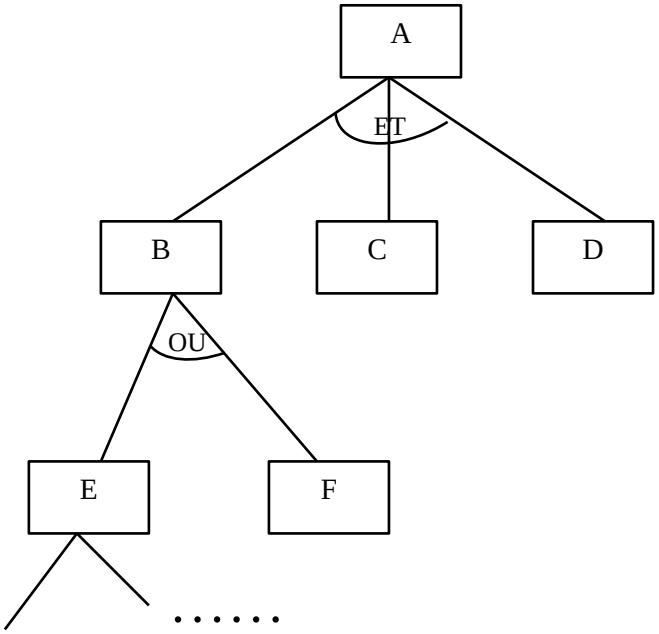


Figure 2: Modèle de buts montrant le AND/OR raffinement des buts

Le processus de raffinement est guidé généralement par deux questions : « pourquoi ? » et « comment ? ». Les buts des niveaux supérieurs sont identifiés en posant la question « pourquoi les parties prenantes veulent atteindre un but » sur l'ensemble des buts disponibles. Alors que les buts qui sont plus concrets sont identifiés en posant la question « comment un but peut être atteint ? ». Par exemple, poser la question « comment assurer la transmission des données dans un réseau ? » donne les deux buts WIFI et Bluetooth. Ces buts identifiés doivent être atteints grâce à la coopération de divers agents, notamment des composants logiciels, des périphériques externes ou des utilisateurs. Chaque agent se voit attribuer une responsabilité qui limite son comportement pour assurer son objectif. L'attribution de responsabilité est un critère d'arrêt pour le processus de raffinement.

Pour spécifier formellement les buts, des opérateurs de la logique temporelle sont proposés qui correspondent à quatre patterns temporels de buts, notamment Atteindre, Maintenir, Éviter et Cesser. « Atteindre » nécessite que certaines propriétés soient conservées à un moment donné dans l'avenir.

« Cesser » exige qu'une propriété ne soit plus conservée à l'avenir. « Maintenir » nécessite qu'une propriété doive toujours tenir. Enfin, « Éviter » implique qu'une propriété ne doit jamais tenir. Cela permettra à l'analyste de spécifier des propriétés impliquant des délais en temps réel.

I* : est un Framework qui met l'accent sur les relations intentionnelles entre acteurs stratégiques. Les acteurs ont la liberté d'action mais opèrent au sein d'un réseau de relations sociales. En particulier, ils dépendent les uns des autres pour la réalisation des objectifs, des tâches à exécuter et des ressources à fournir. Ces dépendances sont intentionnelles dans la mesure où elles sont basées sur des concepts sous-jacents tels que but, capacité, engagement, conviction, etc. Les acteurs sont stratégiques en ce qu'ils évaluent leurs relations sociales en termes d'opportunités qu'ils offrent et de vulnérabilités qu'ils peuvent apporter. Les acteurs stratégiques cherchent à protéger ou à promouvoir leurs intérêts.

Deux types de buts sont définis dans I* : les buts concrets, communément appelés buts, et des soft goals. Un but est défini comme « une condition ou une situation dans le monde que les parties prenantes souhaitent atteindre ». Un soft goal est comme un attribut de qualité sur l'un des autres éléments intentionnels, il est similaire à la notion de but, sauf que les critères de satisfaction dont ils dépendent ne sont pas précis et impliquent souvent un compromis avec d'autres soft goals. Cela nous ramène au terme « satisficing » inventé par Hebert Simon pour désigner le manque de précision dans la perception de la satisfaction.

Il propose ainsi deux types de modèles : le modèle de dépendance stratégique (SD) et le modèle de motivation stratégique (SR). Le modèle SD décrit le contexte de l'environnement organisationnel du système en tant que réseau de relations de dépendance entre les acteurs. Ce réseau consiste en un ensemble de nœuds et d'arcs, chaque nœud représente un acteur et chaque arc représentant une dépendance entre deux acteurs. Le modèle (SR) exprime le contexte organisationnel ainsi que les relations internes entre les éléments intentionnels qui peuvent être des buts, des tâches, des soft goals ou des ressources ou des croyances (les hypothèses sur l'environnement). Les motivations sont modélisées à l'aide de relations Means-ends, de décompositions de tâches et de contributions (« break » (--), « hurt » (-), « help » (+), « make » (++)) des soft goals.

D'autre part, d'autres méthodologies de modélisation orientée but sont utilisées telles que TROPOS et REFAS :

TROPOS (Bresciani et al., 2004) est une approche orientée agent, qui étend le Framework i* pour définir les obligations des acteurs « *dependees* » envers d'autres acteurs « *dependers* ». Ainsi le modèle DS (diagrammes d'acteurs sous Tropos) décrit le réseau d'interdépendances entre acteurs, et le modèle RS (diagrammes de buts sous Tropos) décrit et conduit l'analyse des délégations faites par chaque acteur à d'autres acteurs pour atteindre ses buts.

REFAS (Muñoz-Fernández et al., 2015) est un langage de modélisation orienté but qui étend le langage proposé par Sawyer et al. (2012) pour soutenir la spécification des systèmes auto-adaptatifs. REFAS fournit cinq vues pour représenter les différents aspects liés aux exigences telles que le but, le soft goal, opérations ainsi que les hypothèses sur l'environnement. *La vue de variabilité* représente les exigences fonctionnelles sous la forme d'un arbre ET/OU de décomposition de buts en sous-buts en capturant des liens hiérarchiques entre buts et les contraintes entre eux. Le processus de raffinement se termine lorsque tous les buts « feuilles » deviennent opérationnels. *La vue soft goals* représente les exigences non fonctionnelles en tant que soft goals qui sont liés par des relations de contribution. *La vue de contexte* définit les variables de l'environnement et les variables internes au système. *La vue assets* représente les assets du système telles que les composants logiciels, et leurs associations avec la vue de variabilité. Finalement, *la vue de satisfaction des soft goals* permet le raisonnement sur les hypothèses d'environnement et leur implication sur la satisfaction des soft goals par le biais de « soft-dépendances » et « claims ».

1.4. Instrumentation

L'instrumentation est une technique permettant au système de monitoring de rassembler des informations sur le système surveillé par insertion des probes. Cela peut être un bout de code intégré dans le programme sans nécessiter la modification du code source original ou un dispositif matériel.

Nous retenons la définition de la notion d'instrumentation par l'ISO/IEC/IEEE 24765 (2010) : " Devices or instructions installed or inserted into hardware or software to monitor the operation of a system or component".

Le but de l'instrumentation est la maitrise des exécutions du système en exploitant les données collectées. Ces données sont les points clés pour une meilleure compréhension du système instrumenté et la gestion de celui-ci.

L'instrumentation peut être de nature intrusive ou non intrusive : (i) Une instrumentation L'instrumentation intrusive implique une modification du code du système cible pour enregistrer les appels aux composants et la durée de leurs appels. (ii) Une instrumentation non intrusive signifie qu'aucune modification du code du système n'est utilisée pour l'instrumentation. Cela passe généralement par une instrumentation de l'environnement lui-même pour l'interception des appels aux composants.

1.5. MAPE-K loop

La théorie de contrôle fournit aux concepteurs des méthodes et techniques pour les systèmes informatiques. Le paradigme de base du contrôle est la rétroaction. Une boucle de rétroaction définit les relations et l'interdépendance entre deux composants ou plus qui traduisent le changement d'état d'un élément qui affecte un autre. Cette dynamique capturée par les boucles de rétroaction joue un rôle fondamental dans la gestion de l'incertitude et de la dynamique croissantes des systèmes logiciels.

Les boucles MAPE-K ont été largement reconnues comme une approche efficace pour la construction de systèmes auto-adaptatifs (Brun, et al., 2009), car elles garantissent le paradigme de l'auto-adaptation, dont le principe est la séparation des problèmes entre le système géré et la réaction.

Introduit pour la première fois par IBM dans son livre blanc, une boucle MAPE-K (en anglais, Monitor, Analyse, Plan, Execute - Knowledge) est une boucle de rétroaction qui est considérée comme le modèle de référence le plus influent en informatique autonome.

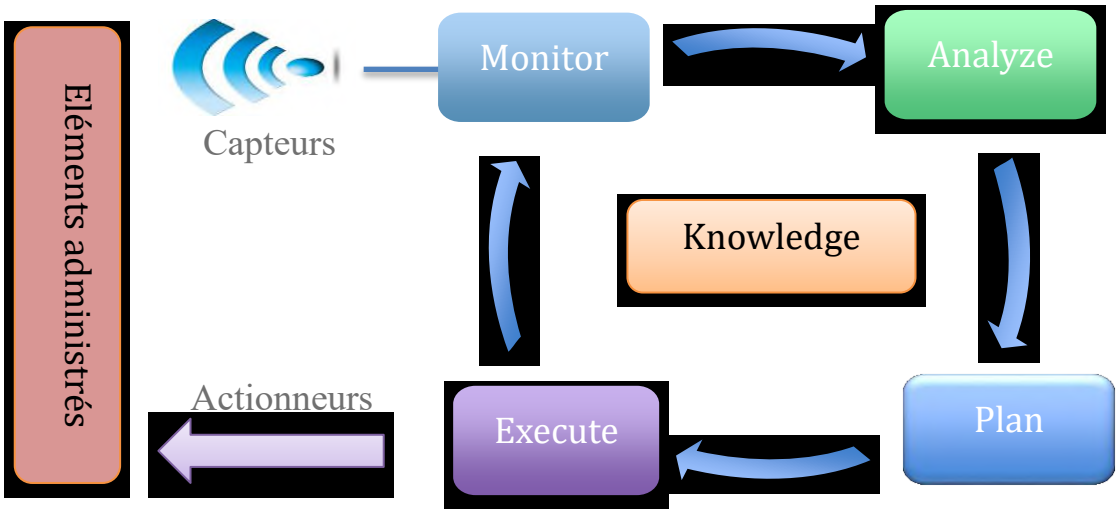


Figure 3 : le principe de la boucle MAPE-K

Elle est conçue comme une séquence de quatre phases (voir Figure 3) : Monitor (M) : recueille des informations à partir du système géré et de son contexte (ou de l'environnement externe) au moyen de capteurs (ou sondes). Analyze (A) : analyse les données pour décider si une adaptation est nécessaire ou non. Plan (P) : décide de la meilleure stratégie d'adaptation pour atteindre un état souhaitable. Execute (E) : exécute la décision via des effecteurs (ou des actionneurs). Toutes ces phases peuvent être complétées par des connaissances supplémentaires sur le système qui ont été mises à jour au moment de l'exécution.

1.6. Evolution

L'évolution est la caractéristique d'un système logiciel capable de répondre à l'évolution de ses exigences et de son environnement d'exécution. Traditionnellement, le problème de l'évolution est abordé dans le cadre de la maintenance des logiciels et traité comme un phénomène post-implémentation pour faire face aux anomalies du système considéré (Mens, 2008). Cependant, la communauté de l'ingénierie logicielle a généralement reconnu que les logiciels implémentant des applications réelles doivent évoluer en permanence en fonction des changements.

Les changements peuvent être traités selon le degré de leur prévisibilité. En effet, certains chercheurs comme McManus & Hastings (2005), de Rocquigny et al. (2008) et Andersson et al. (2009) distinguent entre un changement « inconnu connu » et changement « inconnu inconnu » :

- **Les changements « inconnus connus » (**en anglais, « foreseeable » ou « unknown known ») sont associés en général à des risques qui sont prévisibles mais qui ne sont pas totalement maîtrisable. Par exemple la fiabilité de certains équipements. Ces types d'incertitude peuvent être encadrés contrairement à un changement imprévisible, du fait qu'ils peuvent être reconnus, offrant ainsi la possibilité de les atténuer au moment de l'exécution.

- **Les changements « inconnus inconnus »** (en anglais, « unforeseen » ou « unknown unknown »), ils sont associés à des sources qui sont difficiles à prédire. Il peut s'agir d'un changement radical des exigences du marché suite à une chute d'une compagnie ou de nouveaux types de menaces. Ce type de changements peuvent être capturés par le monitoring en ligne et enregistrés, par exemple dans une base de données, pour une résolution manuelle dans une phase de maintenances mais ne peuvent être résolus automatiquement par des mécanismes en ligne.

Autrement dit, le logiciel en constante évolution ne remplira pas ses exigences et deviendra donc obsolète plus tôt que prévu s'il ne gère pas les sources de changement possibles. Cette conscience a contribué à l'apparition de plusieurs paradigmes traitant l'évolution des logiciels d'une manière itérative, incrémentale ou révolutionnaire tout au long de leurs cycles de vie.

1.7. Conclusion du chapitre

Dans ce chapitre nous avons fourni une introduction à des termes fréquemment utilisés dans la thèse et qui sont liés aux exigences et au monitoring et aux systèmes auto-adaptatifs.

Chapitre 2

2. APPROCHES DU MONITORING DES EXIGENCES - REVUE SYSTEMATIQUE

La notion de monitoring des exigences a été abordée par plusieurs communautés avec des connotations souvent différentes. Cette section dresse un état de l'art multidisciplinaire des notions, méthodes et techniques relatives au monitoring des exigences à l'exécution. Nous portons ici une attention particulière à la définition d'une terminologie uniforme (taxonomie).

Pour ce faire, une revue de littérature a été menée en suivant les lignes directrices des revues systématiques de Kitchenham et al., (2009). La revue vise à répondre aux cinq questions suivantes :

- QR1 : Comment le monitoring des exigences à l'exécution peut être défini et caractérisé ?
- QR2 : quels sont les outils supportant l'analyse des exigences à l'exécution ?
- QR3 : Quels sont les domaines d'application des approches identifiées de monitoring des exigences à l'exécution ?
- QR4 : Quelles sont les méthodes de spécification qui ont été bien appliquées et ont été sujettes à une évaluation rigoureuse ?
- QR5 : Quels sont les points faibles et les points d'amélioration possibles dans la recherche sur le monitoring des exigences à l'exécution ?

L'objectif de la QR1 est d'identifier les différents éléments qui caractérisent le monitoring des exigences à l'exécution, dans le but de pouvoir classifier les approches identifiées selon un vocabulaire commun (définition d'une taxonomie). La QR2 vise à identifier les différents outils pouvant automatiser l'analyse des exigences à l'exécution et garantissant ainsi la validité des résultats. La QR3 vise à identifier les champs d'applications des approches de monitoring existantes. L'objectif de la QR4 est de comprendre comment les méthodes de modélisation et de spécification identifiées ont été évaluées, et quelles sont celles qui ont été bien appliquées selon le score de qualité basé sur le programme d'évaluation critique [7](CASP). Quant à la QR5, le but est d'identifier les lacunes en matière de recherche et les orientations potentielles pour de futures recherches.

Stratégie de la recherche : une recherche automatique a été effectuée pour identifier les articles journaux dans le domaine de monitoring des exigences publiés dans les trois bases électroniques suivantes : IEEE Xplore, ScienceDirect et SpringerLink. En outre, en l'absence des publications

[7] https://casp-uk.net/casp-tools-checklists/

importantes de type conférence que nous connaissions déjà, comme l'article de Fickas et Feather (1995) et celui de Sawyer (2010), une recherche manuelle a été également menée pour ajouter ces publications. Par conséquent, 50 études primaires ont été sélectionnées parmi les 4556 publications de la recherche initiale. Le protocole d'examen et d'extraction des données apparaît dans l'annexe.

Dans la suite de ce chapitre, la section 2.1 présente les éléments de la taxonomie, définie pour guider notre revue systématique. La section 2.2 présente les réponses aux questions de recherche. La section 2.3 analyse le résultat de la revue. La section 2.4 discute de la validité de la recherche. Et finalement une conclusion du chapitre est donnée à la section 2.5

2.1 Taxonomie du monitoring des exigences à l'exécution

La notion de monitoring des exigences a été interprétée de diverses manières en fonction des objectifs visés. Pour présenter un état d'art de manière compréhensible en utilisant les nomenclatures courantes, nous avons utilisé une taxonomie qui intègre les résultats de notre revue systématique.

Le but de la taxonomie est de fournir aux chercheurs et aux praticiens un vocabulaire commun en matière de monitoring des exigences à l'exécution. Ce vocabulaire sera utilisé par la suite pour classifier les approches supportant le monitoring des exigences.

Dimensions de la taxonomie : les éléments de la taxonomie ont été identifiés en suivant le modèle de 5W + 1H (What, Where, When, Why, Who et How) (Jia et al., 2016). Le Tableau 1 montre comment la taxonomie répond aux questions Quoi, Pourquoi, Comment, Quand et Où qui ont guidé l'identification des dimensions de la taxonomie et les éléments de chaque dimension.
Les réponses aux questions dans le Tableau 1 ont été extraites des études primaires et regroupées en fonction de la similarité. Si une réponse n'appartenait pas à une catégorie existante, une nouvelle catégorie a été créée et les réponses aux études précédemment analysées ont été reconsidérées. Les données extraites des études primaires sont présentées dans l'annexe.
La Figure 4 donne un aperçu des dimensions de la taxonomie proposée, à savoir la spécification des exigences, le but du monitoring, l'approche, l'instrumentation et le mode du monitoring. Les éléments constituants chaque dimension sont décrits succinctement ci-après : La section 2.1.1 se focalise sur la spécification des exigences. Elle présente dans un premier temps les classes des exigences qui sont susceptibles d'être spécifiées et surveillées à l'exécution, et détaille ensuite les différentes méthodes de modélisation de ces exigences. La section 2.1.2 liste les buts visés par le monitoring des exigences. La section 2.1.3 détaille les approches existantes pour la mise en œuvre du monitoring des exigences à l'exécution des systèmes cibles. La section 2.1.4 présente les techniques d'instrumentation du monitoringet finalement la section 2.1.5 liste les modes d'exécution du monitoring.

Tableau 1 : Détails de la taxonomie

Dimension	**Éléments de la dimension**	**Description**
Spécification des exigences	Type de propriétés	Que veut – on surveiller ? Quelles sont les propriétés contre lesquelles nous devrions vérifier les exigences en ligne ?
	Méthode de spécification	Quelles sont des méthodes de spécification des propriétés à surveiller ?
But du monitoring		Quel est le but du monitoring des exigences ?
Approche	Implémentation du monitoring	Quels sont les techniques et outils de mise en œuvre du monitoring des exigences à l'exécution ?
	L'emplacement du code du monitoring	Le code du monitoring est-il externe au système cible (communication à distance, par exemple à travers de la socket) ? Ou bien il est intégré au code du système (fonctionnant en même temps que le programme).
	Support d'incertitude	L'approche, supporte-t-elle l'incertitude ? Si oui, quel type d'incertitude ?
	Déclencheur	Indique les éléments qui déclenchent le monitoring des exigences.
Instrumentation	Couches d'abstraction du système	L'instrumentation du monitoring est réalisée dans quelle couche d'abstraction du système ?
Mode de monitoring		Quand le code du monitoring s'exécute ?

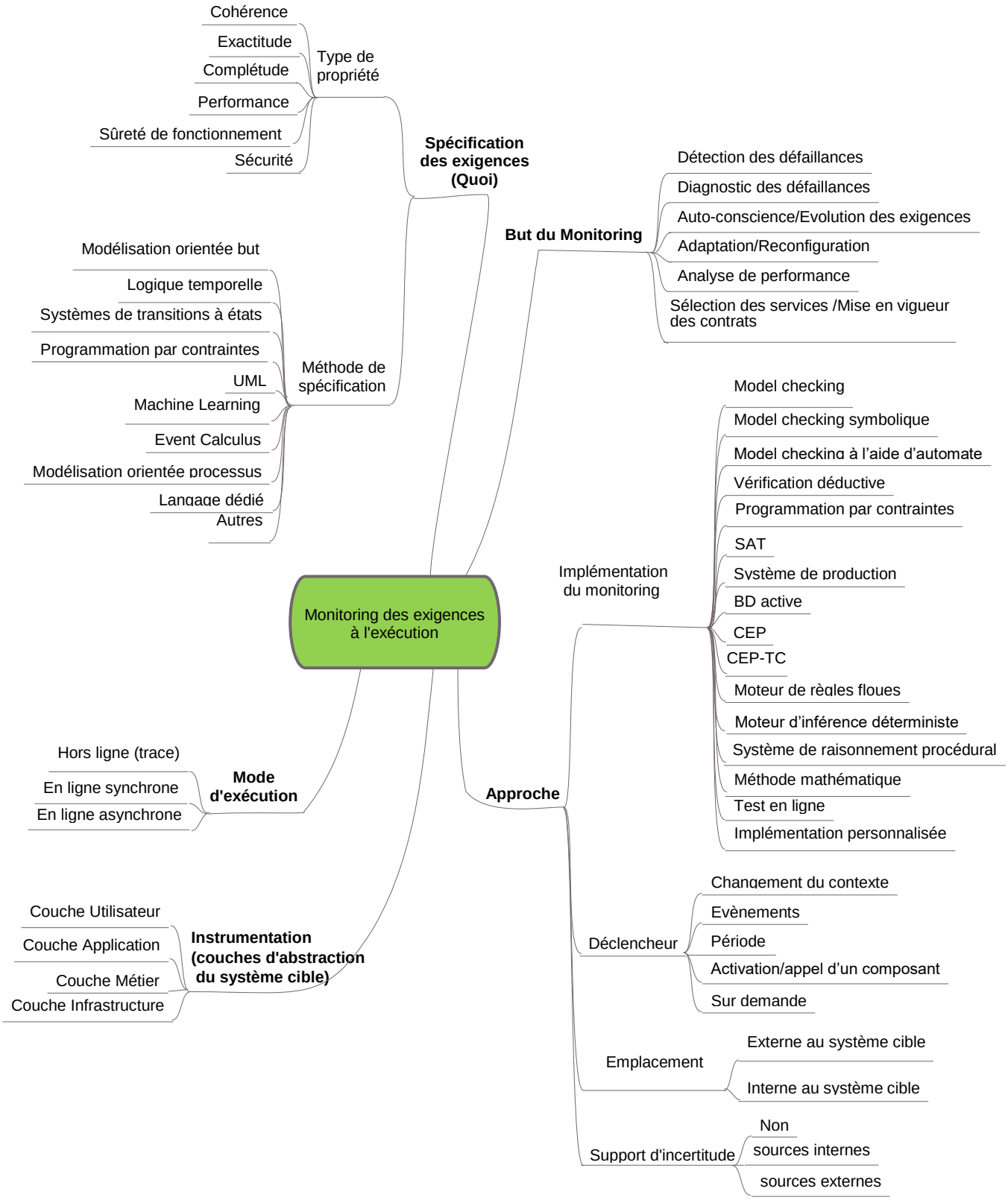


Figure 4 :Taxonomie du monitoring des exigences à l'exécution avec 5 dimensions (la spécification des exigences, l'approche de mise en œuvre, le but du monitoring, le mode d'exécution du monitoring et l'instrumentation).

2.1.1 Spécification des exigences

Cette dimension définit les propriétés à surveiller. Elle contient deux éléments nécessaires pour comparer les approches identifiées : type de propriétés et techniques de spécification.

2.1.1.1 Type de propriétés

Afin de permettre le monitoring des exigences, une analyse des exigences spécifiées est effectuée pour créer une représentation du système qui répondra à ces exigences et qui, dans la mesure du possible, n'implique aucune mise en œuvre spécifique. Il en résulte des propriétés du système mesurables et « monitorables » au moment de l'exécution. Ces propriétés spécifient du point de vue d'analyste et/ou décideur, les caractéristiques fonctionnelles et non-fonctionnelles qu'un système cible doit posséder pour satisfaire les exigences. Généralement, le monitoring concerne des exigences individuelles, souvent non-fonctionnelles comme la performance, la sûreté du fonctionnement et la sécurité. Néanmoins, il peut s'intéresser à des propriétés liées à la cohérence fonctionnelle d'un ensemble d'exigences, notamment la cohérence, l'exactitude et la complétude (en anglais, les 3C : Consistency, Correctness Completeness). En résumé, nous avons recensé 6 types principaux de propriétés qui sont définies comme suit :

Une propriété de performance (comportement dans le temps, utilisation des ressources) indique l'efficacité du système en utilisant les ressources appropriées dans des conditions déterminées et dans un contexte d'utilisation spécifique. Par exemple, le nombre de transactions par seconde ou le temps de rafraîchissement de la mémoire.

Une propriété de sûreté de fonctionnement fournit des critères pour décider si un service est fiable et capable d'éviter des défaillances de service plus fréquentes et plus graves que ce qui est acceptable (Avizienis et al., 2004). La sûreté de fonctionnement englobe généralement des propriétés de qualité comme la disponibilité (les ressources et les services du système sont disponibles aux utilisateurs autorisés), la fiabilité (le système assure la continuité du service), la vivacité (sous certaines conditions, le système se trouvera dans une situation souhaitée) et la maintenabilité (système peut subir des modifications et des réparations).

Une propriété de sécurité indique que quelque chose (mauvais) ne se produit jamais dans une séquence d'événements. La propriété de non-blocage en est un exemple qui stipule que le système ne peut jamais tomber dans une situation où aucun progrès n'est possible. D'autres propriétés de qualité en sont liées comme l'intégrité (absence de modifications non-autorisées par le système) et la confidentialité (absence d'accès non-autorisés par le système).

Une propriété de cohérence (en anglais « consitency ») permet de vérifier si une spécification ne contient pas de contradictions entre les exigences spécifiées (Zowghi & Gervasi, 2003). Il fait également référence à la conformité aux règles et aux contraintes spécifiées.

Une propriété d'exactitude (en anglais « *Correctness »)* permet de savoir si une exigence est bien formée. Notez que des exigences cohérentes peuvent conduire à des solutions incorrectes. D'où l'analyse d'exactitude qui est souvent pragmatique étant liée à la satisfaction des utilisateurs finaux ; elle indique que chaque solution acceptée par les utilisateurs est conforme au modèle des exigences.
Une propriété de complétude (en anglais « *Completeness* ») indique qu'aucune exigence dans le modèle ne manque et donc que chaque solution générée à partir de la spécification est également acceptée par les utilisateurs finaux. Cependant, comme mentionné par Shell (2010), comment pouvons-nous être sûrs que la spécification de l'exigence est complète sans savoir quelle est la spécification de l'exigence complète ? Prouver la complétude absolue est presque impossible, les pratiques en ingénierie se contentent de définir tout ce qui 'peut' être vrai pour prouver d'une complétude relative.

2.1.1.2 Méthodes de spécification

Cet élément décrit les langages de spécification utilisés pour définir les propriétés à vérifier. Certaines approches utilisent un langage spécifique, d'autres adoptent des « langages dédiés » (en anglais, domain specific language ou DSL) qui sont conçus pour répondre aux contraintes d'un domaine d'application précis. Nous avons identifié plusieurs langages dans la littérature qui ont été organisés selon dix classes de méthodes de modélisation des exigences, à savoir modélisation orientée but, logique temporelle, systèmes de transition d'états, Event Calculus, UML, la programmation par contraintes, modélisation orientée processus, machine Learning, langage dédié, et autres. Ces langages sont présentés succinctement comme suit :

A. Modélisation orientée but

Les méthodologies orientées buts sont souvent utilisées pour la spécification des exigences car ils sont plus généralistes et permettent de décrire à la fois les exigences fonctionnelles et non-fonctionnelles. Elles offrent en outre des concepts de haut niveau d'abstraction comme les contraintes structurelles, les notions d'agent et les objets.
Bien que le processus de définition des exigences utilise des concepts similaires dans les différentes méthodologies existantes telles que KAOS, i*/TROPOS et REFAS, à savoir : buts, contraintes/soft goals, actions / tâches/opérations, objets/ressources/assets. La méthodologie de KAOS offre une taxonomie plus riche pour le raisonnement autour des exigences fonctionnelles, alors que les autres méthodologies i*, et TROPOS et REFAS supportent plus les exigences non-fonctionnelles, qui sont représentées par des soft goals. Ces derniers sont considérés comme des facteurs d'évaluation des alternatives (liens Means-ends), permettant aux analystes de justifier les décisions de conception dans le modèle.

B. Systèmes de transition d'états

Les systèmes de transition d'états définissent les exigences fonctionnelles d'un système cible à l'aide d'états et de transition dont les gardes sont des conditions d'environnements d'exécution. Cela inclut les automates finis FSA, FSM et NFA, les variantes de chaines de Markov telles que DTMC et CTMC, et les systèmes de transition d'états étiquetés LTS… etc.

C. Logique temporelle

Les logiques temporelles permettent de spécifier des exigences opérationnelles telles que la disponibilité, la maintenabilité, la fiabilité et la sécurité. Elles sont particulièrement utilisées pour spécifier les exigences dans une perspective de vérification à base de model-checking. Les événements du système sont alors surveillés pour évaluer ces propriétés en fonction des valeurs actuelles des paramètres du système cible.

En principe, la logique temporelle utilise des propositions atomiques P pour faire des déclarations sur une exécution du système représentée par une séquence d'états (Bérard et al., 2013). Ces propositions atomiques (AP) sont combinées par un certain nombre de connecteurs logiques, dont les connecteurs usuels : *et*, *ou*, *non*, *implication*, ainsi que d'autres opérateurs temporels telles que X (« neXt » désigne immédiatement après), G (désigne toujours), U (désigne jusqu'à). Par exemple la logique temporelle linéaire LTL est construite à partir d'un ensemble AP et les opérateurs logiques ¬ et ∨, ainsi que les opérateurs temporels X et U. D'autres langages sont également utilisés pour le monitoring des exigences à l'exécution telles que PCTL [Ref11], PTLTL [Ref25] [Ref23] et RTL [Ref19].

Synthèse : l'écriture des formules de logiques temporelles s'avère un bon moyen de formalisation des exigences non-fonctionnelles temporelles. Néanmoins, la manipulation de ces formules nécessite un certain niveau d'expertise et des compétences particulières. De plus, ces formalismes ne sont pas appropriés pour exprimer les exigences fonctionnelles du système cible. D'autre part, la complexité des modèles créés dépend étroitement de la formulation du problème, plus la logique utilisée est expressive, plus le problème du model-checking devient difficile.

D. UML

Les approches basées sur UML pour le monitoring des exigences, utilisent principalement les diagrammes de séquence ou le langage OCL pour définir et surveiller les exigences fonctionnelles du système cible.

Bien que le langage UML offre un grand degré d'abstraction permettant la modélisation des exigences, il ne permet pas aux analystes de raisonner sur des alternatives et stratégies d'adaptation. S'ajoute à cela l'ambiguïté inhérente au langage UML associée à sa prise en charge d'une modélisation à perspectives multiples qui pose un grand risque d'incohérence (Elaasar & Briand, 2004).

E. Programmation par contraintes

La programmation par contraintes est reconnue pour une résolution efficace des problèmes combinatoires complexes tels que les tâches de planification, d'ordonnancement et d'attribution

(Wallace, 1996). Un problème de satisfaction de contraintes CSP (en anglais : Constraint satisfaction Problem) est en un triplet (V, D et C) où V = {v1, v2} : un ensemble de variables. D (vi) : un ensemble de valeurs qu'une variable vi peut prendre, appelé domaine de vi. C : un ensemble de contraintes, qui sont des relations entre des variables en V qui restreignent leurs domaines.
Une solution d'un CSP consiste en des affectations (V ← a) de chaque variable qui soient cohérentes avec les contraintes sur ces variables.
L'intérêt de la programmation par contrainte dans la modélisation des exigences est qu'elle offre des outils (solveurs) pour automatiser la résolution des contraintes. Nous avons fait le choix d'utiliser la programmation par contraintes pour le raisonnement sur des exigences spécifiées en langage REFAS. Cela nous a permis de tirer profit de l'expressivité de la modélisation par buts et de la capacité de raisonnement de la programmation par contraintes.

F. Event-Calculus

Event-calculus (Shanahan, 1999) est un langage logique pour représenter et raisonner autour des événements. Il spécifie formellement les propriétés à surveiller avec des assertions logiques qui définissent les événements déclencheurs, l'effet de ces événements, et l'action sur les fluents.

G.Modélisation orientée processus

Les approches de monitoring des exigences utilisant la modélisation orientée processus sont centrées sur les architectures orientées service (SOA). Elles se basent sur des langages comme WS-BPEL [Ref28][Ref4][Ref27] et SLAng SLA[Ref17] pour la détection des violations du contrat de niveau de service (SLA) ou pour la sélection et la décomposition des services. Ces langages peuvent exprimer une gamme de contrats de service SLA à faire respecter aux parties prenantes et servent à décrire l'orchestration des processus métiers du système cible. Ces informations sont souvent utilisées pour générer automatiquement une spécification du système, par exemple à partir des processus BPEL, en plus des exigences requises.

H.Machine learning

Les approches exploitant le Machine Learning pour le monitoring des exigences, tirent profit de sa capacité de prédiction et de traitement d'une grande quantité de donnée. Cela conduit à un monitoring intelligent capable d'identifier et de résoudre les problèmes d'insatisfaction des exigences avant qu'ils se concrétisent. Étonnamment, nous avons identifié qu'une seule étude basée sur les classificateurs à base de règles [Ref37].

I. Langage dédié

Les approches basées sur les langages dédiés utilisent des implémentations personnalisées pour le monitoring des exigences à l'exécution.

J. Autres

La catégorie « Autres » représente l'ensemble des approches utilisant des technologies et des concepts ne figurant dans aucune des catégories présentées ci-dessus, par exemple le matching [Ref12] et la programmation par contrat [Ref11].

2.1.2 But du monitoring

Cette dimension définit les objectifs du monitoring des exigences à l'exécution, notamment l'analyse de performance, la détection des défaillances, le diagnostic des défaillances, la sélection des services et la mise en vigueur des contrats (par exemple les SLA pour les systèmes orientés services), l'adaptation ou reconfiguration du système adaptatif considéré et le raisonnement sur les exigences en temps réel afin de valider les exigences, de permettre la conscience des systèmes en leurs exigences, l'évolution et l'adaptation des exigences en temps réel.

2.1.3 Approche

Cette dimension décrit les approches de mise en œuvre du monitoring pour l'analyse des propriétés spécifiées. Nous décrivons brièvement les différents outils et techniques de mise en œuvre et les aspects spécifiques aux architectures utilisées. Ces aspects incluent l'emplacement du code du monitoring, le déclencheur des activités du moniteur, et le type d'incertitude supportée s'il en est.

2.1.3.1 Implémentation du monitoring

Cet élément décrit les techniques et outils utilisés pour raisonner sur des propriétés spécifiées dans le but de surveiller en continue la validité du système par rapport à ces propriétés.

L'automatisation du raisonnement est généralement basée sur des méthodes mathématiques, logiques, informatiques et formelles. Nous proposons de les classer en 11 catégories (A-K) qui sont décrites ci-après, à savoir le Model-checking, le Model-checking symbolique, le Model-checking à l'aide d'automate, la vérification déductive, la programmation par contraintes, le problème de satisfiabilité (SAT), les systèmes à base de règles, les systèmes de raisonnement procéduraux, les méthodes mathématiques, les méthodes basées sur les tests en ligne, et finalement l'implémentation personnalisée.

- **Model-checking**

Le model-checking est une méthode puissante proposée par Clarke et al. (1999) pour analyser exhaustivement l'évolution d'un système dont l'espace des états est fini. Cela consiste à vérifier si un modèle mathématique du système satisfait ou non une propriété qui est généralement présentée sous la forme d'une formule de la logique temporelle (par LTL ou CTL). L'avantage de cette technique est qu'elle offre une analyse automatique, rapide et sûre des propriétés. Cependant, cette analyse est faite sur un modèle du programme à vérifier.

- **Model-checking symbolique**

Cette méthode vise à représenter le système d'une manière symbolique et compacte par plusieurs techniques pour réduire l'espace du model-checking et éviter ainsi le problème d'explosion combinatoire du model-checking traditionnel. Ainsi, plusieurs approches ont vu le jour telles que le BDD-MC par McMillan (1992) qui propose de représenter le système à l'aide des fonctions booléennes sous forme d'un *diagramme de décision binaire (BDD)* (Bryant, 1986), pour une meilleure manipulation des formules booléennes avec les opérateurs de BDD. Une alternative est le model-checking borné (SAT-BMC : SAT based Bounded Model checking) par Biere et al. (2003) qui proposent d'utiliser SAT au lieu de BDD pour manipuler les formules booléennes. L'idée est de chercher des contre-exemples parmi les exécutions du programme dont la taille est limitée à une longueur bornée K. Si aucun bogue n'est trouvé, on augmente k jusqu'à ce qu'un bogue soit trouvé, le problème devient insoluble ou que la limite supérieure soit atteinte. Le problème BMC peut être efficacement réduit à un problème de satisfiabilité propositionnelle dont l'existence d'une exécution vérifiant une certaine propriété est équivalente à la satisfiabilité d'une certaine formule booléenne. La force de cette technique est qu'il peut tirer profit des progrès faits par les solveurs SAT plutôt que de se baser sur une résolution à l'aide de BDD. Une évolution récente de ces techniques est la possibilité de traiter des variables entières ou réelles en plus des variables booléennes ou plus généralement des prédicats pris dans une théorie. On parle alors de problème de satisfiabilité modulo une théorie SMT qui peut être résolue par des solveurs SMT.

Une autre approche de model-checking symbolique se base sur les *techniques d'abstraction et de raffinement*. Elle consiste à une série du raffinement du modèle symbolique si ce dernier se trouve trop grossier. Cette technique, bien qu'elle soit efficace, elle souffre d'une explosion exponentielle de la quantité de mémoire requise pour le stockage des structures qu'elle manipule.

- **Model-checking à l'aide d'automate**

Cette méthode est une autre vision du model-checking, qui est centrée sur la théorie des automates. Elle consiste à construire le modèle du système et la spécification des propriétés à analyser avec des automates (par exemple l'automate de Büchi ou FSM). L'analyse des propriétés se base alors sur le produit des deux automates.

- **Vérification déductive: logique formelle + assistant de théorème**

La vérification déductive permet de vérifier par déduction des formules logiques, qui représentent le programme cible ainsi que les propriétés à vérifier.

Cette approche se base généralement sur le calcul de la plus faible pré-condition et la logique de Hoare pour donner une représentation purement logique et sémantique à un système. L'analyse des propriétés est alors réduite à une démonstration de théorèmes de cohérence et de complétude. Ces théorèmes permettent de formaliser le comportement attendu (spécification) d'un système et leur démonstration permet d'en affirmer leur vérité.

- **Programmation par contraintes**

Cette méthode transforme le problème de monitoring des exigences à l'exécution en un problème de satisfaction de contraintes qui peut être résolu en ligne à l'aide des solveurs CSP.

- **SAT**

La modélisation par SAT consiste à définir un problème de satisfaisabilité (satisfaction booléenne) qui étant donné une formule de logique propositionnelle, détermine s'il existe une solution qui rend la formule vraie. Cette approche est automatique et sûre car l'analyse est automatisée par des solveurs SAT.

- **Systèmes à base de règles**

Cette catégorie représente des approches qui combinent les techniques des systèmes réactifs et les technologies de traitement des événements. Ces techniques ont fait l'objet d'une étude approfondie au cours des dernières décennies (Paschke & Kozlenkov, 2009). Par conséquent, différents formalismes et langues ont vu le jour et qui ont été utilisés pour l'analyse des exigences des systèmes en ligne, notamment les systèmes de production, les bases de données actives, le traitement des événements complexes (CEP), le traitement d'évènements à base de connaissance temporelle (CEP-TK), les moteurs de règles floues et les moteurs d'inférence déterministes :

- Systèmes de production

Ils ont fait l'objet d'une étude approfondie dans les domaines des systèmes experts depuis les années 1980 et ont été appliqués avec succès dans l'industrie. Il s'agit de règles métiers qui sont décrites sous la forme de règles simples : « *SI condition ALORS action »,* sans la clause SINON. Ces règles sont souvent exécutées par un chainage-avant où les conditions changeantes déclenchent des actions de mise à jour.

- Bases de données actives

Elles visent à combiner les techniques des systèmes experts et des bases de données afin de prendre en charge le déclenchement automatique des réactions en réponse à des événements internes ou externes de la base. Ces événements sont décrits selon la syntaxe des règles supportées par la base. Elles sont dans la plupart du temps de type Event-Condition-Action (Paton et al., 1999).

- Traitement des événements complexes (CEP)

Cette technique combine les techniques de traitement d'événements et des systèmes réactives. Elle permet la détection des événements complexes ou des situations (événement complexe + contexte conditionnel) pour déclencher une réaction locale dans un contexte.

- Traitement d'évènements à base de connaissance temporelle (CEP-TK)

Le CEP-TK (en anglais, Temporal Knowledge based Complexe Event Processing) tire ses origines de la représentation des connaissances (KR) et de la programmation logique (LP). Il définit une approche déclarative et sémantique dont le but d'une formalisation de type axiomes d'action / événement (Paschke & Kozlenkov, 2009). L'accent est mis sur le développement d'axiomes pour formaliser les notions d'actions, d'événements et de causalité, où les événements sont caractérisés en termes de conditions nécessaires et suffisantes pour leur occurrence, et où les événements / actions ont un effet sur les états des connaissances réels.

- Moteur de règles floues

Il représente un système de règles qui sont associées à la logique floue. Cette dernière est basée sur l'utilisation de variables linguistiques pour la représentation de la connaissance. Les valeurs de ces variables sont définies par une fonction d'appartenance plutôt que par un intervalle de valeurs.

- Moteur d'inférence déterministe

Cette technique utilise plutôt des approches à base de règles exprimées dans un langage spécifique à un domaine pour mieux décrire les stratégies.

- **Système de raisonnement procédural**

Un système de raisonnement procédural peut être défini comme un cadre logiciel pour la construction des systèmes de raisonnement en temps réel pouvant effectuer des tâches complexes dans des environnements dynamiques. Il repose sur la notion d'agent rationnel ou d'agent intelligent et est souvent modélisé par l'architecture BDI ou le formalisme croyance – désir – intention (BDI- agent : Belief-desire-intention). Cette méthode est très utilisée pour le monitoring des exigences qui sont exprimées en programmation orientée but (par exemple i*).

- **Méthodes mathématiques**

Ces méthodes proposent de formuler le problème d'analyse de la spécification des propriétés sous forme d'un problème d'optimisation multi-objective en se basant par exemple sur la théorie d'utilité.

- **Test en ligne**

Cette méthode se base sur un générateur de test en ligne pour tester des exécutions du système avec un jeu d'entrées, et par la suite comparer le résultat obtenu avec le résultat attendu afin de s'assurer que le système est valide par rapport aux propriétés spécifiées.

- **Implémentation personnalisée**

Elle se base sur des approches de langages spécifiques à un domaine, un langage personnalisé ou sur des formalismes déjà existants du monitoring tels MOP [Ref25] et REqMon [Ref33].

2.1.3.2 Emplacement du code du monitoring

Cet élément décrit la relation entre le monitoring et le système cible, c.-à-d. l'emplacement du code du monitoring par rapport au programme cible. Le code du monitoring peut être « externe » au système cible [Ref28-Ref36], la communication entre eux se fait alors à distance, par exemple à travers des sockets. Il peut être également intégré « interne » au code du système et donc fonctionne en même temps que le programme cible. Par exemple, dans le cas des systèmes auto-adaptatifs, certaines approches proposent un monitoring intégré dans une MAPE loop [Ref46, Ref38, Ref42] qui fait partie du raisonnement autour des stratégies d'adaptation. Il est à noter que dans le cas de la vérification des programmes, l'emplacement « interne » définit un type de monitoring « inline » [Ref24,Ref23] où le corps du raisonnement du monitoring est inséré dans le code du programme cible.

2.1.3.3 Déclencheur

L'élément déclencheur décrit les mécanismes utilisés pour lancer l'évaluation des exigences. La plupart des approches sont basées sur des événements, c'est-à-dire que l'évaluation des propriétés est déclenchée lorsqu'un événement spécifique se produit [Ref12, Ref15, Ref18, Ref19] ou lors d'un changement de structure ou reconfiguration [Ref8, Ref9]. D'autres approches [Ref10, Ref17] utilisent un déclencheur périodique ou sur demande [Ref14, Ref20], alors que dans certaines approches l'évaluation s'active lors de l'appel d'un composant ou méthode surveillés [Ref11, Ref16, Ref21].

2.1.3.4 Support d'incertitude

Cet élément décrit les différents types d'incertitude qui peuvent être supportés (au moins partiellement) par le monitoring pour faire évoluer le système cible en temps réel. En effet, bien que les avantages des systèmes auto-adaptatifs et dynamiques soient nombreux, leur développement s'est avéré plus difficile que les systèmes logiciels traditionnels. L'une des principales causes est que l'auto-adaptation est sujette à l'incertitude (Cheng et al., 2009). Cette dernière caractérise des facettes du système qui ne sont pas maitrisables complétement ce qui résulte à un comportement imprévisible ou à une performance non-optimale. Il existe de nombreuses taxonomies et classification d'incertitude telles que celles de Ramirez et al. (2009) et Xiao (2009). Nous n'avons pas la prétention de couvrir toutes les causes d'incertitude, nous faisons ici distinction entre **« incertitude interne »** et **« incertitude externe »** selon la classification donnée par Esfahani & Malek (2013) qui classent l'incertitude selon deux types de sources de celle-ci :

- Sources « externes » qui proviennent de l'environnement ou du domaine dans lequel le logiciel est déployé. Par exemple, des données incomplètes ou inconnues concernant la fiabilité des capteurs utilisés pour la collecte des données, ou par rapport à la performance du système qui peut dépendre du comportement des utilisateurs finaux, de la congestion du réseau ou des conditions du chargement des services externes intégrés au système.

- Sources « internes » liées à la difficulté de prévoir toutes les stratégies d'adaptation due à un oubli et/ou à une connaissance imparfaite de toutes les variations des composants du système et des exigences lors de la spécification.

Les techniques de la prise en compte d'incertitude varient donc selon le type d'incertitude et le degré de sa prédictibilité. Certaines approches ont exploité les capacités de la logique floue pour exprimer l'imprécision et la variation de l'état d'une chose : autoRELAX [Ref7] utilise le langage RELAX (Whittle et al. 2009) qui est basé sur la logique floue mais qui se démarque par des opérateurs temporels et modulaires (par exemple, «dès que possible», «autant que possible»). Ces opérateurs ont été utilisés pour relaxer les contraintes qui définissent la véracité d'un Claim, permettant ainsi à un système dynamique DAS (en anglais « Dynamique Adaptive System ») de tolérer l'incertitude externe qui pourrait réfuter à tort un Claim et déclencher des adaptations inutiles. FLAGS [Ref43] (Fuzzy Live Adaptation Goals for Self-Adaptive Systems) introduit la notion de buts flous ou adaptatifs qui sont associés à un ensemble de fonctions d'appartenance pour faire face à l'incertitude liée aux exigences ambiguës, en définissant des niveaux de satisfactions partiels des buts. FLAGS est également basé sur la logique temporelle linéaire (LTL) pour faire face à l'incertitude externe en déclenchant dynamiquement des actions d'adaptation du modèle du but de différentes manières, notamment l'ajout / la suppression / la modification des buts ou des agents. D'autre part, quelques approches ont supporté l'incertitude externe liée à l'environnement d'exécution. Par exemple, Acon [Ref37] et [Ref50] propose une adaptation en continu des exigences contextuelles au moment de l'exécution par un apprentissage automatique des données contextuelles à l'aide des classificateurs base de règles, tandis que [Ref50] propose de définir le contexte sur un ensemble de points de variation du modèle du but et de combiner l'analyse de la satisfaction des buts avec les contraintes de contexte. Finalement, quelques approches telle que celle proposée par Sawyer et al. (2012), ont opté pour la logique des prédicats pour raisonner autour les différents degrés de satisfaction des exigences non-fonctionnelles via des affectations de valeurs comme : « Partiellement satisfait » « totalement refusé » « partiellement refusé » « totalement refusé ».

2.1.4 Instrumentation : couches d'abstraction du système

Cette dimension est liée à l'instrumentation du système logiciel qui peut-être réalisée dans différents niveaux d'abstraction du système cible. Plusieurs Framework ont émergé au cours des dernières décennies, tels que ArchiMate, ITIL et TOGAF, offrant une vision holistique d'un système logiciel d'une entreprise moderne qui comprend plusieurs couches. Les principales couches pertinentes pour le monitoring sont : la couche *utilisateur*, la couche *métie*r, la couche *application* et la couche *infrastructure* (voir Figure 5).

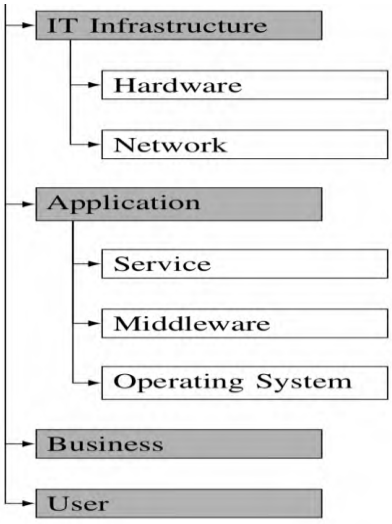


Figure 5: les couches d'abstraction (Faber et al., 2016).

La première couche englobe des informations sur le comportement des utilisateurs et toutes les statistiques relatives à leurs interactions avec le système cible telles que le temps de chargement du système, le temps moyen de visite, la durée de la session. La couche *métier* offre une vue abstraite du système qui englobe les processus métier. Elle fournit des informations sur l'état actuel et les résultats des différentes opérations, processus et transactions. La couche *application* englobe les services, les middlewares, les machines virtuelles et le système d'exploitation. Ces applications offrent généralement des options pour des points d'instrumentation spécifiques permettant d'examiner une multitude de données, telles que la disponibilité et les performances de l'application, le nombre de sessions ouvertes et d'états des transactions. Enfin, au niveau le plus bas de l'application, la couche *infrastructure* comprend les composants matériels du système informatique et les équipements réseaux. L'instrumentation de cette couche est un cas particulier, en effet, toutes les autres couches utilisent une instrumentation logicielle tandis que la couche inférieure utilise l'instrumentation matérielle basée sur des dispositifs spéciaux insérés pour capturer l'état du matériel. Elle fournit des informations sur l'utilisation du processeur ou de la mémoire (la vitesse CPU, la disponibilité et la fiabilité des composants matériels. Ainsi que des métriques décrivant la disponibilité, les délais et l'état de la bande passante dans le réseau informatique.

Pour collecter les données à partir de ces couches, plusieurs mécanismes ont été utilisés dans la littérature. Certaines approches [Ref44, Ref45] ont opté pour l'instrumentation du code source du système cible, tandis que d'autres préfèrent l'utilisation des techniques qui ne nécessitent pas d'intrusion de code source comme l'interception des données et messages envoyés via les bus des données [Ref4] [Ref13] [Ref17] [Ref18] et des logiciels de monitoring dédiés à chaque couche, par

exemple, WMI[8] [Ref11] qui est un système de gestion interne de Windows qui prend en charge la surveillance et le contrôle des ressources systèmes via un ensemble d'interfaces, ou Ganglia[9] [Ref12] qui est utilisé pour capter des statistiques en direct ou enregistrées couvrant des métriques telles que les charges moyennes du processeur ou l'utilisation du réseau pour de nombreux nœuds, ou encore Miscrosoft EIF pour les applications .NET [Ref32] [Ref33].

D'autres approches ont utilisé des sondes [Ref30] ou des capteurs logiciels [Ref7, Ref29, Ref38] pour récupérer des événements et des données d'exécution. Des informations plus détaillées sur les technologies utilisées par les études primaires pour la collecte des données, figurent dans le Tableau A.3 de l'annexe.

2.1.5 Mode de monitoring

Cette dimension indique quand est-ce que le code de monitoring s'exécute. En effet, le monitoring peut être effectué hors ligne [Ref12, Ref13, Ref2, Ref22], ce qui signifie que le code s'exécute après l'exécution du programme cible. Les événements pertinents du système sont enregistrés en tant que trace d'exécution dans un magasin de données permanent. Ces traces sont ensuite transmises au monitoring pour exécuter les tâches programmées telles que la vérification et le diagnostic. Ce type de monitoring est moins intrusif et n'impacte pas la performance du système cible. En revanche, le monitoring « en ligne » effectue une vérification pendant l'exécution du programme cible. L'exécution simultanée du système et du monitoring peut être effectuée de différentes manières : si le programme peut continuer à s'exécuter sans attendre la fin des tâches programmées [Ref12, Ref13, Ref21, Ref22], le monitoring est classé « asynchrone ». Cependant, si le programme doit attendre la fin des tâches programmées [Ref38, Ref42, Ref47], le monitoring est classé « synchrone ».

2.2 Résultats d'analyse des études primaires

Dans cette section, nous apportons des réponses aux questions de recherche présentées dans l'introduction du chapitre 2. Alors que l'extraction des données et les résultats de l'analyse démographique et quantitative sont présentés dans l'annexe.

QR1 : Comment le monitoring des exigences à l'exécution peut être défini et caractérisé ?

Définition 1 : La notion du monitoring des exigences a été introduite pour la première fois par Fickas et al. (1995), pour désigner « *l'insertion de code dans un système en cours d'exécution en vue de recueillir des informations permettant de déterminer si et dans quelle mesure ce système en cours répond à ses exigences* ».

[8] https://docs.microsoft.com/en-us/windows/desktop/wmisdk/about-wmi
[9] http://ganglia.sourceforge.net

Définition 2 : Dans le domaine de la vérification à l'exécution, la notion du monitoring à l'exécution est définie comme : « *un outil qui lit une trace finie et donne un certain verdict* » (Leucker & Schallhart, 2009). Si la trace est considérée en temps réel, on parle de monitoring en ligne.

Il est clair que les deux définitions sont similaires, et que le domaine du monitoring des exigences et celui de la vérification sont étroitement liés. La seule différence est que la vérification à l'exécution s'intéresse plutôt à des modèles de raisonnement par le model-checking et la démonstration de théorème ; mais l'idée générale reste d'utiliser des données collectées du système cible, pour pouvoir analyser en continu la validité du système par rapport aux exigences spécifiées.

Selon Bencomo et al., (2010) et borgida et al. (2013), pour analyser les exigences, le monitoring doit posséder un modèle d'exigences à l'exécution. Ce modèle représente des entités en temps réel qui traduisent les exécutions du système qui sont conformes aux exigences. Ces entités doivent révéler des informations sur elles-mêmes. Nous retenons le terme « models@Runtime » proposé par Bencomo et al. (2010) et noté par la suite par M@RT pour désigner la représentation des exigences à l'exécution.

Un M@RT est construit à partir de la spécification des exigences à surveiller et dépend étroitement de la méthodologie de mise en œuvre du monitoring. À l'exécution, les instances des classes d'exigences dans le M@RT sont créées sur la base des informations collectées du système cible via l'instrumentation. Ces instances stockent les informations sur l'état actuel des exigences et un historique des états passés. Elles seront exploitées en cas de besoin par des mécanismes de raisonnement et de vérification pour atteindre différents buts tels que la détection des défaillances, le diagnostic des défaillances, ou encore pour des buts d'adaptation.

Dans une perspective globale, le monitoring peut être caractérisé par les éléments de la taxonomie définie dans la section 2.1. Cette taxonomie repose sur les concepts clés qui couvrent tous les aspects du monitoring, à savoir, la définition des exigences, les techniques de mise en œuvre du monitoring et l'instrumentation. En effet, certaines approches de monitoring s'appuient sur des langages de spécification et des outils de raisonnement spécifiques, d'autres reposent sur un langage dédié, une implémentation personnalisée ou sur des formalismes déjà existants du monitoring tels que FLEA par Cohen et al. (1997), MOP [Ref25] et REqMon [Ref33].

QR2 : Quels sont les outils supportant l'analyse des exigences à l'exécution ?

L'analyse des études primaires montre que plus que la moitié des approches ne sont pas supportées par un outil d'automatisation. Seulement 22 approches ont été supportées par des solveurs, des interprètes et des model-checkers.

La Figure 6 montre que les Model-checkers sont les outils les plus utilisés pour l'analyse et la vérification des exigences, notamment le model-checker PRISM (4 études sur 8), suivi des solveurs SAT et SMT, qui ont été également beaucoup utilisés. Par ailleurs, quelques études ont opté sur des

formalismes connus, tels que MOP, ExploreMath et ReqMon. Ces formalismes offrent des interfaces de programmation qui simplifie la définition et la génération du code de monitoring.

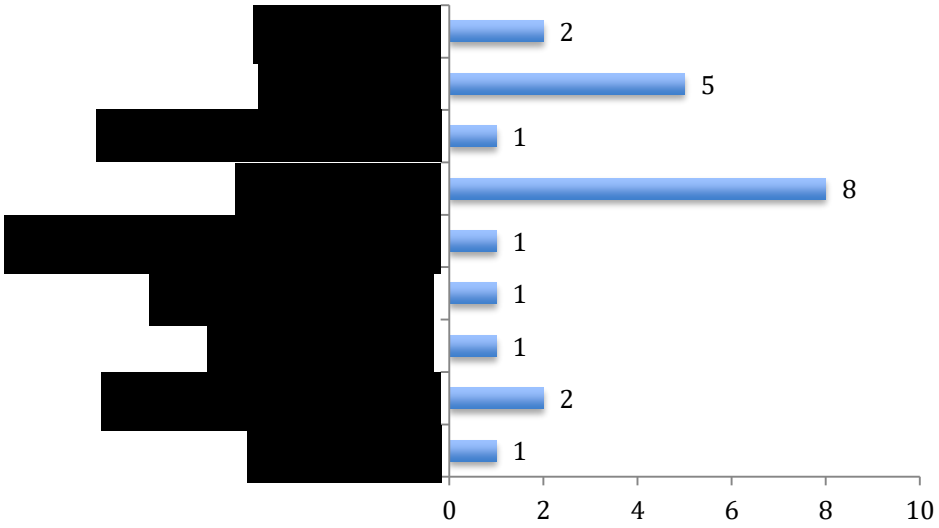


Figure 6 : distribution des études primaires selon l'outil supportant l'automatisation d'analyse des exigences

QR3 : Quels sont les domaines d'application des approches identifiées de monitoring des exigences à l'exécution ?

En raison de la diversité des approches de monitoring. Il n'y a pas à une description exhaustive de leurs domaines applications. En effet, de nombreuses approches ont été développées pour un domaine d'application particulier, tels que les systèmes auto-adaptatifs [Ref3, Ref1, Ref9, Ref29], les attaques malveillantes [Ref44, Ref45], les systèmes embarqués automobiles[Ref34], les systèmes orientés composants [Ref11, Ref21, Ref11] et les systèmes orientés services[Ref4, Ref10, Ref17, Ref35]; voire certaines d'entre elles se sont centrées sur une architecture ou langage de programmation particuliers tels que l'architecture orientée service implémentée par le langage BPEL[Ref27][Ref28], les programmes Java[Ref14, Ref16] ou les programmes SCALA[Ref24]. Tandis que d'autres approches sont plus généralistes et ne sont pas liées à un domaine particulier, par exemple [Ref47, Ref33, Ref24].

La Figure 7 montre que la plupart des domaines d'application sont les systèmes orientés service et les systèmes auto-adaptatifs (9 et 6 études respectivement). Cela est dû à la complexité des environnements dans lesquels ils s'exécutent.

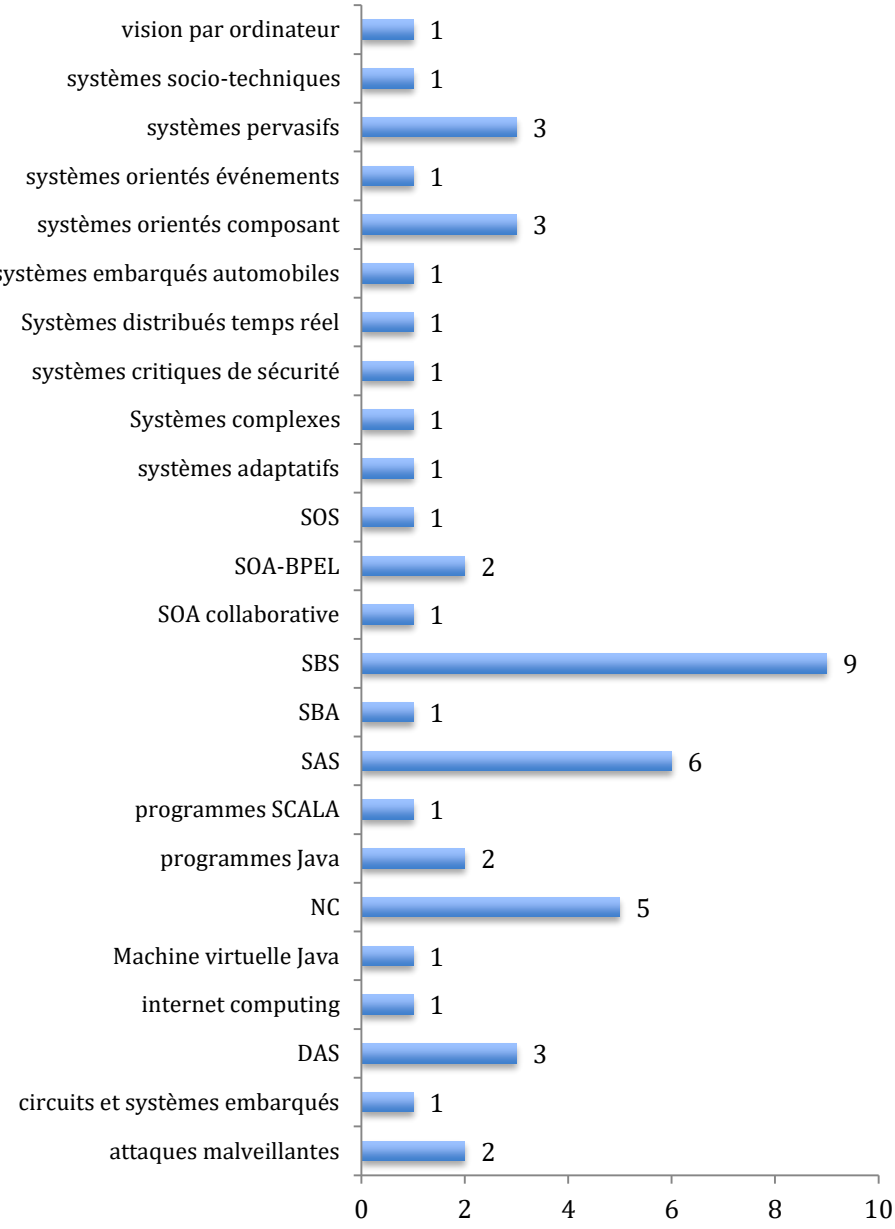


Figure 7 : Distribution des études primaires selon le domaine d'application

D'autre part, une analyse de la corrélation entre le domaine d'application et les techniques de mise en œuvre du monitoring ainsi que les méthodes de spécification des exigences a été réalisée. La Figure 8 révèle qu'il n'y a pas une relation entre les domaines d'application et les techniques de mise en œuvre du monitoring et que la plupart des approches sont basées sur des implémentations personnalisées.

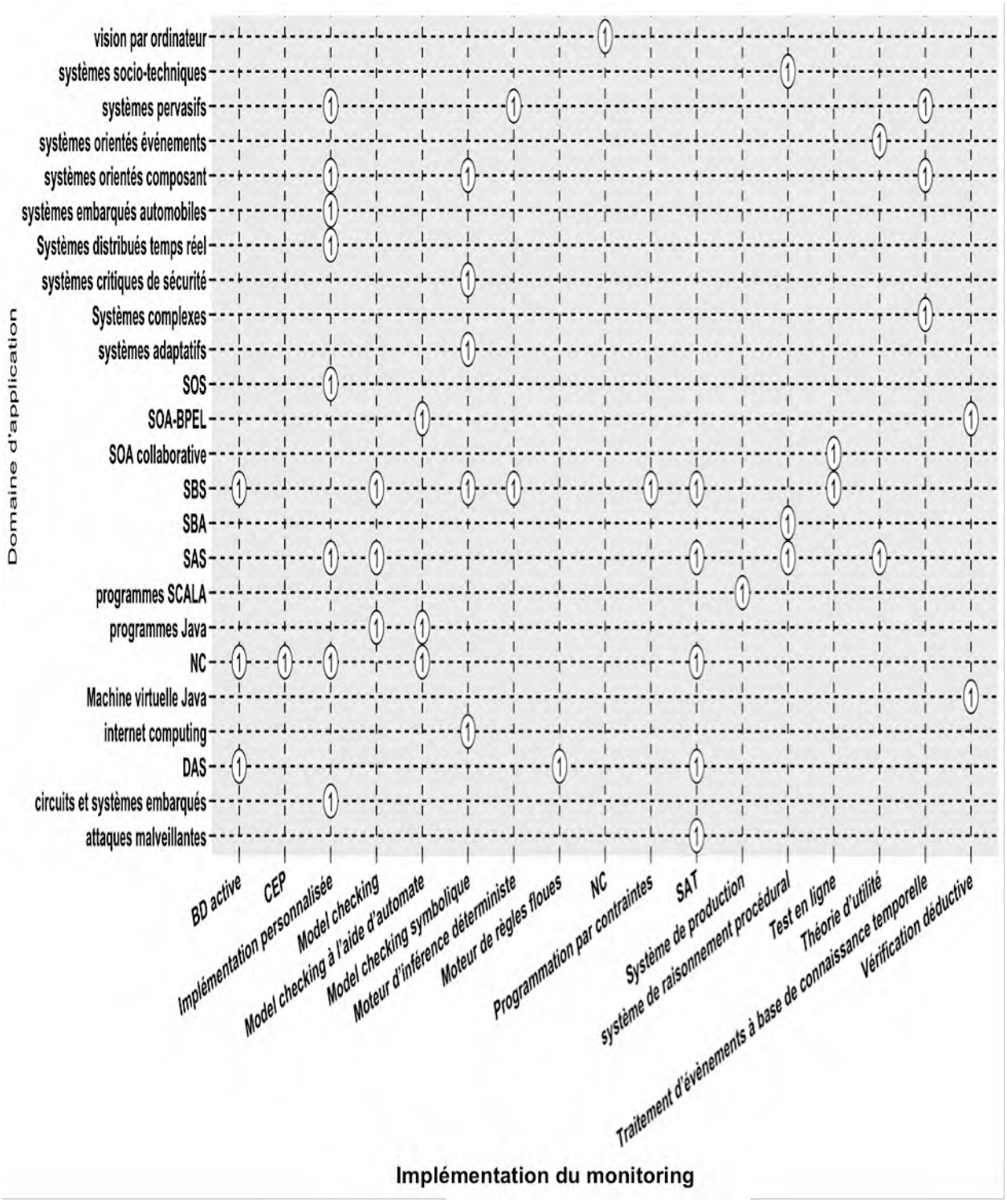


Figure 8 : Corrélation entre l'implémentation du monitoring et le domaine d'application

Finalement, la Figure 9, montre qu'une grande corrélation entre les SASs et la programmation orientée but, et une autre corrélation entre les systèmes orientés service et la logique temporelle ainsi que les systèmes de transition d'états.

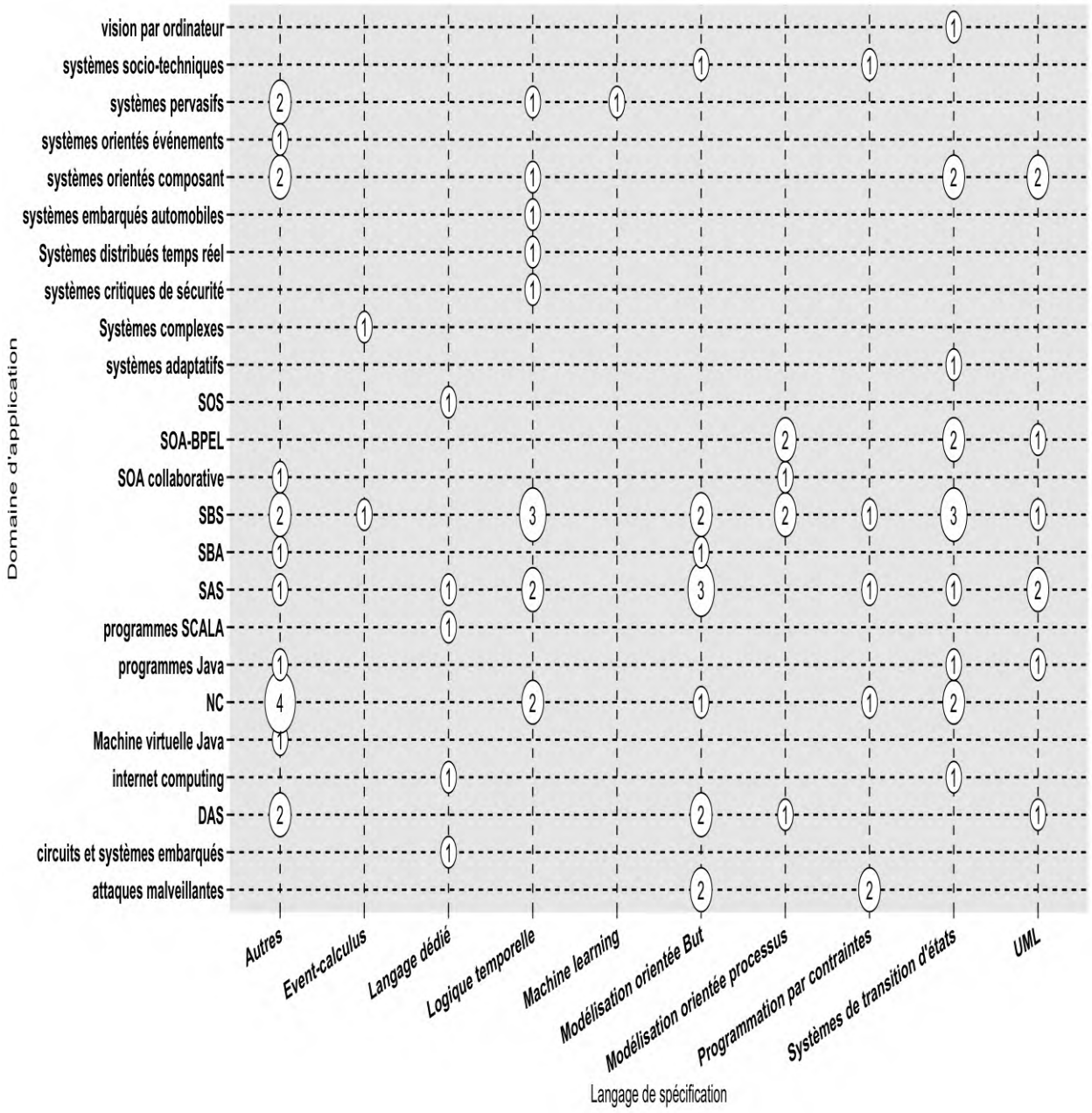


Figure 9 : Corrélation entre les langages de spécification et domaine d'application

QR4 : Quels sont les langages de spécification qui ont été bien appliqués et ont été sujettes à une évaluation rigoureuse ?

L'objectif de cette question est d'identifier les méthodes de spécification qui ont été bien appliquées et bien évaluées. Pour ce faire, une évaluation des études primaires a été effectuée en se basant sur les six questions du programme d'évaluation critique (CASP). Les six questions concernent principalement la clarté de l'étude, la rigueur dans la conception, la mise en œuvre et la validation.
Le résultat d'évaluation est décrit dans le Tableau 2.

Tableau 2 : distribution des études primaires par méthodes de modélisation utilisée et score de qualité obtenu

Méthode de modélisation	Fréquence (%) des études primaires par score de qualité obtenu			
	3	**4**	**5**	**6**
Logique temporelle	41,67%	16,67%	**33,33%**	8,33%
Machine Learning	-	100%	-	-
Modélisation orientée but	33,33%	**33,33%**	25%	8,33%
Modélisation orientée processus	50%	**50%**	-	-
Programmation par contraintes	33,33%	16,67%	**50%**	-
Systèmes de transition d'états	**57,14%**	14,29%	21,43%	7,14%
Event Calculus	50%	-	**50%**	-
UML	**50%**	37,5%	-	12,5%
Langage dédié	40%	20%	**40%**	-
Autres	**66,67%**	22,22%	11,11%	-

D'après le Tableau 2, la programmation par contraintes, la logique temporelle, Event Calculus, les langages dédiés et la modélisation orienté but sont les méthodes de modélisation qui ont obtenu les scores les plus élevés (plus de 40% à partir du score 5) et donc qui ont été bien appliquées. Il en ressort que les langages orientés modèle, qu'ils soient formels ou semi-formels, sont les langages les plus utilisés pour la modélisation des systèmes et les mieux appliqués. Ces langages favorisent l'expressivité au cours de l'activité de la spécification et sont faciles à utiliser. Une pratique courante de l'utilisation de ces langages dans le monitoring consiste à les transformer vers d'autres langages formels pour bénéficier de l'analyse et la vérification automatiques.

QR5 : Quels sont les points faibles et les points d'amélioration possibles dans la recherche sur le monitoring des exigences à l'exécution ?

Pour analyser l'état actuel de la recherche dans le domaine de monitoring des exigences, nous avons effectué une analyse croisée des différentes dimensions de la taxonomie en se basant sur les données extraites des études primaires. Les résultats des différentes corrélations entre les dimensions nous ont permis de dégager les lacunes et les orientations dans ce domaine. En effet, nous avons utilisé : (i) le but de monitoring, et les propriétés sujettes au monitoring, les méthodes de spécification ainsi que le mode de monitoring en tant que dimensions transversales, comme le montrent les Figures 7, 8, et 9 ; (ii) les méthodes de spécification, et le type de propriété, les techniques d'implémentation ainsi que le support d'incertitude comme dimensions transversales, comme le montrent les Figures 10 et 11 et 12. Le nombre des études primaires apparaît dans le cercle de chacune des dimensions transversales. (iii)

les techniques d'implémentation du monitoring et le support d'incertitude, et enfin (iv) les couches d'abstraction ont été corrélées avec les technologies d'instrumentation.

La Figure 10 montre que la détection des défaillances a été l'objet d'une grande partie des études, et que la sûreté de fonctionnement, l'exactitude et la sécurité sont les plus importantes propriétés quel que soit le but de monitoring. En particulier, une grande corrélation peut être observée entre l'exactitude/ la sûreté de fonctionnement et la détection des défaillances, mais aussi entre l'exactitude et l'adaptation/reconfiguration et la conscience des exigences.

D'autre part, un manque significatif peut est observée concernant le support de diagnostic des défaillances (seulement 3 études identifiées).

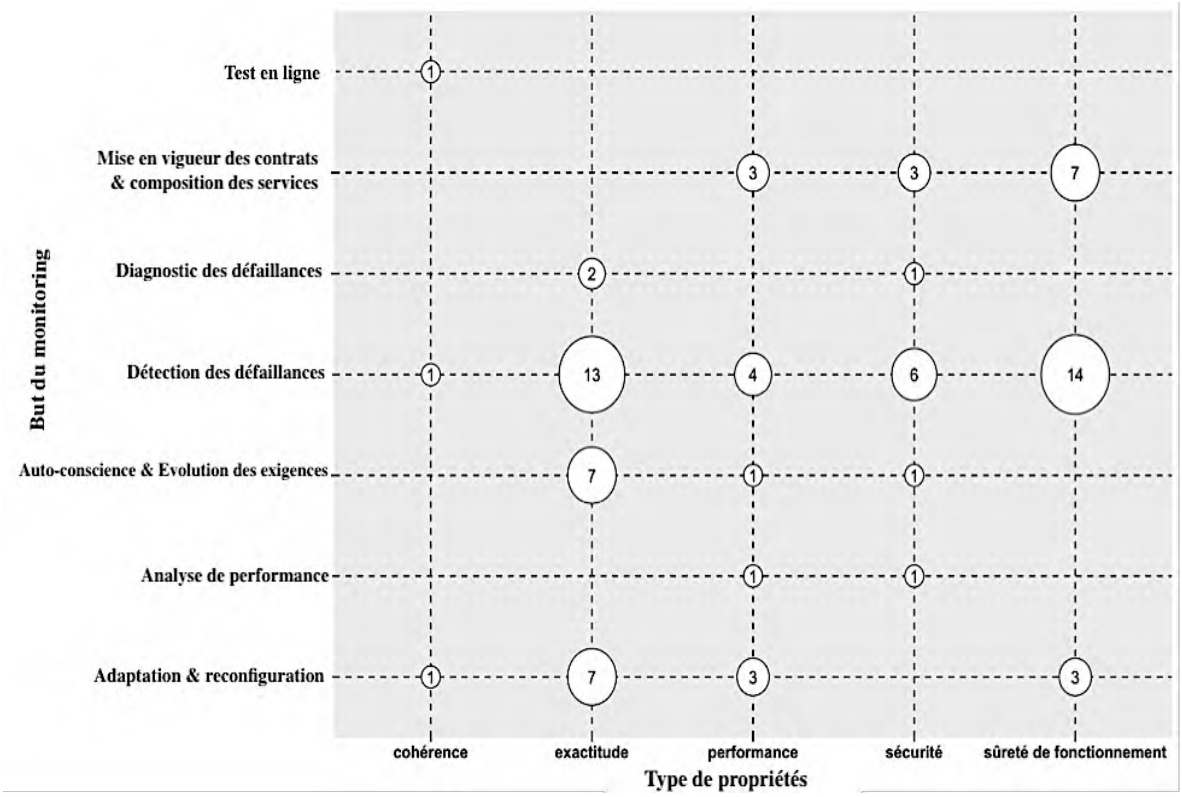


Figure 10 : Corrélation entre but de monitoring et les propriétés sujettes au monitoring

En ce qui concerne les méthodes de spécification des exigences, la Figure 11 montre que la logique temporelle, la modélisation orientée but et les systèmes d'états sont les plus importantes méthodes de spécification des exigences, quel que soit le but de monitoring. Spécialement, il y a une grande corrélation entre la conscience des exigences, l'adaptation et la modélisation orientée but, entre les systèmes de transition d'états et la détection des défaillances et entre la programmation par contraintes, la modélisation orientée but et le diagnostic. Alors que La Machine Learning semble être une lacune dans ce domaine.

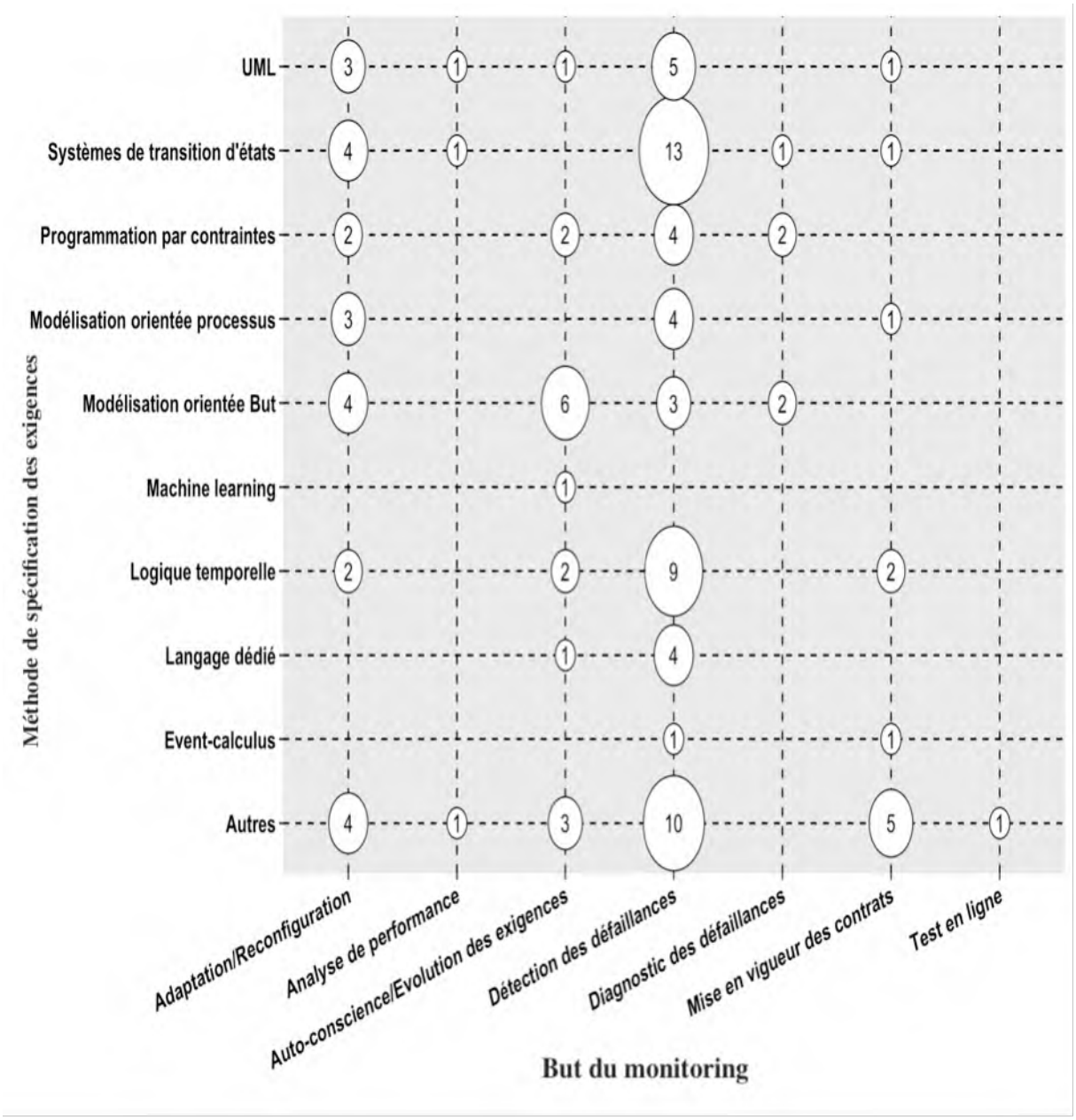


Figure 11 : Corrélation entre but de monitoring et les méthodes de spécification des exigences

D'autre part, la Figure 12 montre une grande corrélation entre le model-checking symbolique et les systèmes de transitions d'états (ce qui est évident), et entre SAT / programmation par contraintes et la programmation orientée but. Elle montre également que la logique temporelle est exploitée souvent dans des implémentations personnalisées et bien évidemment par le model-checking. Il est à noter que le traitement des évènements complexes à l'aide de la logique temporelle est un champ en pleine expansion.

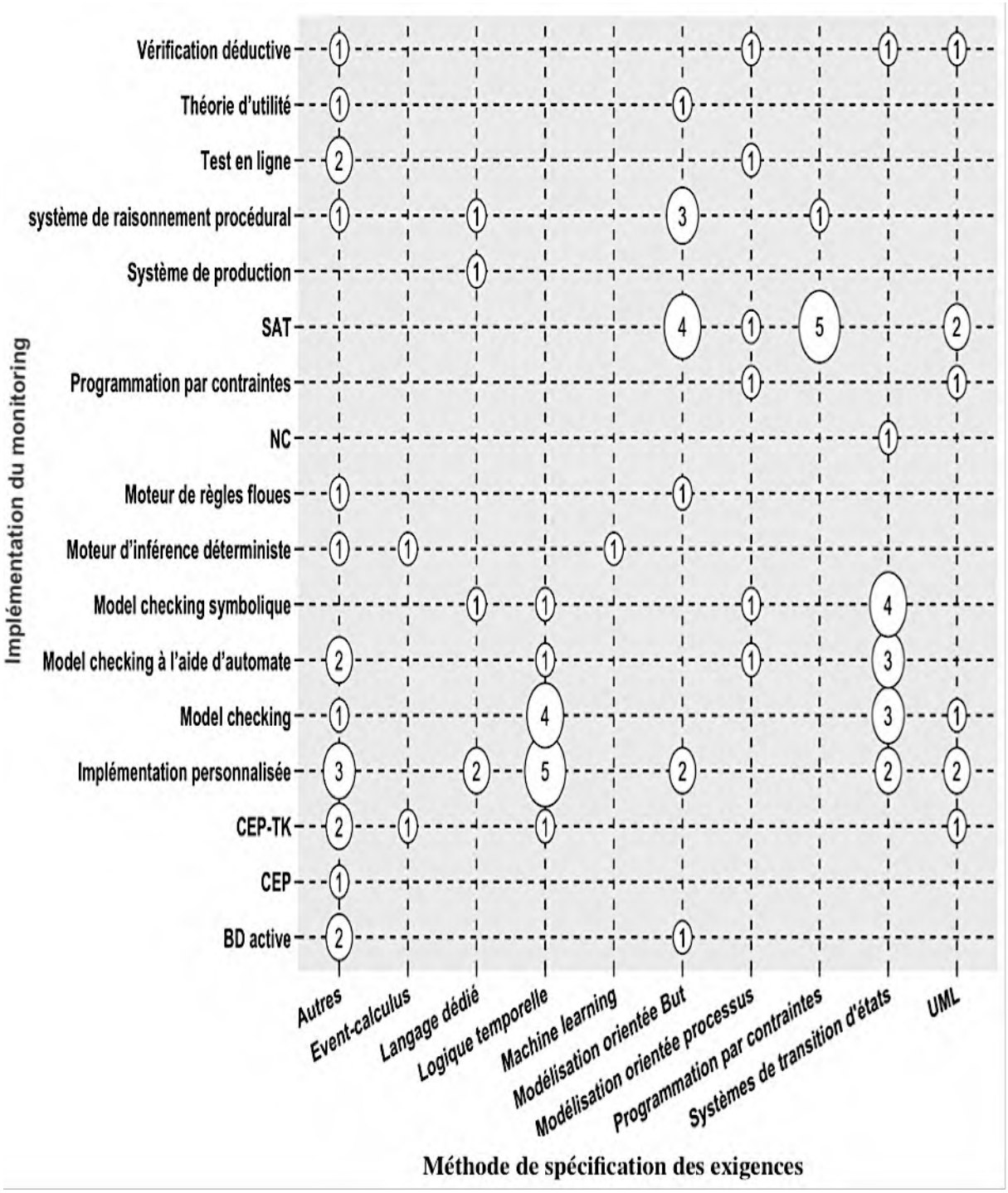


Figure 12 : corrélation entre la méthode de spécification des exigences et l'implémentation du monitoring

Quant au mode de monitoring, le monitoring de type asynchrone est le plus utilisé. Comme le montre la Figure 13, il y a une grande corrélation entre le mode asynchrone et la détection des défaillances. Une grande corrélation existe également entre le mode synchrone et l'adaptation.

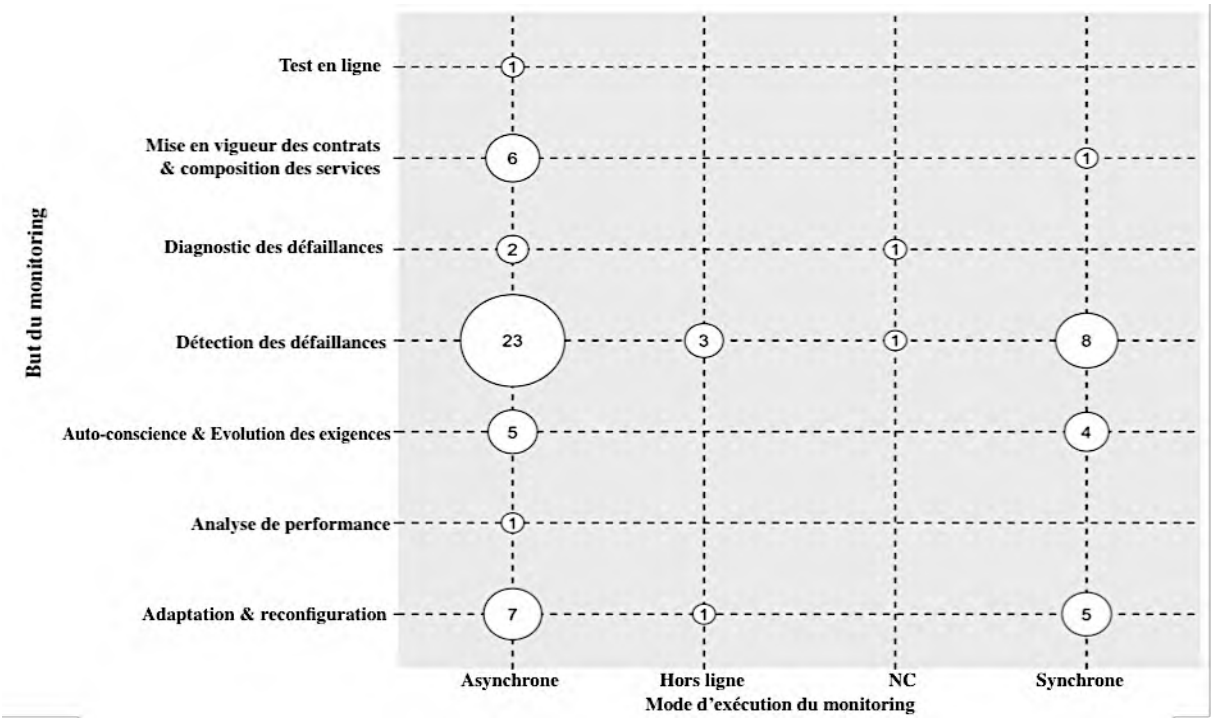


Figure 13 : Corrélation entre but de monitoring et le mode de monitoring

En ce qui concerne le support d'incertitude, la Figure 14 montre que l'incertitude liée à une source interne a été supportée par presque tous les langages de spécification. Particulièrement, une grande corrélation existe entre la modélisation orientée but, la logique temporelle et le support d'incertitude interne. D'autre part, la Machine Learning est étonnamment un mécanisme peu connu pour la gestion d'incertitude dans ce domaine.

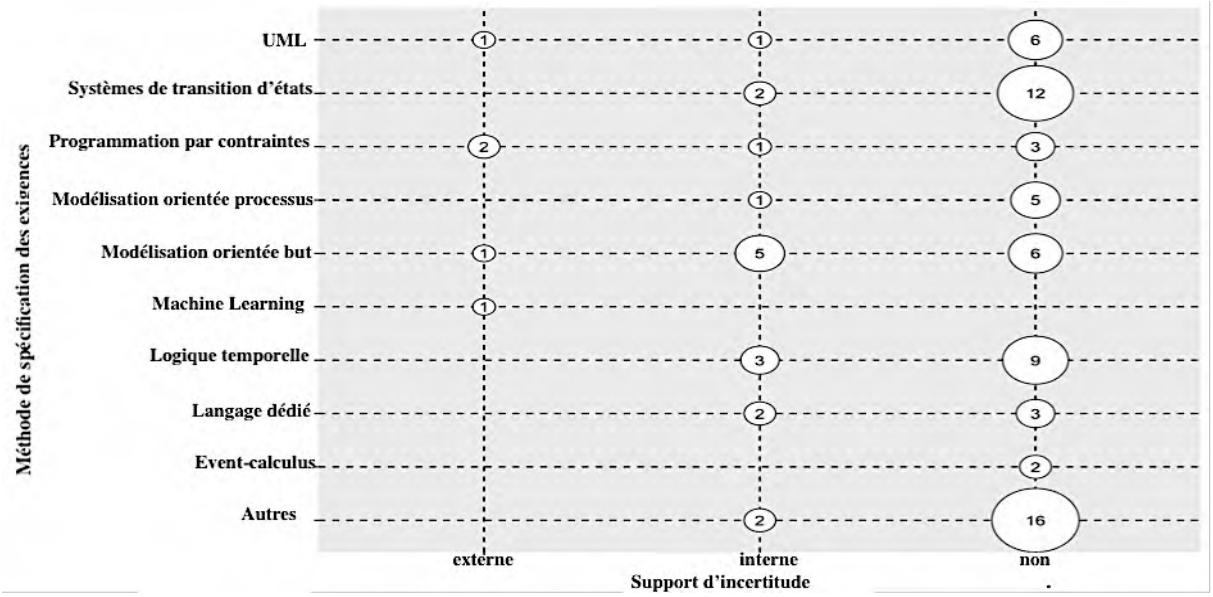


Figure 14 : Corrélation entre les méthodes de spécification et le support d'incertitude

En outre, la Figure 15 montre que l'incertitude est peu supportée dans les différentes implémentations du monitoring. Cependant, une corrélation peut être observée entre la modélisation par SAT et l'incertitude externe, et entre le model-checking, SAT, les systèmes de raisonnement procédural, et l'incertitude interne.

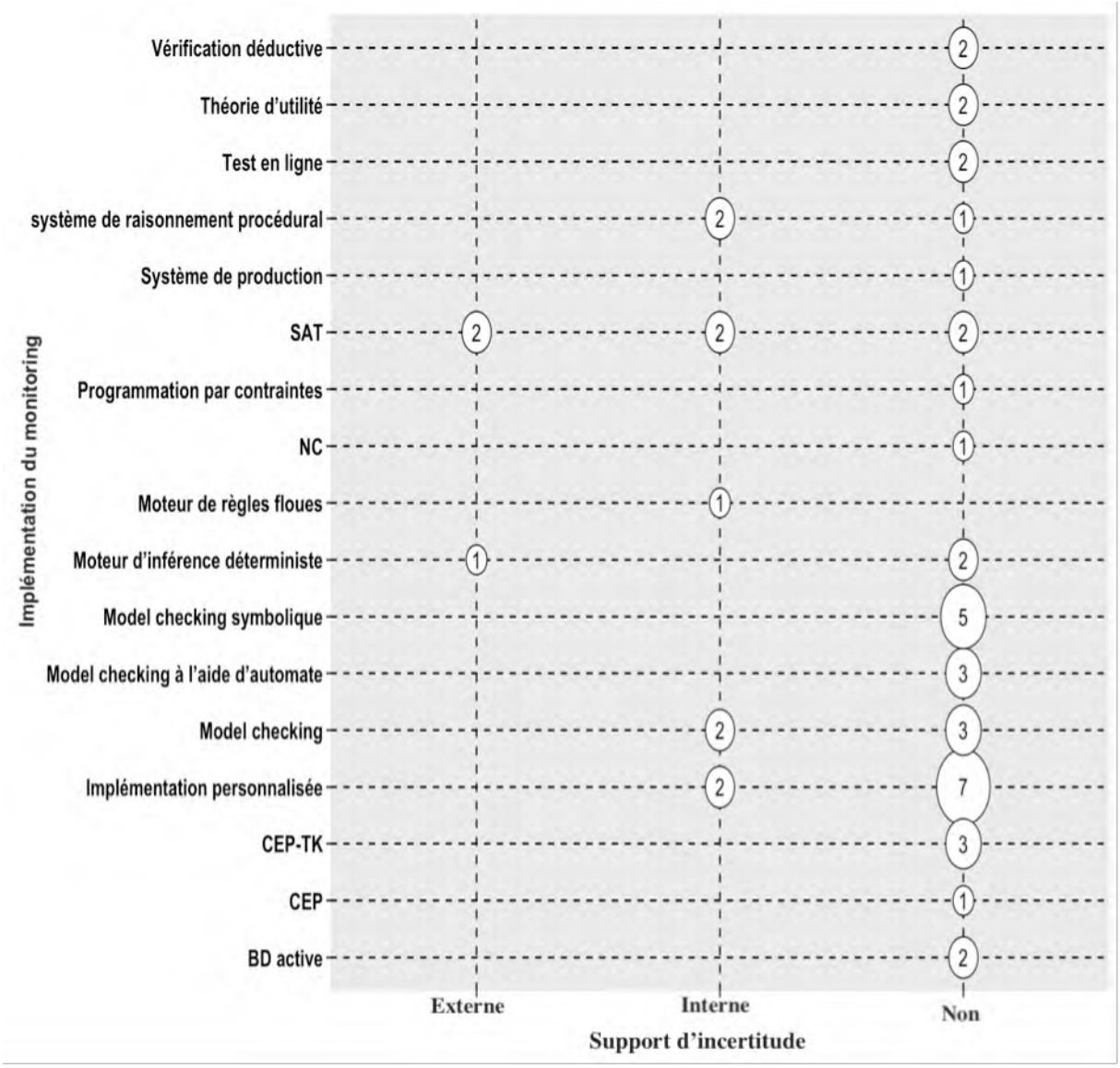


Figure 15 : Corrélation entre l'implémentation du monitoring et le support d'incertitude

Finalement, quant à l'instrumentation (voir Figure 16), les couches application et infrastructure sont souvent considérées pour la collecte des données du système cible. Alors que l'exploitation de la couche utilisateur semble une autre lacune de recherche à combler dans ce domaine.

Pour l'instrumentation de ces couches d'abstraction, le langage AspectJ, l'interception et les capteurs figurent parmi les technologies les plus utilisées. Spécialement, une grande corrélation a été identifiée entre la couche application et l'interception/AspectJ, ainsi que la couche infrastructure et les capteurs.

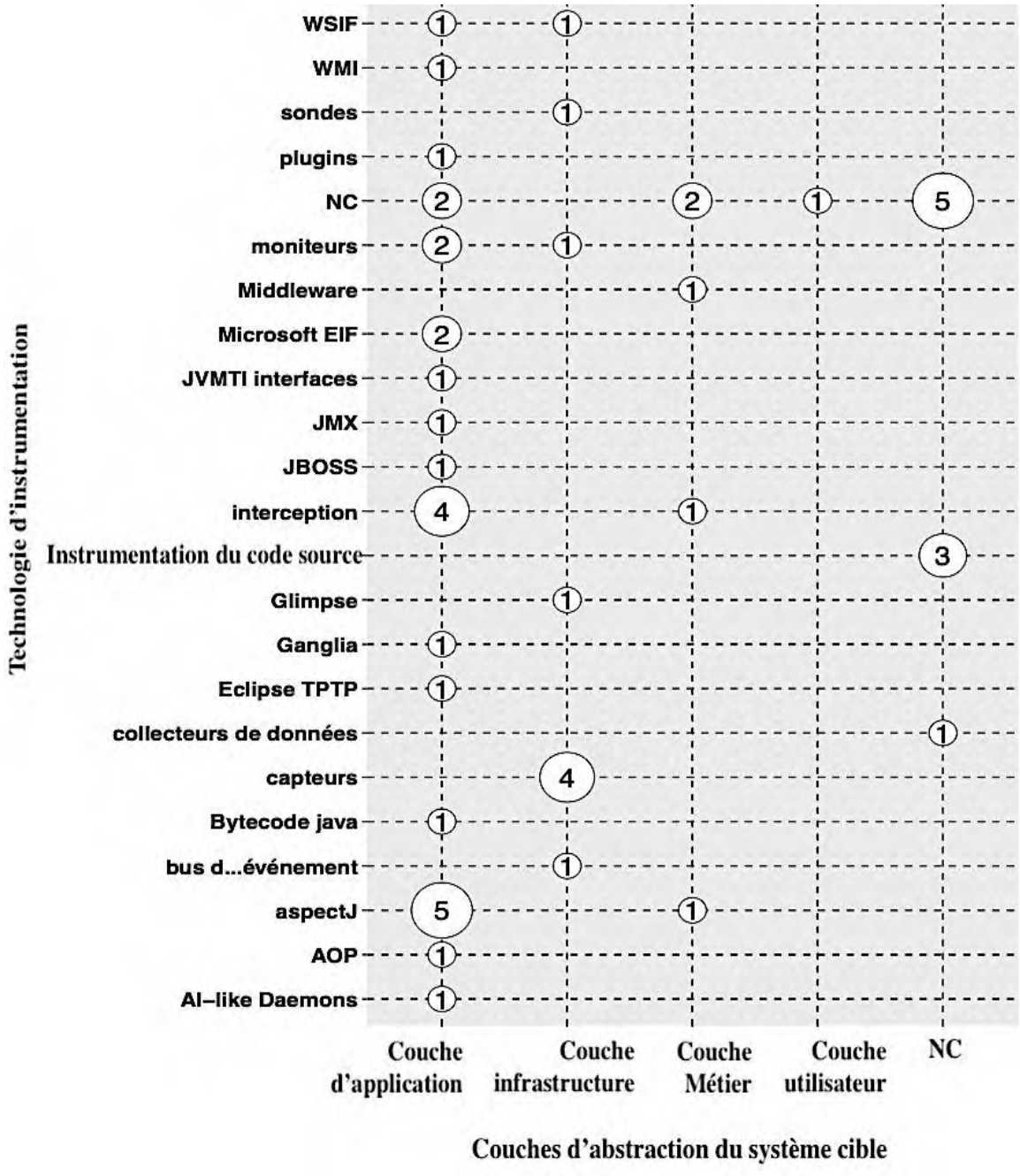


Figure 16 : Corrélation entre les couches d'abstraction et les technologies d'instrumentation

D'autres observations et lacunes :

- Peu d'approches traitent le diagnostic des défaillances, alors que le diagnostic doit être au cœur du processus de l'analyse des exigences à l'exécution pour automatiser et optimiser la reconfiguration des systèmes adaptatifs et l'évolution des exigences. D'autre part, parmi les approches de monitoring des exigences qui ont indiqué leur support à l'adaptation après la détection des non-conformités, peu d'entre elles ont décrit comment l'adaptation est supportée.

- Bien qu'il existe une variété des approches supportant le monitoring des exigences à l'exécution, la plupart des cadres logiciels et des outils ne sont pas évalués sur des cas réels et leurs implémentations, si elles existent, ne sont pas disponibles en ligne. Par conséquent, les chercheurs sont souvent ramenés à « réinventer la roue », ce qui ralentit l'adoption de ces approches dans l'industrie. D'où la nécessité de non seulement proposer une approche expressive, mais de fournir une architecture et une implémentation intelligente et disponible.

- Enfin, un dernier champ qui mérite d'être exploré est le benchmark des systèmes de monitoring. En effet, avec le manque des méthodologies de benchmarks en ingénierie logiciel, qui aident à mesurer correctement la performance des systèmes logiciels et offrent un moyen de comparer les différentes approches existantes d'un domaine (Price, 1989 ; Sachs, 2011), spécialement ceux traitant les systèmes de monitoring des exigences à l'exécution, il n'existe pas un consensus de la meilleure évaluation de ce type de monitoring (Waller & Hasselbring, 2013).

2.3 Discussion

Le monitoring des exigences à l'exécution est un concept délicat à cerner avec précision vu la variété des approches identifiées. Ces approches visent généralement à étudier dans le temps le comportement des systèmes complexes sur la base des propriétés spécifiques ou des invariants. Le monitoring peut être une fin en soi ou une étape vers d'autres buts comme l'évolution, le diagnostic ou la réparation voire la reconfiguration. Les résultats de la revue systématique ont permis de dégager les pratiques et directions dans ce domaine en s'appuyant sur une analyse plus au moins quantitative. Dans cette section, nous présentons en détail les approches qui sont étroitement liées à nos travaux de recherche et la relation entre elles.

Dans le cadre de cette thèse, les travaux de recherche sur le monitoring des exigences s'appuient sur l'expressivité de la modélisation orientée buts, et les spécificités de la programmation par contraintes et son support au raisonnement logique qui mène à une spécification plus efficace des problèmes étudiés :

Approches de monitoring orientées buts

Généralement, le principal avantage de l'utilisation des modèles réside dans le fait que la spécification est souvent moins liée à la technologie d'implémentation et beaucoup plus proche du domaine du problème. Cela rend les modèles moins sensibles à la technologie informatique utilisée, et aux changements évolutifs de cette technologie. Spécialement, la modélisation orientée but, et contrairement aux méthodes traditionnelles comme UML, est reconnu par sa capacité à exprimer des concepts abstraits tels que l'intention et le but. Ce dernier permet la validation de la complétude et de la cohérence des exigences qui leur sont associées.

Baresi et Pasquale (2010) [Ref43] proposent la notion des buts flous ou adaptatifs pour représenter l'incertitude liée aux buts. En effet, FLAGS fait la distinction entre un but précis et un but flou. La

satisfaction du premier peut être représentée par la logique LTL comme dans KAOS, alors que la satisfaction du deuxième est modélisée par une fonction d'appartenance issue de la théorie de la logique floue. Les buts adaptatifs définissent des contre-mesures à l'aide des règles Event-Condition-Action qui sont exécutées lorsque les buts ne sont pas atteints. En revanche, aucun détail sur la manière d'intégrer ces buts adaptatifs dans un modèle de buts ni une implémentation ou évaluation de la solution proposée n'a été fourni explicitement.

Welsh, Sawyer & Bencomo (2011) et Welsh et al. (2014) [Ref1] ont opté pour les modèles i * pour la spécification des exigences non-fonctionnelles. Ces modèles sont étendus par des claims qui définissent l'impact des différentes stratégies du système cible sur la satisfaction des soft goals. A l'exécution, les agents BDI surveillent la véracité de chaque claim en se basant sur les données collectées du contexte. Ce qui contribue à justifier le choix d'une stratégie (la notion d'auto-justification) sous incertitude et amener le système à s'adapter pour optimiser ces objectifs dans de nouvelles conditions de contexte. Nous utilisons également les claims pour lier les choix stratégiques à la satisfaction des buts, cependant notre approche est totalement automatisée par des règles de transformation du modèle de buts vers un programme de contraintes, à l'encontre du système BDI qui en plus de la difficulté de sa mise en œuvre, son comportement est souvent imprévisible quand les agents sont liés à des désirs et des besoins conflictuels (Taillandier, et al., 2012 ; Sardina, et al., 2006).

Ali, Dalpiaz & Giorgini (2010) [Ref50] proposent de définir des contraintes de contexte et les lier à des points de variation du modèle de buts TROPOS. L'idée centrale est de supporter le raisonnement sur la variabilité du contexte et son impact sur la satisfaction des exigences. Tandis que le monitoring et le diagnostic des non-conformités ne sont pas des préoccupations dans leur approche.

Dans la même veine Dalpiaz, Giorgini & Mylopoulos (2013) [Ref46] propose une architecture BDI à partir d'un modèle de buts TROPOS. Les auteurs spécifient en détail des plans sous forme d'un ensemble d'actions à exécuter par les agents, des préconditions et des postconditions ainsi que des délais pour l'exécution de ces actions. Si l'une des actions d'un plan n'est pas remplie, cela abouti à un échec du plan. Les auteurs définissent des algorithmes de diagnostic et de reconfiguration pour compenser les tâches ayant échoué ou celles ayant déjà commencé et qui seront annulées. Le principe consiste à comparer les différentes alternatives et leur contribution à la satisfaction des soft goals tout en examinant les relations sociales entre les différents acteurs du système cible. Une comparaison entre les performances de ces deux travaux et de notre approche est donnée dans la section 6.6.

Approches de vérification des LdPs basées sur la satisfaction des contraintes

Notre approche est fondée sur des travaux de recherche antérieurs dans le domaine des LdPs qui proposent de transformer le modèle de LdP en logique propositionnelle ou en programmes de satisfaction des contraintes. L'objectif de ces représentations est d'exploiter l'expressivité de la logique et de la programmation par contrainte et d'automatiser la vérification des LdPs avec des

systèmes de preuve automatique tels que les solveurs SAT (Metzger et al., 2007), les diagrammes de décision binaire (BDD) (Trinidad et al., 2008), les solveurs CSP (Benavides et al. 2010 ; Mazo et al., (2011a ; 2012) ; Salinesi et al., (2010, 2011) ; White et al. 2008), les solveurs SMV (Zhang et al. , 2004), et les interprètes OCL (Vierhauser et al. , 2010).

Selon Mazo (2011b) et Benavides et al. (2010), la plupart des approches existantes transforment les LdPs en un programme de satisfiabilité boolèene. Ce faisant ces approches n'exploitent pas pleinement la richesse de la programmation par contraintes, comme la possibilité de définir des concepts numériques tels que la cardinalité d'une caractéristique et le groupe de cardinalités.
Notre approche est similaire aux travaux supportant la programmation par contraintes dans la mesure où elle propose de transformer la spécification des exigences en un programme logique de contraintes en domaines finis. Cependant, les travaux dans ce domaine sont centrés sur la vérification de la consistance des Laps en temps de spécification et n'ont pas pour vocation le monitoring et la validation des exigences à l'exécution. De plus, les règles de transformation proposées sont étroitement liées à REFAS et les propriétés de son méta-modèle. Ce langage est très récent (2015) et à notre connaissance n'est pas encore exploité pour des fins de monitoring et de diagnostic des exigences.

Approches de monitoring basées sur la satisfaction des contraintes

Wang et Mylopoulos (2009) [Ref45] proposent de représenter les exigences sous forme du modèle de buts défini par Giorgini et al. (2002). Chaque but se voit attribuer une précondition, une postcondition et un monitor pour le surveiller. A l'exécution les données de journalisation sont générées sous forme de formules propositionnelles et sont analysées par un solveur SAT. Ainsi, si un but échoue, un diagnostic à différents niveaux de granularité est fourni et une reconfiguration du système est déclenchée sans les buts défaillants. Malheureusement, l'approche de Wang et Mylopoulos (2009) se limite au monitoring et au diagnostic des exigences fonctionnelles du système qui sont directement observables (par exemple, surveiller si un serveur démarre correctement) et ne permet pas de gérer les changements imprévus de l'environnement. Alors que notre approche utilise les claims comme marqueurs d'incertitude et gère les sources imprévisibles d'incertitude en les sauvegardant dans une base de données qui déclenche un besoin d'évolution manuelle de la spécification. De plus, alors que leur approche exploite la structure arborescente des modèles de buts et associe une précondition et une postcondition pour chaque liaison, notre approche transforme chaque liaison en une formule logique, ainsi en cas de changement dans le modèle de buts, le changement est directement répercuté sur le programme de contraintes.

Souza & Mylopoulos (2009) [Ref44] proposent d'étendre l'approche de Wang et Mylopoulos (2009) par la notion d'anti-buts pour représenter non seulement des exigences des parties prenantes (buts),

mais également les intentions des attaquants. Le but est de favoriser le monitoring et le diagnostic des attaques malveillantes.

Peng et al. (2012) [Ref47] repose sur un contrôleur PID pour ajuster les niveaux de satisfaction des soft goals en cas de conflit entre les soft goals. Leur approche repose également sur la transformation du modèle de buts en un ensemble de contraintes logiques sous la forme normale conjonctive, à l'aide du codage proposé par Sebastiani, Giorgini, & Mylopoulos (2004). La configuration est ensuite prise en charge par un solveur SAT. Le principe consiste à supprimer le but le moins classé et d'exécuter le solveur SAT d'une manière itérative jusqu'à ce qu'une configuration optimale soit trouvée. Malheureusement leur approche ne propose aucun dispositif pour le diagnostic des exigences et le processus de configuration repose sur une approche ad hoc (suppression des buts d'une manière itérative).

Enfin, Salifu et al. (2012) [Ref3] présente un formalisme personnalisé basé sur la logique propositionnelle. Cependant, l'approche a été validée par simuler du comportement du système en utilisant statechart UML.

2.4 Validité de la recherche

La validité de construction relative aux travaux examinés et à la collecte de données

La recherche était basée uniquement sur des métadonnées (résumé, titre et mots-clés) et aurait peut-être manqué certaines études pertinentes qui considèrent le monitoring des exigences à l'exécution mais qui n'ont pas explicitement mentionné les mots-clés utilisés dans leurs titres, résumés et mots-clés. En outre, comme la sélection des études primaires est subjective (faite un humain), nous ne pouvons pas garantir que toutes les études pertinentes ont été sélectionnées. Certaines publications pourraient avoir été exclues alors qu'elles sont pertinentes Pour éviter ce biais, nous avons défini des critères d'inclusion et d'exclusions claires permet d'atténuer la subjectivité humaine.

D'autre part, une autre menace à la validité de construction est liée à l'inexactitude des données extraites qui peut être liée à la subjectivité humaine ou à la manière dont les auteurs des études primaires présentaient leurs approches.

Validité interne relative à la méthode de la revue

Les questions de recherches définies n'auraient peut-être pas couvert l'ensemble du domaine de recherche. Notre revue pourrait néanmoins servir à identifier et comprendre les pratiques utilisées pour le monitoring des exigences à l'exécution. D'autre part, nous reconnaissons que les études primaires incluses pourraient ne pas couvrir l'ensemble du domaine de recherche, cependant, nous sommes confidents que ces études constituent un bon représentant des recherches effectuées dans ce domaine. Finalement, Une menace importante à la validité interne, est de savoir si la taxonomie construite est solide et complète pour l'analyse et la classification. Cependant, étant donné que la taxonomie a été

construite en répandant aux six questions (Quoi, Où, Quand, Pourquoi, qui et comment) et en adoptant une démarche itérative en se basant sur l'analyse des études primaires, nous pensons que la taxonomie couvre les concepts clés relatifs au monitoring des exigences à l'exécution.

Validité externe relative à la généralisation.

Étant donné que la revue couvre seulement le domaine de recherche académique avec des articles journaux, et quelques cas réels issus de l'industrie, Il y a un risque que les pratiques industrielles ne soient pas incluses dans la revue. Nous ne pouvons pas donc généraliser le résultat de cette revue.

2.5 Conclusion du chapitre

Ce chapitre a contribué à une revue systématique sur le monitoring des exigences à l'exécution. Une taxonomie a été proposée pour analyser et comparer les approches supportant ce domaine de recherche. Une telle taxonomie nous a ouvert la voie à une meilleure compréhension des pratiques utilisées pour le monitoring des exigences à l'exécution. De plus, une analyse croisée des différentes dimensions de la taxonomie nous a permis de dégager des lacunes et des orientations pour des recherches ultérieures.

Nous rappelons que cette thèse vise à intégrer des mécanismes de monitoring pour surveiller la validation d'un système auto-adaptatif par rapport à des exigences spécifiées. Le but par la suite est de pouvoir diagnostiquer les sources des défaillances détectées et de planifier des remédiations pour optimiser le système cible. À la lumière des résultats de la RS, nous optons pour les solutions suivantes :

[1]L'utilisation des méthodes orientées but pour la spécification des exigences. Ces méthodes sont plus généralistes car ils permettent de décrire à la fois les exigences fonctionnelles et non-fonctionnelles. Ils offrent en outre des concepts de haut niveau d'abstraction comme les contraintes structurelles.

[2]La transformation automatique des modèles de buts en des modèles formels supportés par des outils d'analyse et de vérification. Ce qui va nous permettre d'automatiser notre approche et d'avoir des résultats sûrs.

Deuxième Partie : Approche proposée

Cette partie présente notre approche pour le monitoring des exigences à l'exécution, dédiée aux systèmes auto-adaptatifs :

- Le chapitre 3 présente notre approche appelée RMAS (en anglais, Requirements Monitoring for Adaptive Systems) qui se compose de deux phases principales : la phase de spécification est basée sur la modélisation orientée buts et la programmation logique par contraintes, et la phase d'exécution propose des algorithmes d'automatisation d'analyse des exigences à l'exécution.
- Le chapitre 4 présente une implémentation des modèles présentés dans le chapitre 3.

Chapitre 3

3. PRESENTATION DE L'APPROCHE

Ce chapitre détaille la première contribution de notre approche liée à la formulation de la spécification d'exigences et le monitoring des exigences à l'exécution des SASs. Ce chapitre apporte donc des réponses aux questions de recherche QR1 et QR2, énoncées dans l'introduction.

Les réponses à ces questions sont présentées dans les sections suivantes : la section 3.1 présente un exemple d'un système hypermédia adaptatif qui va servir à illustrer notre approche. La section 3.2 présente l'approche générale. La section 3.3 apporte une réponse à la QR1 (voir définition 8) par le biais d'une spécification axée sur les modèles de buts, l'analyse de la variabilité et la programmation logique par contraintes, et guidée par REFAS (Muñoz-Fernández et al., 2015), un nouveau langage proposé par une équipe de notre laboratoire de recherche CRI. La section 3.4 répond à la QR2. Enfin, nous terminons le chapitre par une conclusion

3.1 Exemple illustratif

3.1.1 Présentation de l'exemple

Nous nous appuyons sur un exemple illustratif INSPIREus (Papanikolaou, et al. 2003) pour expliquer notre approche.

INSPIREus est un système hypermédia adaptatif dédié à l'auto-formation. Il offre un environnement personnalisé et une navigation adaptée à chaque apprenant. Il propose en outre des fonctionnalités de collaboration et un processus flexible d'édition du contenu permettant aux utilisateurs d'explorer librement le contenu des cours et de refléter leur perspective pédagogique sur le développement du contenu.

Le contenu des modules de connaissance est organisé en trois niveaux d'acquisition des connaissances rappeler, utiliser et trouver, allant du simple au complexe. Le niveau "rappeler" est lié à la capacité des apprenants à se rappeler leurs connaissances ; le niveau "utiliser" est lié la capacité des apprenants à appliquer la théorie à travers des études de cas, et le niveau 'Trouver' est lié à la capacité des apprenants à proposer et à résoudre des problèmes originaux.

Tous les apprenants reçoivent les mêmes contenus, à savoir « théorie », « exemple », « exercices » et « activités ». Toutefois, la méthode et l'ordre de leur présentation sur chaque page sont personnalisés pour chaque apprenant. Le choix de la structure dépend du niveau d'apprentissage 'KL' et du style d'apprentissage 'LS' des apprenants. KL comprend trois niveaux {faible, moyen, élevé}. Et LS est défini selon le modèle de Honey & Mumford (1992) qui distinguent quatre styles d'apprentissage {actif, réflexif, théoricien et pragmatique} : les actifs préfèrent assimiler de nouvelles informations à travers des activités. Les réflexifs restent à l'écart des gens et apprennent mieux en observant quelqu'un d'autre, et en réfléchissant à ce qui se passe, les pragmatiques préfèrent apprendre par la

pratique. Et les théoriciens apprennent par la théorie et aiment les problèmes complexes où ils peuvent utiliser leurs compétences et leurs connaissances.

Partant de ces critères, l'objectif principal est de faire correspondre les préférences d'apprentissage dominantes des apprenants à un séquencement approprié des ressources pédagogiques. Le Tableau 3 résume les stratégies d'adaptation adoptées pour chaque style d'apprentissage.

Tableau 3 : les stratégies d'adaptation de la présentation des pages de connaissance (choix de modules et ordre d'apparence)

Niveau d'acquisition / style d'apprentissage	Niveau "rappeler"	Niveau "utiliser"
actif	(Q, E, T)*	(A, Ex, T, E) *
réflexif	(T, Ex, Q)*	(Ex, T, E, A) *
théoricien	(Q, T, Ex) *	(T, Ex, E, A) *
pragmatique	(Ex, T, Q) *	(E, Ex, T, A) *

* Q : module question, E : module exercices, Ex : module exemple, T : module théorie, A : module activités.

Ainsi, il existe deux stratégies principales de présentation des modules sur une page du niveau « rappeler » : (i) La présentation « inquisitoire » (Q, E, T) ou (Q, T, Ex) commence par une « question » visant à attirer l'attention de l'apprenant, puis enchaîne respectivement avec les modules « exercices » ou « théorie » associés respectivement aux styles actif et théoricien. (ii) La présentation « expositoire » (Ex, T, Q) ou (T, Ex, Q) commence respectivement par les modules « exemple » ou « théorie » associés respectivement aux styles pragmatique et réfléchi, puis le module « question » apparaît comme une auto-évaluation qui incite à la réflexion.

D'autre part, quatre stratégies sont appliquées sur une page du niveau « utiliser » : (1) une présentation –orientée activités, dédiée au style actif, qui commence par le module « activités » en haut de la page, suivi des autres modules, (2) une présentation orientée exemple, dédiée au style réfléchi, qui commence par des exemples en haut de la page, suivie des autres modules. De même (3) une présentation orientée exercices, dédiée au style pragmatique, qui commence par le module « exercices » ; et (4) une présentation orientée théorie, dédiée au style théoricien, qui commence par le module « théorie ».

3.1.2 Exemples des exigences à surveiller

Nous avons fait le choix des exigences générales issues des problèmes signalés dans les travaux de recherche sur les AESs, que nous les avons adaptées à notre contexte comme suit :

Exig1 : INSPIREus doit fournir aux apprenants des ressources pédagogiques qui correspondent à leurs styles d'apprentissages.

INSPIREus donne aux apprenants un contrôle sur l'environnement d'apprentissage leur permettant de modifier leur style et niveau d'apprentissage. Ainsi, pendant son apprentissage, il se peut qu'un apprenant apporte une modification de style qui ne soit pas dans son intérêt, soit parce qu'il/elle ne maîtrise pas son style d'apprentissage ou par ignorance de l'effet de cette modification. Le système doit être plus flexible pour supporter chaque apprenant dans son choix et générer une ressource pédagogique adaptée à son « style d'apprentissage efficace », ce dernier pouvant être détecté à l'aide des fichiers de journalisation du système. Par exemple, si un apprenant modifie son style d'apprentissage en « actif » alors que l'historique de ces interactions indique que cet apprenant est plutôt réflexif dans le sens où il est plus productif en suivant des exemples; le système doit capter cette information en lui recommandant des modules exemple plutôt que de modules « activité ».

Exig2 : INSPIREus doit améliorer la navigation des apprenants dans le cours.

Exig3 : INSPIREus doit améliorer la communication entre les apprenants.

Le système doit offrir un environnement d'apprentissage en ligne favorisant les échanges et les interactions entre les apprenants. Par exemple, en créant des sujets d'intérêt sur le forum ou en les guidant vers des groupes d'apprentissage plus compatibles.

Exig4 : INSPIREus doit réguler la liberté donnée aux apprenants dans le contrôle de son apprentissage.

Plusieurs travaux de recherche (Devoper & Quintin, 1992 ; Specht, 1998) ont étudié l'impact de la liberté donné aux apprenants dans le contrôle de son apprentissage, notamment la liberté de changer son niveau d'apprentissage, d'activer / désactiver le comportement adaptatif du système. Ces études révèlent que les apprenants ne savent généralement pas ce qu'ils veulent et ne maîtrisent pas leurs compétences. Par conséquent, la décision de rendre le modèle d'apprenant accessible aux apprenants peut entraîner des échecs et des décisions inexactes de la part des apprenants. Néanmoins, le fait de les restreindre durant leurs apprentissages peut également avoir un impact négatif leur poussant à perdre confiance dans le système adaptatif et à cesser de l'utiliser. Par conséquent, le degré de liberté doit dépendre du niveau de connaissances des apprenants et du temps passer dans les supports pédagogiques. Par exemple, si un apprenant passe beaucoup de temps à apprendre sans progrès et qu'il a désactivé le comportement adaptatif du système, le système doit le dissuader de désactiver l'option adaptative par des messages ou limiter le degré de son contrôle sur son apprentissage.

3.2 Approche générale : RMAS

Cette section présente notre approche de monitoring des exigences en ligne, appelée RMAS (en anglais, Requirements Monitoring for Adaptive Systems). La Figure 17 donne un aperçu général de

l'approche qui est structurée en deux phases : la phase de spécification des exigences du monitoring, et la phase de monitoring en ligne.

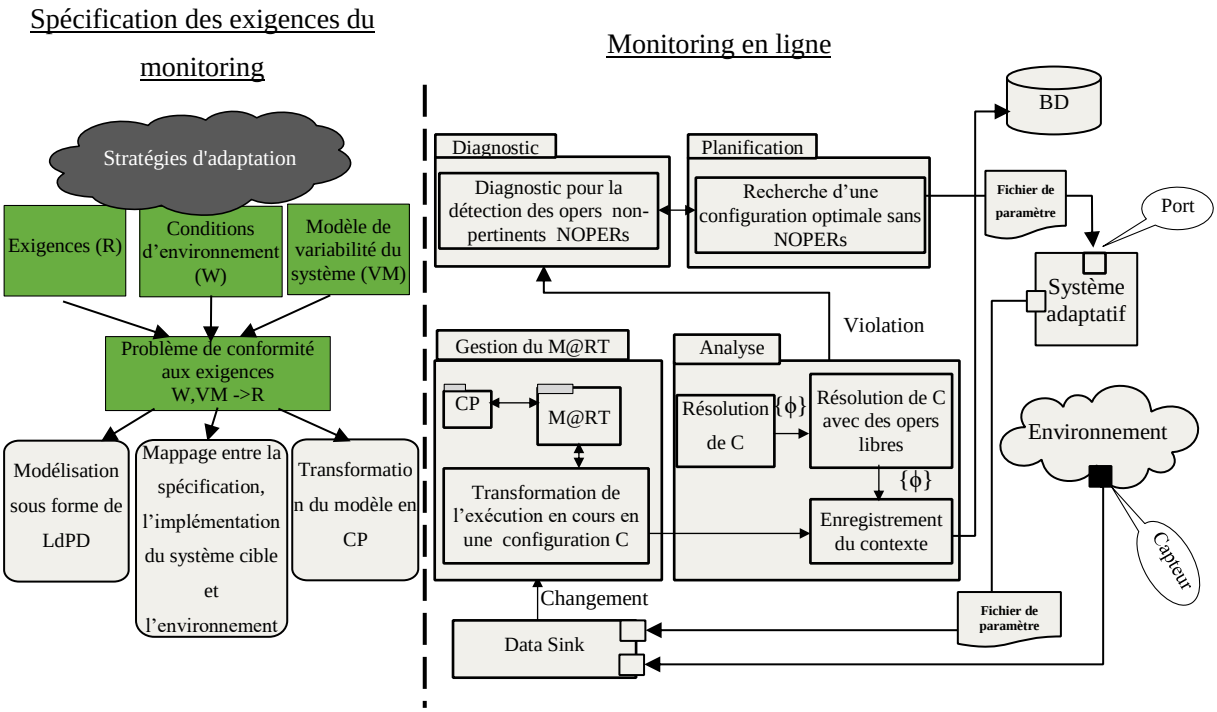


Figure 17 : aperçu général de l'approche du monitoring des exigences en ligne

La phase de spécification des exigences du monitoring vise à représenter les décisions d'assemblages d'exigences (R) sur la base des stratégies d'adaptation du système cible.

Ce processus d'assemblage est reconnu par sa difficulté qui émane de la complexité des SASs et de leurs environnements d'exécution ainsi que de nombreuses dépendances entre les exigences qui font qu'une exigence peut aller à l'encontre d'autres exigences.

La plupart des méthodes existantes permettent une spécification procédurale basée plus sur les règles de décisions que sur les exigences. Cela implique une énumération exhaustive de toutes les règles. Ces dernières augmentent d'une façon exponentielle avec le nombre d'exigences considérées.

Pour pallier ces difficultés, la formulation de la spécification dans le cadre de cette thèse est basée sur deux stratégies :

(1) La définition d'une spécification de haut niveau orientée but qui traduit les décisions d'assemblages d'exigences sous forme d'une LdPD. Cette dernière contribue à capturer la variabilité du système (VM) et de son environnement d'exécution (VW) et les connecte aux exigences (R) à l'aide d'une approche ascendante, où les buts du haut niveau sont introduits pour justifier les buts de bas niveaux décrivant les sous-systèmes et les composants du système.

(2) La génération d'une spécification formelle, à partir du modèle crée, sous forme d'un programme logique de contraintes CP. Cette génération est faite sur la base des règles de transformation définies

dans le codage REFAS-CP (voir section 3.2.1.3). Ce faisant, notre but est de tirer profit de la programmation par contraintes qui permet d'enrichir la spécification par l'ajout de contraintes à tout moment, et de déléguer la complexité de la résolution des dépendances entre les différentes contraintes à un solveur de contraintes. Ainsi, les activités d'analyse des exigences et de planification de configurations optimales à l'exécution peuvent être automatisées sans l'obligation de définir des aspects opérationnels de la résolution.

La phase du monitoring en ligne est assurée par quatre composants qui ont le comportement préconisé par la boucle MAPE-K : un composant de gestion du M@RT qui reflète les changements surveillés sur le modèle de la spécification. Ces changements sont de deux types, d'une part, la configuration du système qui traduit les stratégies d'adaptation et d'autre part les changements des conditions d'environnement d'exécution qui peuvent influencer le comportement du système. Un composant d'analyse qui vérifie la conformité du système aux exigences sur la base des données collectées. Si le système ne satisfait pas les exigences, le composant d'analyse déclenche la planification des remédiations pour faire évoluer le système. Un composant de diagnostic qui analyse les causes des non-conformités retrouvées. Et finalement, un composant de planification qui détermine les modifications à suivre pour satisfaire au mieux les exigences. Dans le cas d'impossibilité de remédiation, les informations d'environnement sont stockées dans une base de données, pour des mises à jour de la spécification en mode hors ligne. Ce faisant, notre approche supporte l'incertitude externe liée aux changements d'environnement.

Notez que le but ultime ici étant le monitoring des exigences et la planification des remédiations. Aucun mécanisme pour l'exécution de ces remédiations n'est programmé. Cette tâche est supposée assurer par les administrateurs du système cible et n'est pas une préoccupation dans cette thèse.

3.2.1 Spécification des exigences du monitoring

Cette section décrit les processus de spécification des exigences et de sa transformation en un programme de contraintes.

3.2.1.1 Modélisation des exigences

Une spécification des exigences définit le comportement attendu du système à partir des exigences à surveiller. Elle constitue la base du raisonnement du monitoring en ligne. Notre objectif est de définir des politiques d'analyse et d'optimisation qui définissent un comportement sensible au contexte qui satisfait au mieux les exigences en temps réel.

À cet objectif, nous exploitons le paradigme LdPD qui se base sur l'analyse de la variabilité pour capturer les différentes parties dynamiques du système qui s'adapte aux changements de l'environnement d'exécution. L'adaptation est définie par des politiques de reconfiguration qui détermine la configuration du système qui satisfait les changements en temps réel.

Nous proposons une modélisation guidée par le langage REFAS et sa vision multi-vues qui favorise la modélisation de la variabilité du système et de son environnement d'exécution sous la forme des graphes dans lesquels les arcs définissent les contraintes décrivant les règles de dépendance entre les points de variation. Nous définissons dans ce qui suit les différentes vues de REFAS, notamment la vue de variabilité, la vue des soft goals, la vue de contexte et la vue La vue « satisficing soft goals » :

La vue de variabilité est utilisée pour modéliser le comportement du système, c.à.d. ce qu'il fait et le relie aux exigences non-fonctionnelles sous forme de buts. Cette vue représente l'ensemble des chemins possibles pour l'opérationnalisation d'un but racine par un processus de ET/OU raffinement en sous-buts. Le processus de raffinement s'arrête lorsque tous les buts « feuilles » sont opérationnels.
La Figure 18 illustre la vue de variabilité du système INSPIREus. Elle part d'une racine qui représente le but principal « supporter le processus d'apprentissage » qui est raffiné en quatre sous-buts « personnaliser la présentation des ressources matérielles », « supporter le séquencement de ces ressources », « encourager la collaboration », et « supporter le partage de contrôle entre le système et les apprenants ». Le premier but est raffiné en trois buts feuilles : « personnaliser les ressources du niveau *rappeler* », « personnaliser les ressources du niveau *utiliser* », et « mettre à jour le style d'apprentissage ». Chaque but feuille est assigné à un ou plusieurs opérations, appelé OPER. L'ensemble des opérations choisies détermine la configuration du système en cours.

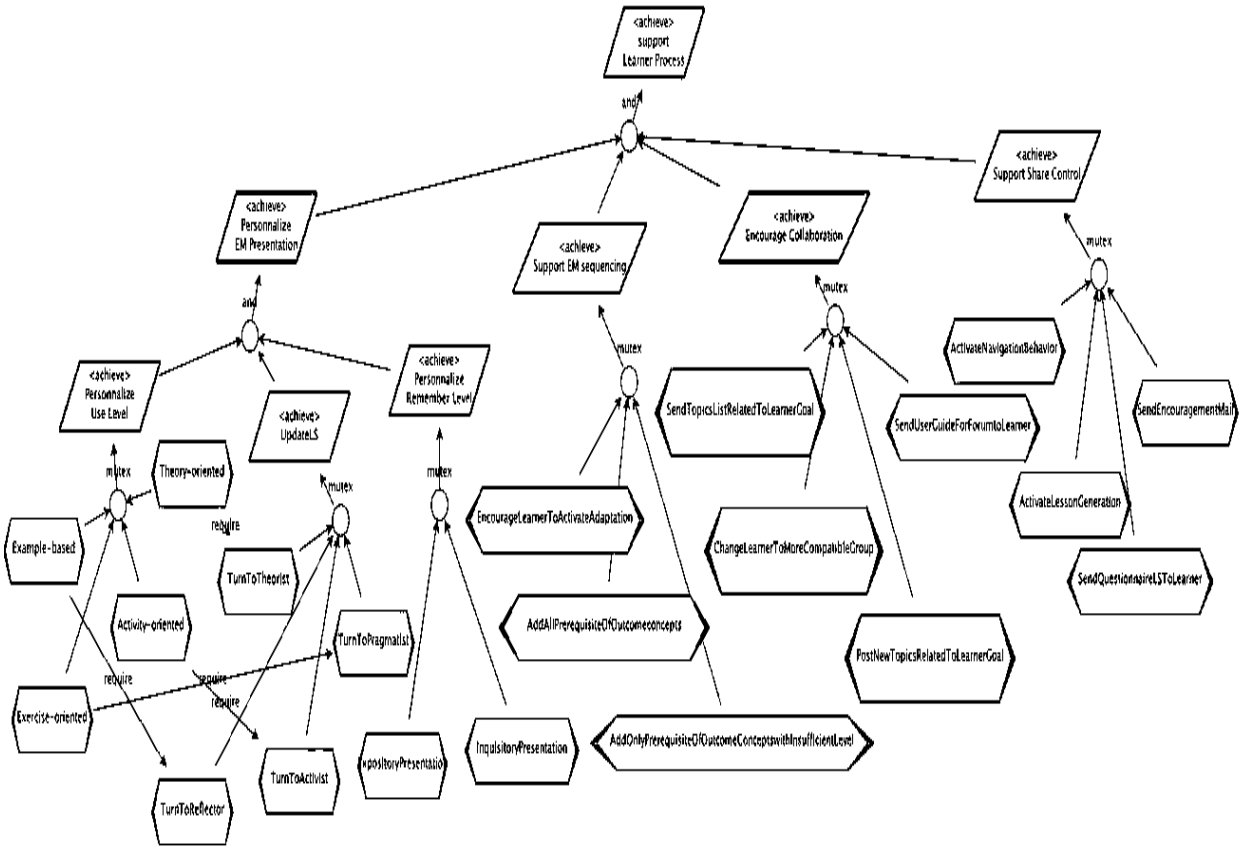


Figure 18 : la vue de variabilité appliquée à INSPIREus

Par exemple, le but « personnaliser les ressources du niveau *utiliser* » peut-être opérationnaliser par « une présentation orientée exemple », « une présentation orientée activité », « une présentation orientée exercice », ou (exclusive) « une présentation orientée théorie ». En plus des relations de décomposition, des contraintes telles que « require » et « exclude » peut être utilisée pour restreindre ou inclure des OPERs. Ce type de contraintes peut réduire considérablement l'espace de configuration lors de la planification des configurations alternatives. Par exemple, si la « présentation orientée théorie » est adoptée alors le style d'apprentissage doit être mis à « *réflexif* ».

La vue des soft goals est utilisée pour modéliser les exigences non-fonctionnelles sous forme de soft goals. Cette vue représente les soft goals et les relations de contribution entre eux. Contrairement à un but « hard », la satisfaction d'un soft goal est jugée par un niveau « suffisamment bon », on parle de « satisficement » qui implique souvent un compromis avec d'autres soft goals.

La Figure 19 présente les soft goals liés aux exigences {exig1, exig2, exig3, exig4}. Par exemple, le soft goal « optimiser l'exactitude des contenus du niveau *rappeler* pour les apprenants RP » (Réflexifs ou Pragmatiques) ou le soft goal « optimiser l'exactitude des contenus du niveau *rappeler* pour les apprenants AT » *(*Activistes ou Théoricien*)* contribue à la satisfaction du soft goal « obligatoire » « optimiser l'exactitude des contenus du niveau *rappeler* » par un niveau de satisfaction 4.

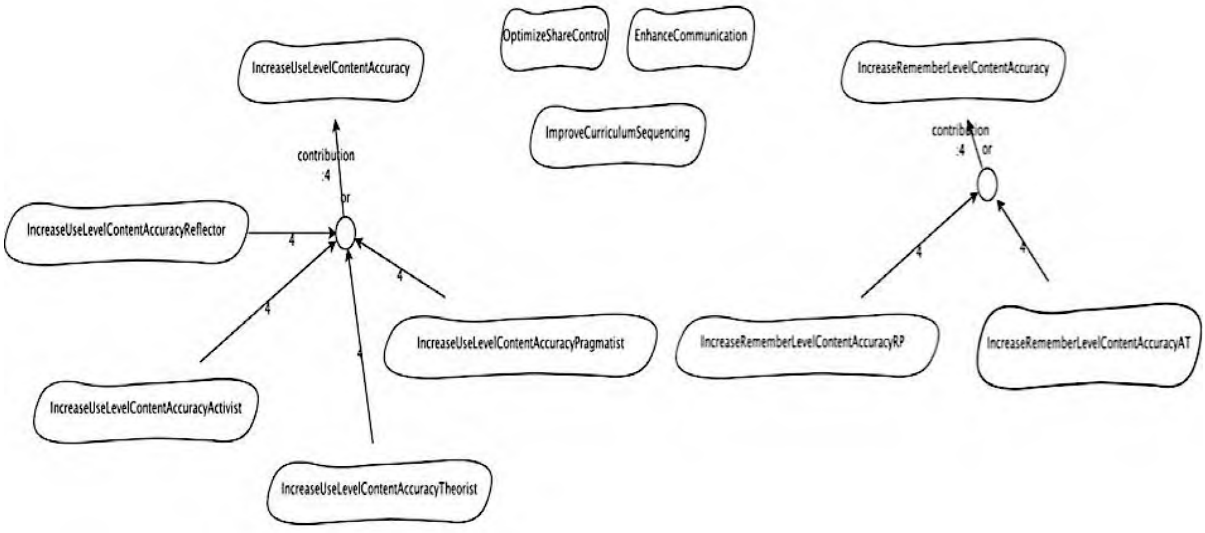


Figure 19 : la vue des soft goals appliquée à INSPIREus

La vue de contexte est utilisée pour définir les variables de contexte liées à l'environnement et au système et qui influencent la satisfaction des exigences. Ces variables prennent leurs valeurs dans un domaine fini.

La Figure 20 décrit les variables de contexte dans notre cas d'étude. Par exemple, les variables exemple-rank, exercie-rank, activity-rank et theory-rank sont des entiers qui prennent leurs valeurs sur une échelle de 1 à 4. Elles indiquent chacune un ordre lié au nombre de fois qu'un apprenant a accédé

à un type de ressource particulier (exemple, exercices, activité, théorie respectivement) par rapport aux autres types. L'intérêt de ces variables est qu'elles permettent d'identifier le style d'apprentissage d'un apprenant. Supposons par exemple que pour un apprenant exemple-rank = 1, cela indique que cet apprenant utilise principalement les exemples pendant son apprentissage et qu'il aime apprendre en observant et en réfléchissant à travers ces exemples. Nous concluons alors que son style d'apprentissage est réflexif.

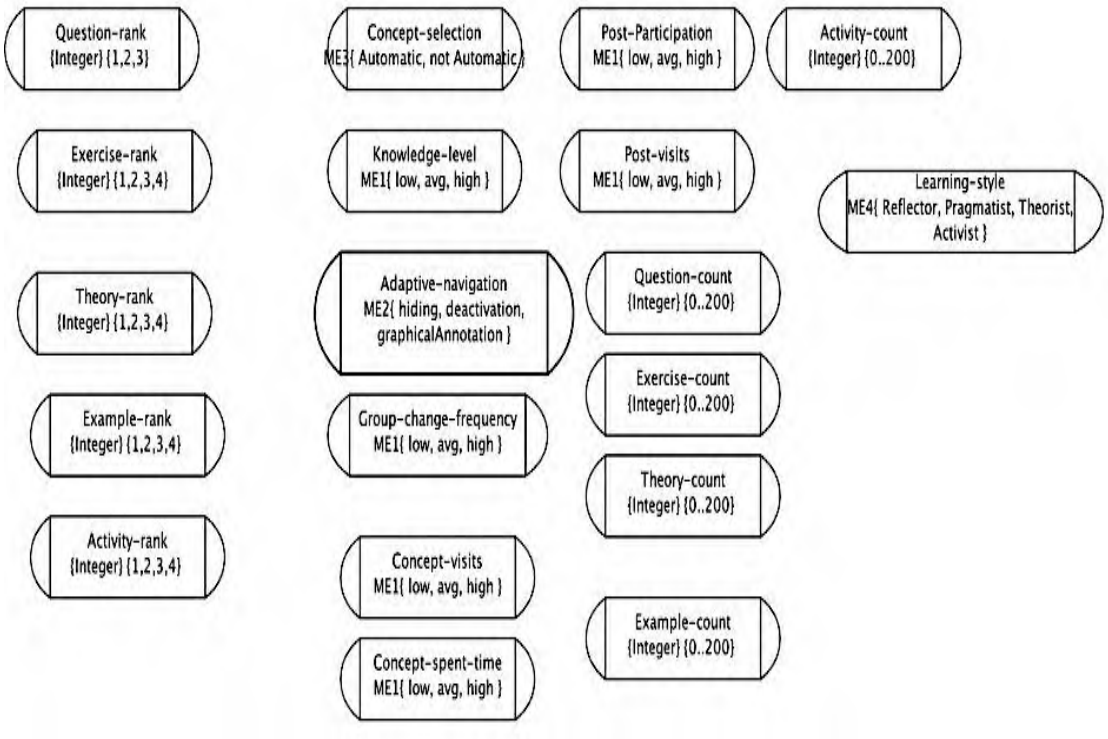


Figure 20 : la vue de contexte appliquée à INSPIREus

La vue « satisficing soft-goals » sert de liaison entre les composants des différentes vues. En effet, la vue permet à l'analyste de raisonner sur la variabilité de l'environnement par le biais de « soft-dépendances » et la variabilité du système par le biais de « claims » et leur implication sur la satisfaction des soft-goals. Un *soft-dépendance* est un prédicat qui indique les niveaux de satisfaction requis des soft-goals pour les conditions de contexte qui leur sont associées. Un *claim* est un prédicat qui indique le niveau de satisfaction du soft goal « attendu » de l'OPER qui lui est associée. Ces niveaux de satisfaction prennent leurs valeurs sur une échelle de 1 à 4, allant de l'insatisfaction (1) à la satisfaction complète (4). Le but ici est de fournir au monitoring une base pour recalculer continuellement les niveaux de satisfactions requis à chaque changement de contexte et d'évaluer si les OPERs qui constituent la configuration en cours satisfont ce contexte et, si nécessaire de planifier des configurations qui optimisent au mieux les soft-goals à travers un compromis entre les contraintes portant sur les *soft-dépendances,* les *claims* et les contraintes spécifiques liées à la composition des OPERs. Dans le cadre de cette thèse, nous proposons d'implémenter ce raisonnement sous forme d'un

programme de contraintes, où l'évaluation et la planification sont automatisées par la résolution du programme de contraintes à l'aide des solveurs de contraintes standards.

Par souci de brièveté et de clarté, la Figure 21 présente un extrait de la vue *satisficing soft-goals* lié à l'exigence exig1.

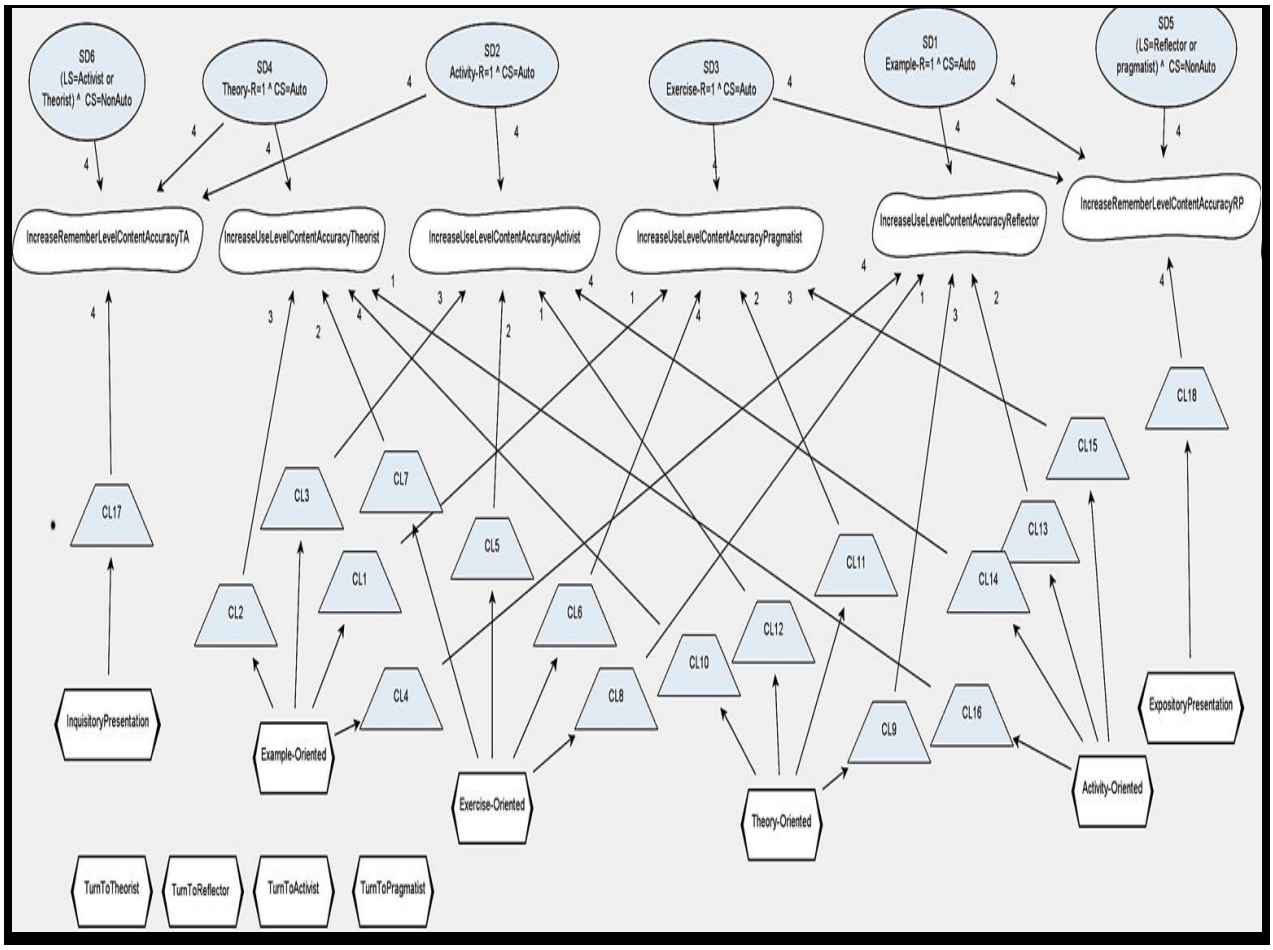


Figure 21 : La vue satisficing soft-goals appliquée à INSPIREus

Le soft-dépendance SD1 spécifie que si exemple-rank = 1 ET que l'apprenant a activé la sélection automatique des concepts, alors le soft goal « optimiser la précision des contenus du niveau *Utiliser* » doit être totalement satisfait (niveau de satisfaction = 4). Ceci est réalisable grâce à l'OPER « *présentation orientée exemple* » comme indiqué par le claim CL4.

3.2.1.2 Support d'incertitude

Alors qu'il est impossible de créer des solutions pour des événements imprévus, il sera possible de se préparer à ses éventuelles occurrences. Cela nécessite une connaissance des événements passés, et la capacité de décrire et de présenter des scénarios concevables pour réagir lorsqu'un événement imprévu se produit (Andersen, 2003). Dans notre contexte, l'imprévisibilité émane de l'incertitude inhérente aux systèmes auto-adaptatifs qui découle de leur complexité et leur variabilité, ainsi qu'aux changements fréquents dans l'environnement d'exécution Dans cette thèse, nous traitons

principalement l'incertitude liée (i) aux changements dynamiques des exigences et (ii) aux adaptations qui n'étaient pas prévues lors de la conception du système.

La gestion de l'incertitude dans le cadre de cette thèse commence dès le moment de formulation de la spécification. En effet, nous proposons de modéliser le monitoring sous forme de ligne de produits logiciels dynamique, et donc de raisonner sur la variabilité du système cible et de son environnement d'exécution. La gestion de la variabilité permet de concevoir des scénarios autour de la variabilité et de prévoir ainsi des éventuels besoins d'adaptation. D'autre part, nous avons conçu des marqueurs d'incertitude en utilisant des claims, liées aux points variables du système cible, pour représenter des hypothèses sur les exigences. En outre, Nous avons défini les conditions nécessaires et suffisantes pour l'activation du monitoring des exigences pour réduire le coût du monitoring. À l'exécution du système, Le monitoring permet de surveiller et vérifier en temps réel les claims selon des conditions d'environnement et contribuera alors à atténuer le risque que le système se comporte de manière inappropriée, et aide le système à évoluer avec le temps.

Cependant, comme nous ne pouvons pas prédire totalement le futur. Nous avons prévu de stocker les situations nouvelles détectées par le monitoring, liées aux adaptations qui n'étaient pas prévues lors de la conception pour qu'elles soient traitées dans des phases de maintenance.

3.2.1.3 Formalisation de la spécification

La programmation logique par contraintes est une technique issue de la programmation logique et de l'intelligence artificielle pour résoudre des problèmes combinatoires complexes d'optimisation et de planification. Il existe aujourd'hui un grand nombre de solveurs robustes qui automatisent la résolution par des algorithmes efficaces de recherche et de propagation de contraintes.

La modélisation du problème par contraintes (CP) est définie par un triplet (V, D, C) qui représente un ensemble de variables V qui prennent leurs valeurs dans un domaine D, et un ensemble de contraintes (formules logiques) qui définissent la relation entre les variables V.

Une solution d'un CP est une affectation de valeurs à un ensemble de variables dans V, qui est conforme à toutes les contraintes dans C. Un problème d'analyse de la conformité aux exigences peut être alors assimilé à un problème de satisfaction de contraintes.

Dans cette section, nous décrivons comment nous exploitons les similitudes entre la programmation logique par contraintes et le problème de conformité aux exigences pour définir une spécification formelle par contraintes à partir des modèles REFAS. La spécification permet de définir les contraintes qui régissent la composition des éléments d'une configuration, notamment les contraintes d'environnement, les contraintes de compatibilité et les contraintes de contribution à la satisfaction de soft-goals définies par les soft-dépendances et les claims.

Un codage REFAS-CP est proposé pour définir les règles de transformation des modèles REFAS en un programme logique de contraintes en domaines finis. Tout au long des étapes du codage, nous

fournissons des définitions des méta-modèles REFAS et les formalisons à l'aide des règles de transformation.

Notre codage REFAS-CP s'effectue en deux étapes : Premièrement, à partir d'une spécification S par REFAS, nous dérivons les variables V et leurs domaines finis D. Deuxièmement, nous dérivons les contraintes Co entre ces variables sous forme d'une formule de logique propositionnelle utilisant des connecteurs logiques entre les variables, selon les règles de transformation suivantes :

Définitions 1 : un méta-concept C désigne l'ensemble des concepts du langage REFAS, notamment goal, OPER, soft goal, claim, variable de contexte, soft-dépendance, assumption, asset et concern level. Ces concepts sont les instances possibles de C.

Avant de continuer les définitions, Nous allons présenter quelques notations :

Soient InstG, InstOPER, InstSG, InstCL, InstX, InstSD, InstAsump, InstAsset et InstConc, des ensembles des variables représentant les éléments de la spécification S qui sont respectivement de type goal, OPER, soft goal, claim, variable de contexte, soft-dépendance, asssumption, asset et concern level, tels que :

InstG = $\{g_i, ..., gn\}$.

InstOPER = $\{oper_i, ..., operm\}$.

InstSG = $\{sg_i, ..., sgk\}$.

InstCL= $\{cl_i, ..., clr\}$.

InstX= $\{x_i, ..., xs\}$.

InstSD = $\{sd_i, ..., sdt\}$.

InstAsump = $\{asump_i, ..., asumpu\}$.

InstAsset = $\{asset_i, ..., assetv\}$.

InstConc = $\{conc_i, ..., concs\}$.

Et soit également InstConcept un ensemble des variables représentant toutes les instances des concepts constituant la spécification S, tel que :

InstConcep =

$\{InstG \cup InstOPER \cup InstX \cup InstSG \cup InstCL \cup InstSD \cup InstAsump \cup InstAsset \cup InstConc\}$.

Règle 1 : Chaque élément de InstConcept définit une variable booléenne $instconcept_i$, où $instconcept_i \in D= \{0,1\}$ indique si la variable est sélectionnée ou désélectionnée dans une configuration du système.

Définition 2 : un méta-attribut A est une caractéristique d'un méta-concept C ou d'une méta-association R (voir définition 3) prenant ses valeurs dans un domaine D.

Notez que nous nous intéressons ici aux instances clés de A, les instances comme Nom et Id ne sont pas représentées dans le CP. Ainsi, les attributs considérés de C sont : ExpectedLevel, RequiredLevel, ConditionalExpression et SatisficingType. Ils sont définis comme suit :

RequiredLevel ϵ J = {1,2,3,4} est un attribut de la méta-association SD_SG entre un sd_i et un soft goal sg_j qui définit le niveau de satisfaction requis du sg_j dans les conditions de contexte données par sd_i, allant de l'insatisfaction complète (1) à la satisfaction complète (4).

ExpectedLevel ϵ I = {1,2,3,4} est un attribut de la méta-association SG_CL entre un claim cl_i et un soft goal sg_j qui définit le niveau de satisfaction attendu de l' $oper_l$ lié à cl_i, allant de l'insatisfaction complète (1) à la satisfaction complète (4).

ConditionalExpression est une chaîne de caractères qui représente une formule logique valide "Ex". Il peut être un attribut d'un soft-dépendance ou d'un claim.

SatisficingType ϵ K = {0,1, 2} est un attribut d'un soft goal sg_i, où 0 correspond à la contrainte «Achieve as close as possible», 1 correspond à la contrainte " Achieve as low as possible ", et 2 correspond à la contrainte " Achieve as high as possible ".

Soit InstSD_SG, InstSG_CL les variables qui renvoient respectivement aux deux ensembles de toutes les instances des méta associations SD_SG et SG_CL.

Règle 2.1: les attributs *ExpectedLevel* de chaque association $sg_j_cl_i$, *RequiredLevel* de chaque association $sd_k_sg_l$, et SatisficingType de chaque sg_m, définissent respectivement les variables ExpectedLevel_$sg_j_cl_i$, RequiredLevel_$sd_k_sg_l$, et SatisficingType_sg_m qui prennent respectivement leurs valeurs dans I, J et K.

Soient RequiredLevel_sg_j et ObtainedLevel_sg_j les variables qui désignent respectivement le niveau de satisfaction requis et le niveau de satisfaction obtenu du soft goal dans une configuration donnée. Les valeurs de ces deux variables sont déduites respectivement des variables RequiredLevel_$sd_k_sg_l$ et ExpectedLevel_$sg_j_cl_i$. Ces dernières désignent respectivement le niveau de satisfaction requis et attendu d'un soft goal selon les prédicats sd_K et cl_i et dont leurs valeurs ne changent pas d'une configuration à une autre.

Règle 2.2: l'attribut ConditionalExpression de chaque sd_j (respectivement cl_k) définit une contrainte qui active respectivement la sélection/désélection du sd_j (respectivement cl_k) :

Co1 : Ex $\Rightarrow$ sd_j (respectivement cl_k) =1

Règle 2.3 : l'attribut SatisficingType_sg_j de sg_j définit un ensemble de contraintes selon la valeur qu'il peut prendre:

∀ la valeur de SatisficingType_sg_j :

$$\text{Co2: } sd_i = 1 \Rightarrow \text{ObtainedLevel_}sg_j \geq \text{RequiredLevel_}sd_i_sg_j$$

Et selon la valeur de SatisficingType_sg_j, les contraintes suivantes sont définies :

- Si SatisficingType_sg_j= 0 ("Achieve as close as possible") alors:

$$\text{Co3: } sd_i = 1 \Rightarrow \text{RequiredLevel_}sg_j = \text{RequiredLevel_}sd_i_sg_j$$

$$\text{Co4: } cl_i = 1 \Rightarrow \text{ObtainedLevel_}sg_j = \text{ExpectedLevel_}sg_j_cl_i$$

- Si SatisficingType_sg_j= 1 ("Achieve as low as possible") alors:

$$\text{Co5: } sd_i = 1 \Rightarrow \text{RequiredLevel_}sg_j \leq \text{RequiredLevel_}sd_i_sg_j$$

$$\text{Co6: } cl_i = 1 \Rightarrow \text{ObtainedLevel_}sg_j \leq \text{ExpectedLevel_}sg_j_cl_i$$

- Si SatisficingType_sg_j= 2 ("Achieve as high as possible") alors:

$$\text{Co7: } sd_i = 1 \Rightarrow \text{RequiredLevel_}sg_j \geq \text{RequiredLevel_}sd_i_sg_j$$

$$\text{Co8: } cl_i = 1 \Rightarrow \text{ObtainedLevel_}sg_j \geq \text{ExpectedLevel_}sg_j_cl_i$$

Définition 3 : une méta association R est une relation entre des méta-concepts: R⊆ C^n. Les modèles REFAS peuvent être vus comme des graphes orientés. Par conséquent, une instance de R peut être définie par un tuple <Cparent, {Cchild-1, Cchild-2… Cchild-n}> où Cparent et Cchild-i sont les noeuds sources et cibles de l'arête i. Le méta-modèle REFAS prend en charge différents types de méta associations, notamment :

Définition 4: (Contribution [a, b] ⊆ **InstSG** × **InstSG**): Contribution [a,b] est une méta association qui représente des contributions entre les soft-goals. Elle indique que si le niveau de satisfaction attendu 'a' est atteint par le soft goal père cela contribue à la satisfaction du soft goal fils par un niveau 'b'.

Règle 4 : soit $ctr_i =< \text{sg}_\text{p}, \text{sg}_\text{ch} >$ une instance de Contribution[a, b] , où $\text{sg}_\text{p}, \text{sg}_\text{ch} \in$ InstSG et $\text{a}, \text{b} \in \{1..4\}$:

$$Co9: \quad ExpectedLevel_\text{sg}_p = a \implies ExpectedLevel_\text{sg}_{ch} = b$$

Définition 5 : *Means-end* est une méta association qui définit deux types de relation entre des concepts de type Variability Vr : obligatoire et optionnelle.

Optionnelle ⊆ **Vr x Vr** : indique que si le noeud parent est sélectionné alors le noeud fils peut être sélectionné.

Règle 5.1 : Soit $meansOpt_i =< \text{V}_\text{p}, \text{V}_\text{ch} >$une instance de la méta association optionnelle, où $\text{V}_\text{p}, \text{V}_\text{ch} \in Vr$ et Vr= {InstG $InstG \cup InstOPER \cup InstAsump$} :

$$Co10: \ \text{V}_\text{ch} \implies \text{V}_\text{p}$$

Obligatoire ⊆ **Vr x Vr** : indique que si le noeud parent est sélectionné alors le noeud fils doit être sélectionné.

Règle 5.2 : soit $meansOb_i =< V_p, V_{ch} >$ une instance de la méta association obligatoire $V_p, V_{ch} \in Vr$:

$$Co11 : (V_{ch} \implies V_p)\ AND\ (V_p \implies V_{ch})$$

Definition 6 une méta association *Traversal* définit deux types de relation entre des OPERs: Inclure et Exclure

Inclure ⊆ **InstOPER** × **InstOPER** : indique que si un $OPER_i$ est sélectionné alors $OPER_j$ doit être sélectionné.

Règle 6.1 : soit $< OPER_i, OPER_j >$ une instance de la méta association inclure, où $OPER_i, OPER_j \in$ InstOPER : $\quad Co12: OPER_i \implies OPER_j$

Exclure ⊆ **InstOPER** × **InstOPER** : indique que si un $OPER_i$ est sélectionné alors $OPER_j$ ne doit être sélectionné.

Règle 6.2 : soit $< OPER_i, OPER_j >$ est une instance de la méta association Exclure où $OPER_i, OPER_j \in$ InstOPER : $\quad Co13: OPER_i \oplus OPER_j$

Définition 7 : une association *Group* (GA) est une *méta association de type* many-to-many qui limite la cardinalité des différentes *MetaPairwiseAssociations* (one-to-one méta association entre des méta-concepts C) : GA ⊆ InstConcept x InstConcept_{alt}, où InstConcept_{alt} désigne toutes les alternatives possibles à partir du noeud source.

GA définit 4 types de décomposition: OU, ET, Mutex et Range [m,n].

OU : indique qu'au moins une des alternatives possibles doit être sélectionnée.

Règle 7.1: soit $< c, \{Alt_1 \dots, Alt_n\} >$ une instance de OU, où $c \in$ InstConcept :
Si c est sélectionné alors $\exists\ Alt_i \in \text{InstConcept}_{alt},\ tq\ < C, Alt_i >$ est selectionnée

$$Co14: C \Leftrightarrow \sum_{i=0}^{n} Alt_i \leq n$$

ET : indique que toutes les alternatives doivent être sélectionnées

Règle 7.2: soit $< c, \{Alt_1 \dots, Alt_n\} >$ une instance de ET, où $c \in$ InstConcept :
Si C est sélectionné alors $\forall\ Alt_i \in \text{InstConcept}_{alt}\ < C, Alt_i >$ est sélectionné

$$Co15: C \Leftrightarrow \sum_{i=0}^{n} Alt_i = n$$

Mutex : indique que seulement une des alternatives possibles doit être sélectionnée

Règle 7.3: soit $< c, \{\text{Alt}_1 \dots, \text{Alt}_n\} >$ une instance de Mutex, où $c \in \text{InstConcept}$ et $\text{Alt}_i \in \text{InstConcept}_{\text{alt}}$.

$$\text{Si} < c, \text{Alt}_i > \text{est sélectionné, alors} \forall\ \text{Alt}_j \in \text{Instconcept}_{\text{alt}} \setminus \{\text{Alt}_i\} < c, \text{Alt}_j > \text{n'est pas sélectionnée}$$

$$\text{Co16: C} \Leftrightarrow \sum_{i=0}^{n} \text{Alt}_i = 1$$

Range [m,n] définit un nombre minimum 'm' et un nombre maximum 'n' des alternatives qui peuvent être sélectionnées.

Règle 7.4: soit $< c, \{\text{Alt}_1 \dots, \text{Alt}_n\} >$ une instance de Range[m,n],

où $c \in \text{InstConcept}$ et $\text{Alt}_i \in \text{InstConcept}_{\text{alt}}$:

Si c est seléctionné $\Leftrightarrow$ for $i = 1..n$, Si $< c, \text{Alt}_i >$ est seléctionée alors $\min \sum_{i=0}^{n} Alt_i = m$ et $\max \sum_{i=0}^{n} Alt_i = n$:

$$\text{Co17: C} \Leftrightarrow \sum_{i=0}^{n} \text{Alt}_i \geq \text{m} \wedge \sum_{i=0}^{n} \text{Alt}_i \leq \text{n}$$

Définition 8 : le programme de contraintes global, noté CP_global, spécifie la conformité aux exigences à partir des règles 1 à 7.4. Et donc à cette spécification que toutes les configurations du système cible doivent être conformes (réponse à la question QR1).

Règle 8 : Soit CP_global = (V, D, CO), où

$$\begin{cases} V = \text{InstConcept} \cup V1 \cup V2 \\ Co = \bigcup_{i=1}^{17} Co_i \end{cases}$$

Avec $V1 = \cup_{i=1}^{n} \{ExpectedLevel_sg_i, RequiredLevel_sg_i, SatisficingType_sg_i\}$, n=Card (InstSG),

$V2 = \cup_{j=1}^{m}\{\text{ExpectedLevel}_sg_cl_j\} \cup_{k=1}^{l}\{\text{RequiredLevel}_sd_sg_k\}$, m=Card(InstSG_CL), l=Card(InstSD_SG), et D est l'ensemble des domaines finis, définis dans les règles 1 à 7.4.

3.2.2 Monitoring des exigences en ligne

Dans la section précédente, nous avons défini des règles de transformation pour construire une spécification des exigences sous forme d'un programme de contraintes sur des domaines finis. Dans cette section, nous montrerons comment le programme généré, noté CP_global, peut être utilisé dans les activités du monitoring assurées en ligne par quatre composants: le composant de la gestion du M@RT, le composant d'analyse des exigences, le composant de diagnostic des non-conformités et le composant de planification des solutions d'optimisation.

3.2.2.1 Manipulation du M@RT

La conscience des exigences à l'exécution est une évolution de la notion de réflexion informatique par Smith (1982) et Maes (1987) qui désigne la capacité d'une entité de se représenter et d'agir de manière autonome. Cette notion a été reprise dans le domaine d'ingénierie des exigences pour permettre aux systèmes logiciels de raisonner sur leurs exigences par le biais du M@RT. Ce dernier sert à maintenir la relation entre les exigences du système et le code qui l'implémente à l'aide des entités qui sont capables de révéler des informations sur elles-mêmes pendant l'exécution du système.

Dans cette thèse, ces deux notions ont été exploitées pour permettre une flexibilité de monitoring lors des changements dans les exigences ou l'état du système. En effet, dans notre approche, les activités d'analyse autour des exigences sont basées principalement sur un programme de contraintes qui est généré automatiquement à partir d'une spécification par REFAS. Ainsi, le monitoring sera en mesure de supporter de nouvelles exigences en apportant des changements au niveau de la spécification sans toucher à sa structure qui, elle, restera inchangée.

Les M@RT dans notre cas, le méta-modèle REFAS est représenté sous forme d'un ensemble de classes décrivant les éléments du méta-modèle. La spécification par REFAS est stockée dans la mémoire sous forme de graphes et des instances des classes du méta-modèle REFAS.

A l'exécution, la manipulation du M@RT est accordée à un composant de gestion de modèles qui a la responsabilité de :

a) Transformer le M@RT en un programme de contraintes à l'aide du codage REFAS-CP. Cette génération est faite une seule fois, à chaque déploiement d'une nouvelle spécification en cas de changement dans les exigences ou dans l'implémentation du système cible. Ce faisant, le monitoring sera en mesure de supporter l'évolution du système cible.

b) Enrichir continuellement le M@RT avec les informations collectées sur le système et son environnement depuis les data Sink (voir Figure 17). À cette fin, les méta-concepts du méta-modèle REFAS sont étendus avec des attributs de mappage sous forme de fonctions mathématiques décrivant la relation entre les données collectées et les entités constituant le M@RT, principalement ceux décrivant le système (OPERs) et son environnement (les variables du contexte).

c) Extraire la configuration en cours du système à partir du M@RT, à chaque changement qui déclenche l'analyse des exigences. Cette configuration sera utilisée par le composant d'analyse pour vérifier la conformité du système aux exigences.

3.2.2.2 Analyse des exigences

Partant du programme de contraintes CP_global = (V, D, CO) qui traduit la spécification des exigences.

Le composant d'analyse des exigences assure la vérification de la conformité du système à la spécification des exigences à l'aide d'un solveur de contraintes au moyen de ses mécanismes de

résolution de contraintes, de substitution et d'unification des variables. Comme le montre la Figure 17, le composant d'analyse reçoit la configuration en cours C (totale ou partielle) du système cible et résout le programme CP_global à l'aide d'un solveur de contraintes (par exemple SWI-Prolog) selon les étapes suivantes :

1- La configuration du système est transformée en une solution partielle S du programme CP_global (voir l'algorithme 1, chapitre 4), à l'aide des affectations partielles des variables de CP_global qui représentent les OPERs et les variables de contexte dans REFAS, à savoir les deux ensemble InstOPER et InstX (voir définition 1, section 3.2.1.3).

2- Le composant d'analyse prépare la requête pour vérifier la solution S et lance l'exécution du programme à l'aide d'un solveur de contraintes. La configuration est dite valide si les éléments la constituant satisfont toutes les contraintes CO. Cela signifie que le système est conforme aux exigences spécifiées. Si la résolution n'admet aucune solution, la configuration est alors dite invalide. Cela signifie que le système n'est pas conforme aux exigences spécifiées.

3- le composant d'analyse décide de l'action suivante selon le résultat de l'étape 2, 2 cas de figure alors se présentent : (i) Si le système est conforme aux exigences, le composant d'analyse termine l'analyse sans faire appel au composant de diagnostic, et attend de nouvelles notifications du composant de gestion de modèles. (ii) Dans le cas contraire, le processus de diagnostic est lancé.

3.2.2.3 Diagnostic des causes des non-conformités

La résolution des contraintes par solveur peut amener à découvrir des problèmes de conformité aux exigences. Le cas échéant, le composant de planification est lancé pour trouver la meilleure configuration qui satisfait le contexte en cours. Cependant, cette tâche n'est pas évidente quand le solveur de contrainte est incapable de déterminer un ensemble fini de solutions au problème de contraintes, et n'est pas en mesure d'indiquer la nature des problèmes rencontrés. Dans ce cas, un diagnostic est nécessaire pour la détection des composants qui empêchent la satisfaction des exigences pour fournir une explication de la nature des non-conformités, et pouvoir interroger à nouveau le solveur de contraintes.

Dans le cadre de cette thèse, 5 types de non-conformité sont considérés :

Type 1: des conditions d'environnement qui ne sont pas prévues lors de la spécification des exigences.

Type 2: la configuration ne satisfait pas suffisamment les soft goals.

Type 3: la configuration manque des OPERs requis par d'autres OPERs (liés par des relations « and » ou « require »).

Type 4: la configuration contient des OPERs incompatibles (liés par des relations « exclude » ou « mutex »).

Type 5: la configuration ne satisfait pas tous les buts. Cela peut se produire si certains buts obligatoires ne sont pas opérationnels dans la configuration en cours. Les causes possibles peuvent

être que les opérations en question sont supprimées, non-fonctionnelles, ou qu'elles ne sont pas sélectionnées lors des actions d'adaptations.

La méthodologie de diagnostic de ces types de non-conformité est la suivante :

Etape 1 : le composant de diagnostic vérifie d'abord l'occurrence de la non-conformité du type 1, c.-à-d. celle liée à la non-prise en charge des conditions d'environnements par la spécification. Cela se fait par la résolution du programme CP_global avec seulement les variables de contexte observées et les soft goals à satisfaire. Le but est de libérer les buts afin de retourner toutes les configurations prévues sous ces conditions. Si la résolution retourne des solutions, le composant continue le diagnostic des autres types de non-conformité sur la base des solutions retournées, notées P. Dans le cas contraire, le composant de diagnostic termine le processus diagnostic et enregistre les conditions de contexte observées dans une base de données. Cette dernière sera consultée régulièrement par les administrateurs du système cible, afin de planifier des modifications au niveau du modèle de spécification pour inclure les conditions de l'environnement, et, si nécessaire, ajouter de nouvelles contraintes pour satisfaire les exigences par rapport à ces nouvelles conditions.

Etape 2 : le composant de diagnostic vérifie les non-conformités de type 2 dans le but de retourner un rapport décrivant le verdict des claims, des soft-dépendances, des buts, des OPERs et des soft goals liés à S.

Le scénario suivant est utilisé pour illustrer la méthodologie de diagnostic.
Supposons que S est la configuration en cours du système INSPIREus telle que :

$\boldsymbol{S} = (\text{OPER2} = \text{Exercie} - \text{based}, \text{OPER10} = \text{Inquisitory Preparation}, \text{OPER8} = \text{Turn to Pragmatist})$
Et les conditions du contexte sont : $\text{Exercise} - \text{rank} = 3, \text{theory} - \text{rank} = 1, \text{Exemple} - \text{rank} = 2, \text{Activity} - \text{rank} = 4,$
Concept-selection=automatic et Learning style=pragmatiste, et Satisficing-Type=0

Partant du modèle M@RT, S la solution partielle qui viole les contraintes de CP_global et P l'ensemble des solutions fournies par le composant d'analyse.
- D'abord, le composant de diagnostic déduit les affectations possibles à partir des valeurs des variables collectées à partir du M@RT illustré dans les Figures 2, 3, 4 et 5.

Les affectations suivantes sont alors déduites à partir des valeurs du scénario:

- $\text{OPER2} = 1 \implies \text{CL5} = 1 \text{ AND CL6} = 1 \text{ AND CL7} = 1 \text{ AND CL8} = 1$
- $\text{OPER10} = 1 \implies \text{CL17} = 1$

- $\text{CL5} = 1 \implies \text{ObtainedLevel_SG}_6 = 2 \text{ AND ExpectedLevel_SG6_CL5} = 2$
- $\text{CL6} = 1 \implies \text{ObtainedLevel_SG}_8 = 4 \text{ AND ExpectedLevel_SG8_CL6} = 4$
- $\text{CL7} = 1 \implies \text{ObtainedLevel_SG}_7 = 2 \text{ AND ExpectedLevel_SG7_CL7} = 2$
- $\text{CL8} = 1 \implies \text{ObtainedLevel_SG}_5 = 1 \text{ AND ExpectedLevel_SG5_CL8} = 1$

- $CL17 = 1 \Longrightarrow ObtainedLevel_SG_4 = 4$ AND $ExpectedLevel_SG4_CL17 = 1$
- $Theory - rank = 1$ AND $Concept\ Selection = Automatic \ => \ SD4 = 1$
- $SD4 = 1 \Longrightarrow SG7 = 1$ AND $RequiredLevel_SG7 = 4$ AND $RequiredLevel_SD4_SG7 = 4$
- $Learning\ style = pragmatiste$ AND $Concept\ Selection = automatic \ => \ SD6 = 1$
- $SD6 = 1 \Longrightarrow SG4 = 1$ AND $RequiredLevel_SD6_SG4 = 4$ AND $RequiredLevel_SG4 = 4$

- Après, à partir du type de satisfaction dans chaque soft-goal est 0, le composant de diagnostic génère les contraintes qui conditionnent la satisfaction des soft goals.

Les contraintes suivantes sont alors déduites :

- $CL5 = 1 \Longrightarrow ObtainedLevel_SG_6 = ExpectedLevel_SG_6_CL_5$
- $CL6 = 1 \Longrightarrow ObtainedLevel_SG_8 = ExpectedLevel_SG_8_CL_6$
- $CL7 = 1 \Longrightarrow ObtainedLevel_SG_7 = ExpectedLevel_SG_7_CL_7$
- $CL8 = 1 \Longrightarrow ObtainedLevel_SG_5 = ExpectedLevel_SG_5_CL_8$
- $CL17 = 1 \Longrightarrow ObtainedLevel_SG_4 = ExpectedLevel_SG_4_CL_{17}$
- $SD4 = 1 \Rightarrow RequiredLevel_SG_7 = RequiredLevel_SD_4_SG_7$
- $SD6 = 1 \Rightarrow RequiredLevel_SG_4 = RequiredLevel_SD_6_SG_4$

- Ensuite, le composant de diagnostic vérifie les contraintes générées afin de déduire le verdict des variables liées aux soft goals :

Soient :

VerdictSG = {Vsg_1, ..., Vsgk} l'ensemble des variables booléennes Vsg_i qui décrivent chacune si un soft-goal est satisfait ou insatisfait.

VerdictCL= {Vcl_1, ..., Vclr} l'ensemble des variables booléennes Vcl_i qui décrivent chacune si un claim est vrai ou faux.

Les contraintes (14), (16), (1) et (18) ne sont pas respectées. Par conséquent, la configuration du scénario ne satisfait pas les soft goals SG4 et SG7 => **Vsg_4=0** et **Vsg_7=0.**

Ces contraintes ne sont pas respectées par les claims CL7 et CL17 => **Vcl_7=0** et **Vcl_{17}=0.**

- Finalement, le composant parcourt les graphes du M@RT pour déduire le verdict des buts et des OPERs.

Soient :

VerdictG = {Vg_1, ..., Vgn} l'ensemble des variables booléennes Vg_i qui décrivent chacune si un but est satisfait ou insatisfait.

VerdictOPER = {$Voper_1$, ..., $Voper_m$} l'ensemble des variables booléennes $Voper_i$ qui décrivent chacune si un OPER est « pertinent » dans le contexte en cours ou « non pertinent ».

L'exploration des claims conduit aux affections suivantes :

OPER2 est lié au claim CL7 => $\mathbf{Voper_7=0}$

OPER10 est lié au claim CL17 => $\mathbf{Voper_{10}=0}$

Et le parcours du graphe de la vue variabilité à partir des OPERs non pertinents permet de déduire le verdict des buts allant jusqu'au nœud racine.

OPER2 => $\mathbf{Vg_1=0, Vg_4=0}$ et $\mathbf{Vg_8=0.}$

OPER10 => $\mathbf{Vg_3=0.}$

Etape 3: le composant de diagnostic vérifie l'occurrence des non-conformités de types 3 et de type 4 qui représentent les problèmes de compatibilité entre les OPERs de la configuration. Cela consiste à parcourir les éléments du graphe de la vue de variabilité et d'identifier les OPERs qui correspondent aux cas suivants:

- ✓ Les OPERs qui appartiennent à une association de type « exclude ».
- ✓ Les OPERs qui participent en même temps à un groupe d'associations de type « mutex ».
- ✓ Les OPERs qui participent en même temps à un groupe d'associations de type « AND » mais dont quelques éléments du groupe sont manquants.
- ✓ Les OPERs qui participent à une association de type « require», dont le nœud cible n'est pas inclu dans la configuration.

Etape 4 : Le composant de diagnostic parcourt le modèle de variabilité pour vérifier les buts obligatoires qui ne sont pas opérationnels.

Dans le scénario, les buts 3, 6 et 8 ne sont pas opérationnels

⇒ $\mathbf{Vg_5=0, Vg_6=0}$ et $\mathbf{Vg_7=0.}$

3.2.2.4 Planification des solutions optimales

Le but du diagnostic est de déduire les OPERs conduisant à l'échec de la résolution du programme CP_global. La planification d'une solution optimale consiste alors à réparer la configuration en cours pour trouver une solution optimale.

Pour ce faire, notre stratégie consiste à (i) retirer les OPERs non pertinents pour purger l'espace de recherche. (ii) lancer la résolution à partir des OPERs restants de la configuration. Et (iii) calculer à partir des solutions générées par le solveur, la meilleure configuration qui optimise les soft goals, c.-à-d. qui maximise la qualité de configuration, noté Qconfig (voir équation (*)), et qui implique un minimum de changement dans la configuration en cours.

$$Q_{config} = \sum_{i=1}^{n} w_{exig_i} * \text{fulfillmentLevel}_{exig_i} \qquad (*)$$

Où, $\sum_{i=1}^{n} w_{exig_i} = 1$, n = nombre des exigences spécifiées, et

$$\text{fulfillmentLevel}_{exig_i} = \sum_{i=1}^{k} w_{SG_i} * \frac{\text{ObtainedLevel}_{SG_i}}{\text{RequiredLevel}_{SG_i}} \qquad (**)$$

Où, $\sum_{i=1}^{n} w_{sg_i} = 1$ et k=nombre des sg_i liés au $exig_i$

Qconfig mesure la qualité globale de chaque configuration du point de vue de la contribution des éléments de cette configuration à la satisfaction et à l'accomplissement des exigences. Les valeurs de Q_{config} sont comprises entre 1 to 4, où 4 correspond un accomplissement complet de toutes les exigences et les valeurs proches de 1 correspondent à une qualité de configuration médiocre.

Comme l'équation (*) le montre, une politique de priorisation est appliquée aux exigences. Le principe consiste à donner un poids w_{exig_i} à chaque exigence qui indique son importance pour le bon fonctionnement du système selon les hypothèses suivantes :

- L'exigence doit être accomplie « must » (c'est-à-dire qu'elle est nécessaire pour le bon fonctionnement du système et doit être pleinement satisfaite).
- L'exigence devrait être accomplie « should » (c'est-à-dire qu'elle est importante pour le système mais peut être partiellement satisfaite).
- L'exigence peut être accomplie « could » (c'est-à-dire qu'elle n'est pas obligatoire, mais préférable pour améliorer les fonctionnalités du système).

La même politique de priorisation est appliquée aux soft-goals liés à une exigence pour calculer le niveau d'accomplissement d'une exigence, noté $\text{fulfillmentLevel}_{exig_i}$, qui est compris entre 1 to 4, où 1 correspond au non-accomplissement de l'exigence et les valeurs proches de 1 correspondent à un niveau médiocre d'accomplissement.

3.3 Conclusion du chapitre

Dans ce chapitre, nous avons présenté notre approche de monitoring des exigences en ligne qui contribue à évaluer en continue les adaptations des SASs et à supporter l'incertitude dans des environnements changeants. L'approche consiste à spécifier les exigences sous forme d'un programme de contraintes traduisant les variations du système et de son environnent d'exécution. Les variations sont modélisées sous forme de ligne de produits logicielle dynamique. Nous avons montré l'intérêt de : (i) l'analyse de la variabilité qui évite essentiellement d'énumérer d'une manière procédurale toutes les situations d'adaptation, et (ii) la programmation par contraintes qui supporte la vérification de la spécification et l'ajout des contraintes binaire, n-aire ainsi que des contraintes sur des entiers, réel...etc., ce qui favorise l'enrichissement à tout moment de la spécification. Ainsi, l'analyse des exigences est effectuée pendant l'exécution du système par l'intermédiaire d'un solveur de contrainte et du gestionnaire du M@RT.

Il est à noter par ailleurs que qu'il faut faire la distinction entre l'adaptation qui est un processus itératif d'apprentissage, et le monitoring des exigences qui contribue à la gestion des adaptations. Le

monitoring en ligne est centré sur la modélisation des exigences et la compréhension des stratégies d'adaptation afin de déterminer les modifications pour une adaptation appropriée.

Chapitre 4

4. MISE EN ŒUVRE

Ce chapitre présente la mise en œuvre de notre approche de monitoring des exigences en ligne. RMAS, est un prototype développé dans le cadre de cette thèse afin d'automatiser l'analyse en ligne des exigences des systèmes auto-adaptatifs : Toutes les stratégies décrites précédemment sont entièrement implémentées, sauf l'interface graphique qui manque pour la présentation des résultats du monitoring. Une description détaillée de l'outil est fournie tout au long de ce chapitre. La section 4.1 présente l'architecture logique du RMAS. Et la section 4.2 présente les algorithmes mis en œuvre pour automatiser les fonctionnalités du monitoring.

4.1 Architecture

Le prototype RMAS est entièrement développé en Java en se basant sur une grande partie des packages de VariaMos qui ont été réutilisés pour prendre en charge les spécifications REFAS. Le prototype peut être intégré dans le code du système cible ou réside sur un serveur autonome. Comme le montre la Figure 22, l'architecture logique du prototype RMAS est composée de trois packages principaux :

(1) Le package main est le package central qui interagit avec le M@RT pendant l'exécution du système cible pour obtenir les informations sur le système et son environnement, déclencher les activités d'analyse des exigences et de planification des configurations optimales en cas de violations et gère la communication avec le serveur du solveur.

Il existe deux classes principales dans ce package : (i) MonitoringAndDiagnosing qui implémente les algorithmes d'analyse et de diagnostic de la configuration en cours pour fournir un diagnostic complet en cas de violations de contraintes. Et (ii) PlanningOptimalConfiguration qui planifie une configuration pour optimiser le contexte en cours.

(2) Le package modelsAtRuntime a pour but de mettre à jour les instances des classes du méta-Modèle REFAS pour refléter l'état du système et de son environnement d'exécution. Il sert également à générer le programme de contrainte à partir des règles de transformation.

Ce package est basé principalement sur le package « com.VariaMos.dynsup.model », appartenant au code de l'outil VariaMos, pour lire la spécification créée sous VariaMos. Cette spécification portant l'extension « .vmum ». En effet, le package de VariaMos définit un ensemble de classes Java qui représentent le méta-modèle REFAS, ce qui facilite l'enregistrement des graphes en mémoire et leurs prises en charge en ligne. Les règles de transformation définissent dans le chapitre 3 sont également codées en Java pour convertir automatiquement le M@RT en un programme de contraintes CP.

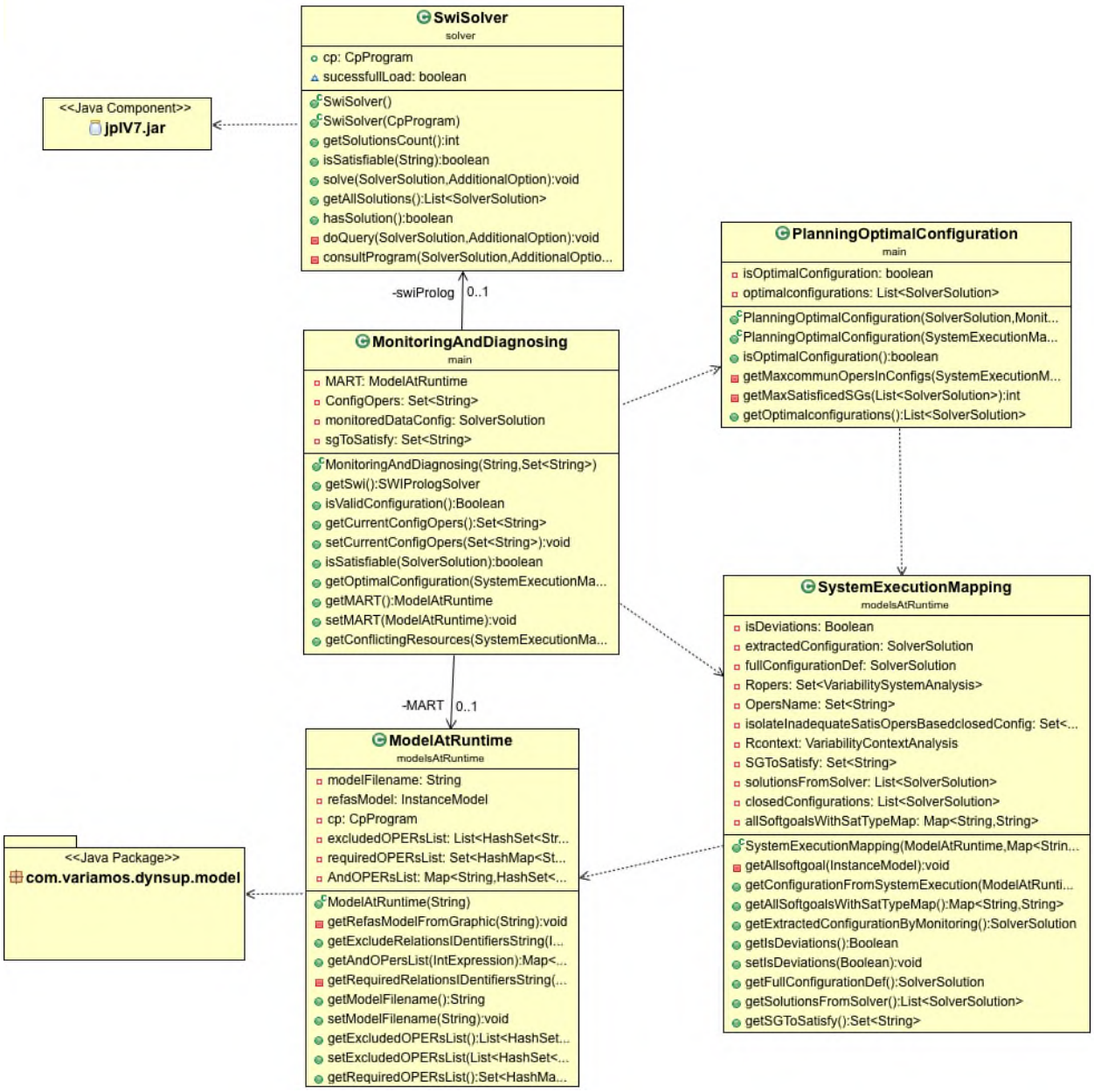


Figure 22 : architecture logique du prototype RMAS

Il existe quatre classes principales dans ce package : (i) ModelAtRuntime utilise les classes du package de VariaMos pour transformer le fichier de spécification (extension « .vmum ») en M@RT, ce dernier représente des instances des classes du méta-modèle. Le M@RT sert à stocker l'état actuel du système et les conditions de son environnement d'exécution, ce qui permet de mettre à jour en continue les valeurs des soft-dépendances et des claims dans le programme de contraintes à partir des règles de transformation. Cette classe définit également une liste des OPERs compatibles entre eux, appelée RequiredOpersList, traduisant les relations « require » et « and » dans la spécification, une autre liste des OPERs incompatibles, appelée ConflictOpersList, traduisant les relations « exclude » et « mutex » dans la spécification. Et une dernière liste de type clé/valeur contenant id de tous les goals de hauts niveaux qui sont obligatoires et leurs niveaux de granularité. Ces trois listes contribuent à

accélérer les algorithmes de diagnostic (voir algorithme 3 et 4) et sont calculées une seule fois lors du premier chargement de la spécification en ligne. (ii) SystemExecutionMapping sert à stocker les informations sur les OPERs actifs et les variables du contexte dans des instances des classes LeafGoalAnalysis et la VariabilityContextAnalysis respectivement, à partir des données collectées (fichiers de journalisations). (iii) LeafGoalAnalysis sert à stocker les informations sur les claims et les soft-goals liés à chaque OPER (qui représente les goals feuilles) ainsi que tous les goals de hauts niveaux liés à l'OPER dans des listes de type clé/valeur. Le but est d'optimiser le temps d'analyse lors des activités de diagnostic. Et (iv) VariabilityContextAnalysis sert à stocker les informations sur les soft-dépendances et les soft-goals liés aux variables de contexte.

(3) Le package Solver interagit avec le composant « JPL » pour interroger le solveur swiProlog durant les activités d'analyse et de diagnostic. JPL est un ensemble de classes Java et de fonctions C fournissant une interface bidirectionnelle entre Java et Prolog. JPL utilise l'interface JNI (Java Native Interface) pour se connecter à un moteur Prolog via l'interface FLI (Prolog Foreign Language Interface).

4.2 Fonctionnalités

Dans cette section, les fonctionnalités fournies par RMAS sont présentées. La Figure 23 résume les principales étapes effectuées par RMAS :

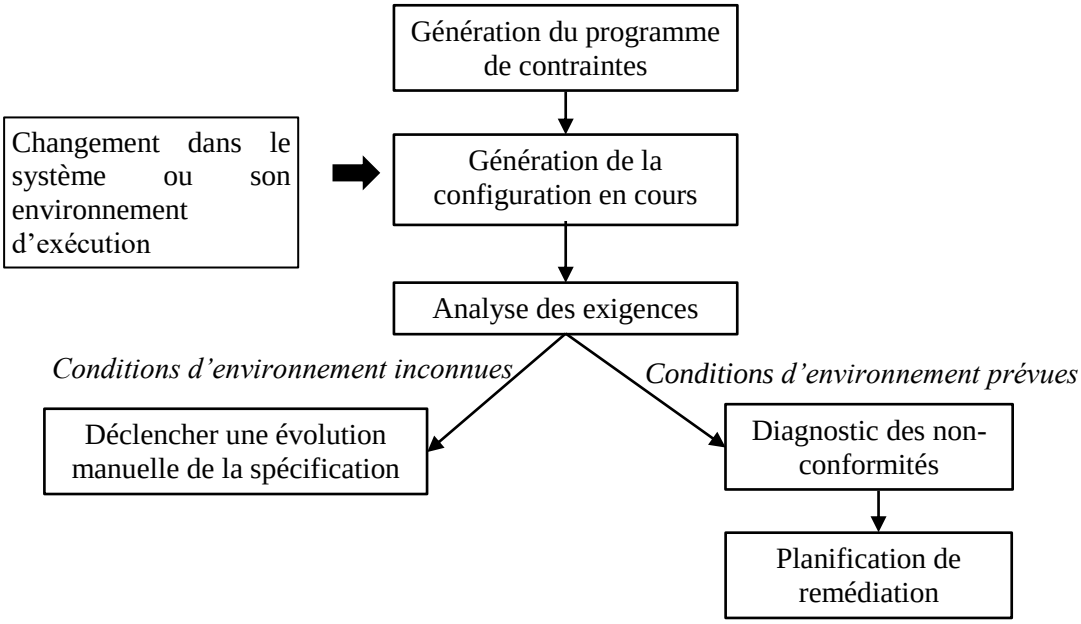


Figure 23 : aperçu des relations entre les fonctionnalités offertes par le prototype RMAS

Génération du programme de contraintes : La première étape du monitoring des exigences est la génération d'un programme de contraintes CP par l'application des règles de transformation REFAS-CP à partir du M@RT. Ce programme est maintenu à jour, à chaque chargement d'une nouvelle spécification d'exigences en ligne.

Génération de la configuration en cours : les instances du M@RT sont enrichies continuellement en temps réel avec les données du système et de son environnement d'exécution. Sur la base de ces données, le monitoring génère une configuration qui se compose des OPERs actifs du système cible et les valeurs des variables du contexte en cours.

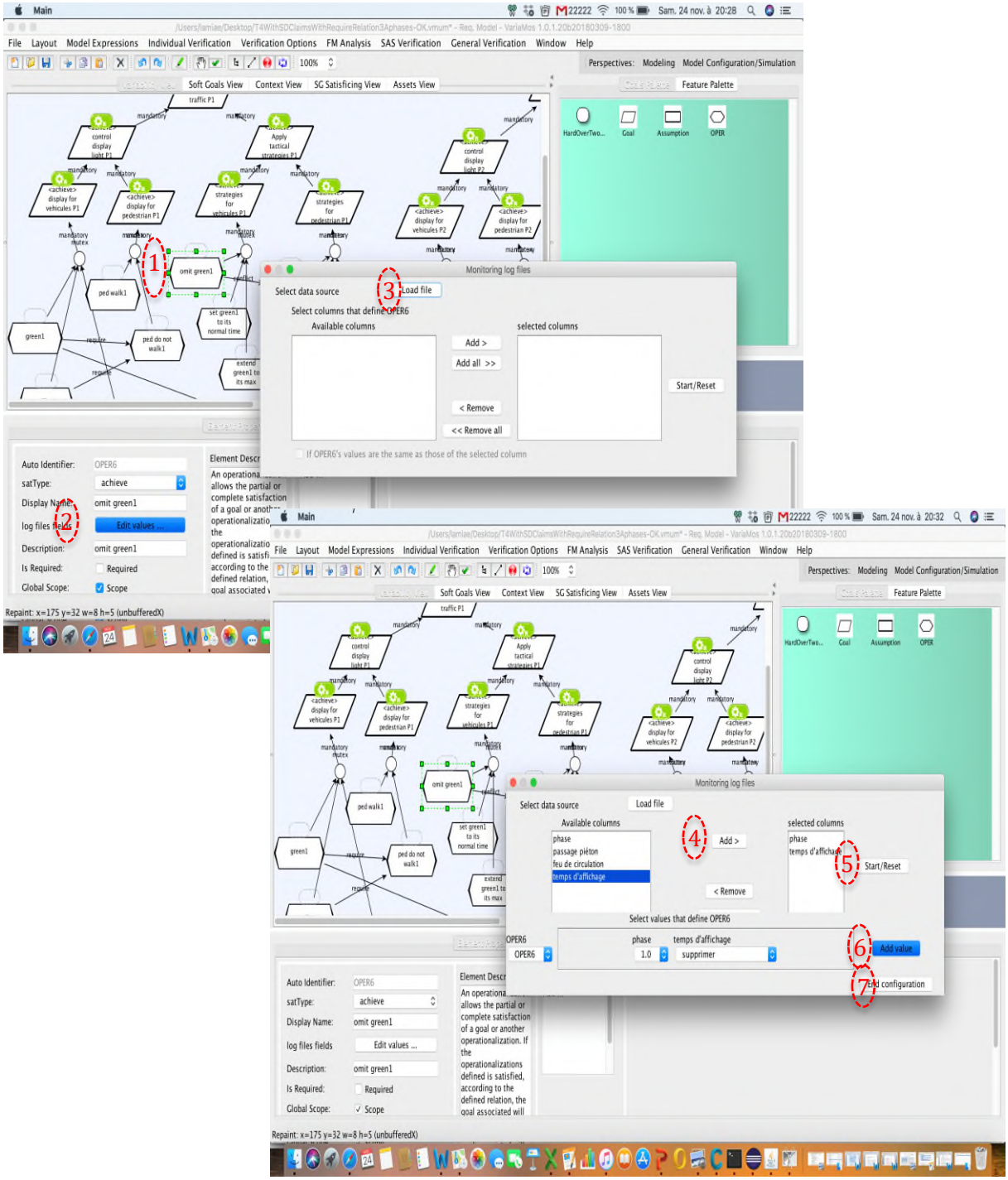


Figure 24 : interface d'initialisation de l'attribut de mappage entre un concept REFAS et les champs des fichiers de journalisation (étapes de 1 à 7 en rouge)

Pour concrétiser cette génération, les méta-concepts du méta-modèle REFAS ont été étendus avec l'ajout d'un attribut, appelé « VAR_MONITORINGFIELDS » qui décrit la relation entre les champs du fichier contenant les données d'exécution et les entités constituant le M@RT. Cet attribut doit être initialisé pour chaque concept de type OPER et variable du contexte via l'outil VariaMos avant le déploiement de la spécification en ligne. À cet effet, le code de l'outil VariaMos est modifié pour inclure une interface (voir Figure 24) permettant de définir une correspondance entre le concept sélectionné et les champs des fichiers de journalisation.

Comme le montre la Figure 24, une interface apparaît en sélectionnant un concept puis en cliquant sur le bouton « Edit Values ». Cette interface permet à l'analyste/développeur de sélectionner le fichier contenant les champs (et leurs valeurs possibles) qui définissent le concept sélectionné. L'interface extrait toutes les colonnes du fichier pour faciliter l'ajout des colonnes concernées. Les valeurs de chaque colonne apparaissent alors dans une liste déroulante pour l'ajout des valeurs correspondantes au concept. Ainsi, par exemple, le concept OPER6 = « Omit green1 » sera sélectionné comme étant un élément de la configuration en cours si les valeurs des deux champs « phase » ET « temps d'affichage » sont respectivement « 1 » et « Supprimer ».

L'analyse des exigences : le monitoring lance l'analyse des exigences à chaque modification observée de la configuration en cours ou des conditions de l'environnement d'exécution. Le cas échéant, la configuration est transformée en une solution partielle selon l'algorithme 1.

Algorithme 1 : Construction d'une solution partielle à tester

```
ENTREE: configMapping : SystemExecutionMapping
SORTIE : S une solution partielle à vérifier avec le solveur
MAKESOLUTION (configMapping) {
foreach (leafGoalsAnalysis op : configMapping.getSelectedOpers ()) {
if (op.getOperName ().startsWith ("OPER")) {
S.add (op.getOperName () ,1) ;
   }        }
foreach (String var : configMapping.getVariableContexte ().keySet ()) {
                    S.add (var, configMapping.getVariableContexte ().get(s)) ;
}
return S ;
}
```

L'algorithme 1 prend en entrée une instance de la classe SystemExecutionMapping et retourne une solution conformément à la notation du programme des contraintes.

La solution S générée est ensuite vérifiée par le solveur de contraintes. Par conséquent, le monitoring termine l'analyse si la résolution admet des solutions. Sinon, déclenche la planification des remédiations.

Diagnostic des exigences : cette activité est déclenchée à chaque fois que la configuration en cours n'est pas conforme aux contraintes du programme CP. Le but est de localiser les causes des non-conformités et de fournir une explication de la nature de ces causes. Dans le cadre de cette thèse, 5 types de non-conformités sont considérés :

Type 1 : Des conditions d'environnement qui ne sont pas prévues lors de la spécification des exigences.

La détection de ce type se fait par la résolution du CP avec une configuration contenant seulement les variables de contexte et les soft goals à satisfaire. Si la résolution admet des solutions I, ces derniers serviront comme une base de diagnostic des autres types de non-conformités. Si au contraire, la résolution ne donne pas de solutions, ces conditions d'environnement sont stockées dans une base de données et une requête de modification manuelle de la spécification est lancée.

Type 2: la configuration ne satisfait pas suffisamment les soft goals. L'algorithme 2 définit le processus de détection de ce type en pseudo-code.

Algorithme 2 : Identification des Opers qui ne satisfont pas les softgoals

```
ENTREE : config : configuration en cours, configMapping : SystemExecutionMapping, solutions : ensemble des solutions I retournées par le diagnostic du type 1
SORTIE : denialNotSatisficeSG : collection de <String, LeafGoalAnalysis>
GETDENIALNOTSATISFICESG (config, configMapping, optimalConfig) {
sgToOpers=new HashMap<String, <String,HashMap<LeafGoasl,Integer>> () ;
//retourne les soft goals obligatoires.
softgoalsTosatisfy= SystemExecutionMapping.getSgToSatisfy () ;
foreach (String sg : softgoalsTosatisfy) {
sgToOpers.put (sg, new HashMap<LeafGoasl,Integer>> () ) ;
    // Récupérer tous les opers liés aux softgoals via des claims.}
foreach (LeafGoalAnalysis oper : configMapping.getRopers ()) {
  OPER=oper.getOperName ();
  claimsValues=oper.getSGToclaimsValues ();
   foreach (String keyClaim : claimsValues.keySet ()) {
       // keyClaim est une chaine de caractère contenant le nom d'un soft-goal et un claim
          (commence avec « CL») lié à ce soft-goal.
SGfromKeyClaim=keyClaim.substring (0, keyClaim.lastIndexOf ("CL"));
```

```
        if (sgToOpers.get (SGfromKeyClaim) !=null) {
 sgToOpers.get (SGfromKeyClaim).put (oper, claimsValues.get (keyClaim)) ;}}    }
// La configuration la plus proche est celle qui contient le plus d'OPERs communs avec la configuration en cours.
closestconfig=getClosestConfiguration (solutions) ;
// Analyse des valeurs récupérées pour chaque softgoal par rapport à la valeur dans la configuration la plus proche.
 foreach (String sg : softgoalsTosatisfy) {
            //représente la valeur optimale requise pour satisfaire un soft goal.
             satValue= closestconfig.getSolverSolution ().get (SGfromKeyClaim
                    +"_ClaimExpLevel").intValue () ;
            if ( !sgToOpers.get (sg).Values ().contains (satValue)) {
                 denialNotSatisficeSG .put (sg, sgToOpers.get (sg).KeySet ()) ;
}
}
return denialNotSatisficeSG
}
```

L'algorithme 2 prend en entrée la configuration en cours et une solution ∈ I. Cette solution servira de base pour déterminer les niveaux de satisfaction à atteindre par le système par rapport au contexte en cours. L'algorithme retourne les OPERs qui n'ont pas satisfait complètement ou suffisamment les soft goals, étant considérés comme des éléments problématiques dans la configuration.

Type 3 : la configuration manque des OPERs requis par d'autres OPERs (liés par des relations « and » ou « require »).

Type 4 : la configuration contient des OPERS incompatibles (liés par des relations « exclude » ou « mutex »).

Type 5 : la configuration ne satisfait pas les goals, cela peut se produire si certains goals obligatoires ne sont pas opérationnels dans la configuration.

L'algorithme 3 définit le processus de diagnostic en pseudo-code. Il prend en entrée la configuration en cours, le programme de contrainte, et renvoie un diagnostic unique pour le système qui spécifie les OPERs qui sont sources de non-conformité et le type de non-conformité détecté.

Algorithme 3 : diagnose des goals de bas niveaux

ENTREE**:** cp : le programme de contraintes, config : la configuration en cours, mart : ModelAtRuntime

requiredOPERsList : Liste des listes des OPERs liés avec la relation «exclude» ou « mutex » dans mart, conflictOpersList : collection des listes de tous les OPERs liés avec la relation «require» ou «and» dans mart, configMapping : SystemExecutionMapping.

SORTIE **:** denialLeafgoalComponents : collection de <String, List<LeafGoalAnalysis >>

DIAGNOSE (cp, config, mart, conflictOpersList, requiredOpersList, configMapping) {

// contextConfig contient seulement les variables de contexte et les soft goals à satisfaire

contexConfig=Utils.getConfigurationWithFreeGoals (config) ;

solver.solve (contexConfig, cp)

if (solver.hasSolution ()) {

altConfigs=solver.getAllSoutions (contexConfig) ;

// Type 2 : fait appel à l'algorithme 2

denialLeafgoalComponents.putAll (getDenialNotSatisficeSG (config, configMapping, configs)) ;

denialNotSatisficeSG= Utils.getSatValuesFromMap (denialLeafgoalComponents)

//Type 3 :

opers=getOPERSConfig () ;

opers.removeAll (DenialNotSatisficeSG); //suppression des opers déjà analysés

foreach (LeafGoalAnalysis op : opers) {

foreach (Set< LeafGoalAnalysis > req : requiredOPERsList) {

if (req.contain (op) {

if (! opers.containsAll (req) {

denialRequiredComponents.add (op) ;}

} } }

denialLeafgoalComponents.addAll ("Required", denialRequiredComponents);

//Type 4 :

opers.removeAll (denialRequiredComponents); //la suppression des opers déjà analysés

foreach (LeafGoalAnalysis op : opers) {

foreach (Set< LeafGoalAnalysis > req : conflictOPERsList) {

if (req.contain (op) {req1=req.remove (op) ;

if (iscontainsAtListOne (opers, req1) {// test si opers contient au moins un oper de req1

denialConflictComponents.add (op) ;}

} } }

```
denialLeafgoalComponents.addAll ("Conflict", denialConflictComponents);
}
return denialLeafgoalComponents;
}
else {
//Type 1 :
bd.addUnknownContext (contexConfig) ;
return null ;
}
```

Une optimisation de l'algorithme 3 consiste à faire un apprentissage des conditions d'environnements déjà analysées pour éviter d'appeler le solveur pour les mêmes conditions. En effet, En cas de modèles de grandes tailles, la résolution des configurations avec seulement les conditions d'environnement demande un temps considérable. D'où l'idée de stocker les solutions retournées pour les utiliser plus tard dans des conditions d'environnements similaires.
L'algorithme 4 ci-dessous décrit l'optimisation de l'algorithme 3.

Algorithme 4 : diagnose des goals de bas niveaux

ENTREE : cp : le programme de contraintes, config : la configuration en cours, mart : ModelAtRuntime
requiredOPERsList : Liste des listes des OPERs liés avec la relation «exclude» ou « mutex » dans mart, conflictOpersList : Liste des listes de tous les OPERs liés avec la relation «require» ou «and» dans mart, configMapping : SystemExecutionMapping, Learningconditions : liste des conditions de l'environnement qui sont déjà analysées par RMAS.
SORTIE : denialLeafgoalComponents : collection de <String, List<LeafGoalAnalysis >>

```
DIAGNOSE (cp, config, mart, conflictOpersList, requiredOpersList, configMapping) {
if (Learningconditions.contain (Currentcontext)) {
altConfigs=getAlternativeSolutions(CurrentContext) ;
}
else {
// contextConfig contient seulement les variables de contexte et les soft-goals à satisfaire
contexConfig=Utils.getConfigurationWithFreeGoals(config) ;
solver.solve(contexConfig, cp) ;
altConfigs=solver.getAllSoutions(contexConfig) ;
}
if (altconfigs !=null && altConfigs>=1) {
```

```
// Type 2 : fait appel à l'algorithme 2
denialLeafgoalComponents.putAll (getDenialNotSatisficeSG (config, configMapping, configs)) ;
denialNotSatisficeSG= Utils.getSatValuesFromMap (denialLeafgoalComponents)
//Type 3 :
opers=getOPERSConfig() ;
opers.removeAll(DenialNotSatisficeSG); //suppression des opers déjà analysés
foreach(LeafGoalAnalysis op :opers) {
    foreach(Set< LeafGoalAnalysis > req : requiredOPERsList ) {
            if(req.contain(op) {
                if(!opers.containsAll(req) {
                    denialRequiredComponents.add(op); }
}
}
}
denialLeafgoalComponents.addAll("Required", denialRequiredComponents);
//Type 4 :
opers.removeAll(denialRequiredComponents);   //la suppression des opers déjà analysés
foreach(LeafGoalAnalysis op :opers) {
     foreach(Set< LeafGoalAnalysis > req : conflictOPERsList ) {
          if(req.contain(op) { req1=req.remove(op) ;
               if(iscontainsAtListOne(opers,req1) { // test si opers contient au moins un oper de req1
                  denialConflictComponents.add(op) ;
}
}
 }
}
denialLeafgoalComponents.addAll("Conflict", denialConflictComponents);
}
return denialLeafgoalComponents;
}
else { //Type 1 :
 bd.addUnknownContext(contexConfig) ;
return null ;
}
```

Notez cependant que les algorithmes 3 et 4 ne permettent pas de détecter le cinquième type de non-conformité, concernant les buts de hauts niveaux qui ne sont pas opérationnels. En effet, le diagnostic étant basé sur les buts de bas niveaux, les buts de hauts niveaux ne peuvent pas être observés. Ces derniers sont détectés par l'algorithme 5, qui retourne un diagnostic à différents niveaux de granularité partant des buts de bas niveaux, jusqu'au niveau désiré. L'avantage de monitoring à une granularité élevée est qu'il fournit aux développeurs/analystes un historique détaillé leur permettant de comprendre les causes réelles, à hauts niveaux, des non-conformités aux exigences pour améliorer les stratégies d'adaptation dans des phases de maintenances programmées.

Algorithme 5 : diagnose à différents niveaux de granularité

ENTREE : cp : le programme de contraintes, config : la configuration à tester, mart : ModelAtRuntime, level : le niveau de granularité désiré.

SORTIE : denialGoalsWithLevels : collection <Integer, String>

DIAGNOSEALL (cp,config, level) {

// getOpersSet transforme la collection retourner par l'algorithm 3 « diagnose » en une liste des opers.

denialLeafgoalComponents=Utils.getOpersSet(diagnose(cp,config) ;

// retourner les Opers de config

opers=getOpersInconfiguration(config) ;

// retourner tous les buts de hauts niveaux qui sont opérationnels par opers.

highLevelGoals=getAllHighLevelGoalsInConfiguration(opers,mart)

foreach(LeafGoalAnalysis oper : denialLeafgoalComponents){

// les OPERs représentent le niveau 1. Le niveau de granularité augmente par 1 en parcourant les nœuds du bas en haut.

denialGoalsWithLevels.add(oper.getOperId, 1) ;

//topGoals est une liste qui stocke tous les buts qui sont opérationnels par les éléments de denialLeafgoalComponents

topGoals.add(oper.getRelatedgoalToRoot(mart, level)) ;**}**

denialGoalsWithLevels.AddAll(topGoals);

// Type 5 :

// cette fonction retourne tous les goals de hauts niveaux qui sont obligatoires en parcourant récursivement les nœuds du mart.

mandatoryGoals=mart.getAllMandatoryTopGoalWithLevels() ;

missMandatoryGoals=mandatoryGoals.KeySet().removeAll(highLevelGoals) ;

foreach(String s : missMandatoryGoals){

denialGoalsWithLevels.add(s, missMandatoryGoals.get(s)) ;}

return denialGoalsWithLevels; }

Cependant, lorsque le niveau de granularité augmente, le temps de traitement augmente considérablement car il demande de parcourir tous les graphes à partir de chaque OPER considéré. Pour optimiser l'algorithme 5, le détail des différents goals liés à chaque OPER est stocké dans une liste à chaque chargement d'une nouvelle spécification REFAS. Ce processus demande un temps de calcul considérable en cas de spécification de grande taille, cependant, il est calculé une seule fois durant tous le cycle de vie d'une spécification et utilisé en cas de besoin sans besoin de parcourir à chaque fois les graphes de la spécification.

Planification d'une solution optimale : la stratégie de planification d'une solution optimale est implémentée avec l'algorithme 6. Il consiste à chercher parmi les solutions alternatives de I, celles qui optimisent la mesure de « qualité » selon l'équation (*) (voir section 3.2.2.4) tout en impliquant un minimum de changement dans les OPERs.

Algorithme 6 : planification d'une solution optimale

ENTREE : config : configuration en cours, altConfigs : ensemble des solutions I retournées par le diagnostic du type 1

SORTIE : optimal_configurations : liste de <SolverSolution>

```
GETOPTIMALCONFIG (config) {
// calcule la valeur maximale des qualités calculées à partir de l'équation (*)
q=getMaxQualityConfiguration (altConfigs) ;
//calcule le nombre max d'opers communs avec la configuration en cours et retire en même temps les configurations qui ont qualité <q.
MaxOpersNB=getMaxOperInConfigs(context);
optimal_configurations =new ArrayList<SolverSolution>();
foreach(SolverSolution c : altConfigs){
opersC= UtilsLAm.getOPERConfiguration(c);
opersC.retainAll(opers);
if(opersC.size()>= MaxOpersNB){
        optimal_configurations .add(c);
}
 }
return optimal_configurations ;
}
```

4.3 Conclusion du chapitre

Dans ce chapitre nous avons décrit l'architecture logique et les fonctionnalités de l'outil

d’automatisation de notre approche RMAS. Les algorithmes implémentés pour chaque fonctionnalité ont été également présentés.

Dans la partie 3, une évaluation de notre approche a porté sur deux cas d’étude : le système INSPIRE décrit précédemment dans le chapitre 3, et un contrôleur adaptatif des feux de circulation.

Partie 3

Evaluation Empirique

Cette partie présente l'évaluation expérimentale de notre approche du monitoring des exigences en ligne. Nous avons adopté les méthodes descriptives et expérimentales du Design Science (HEVNER et al. (2004) et PEFFERS et al. (2007) pour valider notre approche. Ainsi, des scénarios ont été utilisés tout au long de la thèse pour illustrer l'approche proposée. En outre, dans cette partie, deux cas d'études ont été choisis pour appliquer notre approche de monitoring. Ces deux cas sont issus de deux domaines d'application différents pour tester la généricité de notre approche et son applicabilité sur plusieurs domaines d'application, y inclus le domaine d'éducation à distance et le transport intelligent. Précisément, le premier cas d'étude est un système hypermédia adaptatif et le deuxième cas d'étude représente une simulation d'un contrôleur adaptatif d'intersection à feux de circulation**,** tiré du manuel[10] TSTM « traffic light timing manual», publié dans le site de l'administration des autoroutes aux États-Unis. Ces deux cas d'études sont présentés respectivement dans les chapitres 5 et 6.

[10] https://ops.fhwa.dot.gov/publications/fhwahop08024/doc_page.htm

Chapitre 5

5. ETUDE DE CAS 1 : SYSTEME E-LEARNING ADAPTATIF 'INSPIREUS'

Un système e-learning adaptatif a pour but de délivrer des ressources éducatives personnalisées à chaque apprenant. La plupart des stratégies d'adaptation sont centrées sur l'apprenant ou les théories pédagogiques, alors que peu de travaux se sont penchés sur l'évaluation de la qualité des solutions fournies face à l'incertitude des environnements d'apprentissage et l'imprédictibilité du comportement des apprenants. Ces facteurs sont souvent à l'origine d'une définition incohérente des règles d'adaptation ou d'une insuffisance de quelques-unes, ce qui peut influencer l'efficacité de ces systèmes. Par conséquent, il arrive qu'ils soient incapables de trouver les ressources adéquates pour les apprenants malgré leur présence dans l'espace de connaissance.

À travers le cas d'étude —INSPIREus, nous soulignons la nécessité de surveiller et de diagnostiquer les exigences des systèmes e-learning adaptatifs en temps réel, pour une meilleure compréhension de leurs comportements. À cet objectif, une étude empirique a été réalisée sur 6165 traces numériques d'INSPIREus pour vérifier et valider le comportement d'INSPIREus à l'aide de RMAS et en fonction des quatre exigences énoncées précédemment dans la section 3.1.2. Les résultats fournissent des statistiques sur la précision du système adaptatif et soulignent certains problèmes liés à la conformité des exigences au moment de l'exécution.
Dans le reste de ce chapitre nous présentons le protocole de cette première expérimentation et une discussion des résultats trouvés.

5.1 Méthodologie d'expérimentation

5.1.1 Collecte des données

Dans le cadre d'un cours d'informatique dispensé sur INSPIREus par le département d'informatique et de télécommunications de l'Université d'Athènes, Vingt et un apprenants ont suivi le cours qui a duré trois mois à compter de mars 2016. L'historique de navigation et d'interaction entre apprenants, les notes des évaluations ainsi qu'un fichier descriptif de toutes les ressources éducatives disponibles pour les apprenants, ont été utilisés pour mener cette étude empirique. Le Tableau 4 décrit ces trois ensembles de données.

Tableau 4 : description des traces d'apprentissage. (Dounas et al., 2019)

fichier	Description
Fichier log	Décrit toutes les interactions des apprenants avec INSPIREus. il se compose de neuf colonnes comprenant principalement : l'identifiant de l'apprenant, l'identifiant du Module visité, l'heure de visite, les

	opérations (représentent les modifications apportées au système par les apprenants ou par le système au système).
Fichier d'évaluations	Stocke les informations sur les notes de chaque concept étudié. Il se compose de 5 colonnes : apprenant-ID, le titre du module, l'heure d'évaluation, le titre du concept et la note final de chaque concept.
Fichier descriptif	Décrit les ressources éducatives disponibles pour chaque concept, leur temps d'étude alloué et leur type (activité, exemple, exercice, question ou théorie) et le niveau d'acquisition requis (mémorisation, utilisation ou recherche).

5.1.2 Traitement des données

Le processus d'analyse des données collectées comprend trois phases (voir la Figure 25) :

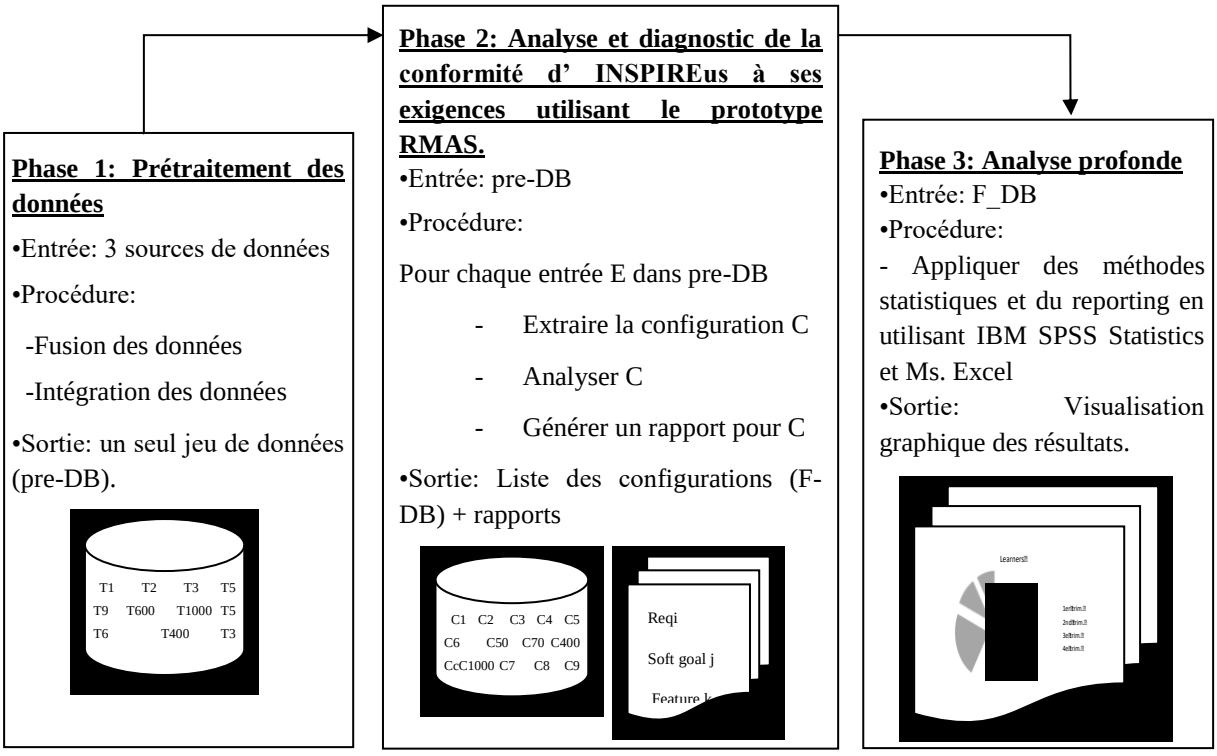


Figure 25 : les phases d'analyse des données. (Dounas et al., 2019)

La phase de prétraitement, il s'agit de fusionner les trois jeux de données issues des trois fichiers en une seule base de données. L'outil Talend Open studio a été utilisé pour cette fin et pour nettoyer les

données des redondances et d'assurer leur intégrité. La base de données finale représente un total de 6 165 entrées.

Après le prétraitement automatique des données, nous avons éliminé manuellement un seul apprenant sur 21 en raison du manque d'assiduité au cours. En effet, cet apprenant a passé un total de 30 minutes d'étude contre 3 h 30 min dus et n'a pas fait d'évaluation.

La phase d'analyse, au cours de laquelle les 6165 entrées ont été analysées par RMAS afin d'obtenir une connaissance de la manière dont INSPIREus remplit ses exigences au moment de l'exécution.

Pour ce faire, chaque entrée a été transformée en une configuration à l'aide de notre stratégie de mappage expliqué dans la section 4.2. Le résultat d'analyse de chaque configuration, indique les buts et les soft goals qui ne sont pas satisfaits, et le cas échéant, les composants de la configuration qui sont causes des non-conformités aux exigences.

La phase d'analyse approfondie vise à ressortir des statistiques plus approfondies afin de comprendre le comportement d'INSPIREus. Les outils utilisés dans cette phase sont IBM SPSS Statistics et Ms. Excel.

5.2 Résultats

Statistiques descriptives

Une analyse préliminaire des données à permis d'identifier 8 apprenants actifs, 2 pragmatiques, 4 réflexifs et 6 théoriciens. Seuls 4 apprenants ont préféré spécifier manuellement leurs styles d'apprentissage dans le modèle d'apprenant, tandis que 16 apprenants ont pris le questionnaire de Honey et Mumford lors de leur première connexion. Le cours propose 3 concepts à étudier. Le temps d'étude moyen était de 440,53 min (ET = 422,07) par rapport au temps alloué (temps proposé par un enseignant) = 394 min. En règle générale, tous les apprenants ont réussi l'examen final avec une moyenne de 11,67 sur 12 (écart-type = 1,05).

Précision et Rappel

La précision et le rappel sont généralement les mesures les plus utilisées pour évaluer l'efficacité d'une stratégie, respectivement en termes d'exactitude et d'exhaustivité.

La précision dans cette étude est définie par le rapport entre les configurations valides retrouvées, c.-à-d. celles évaluées comme satisfont toutes les exigences par RMAS et le total des configurations collectées:

$$Précision = \frac{configurations\ valides\ retrouvées}{total\ des\ configurations\ proposées}$$

Quant au rappel, il correspond au nombre des configurations valides retrouvées au regard du nombre total des configurations valides que possède la base de données y compris celles qui ne sont pas proposées aux apprenants.

$$\text{Rappel} = \frac{\text{configurations valides retrouvées}}{\text{total des configurations valides}}$$

Comme le total des configurations valides n'est pas connu, le rappel n'a pas été calculé pour cette étude. Seule la précision a été calculée pour chaque apprenant.

Comme le montre le Tableau 5, la précision était assez bonne (jusqu'à 68%) sauf pour les apprenants 5, 6 et 8 où la précision était entre 30% et 35%.

Tableau 5 : la précision du système INSPIREus pour chaque apprenant. (Dounas et al., 2019)

Apprenant-ID	Total des Configurations	Configurations valides retrouvées	Précision (%)
0	181	109	60,22
1	643	603	**91,29**
2	956	589	61,61
3	266	250	**93,98**
4	350	216	61,71
5	191	63	34,55
6	435	132	33,1
7	165	98	59,39
8	286	89	30,77
9	161	110	68,32
10	351	164	**88,03**
11	286	233	**81,47**
12	105	58	55,24
13	451	443	**98,23**
14	393	353	76,34
15	221	135	61,09
16	90	79	**87,78**
17	193	137	70,47
18	208	208	**100**
19	233	110	47,21

D'autre part, en regroupant la précision en fonction du style d'apprentissage (voir le tableau 6), les résultats montrent des variations concernant la précision du système au sein du même groupe d'apprentissage. Par exemple, une variation considérable de la précision du système peut être observée entre les apprenants actifs (variance = 301,83).

Tableau 6 : statistiques descriptives de la précision par groupe de style d'apprentissage. (Dounas et al., 2019)

Groupe de style d'apprentissage	Nombre d'apprenants	Min	Max	Moyenne de la précision	Variance
Actif	8	47,21	93,98	71,34	301,83
Pragmatique	2	55,24	70,47	62,85	115,97
Réflexif	4	30,77	60,22	39,66	190,29
Théoricien	6	61,61	100	84,28	208,13
Total	20	30,77	100	68,04	462,40

En outre, l'ANOVA a montré une différence significative dans la précision moyenne entre au moins deux groupes d'apprentissage (μréflexif # μthéoricien # μpragmatique # μactif) : F (3.238887) = 6. 8679, valeur de p = 0.03 <0.05.

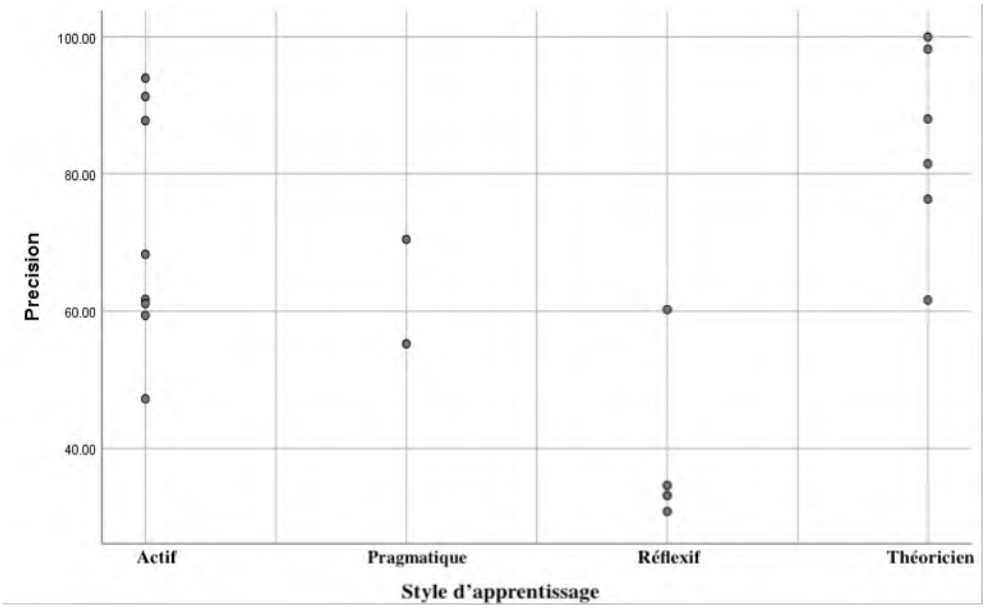


Figure 26 : illustration de la précision pour chaque apprenant groupée par style d'apprentissage. (Dounas et al., 2019)

En particulier, comme illustré par le nuage des points dans la Figure 26 ci-dessous, le système était plus efficace pour les apprenants théoriciens et moins efficace pour les apprenants réflexifs.

Accomplissement des exigences

Le niveau d'accomplissement mesure le degré de satisfaction de l'exigence au regard des niveaux de satisfaction des soft goals qui y sont liées. Sa valeur est calculée selon l'équation (**) (voir section 3.2.2.4) et est comprise entre 1 et 4, allant du non-accomplissement de l'exigence à un accomplissement complet de l'exigence.

Le Tableau 7 présente la politique de priorisation appliquée aux soft goals et aux exigences et qui ont permis le calcul du niveau d'accomplissement des exigences pour chaque configuration.

Tableau 7 : politique de la priorisation des exigences et des soft goals

Exigence	**Exig1**		**Exig2**	**Exig3**	**Req4**
Poids de l'exigence	0.4		0.3	0.2	0.1
Soft-goal lié à l'exigence	SG1	SG2	SG3	SG5	SG4
Poids du soft-goal	0.6	0.4	1	1	1

Par exemple, le niveau d'accomplissement de l'exigence Exig1 est calculé par l'équation suivante :

$$\text{fulfillmentLevel}_{\text{exig}_1} = 0.6 \times \frac{\text{ObtainedLevel}_{\text{SG}_1}}{\text{RequiredLevel}_{\text{SG}_1}} + 0.4 \times \frac{\text{ObtainedLevel}_{\text{SG}_2}}{\text{RequiredLevel}_{\text{SG}_2}}$$

Une normalisation a été effectuée pour ramener les valeurs de fulffilementLevel$_{\text{exigi}}$ entre 0 et 1 selon la méthode de normalisation Min-Max.

Les résultats obtenus sont résumés et illustrés par la Figure 27 et la Figure 28.

Pour aider à comprendre les résultats, nous avons défini quatre classes pour le niveau d'accomplissement :

La classe A représente un bon niveau d'accomplissement ∈ [0.7, 1]

La classe B représente un niveau de d'accomplissement suspecté ∈ [0.5, 0.7 [

La classe C représente un niveau d'alerte d'accomplissement ∈ [0.4, 0.5 [

La classe D représente un niveau d'accomplissement insuffisant ∈ [0, 0.4 [

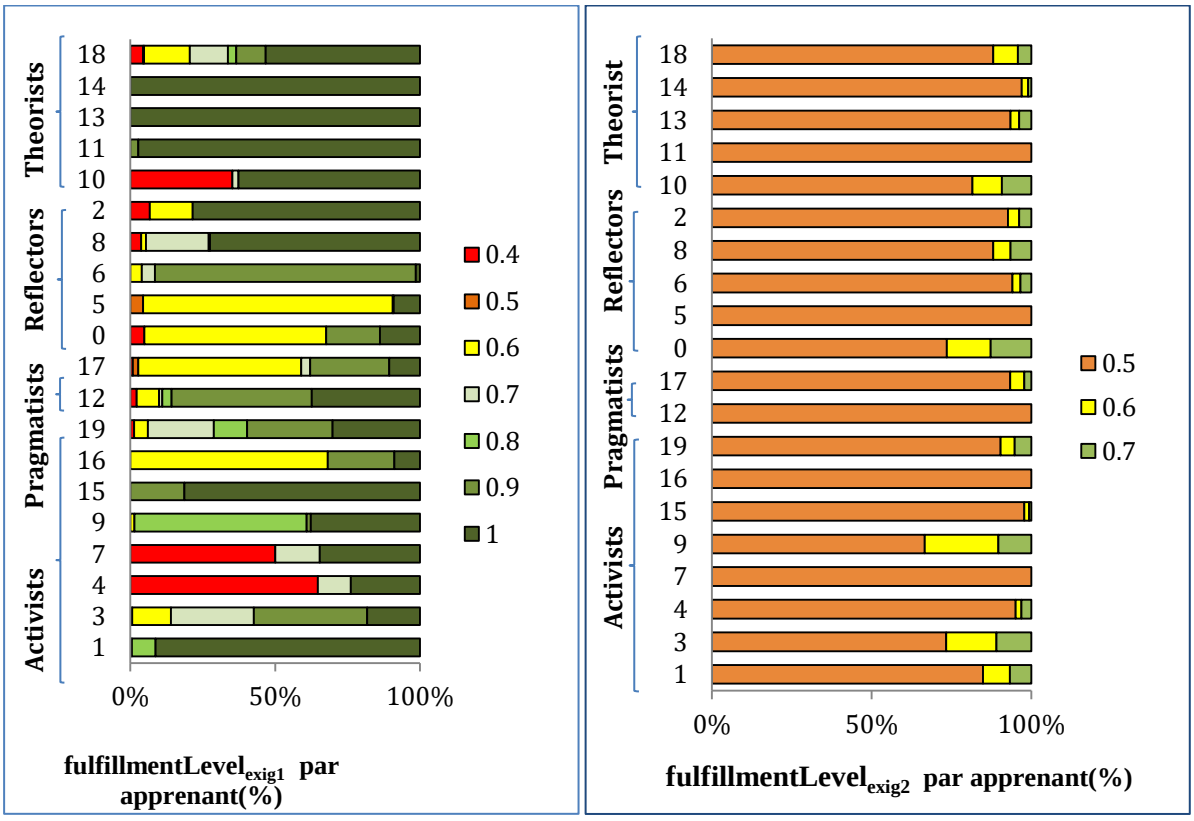


Figure 27 : Distribution du niveau d'accomplissement des exigences Exig3, et Exig4 par apprenant (code des couleurs : 'nuances du vert' Classe A ; 'jaune' Classe B ; 'orange' Classe C ; and 'rouge' Classe D). (Dounas et al., 2019)

Ainsi, comme le montre la Figure 27, l'exigence Exig1 est remplie avec des niveaux assez bons (voir code couleur) par toutes les configurations proposées aux apprenants, à l'exception de celles proposées aux apprenants « 4 » et « 7 ». L'exigence Exig2 était partiellement satisfaite, mais sa valeur s'améliore au fur et à mesure que les apprenants avancent dans le cours, à l'exception des apprenants « 5 », «7» et « 11 ».

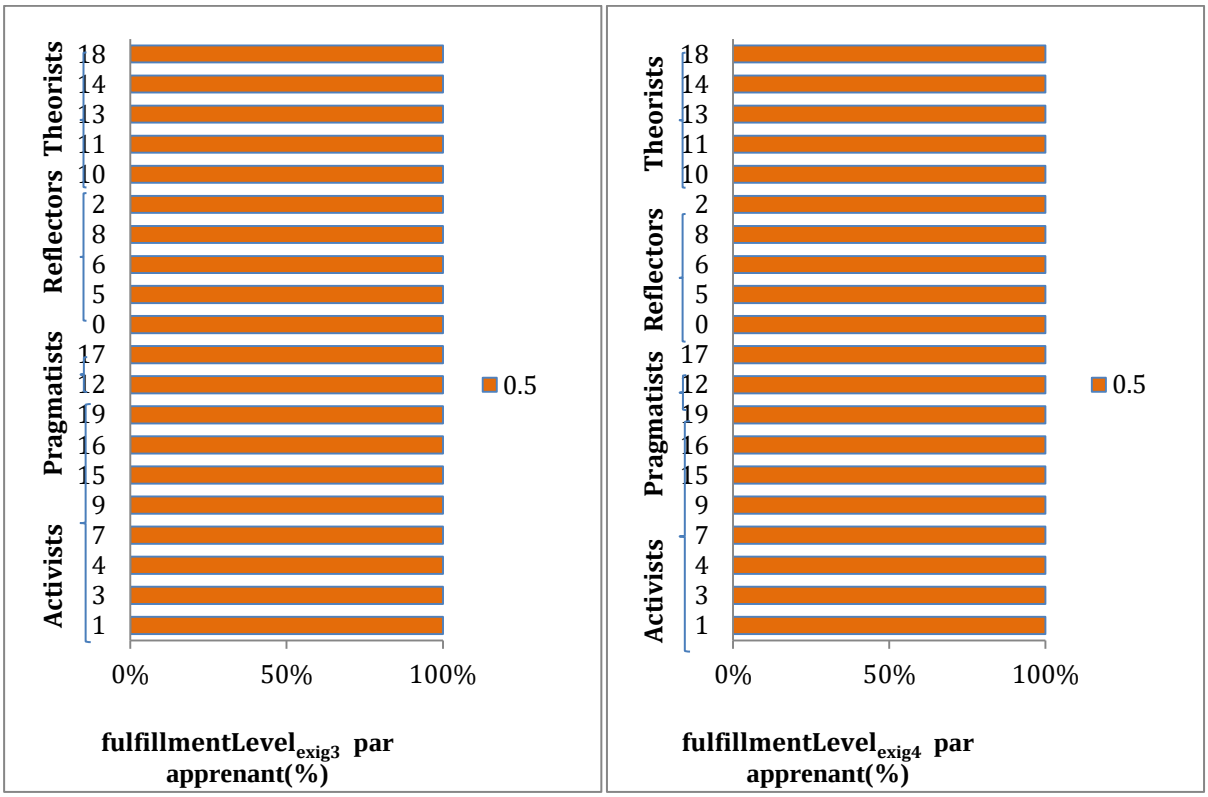


Figure 28 : Distribution du niveau d'accomplissement des exigences Exig1, Exig2 par apprenant (code des couleurs : 'nuances du vert' Classe A ; 'jaune' Classe B ; 'orange' Classe C ; and 'rouge' Classe D. (Dounas et al., 2019)

D'autre part, comme le montre la Figure 28, les exigences Exig3 et Exig4 sont partiellement remplies avec une valeur unique (=0,5) qui indique que le système n'a pas adapté ses stratégies pour prendre en charge respectivement la « communication » et le « contrôle de partage entre les apprenants et le système ». D'où le besoin des actions d'amélioration au niveau de ces deux exigences.

Qualité d'une configuration : QC

La qualité d'une configuration traduit à quel degré les éléments de la configuration satisfont les exigences. Elle est calculée selon l'équation (*) (voir section 3.2.2.4). La politique de priorisation est montrée dans le Tableau 4 ci-dessous. En effet, l'exigence Exig1 a le plus grand poids (=0.4) du fait de son importance pour le processus d'adaptation. Ce dernier se base principalement sur le style d'apprentissage pour adapter les ressources éducatives à chaque apprenant.

Les valeurs de Qc sont comprises entre 1 to 4, où les valeurs proches de 1 correspondent à des qualités médiocres. Une normalisation a été effectuée pour ramener les valeurs de Qc entre 0 et 1 selon la méthode de normalisation Min-Max.

Les résultats obtenus sont résumés et illustrés par la Figure 29.

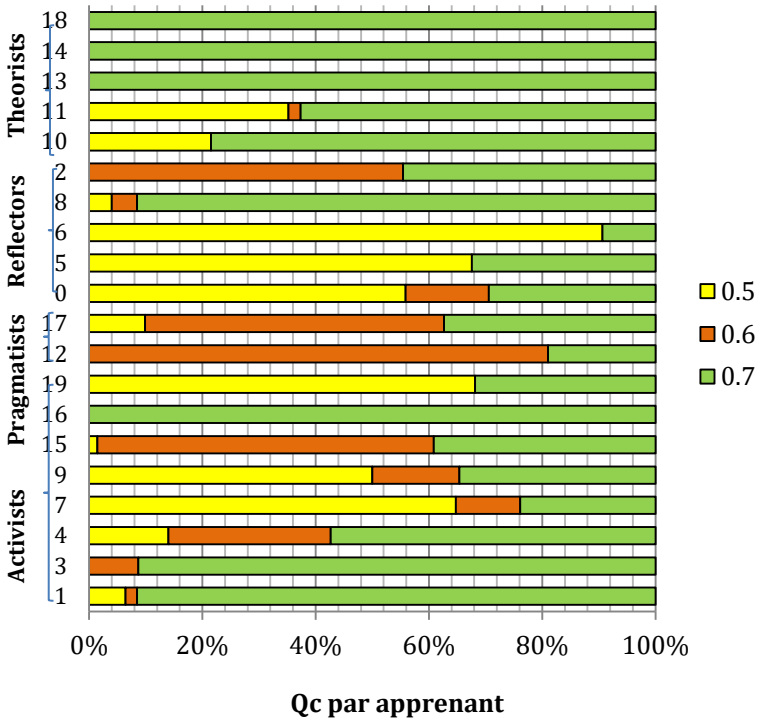


Figure 29 : Distribution de la qualité des configurations par apprenant (code des couleurs : 'nuances du vert' Classe A; 'jaune' Classe B; 'orange' Classe C; and 'rouge' Classe D. (Dounas et al., 2019)

Comme le montre la Figure 29, la qualité des configurations est en générale satisfaisante (62,28% des configurations ont une qualité entre 0,7 sur 1), mais les résultats étaient moins satisfaisants pour presque tous les apprenants réflexifs et certains apprenants actifs (les apprenants «7» et «9»).

Diagnostic des violations des exigences

Le diagnostic des exigences permet de détecter les éléments d'une configuration qui sont sources des violations des exigences. Par souci de brièveté, nous analysons ici que l'exigence Exig1.

La Figure 30 décrit les OPERS qui ont contribué aux violations de l'exigence Exig1.

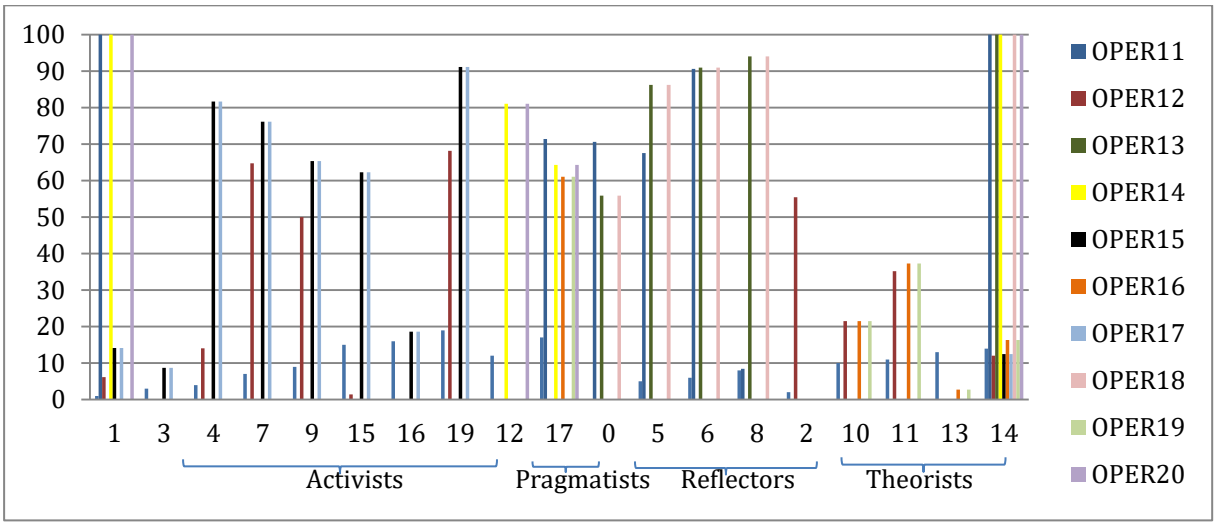


Figure 30 : distribution des OPERs sources de violation de l'exigence Exig1 par apprenant (Dounas et al., 2019)

Par exemple, OPER13= « présentation orientée exemple », OPER11= « présentation expositoire » et OPER18= « appliquer le style réfléchi », ont contribué à l'insatisfaction de l'exigence Exig 1 dans 90% des configurations proposées à l'apprenant 6. Pour vérifier ces résultats, nous avons consulté les fichiers de journalisations. En effet, le système a considéré l'apprenant 6 comme un réflexif alors que l'historique de ses interactions prouve qu'il est plutôt un théoricien car il préfère étudier en utilisant la théorie plutôt que des exemples. De même, OPER12= « présentation orientée théorie » était source des violations de l'exigence exig1 dans 56% des configurations proposées pour l'apprenant 2.
En Suivant ce raisonnement, les analystes / développeurs peuvent améliorer les stratégies d'adaptation du système en tenant compte de certains paramètres imprévus au moment de la conception (ici, il est nécessaire de mettre à jour en continu le style d'apprentissage et de ne pas s'appuyer sur le style d'apprentissage affecté lors de la première utilisation du système.

5.3 Discussion

Le résultat d'analyse des exigences a montré : (i) une bonne qualité des configurations d'INSPIRE (comprises entre 0,7 sur 1) en général. Cependant, des différences dans l'efficacité d'INSPIREus entre les groupes de styles d'apprentissage. En effet, INSPIREus était plus précis pour les apprenants théoriciens et moins précis pour les réflexifs. Des différences considérables au sein des groupes de styles d'apprentissage ont également été détectées. (ii) des violations au niveau des exigences. Par exemple, les exigences Exig1 et Exig2 ont été satisfaites par des niveaux assez bons pour presque tous les apprenants, alors les exigences Exig3 et Exig4 n'ont pas été prises en charge complètement par le système. (iii) Le diagnostic des violations des exigences dans les configurations étudiées a identifié des recommandations des ressources qui ne sont pas à l'intérêt des apprenants. Par exemple, certains apprenants ont préféré étudier avec un format de média différent de celui proposé INSPIREus. Une explication possible serait que les apprenants ne sont pas simplement réflexifs, théoriciens, actifs ou pragmatiques, mais plutôt une combinaison de tous ces styles. Le système adaptatif doit être suffisamment souple pour répondre aux besoins de chaque apprenant et ne pas être le traité comme une « classe » de style d'apprentissage.

5.4 Validité de recherche

Comme pour toute évaluation expérimentale, certaines menaces pourraient affecter la validité de nos résultats. En effet, il existe trois limitations mineures concernant la méthode de l'expérimentation :

Validité interne

La première limitation est liée à l'utilisation des fichiers de journalisation. Bien qu'ils contiennent des informations riches sur les apprenants et le comportement du système, ils contiennent inévitablement des informations incohérentes. En conséquence, un prétraitement de l'ensemble de données a été effectué pour éliminer les valeurs manquantes après la traduction des données du grec en anglais. La traduction a été vérifiée ultérieurement par l'administrateur d'INSPIREus, ainsi que des informations

supplémentaires ont été ajoutées pour consolider les données. De plus, pour éviter données aberrantes, des apprenants qui n'étaient pas engagés dans le cours ont été éliminés de cette étude (un apprenant sur 21 a été éliminé).

La deuxième limitation est liée à la spécification des exigences. Dans cette étude, les exigences ont été identifiées sur la base de problèmes généraux liés aux systèmes e-learning adaptatif et non extraites directement de la spécification du cas d'étude. Cette séparation favorise la comparaison de la qualité des systèmes entre eux, par rapport à des exigences générales. Inévitablement, certaines hypothèses, qui pourraient être subjectives, ont été ajoutées pour associer les fonctionnalités des systèmes aux exigences surveillés. Par conséquent, nous avons validé le modèle par des enseignants avant son utilisation pour assurer sa cohérence.

Validité externe

Enfin, la troisième limitation est liée à la taille de la source des données. En effet, bien que la taille de l'ensemble des configurations soit 6165. Ces configurations ont été collectées de seulement 20 apprenants, ce qui nous empêche de pouvoir généraliser les résultats à une population plus large sans le besoin de recherche ultérieure avec des échantillons plus larges.

5.5 Conclusion du chapitre

Ce chapitre a présenté une première étude de cas qui a mis en évidence le potentiel du monitoring des exigences l'exécution pour améliorer la conception des systèmes adaptatifs et permettre leur évolution. Sur la base d'une étude expérimentale, des problèmes liés à la conformité d'un système e-learning adaptatif à quatre exigences ont été identifiés, ce qui nous a permis d'expliquer les sources de ces problèmes par le biais d'un diagnostic des exigences. Les analystes et les développeurs du système cible peuvent exploiter le résultat pour programmer des améliorations des stratégies d'adaptation à la volée ou dans une phase de maintenance.

Dans le chapitre suivant, une autre étude de cas a été menée dans le domaine des transports intelligents.

Chapitre 6

6. ETUDE DE CAS 2 : CONTROLEUR ADAPTATIF D'INTERSECTION A FEUX DE CIRCULATION

La gestion dynamique du trafic routier est un levier d'action pour assurer la sécurité de la voirie, atténuer la nuisance environnementale (émission du CO2) par la réduction des congestions du trafic, et offrir un confort aux usagers. Parmi les différents moyens d'action, un contrôleur des feux de circulation permet de gérer la congestion du trafic en supprimant certains points de conflit entre les différents mouvements de circulation aux carrefours, et d'optimiser le temps d'attente aux feux.
Il existe trois types de contrôleurs des feux :

- Un contrôleur prédéterminé est basé sur une commande prédéfinie qui consiste en une séquence de phases prédéterminées avec des intervalles fixes des feux verts, jaune et rouge et une durée de cycle fixe.

- Contrairement à un contrôleur de feux précalculé, qui requiert des phases de réactualisation et de maintenances régulières, un contrôleur, dit adaptatif, se base sur un plan dynamique qui s'adapte en continue aux variations des conditions de la circulation dans la voirie pour gérer les conflits entre deux mouvements (entre véhicules ou entre véhicules et piétons). Le principe d'adaptation est simple, Le contrôleur reçoit des informations sur le trafic en temps réel issues des capteurs électromagnétiques (les plus utilisés), des radars, des caméras et récemment par des détecteurs ultrasons, et utilise ces informations pour déterminer un plan d'écoulement du trafic qui consiste à varier la durée du cycle, la durée de vert, la durée du rouge et la synchronisation entre les feux. Ce plan doit être adapté à la géométrie et la capacité du carrefour. Les plans générés par les contrôleurs adaptatifs dépendent également de plusieurs facteurs imprédictibles tels que la réaction du conducteur ou les accidents de routes. Lorsque ces facteurs changent, les performances du système ne peuvent être améliorées qu'en modifiant les paramètres de contrôle à l'aide d'un monitoring en continue des exigences. D'où le choix de la modélisation de ce type de systèmes.

La littérature distingue deux types de contrôleur adaptatif :

- Avec un contrôleur semi-adaptatif, un mouvement dans des rues principales reçoit un vert sauf s'il y a un appel contradictoire à une phase de mouvement mineur. Le cas échéant, le contrôleur utilise l'information de détection uniquement dans les rues secondaires et ne considère pas les informations de détections dans les rues principales. Ce type de contrôleur convient mieux aux intersections isolées avec une route principale à basse vitesse et un volume de carrefour plus léger.

- Un contrôleur entièrement adaptatif utilise les informations de détection pour tous les mouvements de circulation sans faire la distinction entre les rues principales et les rues secondaires. Ce type de contrôleur est parfaitement adapté aux intersections isolées où la demande de circulation et les tendances varient considérablement au cours de la journée.

Dans le cas du carrefour choisi pour l'expérience, il s'agit d'une régulation entièrement adaptative dans la mesure où tous les feux sont commutés en temps réel en fonction des données collectées dans les voies. Le détail de la stratégie est tiré du manuel TSTM qui explique en détail les procédures utilisées pour générer des plans dynamiques de commande des feux et la politique de synchronisation des signaux à respecter qui découlent des objectives globaux du transport régional nord-américain. Ce qui a facilité la phase de modélisation du système et des exigences. Enfin, l'intérêt pour un contrôleur adaptatif est qu'il est facile de créer des modèles de buts de tailles moyennes et grandes cohérentes par clone, si on suppose que le contrôleur gère plusieurs carrefours en même temps. Cela a pour but d'évaluer le passage à l'échelle du prototype RMAS.

6.1 Notions fondamentales concernant le trafic aux carrefours à feux

Dans cette section, nous définissons les notions fondamentales utilisées dans l'ingénierie de transport, à savoir carrefour, mouvement, courant, cycle, intervalle et phase.

Un carrefour est situé à la rencontre de plusieurs branches, déterminant des voies d'entrées et de sorties. Il peut revêtir de plusieurs formes tant dans le nombre des branches, la répartition et l'épaisseur des voies (par exemple 2 voies, 1× 2 voies, 2 × 2 voies, etc.), que dans les règles de franchissement qui caractérisent des courants de flux qui sont soit direct, soit des courants de tourne-à-gauche ou des courants de tourne-à-droite.

Ces paramètres ont un impact sur la durée de chaque feu et la durée du cycle (enchaînement des différentes phases des feux) en général. Par exemple, un mouvement desservi par deux voies plutôt que par une, a une capacité plus grande et nécessite donc moins de temps vert pour l'écoulement du trafic. D'autre part, l'augmentation du nombre de voies sur une branche augmente également le temps de passage pour les piétons sur cette branche et donc la durée du feu rouge.

Pour la gestion des flux d'entrée et sorties, plusieurs types de signaux de circulation peuvent être utilisés :

Les signaux tricolores circulaires : sont composés de trois états : vert, jaune et rouge pour les mouvements dans le sens direct.

- Le feu vert autorise le passage des véhicules tout droit, mais ne donne pas la priorité de tourne-à-gauche.
- Le feu jaune interdit le passage aux véhicules et aux piétons. Il sert à dégager les véhicules déjà engagés et à les informer d'un passage imminent au feu rouge en moyenne dans 3 à 5 secondes. Cependant, pour limiter le risque de collisions par l'arrière, le passage est autorisé dans le cas où les conducteurs estiment qu'il ne pourrait pas s'arrêter à temps au niveau de la ligne d'effet de signal.
- Le feu rouge indique une interdiction de passer. Cependant, en Amérique du Nord, il est possible de tourne-à-droite en laissant la priorité aux piétons.

- Le jaune clignotant : peut être utilisé dans la nuit ou dans des périodes où la circulation est moins importante, pour rappeler le danger lié aux intersections et autoriser le passage des véhicules sous réserve de respecter les règles de la priorité à droite.

Les signaux directionnels tricolores pour tourner à gauche ou « le tourne-à-gauche protégé » : Ils sont destinés aux véhicules qui ont pour destination la direction indiquée par la flèche (ici à gauche). Ils se composent de trois flèches lumineuses verte, jaune et rouge. La plupart du temps les permissions de tourner sont uniquement sur une flèche verte.

- La flèche verte autorise aux véhicules de tourner à gauche.
- La flèche jaune indique un passage imminent au rouge.
- La flèche rouge interdit aux véhicules de tourner à gauche.

Les piétons doivent également suivre les feux de circulation. Certaines intersections, ont des panneaux WALK et DON'T WALK activés avec un bouton-poussoir sur un poteau. La signification des signaux pour piétons est :
WALK indique aux piétons qu'ils peuvent légalement.
DON'T WALK indique aux piétons qu'ils doivent rester sur le trottoir.

Enfin, Les feux de circulation sont caractérisés par le plan de feux, l'intervalle, et la phase, qui sont définis comme suit:
Le plan de feu est une ligne du temps qui voit se succéder les phases de feux d'un carrefour. Un plan cyclique est décrit par une séquence d'états prédéfinis de l'ensemble des feux du carrefour.
Un intervalle est défini selon la norme NTCIP 1202 comme « *une période du cycle pendant laquelle les indications de signal ne changent pas* ». La longueur de l'intervalle dépend de plusieurs paramètres. Par exemple la longueur de l'intervalle du vert pour les véhicules est contrôlée par la durée du vert minimale, la durée du vert maximale et le temps de passage.
Une phase est la partie du cycle donnée à un mouvement ou une combinaison de mouvements non conflictuels au cours d'un ou de plusieurs intervalles. **Le mouvement** étant un ensemble de véhicules qui proviennent de la même origine et ont la même destination, ou un ensemble de piétons sans origine ou destination.

6.2 La régularisation du trafic

L'emploi des feux de circulation a pour but d'assurer la sécurité des piétons et des conducteurs et d'améliorer la fluidité de la circulation. Les plans de commande des feux générés doivent être fiables et respectent les politiques régionales et locales dont voici quelques exemples :

Exig 1 : Le contrôleur doit déterminer une durée appropriée de chaque feu qui dessert le mieux les usagers attendant dans le carrefour (véhicules ou piétons). Si par exemple, M. véhicules arrivent alors que le feu est rouge, le vert suivant persistera p secondes jusqu'au départ de ces véhicules.

Exig 2 : La durée minimale du vert doit être respectée. Elle est en moyenne de 5 secondes.

Exig 3 : la synchronisation des signaux entre les différentes phases ne doit pas présenter un risque pour les usagers.

Exig 4 : Tout conflit entre la demande de piétons et le volume de trafic doit être décidé en faveur des piétons.

Exig 5 : À un certain moment de la journée, l'intersection suspend automatiquement le service normal et son voyant clignote en jaune.

Exig 6 : Le piéton ne devrait plus attendre longtemps pour traverser après avoir appuyé sur un bouton-poussoir.

Exig 7 : Seul le bouton-poussoir intercepté alors que le voyant est rouge est pris en charge.

Exig 8 : le contrôleur doit minimiser les longueurs des cycles. En effet, les durées des cycles trop longues peuvent aggraver la congestion plutôt que de la réduire en raison des longues files d'attente que vont engendrer dans les rues secondaires. En général, il est préférable que les longueurs de cycle des intersections classiques à quatre branches ne dépassent pas 120 secondes.

Ces politiques de la régularisation du trafic sont respectées en utilisant des stratégies de régularisation qui consistent à varier les durées du vert, jaune et rouge selon les procédures suivantes :

- Prolonger la durée du vert : En principe, le contrôleur commence par allouer un temps de vert minimal. Et pendant la circulation, ce temps du vert peut être prolongé par une ou plusieurs unités d'extension pour répondre à la demande des véhicules dans une phase. Cependant, la durée du vert ne doit pas dépasser une valeur de vert maximale qui est définie préalablement, avec la durée minimale. Ces deux valeurs limites sont susceptibles de changer manuellement ou automatiquement en considérant les statistiques de trafic sur une longue période.
- Prolonger la durée du rouge pour répondre aux besoins des piétons.
- Supprimer une période de feu dans une phase: consiste à couper ou supprimer une période du feu pour répondre à une priorité. À l'exception du temps minimal du vert qui est règlementé et qui n'est pas autorisé de le couper sous aucune priorité.
- Programmer un rouge de dégagement: consiste à définir une durée d'écoulement permettant à un piéton engagé à la dernière seconde de vert « walk » de passer en toute sécurité, même si le feu est passé au rouge « don't walk » pour lui ; Ou à un véhicule engagé à la dernière seconde du jaune de passer en toute sécurité. Ces durées sont calculées à raison de 10 m/s pour les véhicules et 1 m/s pour les piétons.
- Programmer la période du jaune : généralement la période du jaune est fixe dans la plupart des systèmes de feux. Cependant, elle peut être calculée sur la base des données collectées du trafic.

- Activation du jaune clignotant dans la nuit ou dans des périodes du jour connu par un trafic très faible.
- Synchroniser les signaux: le contrôleur calcule les paramètres de synchronisation appropriés pour éliminer l'occurrence de deux signaux contradictoires qui représentent un risque pour les usagers de la route.
- Rétrécir et allonger le cycle des feux.

6.3 Description générale du système du monitoring pour un contrôleur de feux adaptatif

Comme nous l'avons évoqué précédemment, le but du monitoring des exigences consiste à s'assurer que les plans des feux générés sont effectifs et fiables et à identifier les raisons des cycles longues.
Le fonctionnement général se fait itérativement, via une boucle reliant le système du monitoring et l'environnement qui dans notre contexte est constitué du carrefour, du contrôleur de l'intersection et des capteurs de détection (voir Figure 31). Le système du monitoring se base sur les informations ci-dessous pour analyser les exigences :

- Les états courants des feux du carrefour.
- La durée de chaque feu.
- Le temps d'attente des piétons.
- Le plan des feux.
- Le temps effectif du vert (en anglais, effective green time) qui représente la quantité du temps utilisable par les véhicules, c'est-à-dire la durée entre le premier départ des véhicules et la fin du feu jaune.
- Le débit de saturation (en anglais, saturation flow rate) qui est la marge en secondes entre les véhicules sortant d'une file d'attente, divisée en 3600 secondes. Il caractérise l'écoulement des véhicules. Le HCM fournit une technique standardisée pour mesurer le débit de saturation. Il est basé sur la mesure de l'avancement entre les véhicules partant de la barre d'arrêt, limitée aux véhicules situés entre la quatrième position de la file d'attente. Cette pratique vise à minimiser les effets du temps perdu au démarrage et la fin de la file d'attente. Par exemple, Les véhicules sortant d'une file d'attente avec une avance moyenne de 2 secondes ont un débit de saturation de 3 600 / 2 = 1800 véhicules par heure et par voie.

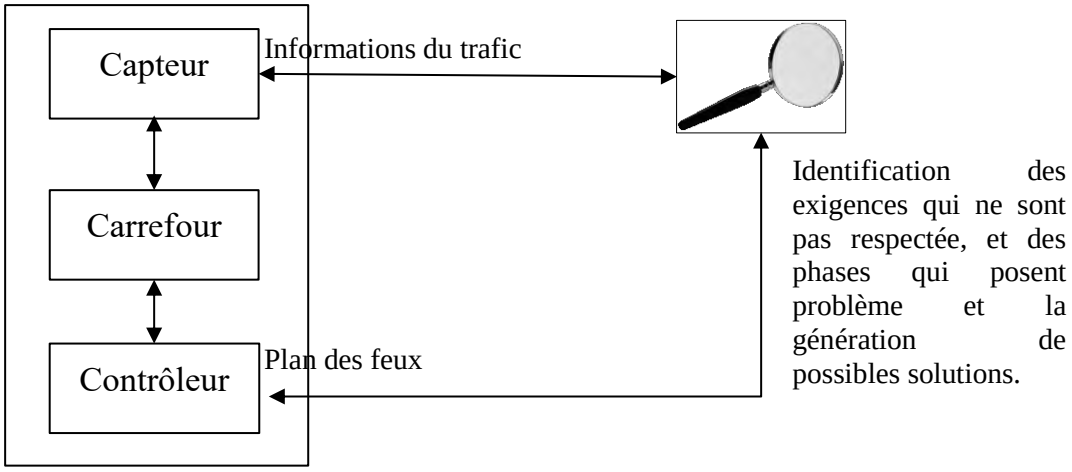


Figure 31 : fonctionnement général d'un monitoring des exigences pour un système de contrôle de feux de circulation

Le monitoring permet de détecter les violations des exigences – dans notre cas il s'agit de la sécurité et de la fluidité ; et d'identifier les phases et les stratégies qui sont sources de violations ainsi que de possibles stratégies de remédiation.

À cet objectif, un modèle de variabilité est créé pour décrire les procédures de commandes de feux suivies pour la régularisation du trafic permettant de générer un plan des feux. Le modèle de soft goals décrit les critères d'optimisation, le modèle de contexte décrit les variables de contexte décrivant l'état du trafic. Enfin, le modèle satisficing soft goals décrit les stratégies de préférences des différents buts de bas niveaux sur la base de son impact sur la satisfaction des soft goals.

Le modèle du carrefour sur lequel s'appuie cette expérience est représenté dans la Figure 32. Il comporte quatre branches à deux sens. Chacune de ces branches est décomposée en deux voies entrantes et deux voies sortantes. La voie entrante de gauche est réservée aux véhicules tournant à gauche et la voie de droite aux véhicules allant tout droit ou tournant à droite.

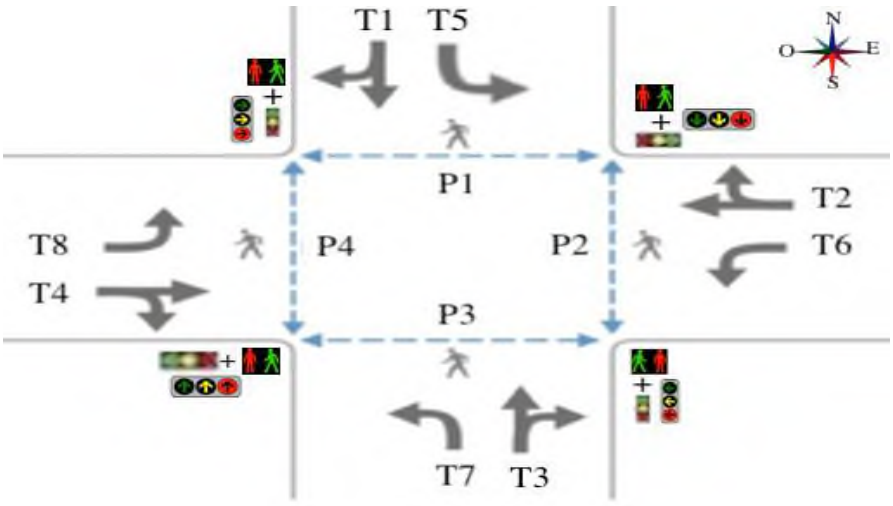


Figure 32 : Mouvements typiques des véhicules et des piétons dans une intersection à quatre branches aux deux sens et un virage à gauche protégé

Dans ce type de carrefour, 12 mouvements sont possibles :

- 8 mouvements de véhicules : les mouvements directs et les mouvements de tourne-à-droite sont contrôlés par un même feu et partagent donc les mêmes phases, alors que les mouvements de tourne-à-gauche sont contrôlés par les flèches directionnelles.
- 4 mouvements de piétons, contrôlés par des panneaux lumineux de « Walk » et « Don't Walk ».

Pendant la phase de modélisation, notre but étant de créer des modèles d'exigences de différentes tailles, afin d'évaluer la performance du monitoring et sa mise à l'échelle. Pour ce faire, des critères spécifiques tels que le sens des voies, le nombre de passages piétons, du nombre et de la distribution des feux dans le carrefour, ont été personnalisés dans chaque modèle. Comme le montre le Tableau 8, 15 modèles ont été créés. Les modèles de M1 à M7 concernent un carrefour isolé. Tandis que les modèles de M8 à M15 concernent plusieurs carrefours en coordination.

Tableau 8 : nombre de phases contrôlées par le contrôleur dans chaque modèle de carrefours.

Modèle	# phases	#buts	#OPERs	# buts total
M1	4	16	24	40
M2	6	22	33	55
M3	8	29	44	73
M4	9	35	52	87
M5	10	41	61	102
M6	11	47	68	115
M7	12	53	76	129
M8	14	61	87	148
M9	16	68	98	166
M10	18	75	109	184
M11	20	82	120	202
M12	21	88	129	217
M13	22	94	137	231
M14	23	100	144	244
M15	24	107	152	259

Par exemple, M1 décrit un système central qui contrôle un carrefour à sens unique avec un cycle de 4 phases (T1, T2, P1 et P2), M2 décrit un système central qui contrôle un carrefour à double sens, avec des virages à gauche permissifs. Et M3 décrit un système central qui contrôle un carrefour à double sens, avec les virages à gauche protégés par des feux. Le reste des modèles représentent la coordination en plusieurs carrefours et sont construits par le clone du M1 plusieurs fois dans le but de créer des modèles de grandes tailles.

6.4 Spécification des exigences du monitoring par REFAS

Dans un but de simplification, les illustrations des différentes vues de REFAS concerneront seulement le modèle M1.

Comme le montre la Figure 33, le carrefour est constitué de quatre phases T1, T2, P1 et P2. La phase T1 comprend un mouvement de véhicules direct et un mouvement de rotation à droite, tandis que la phase T2 a un mouvement de véhicules direct et un mouvement de rotation à gauche. Les phases P1 et P2 comprennent des mouvements de piétons contrôlés par un feu.

On note S= {S1, S2} les deux rues respectivement Nord-Sud et Est-Ouest.

F = {F1, F2, F3, F4} le nombre des feux sur le carrefour.

- Les indications du F1 sont : vert, jaune et rouge, qu'on note respectivement green1, yellow1 et red1.
- Les indications du F2 sont : vert, jaune et rouge, qu'on note respectivement green2, yellow2 et red2.
- De même, les indications du F3 sont walk 3 et don't walk 3.
- Enfin, les indications du F4 sont walk4 et don't walk 4.

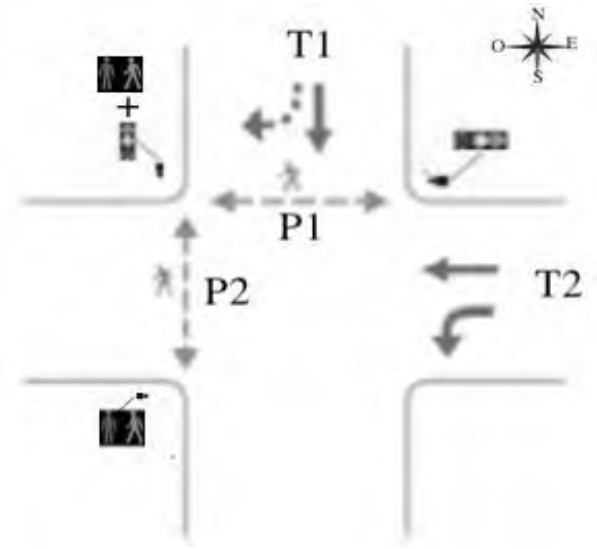


Figure 33 : Schéma de phasage et de déplacement pour l'intersection de deux voies à sens unique

Le contrôleur se base sur les informations de détection des véhicules et piétons pour déterminer les durées du vert, jaune et rouge du feu F1 et F2 qui sont destinés aux véhicules. Les indications des feux P1 et P2, à savoir Walk et don't walk sont déduites directement des durées des feux F3 et F4. Par exemple, si le feu F1 est rouge implique directement que l'état du F3 est walk et s'il est vert implique directement un don't walk.

A. Vue de variabilité

Comme le montre la Figure 34, La décomposition des buts commence par un but racine qui est le contrôle du trafic dans le carrefour décrit ci-dessus. Les stratégies de commande des feux sont de trois Type : (i) les stratégies d'affichage et de synchronisation des feux pour assurer la sécurité des usagers. Dans notre cas ses stratégies sont contrôlées par des relations « require » ou « exclude ». (ii) les

stratégies de calcul de la durée de chaque feu pour répondre à la demande. Cela consiste à allouer un temps minimum et de prolonger ou couper (omit) le temps du feu pendant la circulation pour répondre à une priorité. Le temps alloué ne doit pas dépasser un temps maximum prédéfini. Et (iii) les stratégies de contrôle de la longueur de cycle, en autorisant les extensions du cycle ou la réduction de sa longueur.

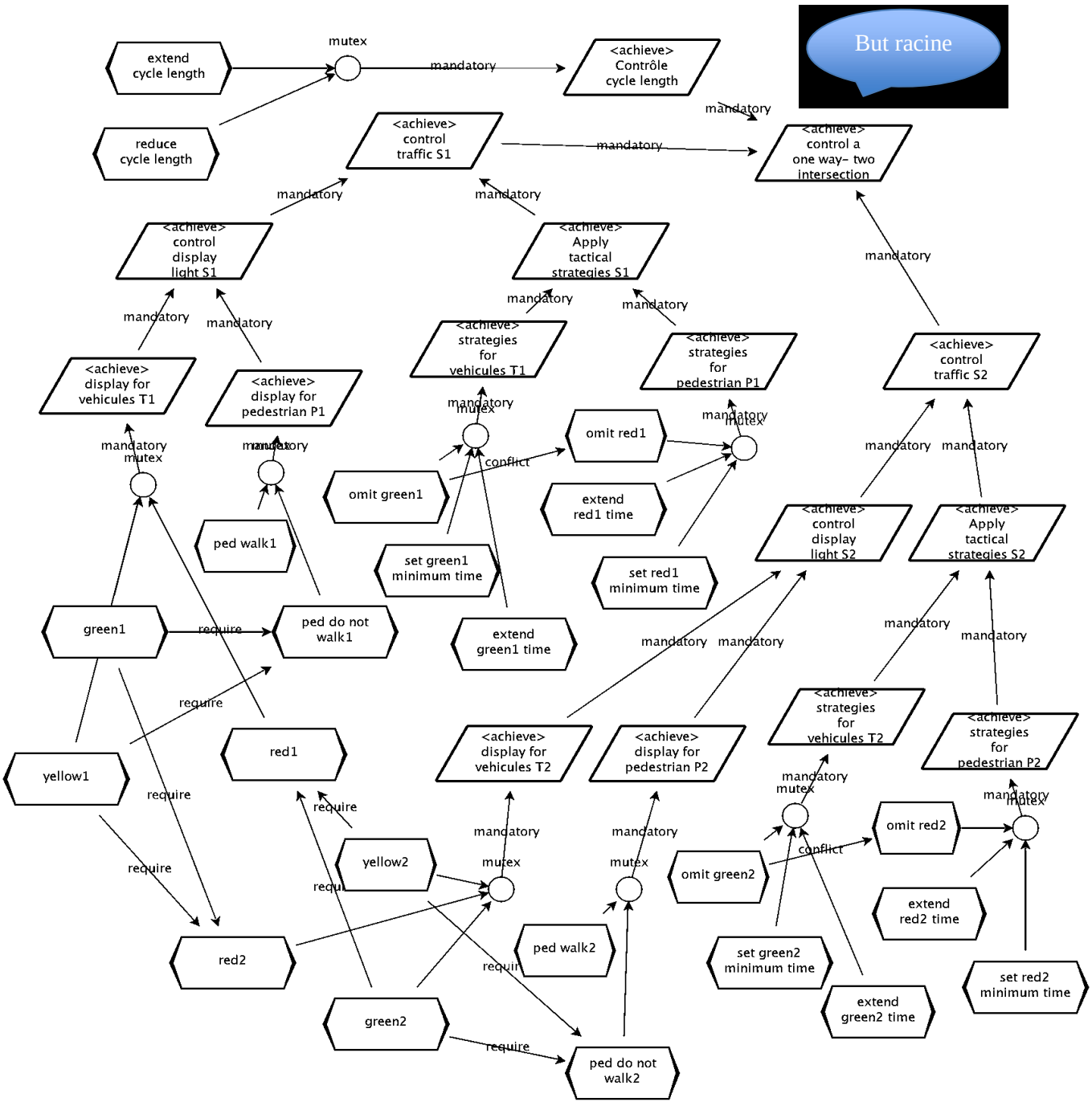


Figure 34 : la variabilité pour le contrôleur de carrefour à feux

B. La vue soft goals

Les enjeux importants de contrôle de feux sont : la sécurité et la fluidité du trafic. La sécurité est assurée par une synchronisation des feux sans risque alors que la fluidité du trafic est assurée par les différentes stratégies de commande de feu pour générer un plan des feux. La fluidité est mesurée dans notre cas par le degré de saturation de chaque voie, le degré de saturation du carrefour ainsi que le temps d'attente des piétons.

Comme le montre la Figure 35, Les critères d'optimisation pris en charge par le monitoring sont :

A. Soft-goal 1 : Minimiser la saturation de la voie 1
B. Soft-goal 2 : Minimiser la saturation de la voie 2
C. Soft-goal 3 : Minimiser le temps d'attente des piétons dans P3.
D. Soft-goal 4 : Minimiser le temps d'attente des piétons dans P4.
E. Softg-goal 5: Minimiser la longeur du cycle.
F. Soft-goal 6: réduire l'émission du CO2 par les véhicules. En effet, les feux mal réglés peuvent entraîner le triplement de la consommation de carburant, donc les émissions de CO_2, lorsque la circulation est saturée.

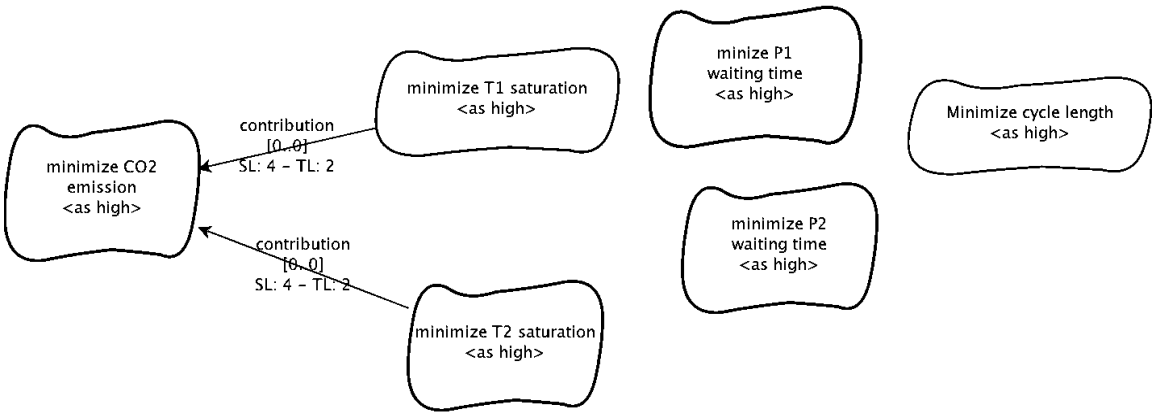


Figure 35 : la vue soft goals pour le contrôleur de carrefour à feux

Les soft goals 1 et 2 contribuent à la satisfaction du soft-goal 6 avec un niveau 2. En effet, une circulation très perturbée implique des « stop-and-go » et donc une forte émission du CO2.

C. La vue de contexte

La multiplicité des enjeux et des types d'usagers amènent les ingénieurs des feux de trafic à considérer plusieurs critères pour la régularisation du trafic. Ces critères sont classés selon l'approche de régularisation adoptée :

- Les approches macroscopiques se basent sur les mesures de la quantité ou de la qualité du trafic telles que le débit et la densité du trafic, la longueur moyenne d'une file d'attente.

- Les approches microscopiques capturent les mouvements des véhicules individuellement, ainsi des critères telles que l'avancement dans le temps ou l'espace, qui sont inclus dans les mesures de l'état du trafic.

Selon le manuel de capacité routière MHC 2000, le degré de saturation est l'un des mesures qui est couramment utilisé pour évaluer les opérations d'intersection signalisée. Pour ce cas d'étude, le degré de la saturation est donc pris en considération avec le temps d'attente des piétons.

La Figure 36 présente les variables du contexte. On note :

- SV1, PT1 le degré de saturation dans la voie 1, et le temps d'attente des piétons dans la ligne perpendiculaire à la voie 1.
- SV2, PT2 le degré de saturation dans la voie 2, et le temps d'attente des piétons dans la ligne perpendiculaire à la voie 2.
- XC le degré de saturation critique dans tout le carrefour.

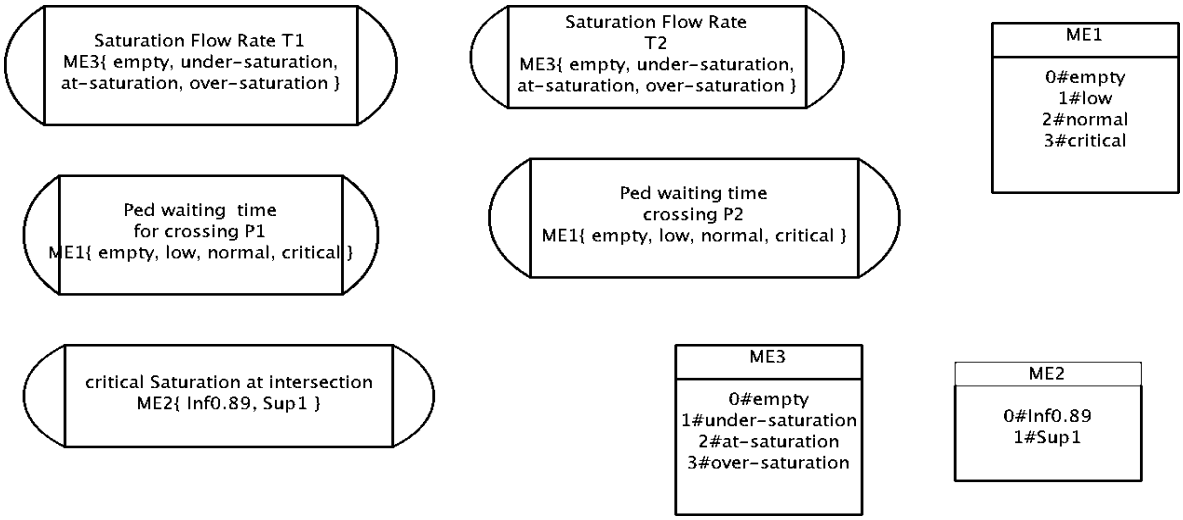


Figure 36 : la vue de contexte du trafic

Temps d'attente des piétons : PT

Le temps d'attente des piétons en attente entre deux phases est calculé sur les informations reçues par les détecteurs de piétons. Elle est mesurée à partir de la période du vert.

Une abstraction mathématique est appliquée au temps d'attente. Comme indiqué dans le Tableau 9, le temps « empty » signifie que PT vaut zéro en l'absence des piétons en attente de traverser. Le temps est « low » si la durée d'attente est inférieure au temps minimal du vert. Le temps d'attente est « normal » ou dans les normes si la durée ne dépasse pas 30 secondes. En effet, selon Battaglia Lagarde (2014), les piétons deviennent impatients quand le retard moyen est supérieur à 30 secs et ils ne suivent plus les indications. Par conséquent, Le temps est classé critique si PT >30 secs.

Tableau 9 : Abstraction mathématique du temps d'attente des piétons

Classe	Intervalle du temps (sec)
Empty	PT=0
Low	0< PT <= temps minimal du vert
Normal	Temps minimal du vert< PT <30
critical	PT > 30

Degré de la saturation d'une voie : SV

Le degré de la saturation d'une voie, connu sous le nom « ratio v/c » (volume/capacité), sert à déterminer l'état d'une voie. Elle est calculée en divisant le nombre des véhicules traversant sur la capacité de la voie (définie ci-dessous).

- La capacité d'une voie représente le nombre de véhicules pouvant théoriquement passer. Elle est définie par l'équation suivante :

$$c = s \times \frac{g}{C}$$

Ou g est la durée du vert effectif (en sec), C la durée du cycle (en sec) et s est le débit de saturation (en véh/h). Ceux-ci sont susmentionnés et illustrés par la Figure 37. Les états du feu sont présentés sur l'axe horizontal, et le débit des véhicules sur l'axe vertical.

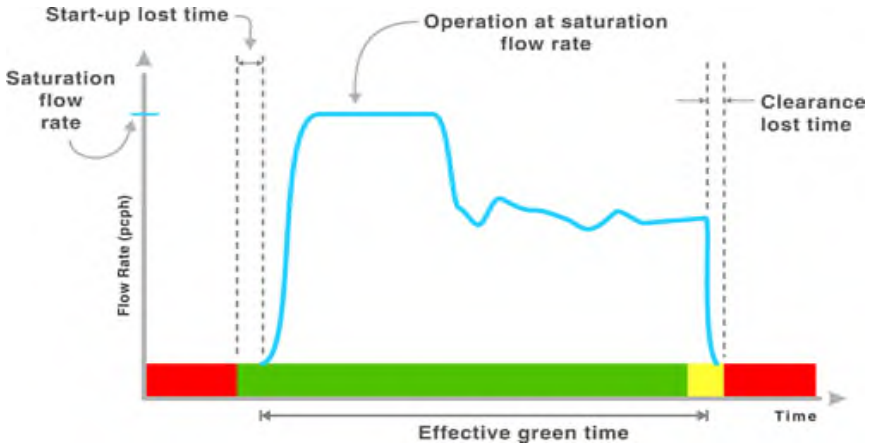


Figure 37: débit de saturation, temps effectif du vert, et temps perdu dans un mouvement[11]

Pendant que le mouvement reçoit une indication rouge, les véhicules arrivent et forment une file d'attente. Avec le passage au vert, le premier véhicule met un retard pour démarrer et les véhicules derrières prennent également du temps à accélérer. Ceci est défini comme le temps perdu au démarrage qui est supposé être d'environ 2 secondes. Selon le HCM 2000, le débit tend à se stabiliser après le cinquième véhicule au débit maximal permis par les conditions, appelé débit de saturation.

[11] https://ops.fhwa.dot.gov/publications/fhwahop08024/chapter3.htm

Ceci est généralement maintenu jusqu'à ce que le dernier véhicule de la file d'attente quitte l'intersection.

Dans le HCM, une voie avec v/c < 0,85 est considérée comme sous-saturée et stable. Un v/c entre 0,85 à 1,00 indique que le flux de trafic devient moins stable. Enfin, un v/c > 1 indique que la demande est supérieure à la capacité, ce qui nécessite des actions d'écoulement de trafic.

D'où l'abstraction mathématique du degré de saturation en 4 intervalles (voir Tableau 10).

Tableau 10 : abstraction mathématique de la SV

Classe	**Intervalle du temps (sec)**
Empty	SV=0
Under-saturation	SV <=0.85
At-saturation	0.85< SV <1
Over-saturation	SV >= 1

Degré de saturation critique : XC

Un autre concept utilisé pour analyser les intersections signalées est le rapport critique (v / c) pour l'intersection dans son ensemble, en considérant uniquement les groupes de voies (ci) ayant le rapport de débit le plus élevé (v / c) pour une phase de signal donnée. Par exemple, avec un signal à deux phases, les groupes de voies opposés se déplacent pendant le même temps vert. Généralement, l'un de ces deux groupes de voies nécessitera plus de temps vert que l'autre et aura un rapport d'écoulement plus élevé. Alors, le groupe de la voie critique qui sera considéré. XC est définie selon l'équation suivante :

$$Xc = \sum_{ci} (\frac{v}{s}) \ (\frac{C}{C-L})$$

Où C est la longueur du cycle (sec), L est la longueur du passage piéton (m).

XC peut être utilisé pour indiquer si les caractéristiques de conception de la géométrie physique et la conception du signal fournissent une capacité suffisante pour l'intersection. Si XC <= 0.85 indique que tous les mouvements dans l'intersection peuvent être adaptés à la longueur de cycle définie.

D. La vue satisficing soft goals

Les relations entre les OPERs et les soft goals ainsi que les variables de contexte et les soft goals sont représentées dans la Figure 38. Par exemple, le claim CL8 indique qu'une réduction de la longueur du cycle contribue à satisfaire le soft-goal SG5 « mimimize cycle length » par un niveau 4. Alors que SD10 spécifie qu'une durée critique de saturation faible (XC<0.85) implique un niveau de satisfaction requis égale à 4 pour SG5. En effet, des valeurs faibles de XC indiquent un besoin de diminuer la durée du cycle pour minimiser le retard à l'intersection.

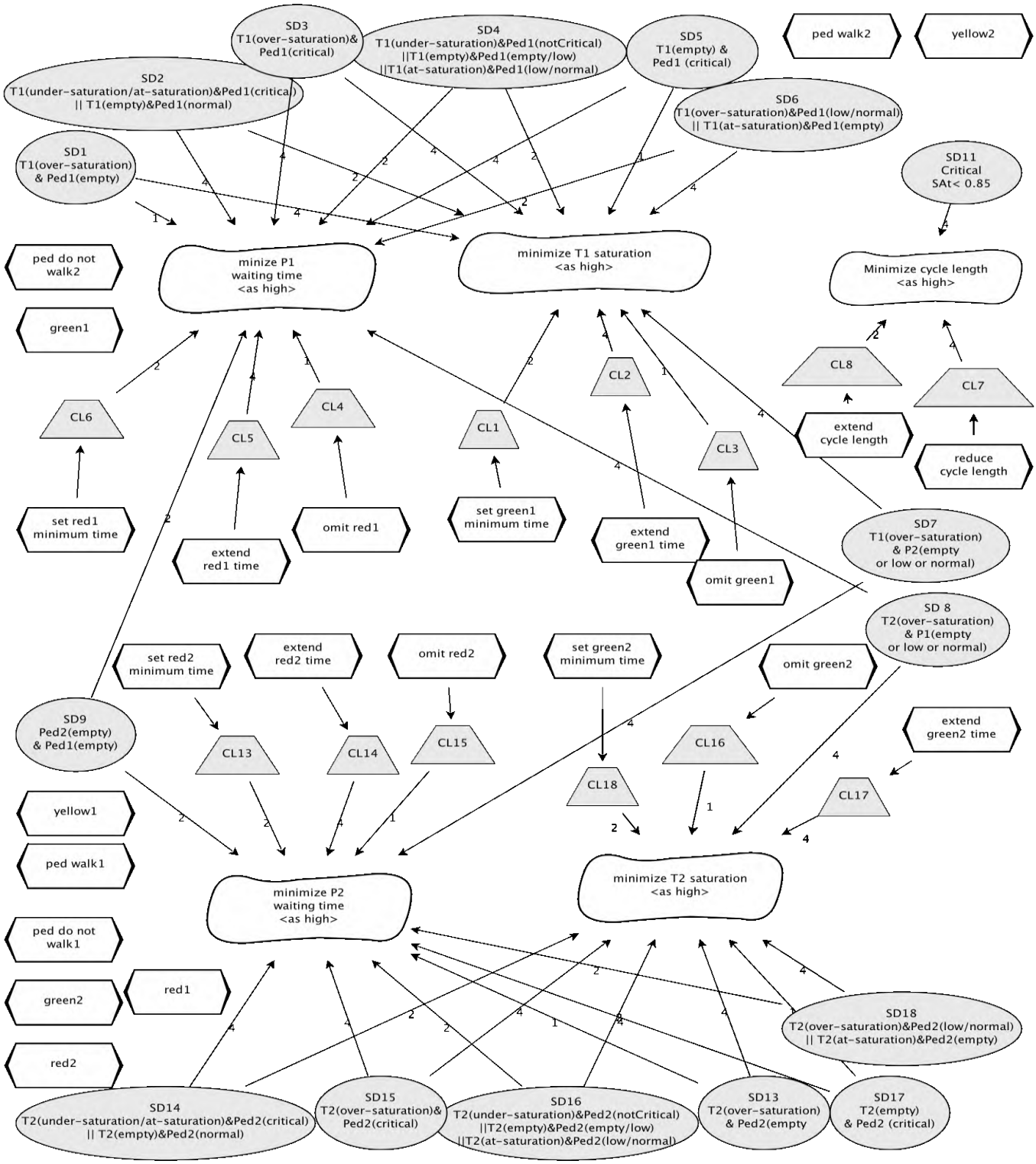


Figure 38 : la vue satisficing soft goals pour le contrôleur de carrefour à feux

6.5 Validation expérimentale

L'expérimentation avec ce cas d'étude a pour but d'évaluer la performance de notre prototype RMAS. Trois séries d'expérimentation ont été conduites : La première et la deuxième série d'expérimentations visent à évaluer le temps d'exécution en simulant des données prévenant du trafic routier. Et La

deuxième série vise à évaluer le passage à l'échelle en mesurant la performance du RMAS face à des modèles d'exigences de tailles industrielles (des TPE/PME).

Toutes les expériences ont été effectuées sur une machine avec un processeur Intel® Core™ i3- 370M (2,4 GHz) et 4 Go de RAM et en utilisant le solveur de contrainte Swi-prolog.

Expérimentation 1

But : Evaluer le temps d'analyse d'une spécification d'exigence en fonction du nombre des liens de contribution entre OPERs, notamment les relations « require », « conflict » et « mutex ». À cet objectif, 9 expériences ont été conduites à partir du modèle d'exigences M2 (voir Tableau 1) mais avec un nombre différent des liens de contribution. De plus, la même expérimentation a été faite sur des modèles d'exigences dérivés du M2 et qui sont obtenus en augmentant le nombre des buts et OPERs, jusqu'à ce que RMAS ne rapporte aucun temps d'exécution. Les résultats de cette première évaluation du RMAS sont présentés dans le Tableau 11. Ce dernier résume le nombre total de buts (#total des buts), le nombre des liens de contribution dans le modèle (#liens de contributions) et finalement le temps d'analyse des exigences et de diagnostic (Tdiag). Ce dernier représente le temps d'analyse par l'algorithme 2 + le temps de diagnostic par l'algorithme 3.

Tableau 11 : la relation entre les liens de contributions et le temps d'analyse des exigences

#total des buts	#liens de contributions	Tdiag (sec)
Modèle d'exigence initial M2		
55 buts	23	0,63
	20	0,72
	17	0,79
	14	0,85
	11	0,88
	8	0,93
	5	1,38
	2	2,19
	0	4,19
Dernier modèle d'exigence dérivé à partir de M2		
73 buts	36	0.52
	32	0,74
	29	0,92
	26	0,96
	23	1,05
	17	2
	12	2,94
	6	15,36
	4	10min ss réponse

Comme le montre le Tableau 11, le temps d'analyse et de diagnostic est inversement proportionnel au nombre des liens de contribution. En effet, plus le nombre des liens de contribution augmente plus

l'espace de recherche rétrécit et plus le nombre des affectations des vérités se réduisent en améliorant ainsi temps de résolution avec le solveur de contraintes.

Le prototype RMAS a réussi à analyser tous les modèles dérivés à partir du M2 (55 buts) allant de 23 liens de contribution jusqu'au 0 lien de contribution. Cependant, arrivant à un modèle de 73 buts, RMAS n'a pas pu terminer l'analyse avec 4 liens de contributions.

Expérimentation 2

But : Evaluer le comportement du RMAS et sa performance avec des erreurs injectées à différentes granularités. 8 expérimentations ont été faites avec un même modèle d'exigences. Ce dernier est dérivé du modèle d'exigences présenté dans la section 6.4 en augmentant seulement le nombre des niveaux dans la vue de variabilité (voir Figure 39). Les autres vues de la spécification REFAS restent les mêmes.

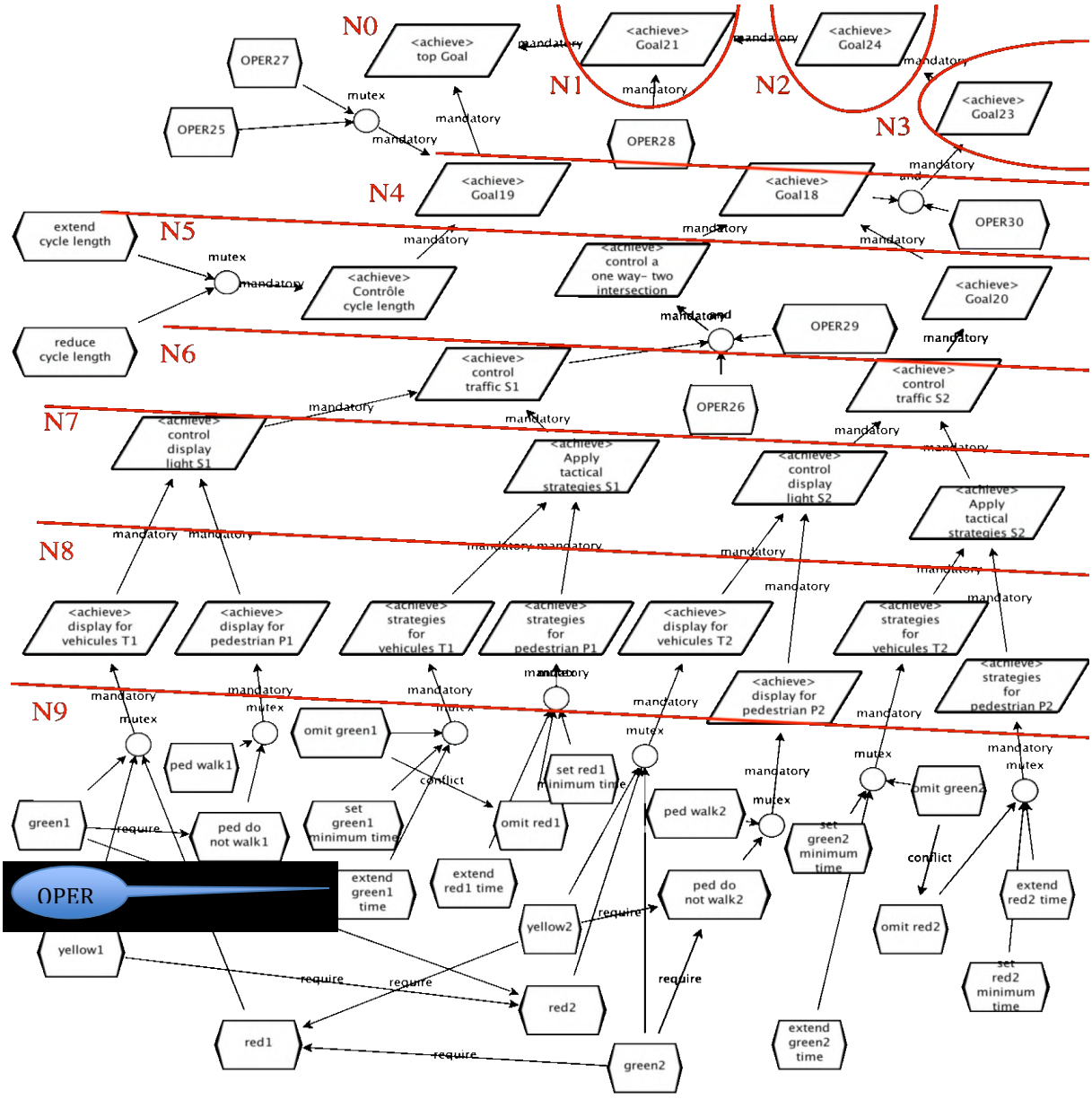


Figure 39: Modèle d'exigences à 10 niveaux

À cet objectif, Nous avons injecté deux erreurs en sélectionnant OPER8= « extend green 1 time » et OPER 11= « extend red1 time » avec les conditions de contexte suivantes : SV1=1, SV2=1, PT1=3, PT2=3, Xc=1. En effet, alors que le la voie 2 est sursaturée et que le temps d'attente des piétons de P2 est critique, le fait d'étendre le temps du vert et du rouge du feu F1 va augmentation la file d'attente dans la voie 2 et le retard s'accumulera, idem pour les piétons.

Sachant que le modèle de variabilité est composé de 10 niveaux, L'OPER8 a été inséré dans chacun de ces niveaux. Ce qui a donné 9 modèles à analyser par RMAS dans les conditions du contexte susmentionné.

Les résultats de cette deuxième évaluation du RMAS sont présentés dans le Tableau 12. Ainsi, le temps de diagnostic (Tdiag) défini précédemment, le temps de planification des solutions optimales par l'algorithme 6 (Toptimal), le nombre des buts retournés par l'algorithme 5 qui représentent les sources de violation des exigences partant des OPERs jusqu'au but racine (#Diag), sont calculés. Le Tableau 5 résume également, le nombre des buts (#Buts), le nombre des OPERs (#OPERs) qui composent la spécification, ainsi que le nombre des littéraux (#Littéraux) et clauses (#Clauses) du programme de contraintes traité par le solveur Swi-Prolog.

Tableau 12 : analyse des exigences à différentes granularités.

	Tdiag (sec)	**Toptimal (sec)**	**#Diag**	**#Buts**	**#OPERs**	**#Clauses**	**#Littéraux**
Niveau 9	0,622	0,015	20	23	30	465	284
Niveau 8	0,573	0,015	19	23	30	460	267
Niveau 7	0,544	0,015	18	23	30	459	285
Niveau 6	0,514	0,015	17	23	30	440	270
Niveau 5	0,505	0,015	16	23	30	436	272
Niveau 4	0,400	0,015	15	23	30	430	268
Niveau 3	0,181	0,015	14	23	30	428	274
Niveau 2	0,135	0,015	13	23	30	420	276
Niveau 1	0,100	0,015	12	23	30	410	266

Comme le montre, le Tableau 12, le nombre des buts retournés par l'algorithme 5 est proportionnel au niveau de granularité. Tandis que le temps d'exécution est inversement proportionnel au niveau de granularité. En effet, plus le niveau de granularité est grand, plus le temps de parcourir le graphe des buts partir de chaque OPER augmente, ce qui influence le temps d'exécution. Pour remédier à ce problème, nous avons proposé une amélioration en stockant le détail des différents niveaux de granularité dans une collection, qui associe à chaque OPER une liste de tous les buts qui leur sont liés. Cette liste sera calculée une seule fois durant tous le cycle de vie d'une spécification — lors de son premier chargement — et utilisé en cas de besoin sans le besoin de parcourir le graphe, en temps réel. Ce qui explique la différence minimale entre le temps de diagnostic des différents niveaux de granularité.

Expérimentation 3

But : Évaluer le comportement du RMAS et sa capacité à gérer des modèles d'exigences de grandes tailles. À cet objectif, les 15 modèles d'exigences présentés dans le Tableau 1 ont été analysés par RMAS. La taille de ces modèles varie de 40 buts jusqu'aux 259 buts. Les métriques de mesure de passage à l'échelle sont :

- Le temps de génération du programme des contraintes (Tencode).
- Le temps de chargement et d'apprentissage (TAp). Cette mesure est définie par l'équation suivante :
    - $\mathrm{TAp} = \mathrm{Tload} + \mathrm{TGranularité} + \mathrm{Tencode} + \mathrm{Treq} + \mathrm{Texclude}$
    - Où Tload représente le temps du chargement de la spécification, TGranularité représente le temps de la génération de tous les buts à différents niveaux de granularité pour chaque OPER, Treq représente le temps génération d'une liste de tous les groupes des OPERS qui sont liés par une relation de type « require » ou « AND ». Et Texclude représente le temps génération d'une liste de tous les groupes des OPERS qui sont liés par une relation de type « conflict» ou « Mutex». Ces deux listes sont utilisées par l'algorithme 4 dans le processus de diagnostic (voir section 4.2).
- Le temps d'analyse et de diagnostic (Tdiag) avec l'algorithme 4.
- Le temps d'analyse et de diagnostic avec apprentissage (TAdiag) c.-à-d. Ce dernier représente le temps d'analyse par l'algorithme 2 + le temps de diagnostic optimisé par l'algorithme 4.
- Le temps de planification des solutions optimales (Toptimal).

Nous avons injecté des erreurs de différents types de non-conformité, RMAS arrive à identifier tous ces types. Les résultats du calcul de ces mesures sont résumés dans les Tableaux 13 et 14. Les données de chargement de la spécification, le codage du programme de contraintes dérivé de la spécification et d'apprentissage ainsi que le nombre des clauses et des littéraux dans chaque programme sont présentées dans le Tableau 13. Alors que les données d'analyse et de diagnostic des exigences à l'aide des données simulées sont présentées dans le Tableau 14.

Tableau 13 : Statistiques d'apprentissage et de préparation lors du chargement d'une spécification d'exigences

	# Total des buts	#Clauses	#Littéraux	TAp (sec)	Tencode (sec)
M1	40	428	248	22,002	0,022
M2	55	424	246	22,474	0,043
M3	73	511	308	25,601	0,06
M4	87	579	356	28,048	0,069
M5	102	648	407	30,056	0,071
M6	115	709	452	33,898	0,079

M7	129	782	500	35,072	0,078
M8	148	877	565	36,061	0,086
M9	166	968	627	42,491	0,094
M10	184	1059	689	47,462	0,101
M11	202	1152	751	50,64	0,111
M12	217	1221	802	52,458	0,127
M13	231	1290	850	57,932	0,141
M14	244	1355	895	60,011	0,151
M15	259	1428	946	62,177	0,159

Comme le montre, le Tableau 14, les temps de chargement et de génération du programme de contraintes sont proportionnels à la taille du modèle d'exigences. Idem pour le nombre des littéraux et des clauses.

De plus, le temps d'analyse et de diagnostic des exigences prouve que notre approche peut être utilisée par des industries de petite et moyenne taille. En effet, pour un modèle d'exigences de 259 buts, notre approche passe seulement 2,59 secondes d'analyse. Cette durée a été largement optimisée par l'algorithme 6 qui se base sur un processus d'apprentissage des conditions de contexte déjà analysées. Par exemple, Tdiag (voir Tableau 7) passe de 2,54 secondes à 1,25 seconde pour le modèle M15 et de 1,9 seconde à 0,663 seconde pour le modèle M10.

Tableau 14 : Mesures de passage à l'échelle du RMAS

	Tdiag (sec)	**TAdiag (sec)**	**Toptimal (sec)**	**#Diag**
M1	0,665	0,3	0,016	2
M2	0,641	0,321	0,027	3
M3	1,036	0,338	0,034	4
M4	1,297	0,35	0,046	4
M5	1,378	0,496	0,05	4
M6	1,508	0,536	0,058	4
M7	1,601	0,594	0,63	4
M8	1,685	0,601	0,07	8
M9	1,846	0,62	0,076	9
M10	1,952	0,663	0,083	10
M11	1,907	0,683	0,09	15
M12	2,181	0,78	0,093	15
M13	2,271	0,92	0,1	15
M14	2,37	1,039	0,12	15
M15	2,54	1,25	0,0172	15

D'autre part, le temps de planification des solutions optimales est très optimisé grâce à notre stratégie de recherche, qui contrairement à la plupart des approches similaires ne se base pas sur le solveur de contraintes mais sur un algorithme de recherche basé sur une fonction d'utilité à partir de la base de toutes les solutions obtenues sans contraintes sur les buts (voir algorithme 2).

6.6 Etude comparative

Les trois séries d'expérimentation visaient à mesurer la performance de RMAS et son passage à l'échelle. Les premiers résultats sont prometteurs et montrent la capacité de RMAS à analyser de grands modèles d'exigences avec un temps d'exécution très encourageant.

Par la suite, nous avons tenté de comparer RMAS avec des approches similaires qui sont les plus connues de l'état de l'art du monitoring des exigences. La Figure 40 montre la performance des quatre approches, notamment RMAS, Wang et al. [Ref45], Ali et al. [Ref50], et Dalpiaz et al. [Ref46].

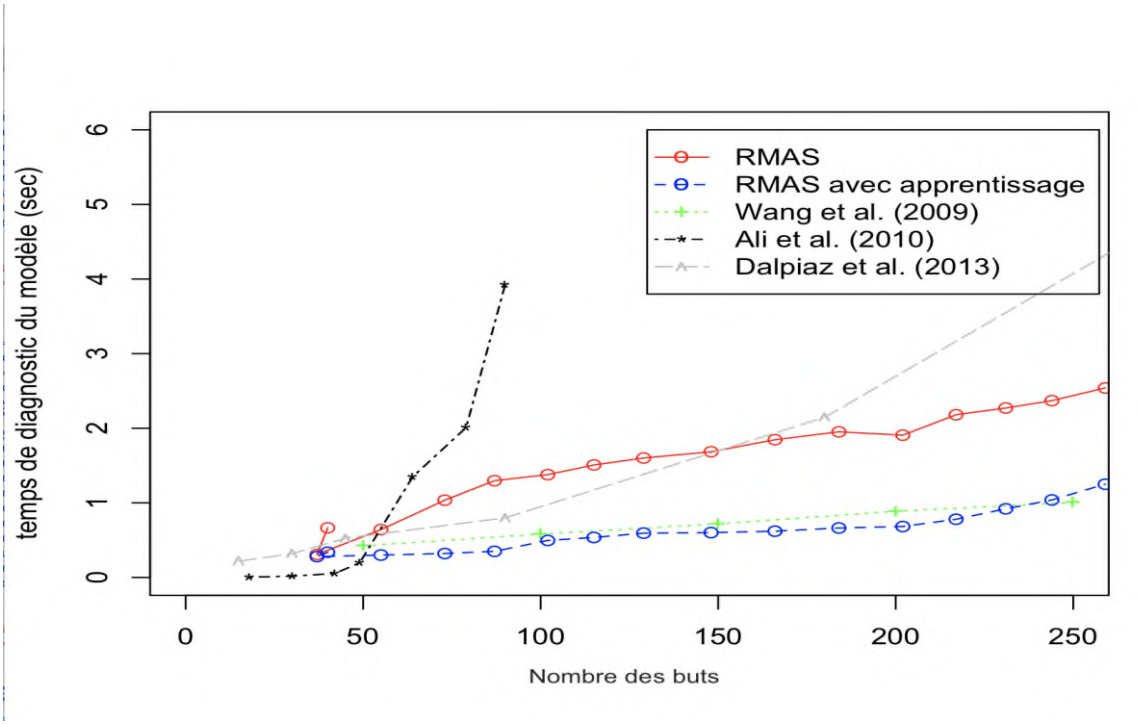


Figure 40 : comparaison du temps de diagnostic de RMAS et les approches de Wang et al. [Ref45], Ali et al. [Ref50], et Dalpiaz et al. [Ref46] face à des modèles d'exigences de tailles différentes.

Nous pouvons constater que RMAS a de meilleures performances que les trois approches dans la plupart des cas. Notons par ailleurs que la performance est très liée aux nombres et à la nature des liens de contribution entre les buts ainsi que le nombre de contexte à analyser, ce qui explique le comportement du prototype de Ali et al. [Ref50] ainsi que celui de Dalpiaz et al. [Ref46]. Ces deux derniers sont basés sur des modèles avec un grand nombre d'alternatives. Par conséquent, il est équitable de mener d'autres expérimentations sur les mêmes modèles, avec une même structure, avant toute généralisation des résultats observés.

6.7 Conclusion du chapitre

Ce chapitre a présenté une deuxième étude de cas qui a montré que notre approche donne de bonne performance et qu'elle permet de s'adapter à des spécifications d'exigences de tailles industrielles (entre 100 buts et 259 buts).

CONCLUSION GENERALE ET PERSPECTIVES

L'une des principales motivations de notre travail est de réduire la complexité de la gestion des SASs face à des conditions inattendues à l'exécution. Ces conditions ont un impact sur la satisfaction des exigences et sur l'efficacité des stratégies d'adaptation en temps réel, ce qui réduit le degré de confiance en ces systèmes et donc leur adoption en industrie.

Cette thèse propose donc une approche et un cadre logiciel de monitoring des exigences des SASs, appelé RMAS. Le but étant de permettre aux SASs de prendre conscience de leurs exigences et de s'auto-configurer pour améliorer leur conformité aux exigences face à l'évolution de celles-ci et à l'incertitude de l'environnement d'exécution. RMAS se compose de 4 éléments : (1) un composant de surveillance des modifications du système cible et son environnement d'exécution, (2) un composant d'analyse des exigences à chaque changement dans l'environnement ou dans la configuration du système cible, (3) un composant de diagnostic des non-conformités à différents niveaux de granularité, et (4) un composant de planification des configurations optimales.

Notre approche a été validée par deux études de cas. Une série d'expérimentations ont été menées afin de tester la généricité de l'approche et la performance du prototype en termes de temps d'exécution et de passage à l'échelle. Le résultat a démontré que notre approche est assez expressive et qu'elle est applicable à des spécifications industrielles de tailles moyennes avec des temps d'exécution raisonnables.

Cette thèse a permis d'apporter plusieurs contributions au monitoring des exigences en ligne, en général, et aux SASs, en particulier :

- Partant du constat que la recherche dans le domaine de monitoring des exigences est assez fragmentée, une revue systématique de la littérature a été effectuée. D'abord, une taxonomie a été définie d'une manière itérative afin d'offrir une meilleure compréhension des pratiques dans ce domaine. Ensuite, une analyse croisée entre les différentes dimensions de la taxonomie a permis de déceler les tendances et les pratiques dans ce domaine et de ressortir les points faibles et quelques points d'amélioration.

Cette revue a montré que plusieurs approches ont été dédiées à un seul type d'architecture ou technologie (par exemple les programmes java) et que seules quelques approches ont fait abstraction à la technologie, et ont étudié les interrelations entre les exigences pour assurer la fiabilité du monitoring. De plus, la plupart des approches identifiées n'ont pas considéré le diagnostic des non-conformités détectées. Enfin parmi les approches qui ont indiqué leur support à l'adaptation, peu d'entre elles ont décrit comment l'adaptation est supportée.

- Deuxièmement, pour répondre aux limitations décelées par la revue de la littérature, une approche générique de formalisation de la spécification d'exigences a été proposée. La méthodologie repose sur un langage orienté buts, appelé REFAS pour spécifier les exigences sous forme d'une LdPD. Le but

est de favoriser le raisonnement sur la variabilité du système cible et de ses stratégies d'adaptation ainsi que la variabilité de l'environnement d'exécution. La spécification est ensuite transformée en un problème de satisfaction de contraintes à l'aide des règles de transformation basées sur le méta-modèle REFAS. Cela a permis d'automatiser l'analyse des exigences à l'aide des solveurs des contraintes standard.

Notre approche est générique dans le sens où le monitoring des exigences supporte tout type de systèmes auto-adaptatifs et que la spécification d'exigence est indépendante à la technologie d'implémentation du système cible. En effet, une extension de l'outil VariaMos a été proposée pour permettre aux développeurs/analystes de faire la correspondance entre les éléments de la spécification et l'implémentation du système cible lors de la spécification.

- Enfin, des algorithmes d'analyse, de diagnostic et de planification des remédiations ont été proposés. Et un prototype a été développé pour concrétiser ces algorithmes et permettre l'évaluation empirique de notre approche.

Limitations de l'approche

Comme tout travail scientifique, il existe quelques limites concernant notre approche :

- La généricité de notre approche dépend en grande partie du travail du développeur et/ ou analyste qui ont la responsabilité de : (i) fournir une spécification d'exigences à l'aide du langage REFAS, (ii) fournir les interfaces de communication avec le système cible pour le monitoring du système et (iii) faire la correspondance entre les éléments de la spécification et l'implémentation du système.
- La complexité d'analyse d'exigences est fortement liée à une bonne spécification d'exigences. En effet, la spécification d'exigences orientée buts repose sur un processus de décomposition des buts en sous-buts. Ce processus est assez subjectif et présente le risque d'une mauvaise décomposition qui manque un nombre suffisant de liens de contribution, ce qui à une influence sur le temps de réponse du monitoring comme cela a été démontré dans le chapitre 6.
- VariaMos est l'outil qui permet la spécification d'exigences en langage REFAS. Cependant, au moment de la publication de cette thèse, cet outil bien qu'il facilite la spécification d'exigences et offre une vérification de la cohérence des exigences, il souffre des problèmes techniques qui empêchent la création des modèles de grandes tailles car il est encore en phase de développement.
- Pour évaluer la performance de prototype, des modèles de buts de grandes tailles (entre 100 et 250) ont été créés par plusieurs clones des modèles de petites tailles au lieu de spécifications de systèmes réels. En effet, Il aurait été difficile d'obtenir des modèles de tailles grandes par modélisation classique. Et vu que le passage à l'échelle de notre prototype a été démontré sur la base de ces modèles. Il faudrait alors expérimenter le prototype sur d'autres cas d'études avant la généralisation des résultats.

Perspectives

Plusieurs pistes de recherche méritent d'être explorées :

Le premier point d'amélioration consiste à explorer le potentiel des algorithmesévolutionnaires et Machine Learning dans le processus de diagnostic et de planification. Ces algorithmes peuvent améliorer le temps de traitement par apprentissage et l'optimisation de la recherche des solutions optimales. D'une part, l'apprentissage servira à ressortir des patterns à partir de l'analyse des modèles d'exigences et des données collectées. Ce qui contribuera à simplifier ces modèles et réduire le temps d'analyse et l'efficacité des algorithmes implémentés. D'autre part, l'exploration des algorithmes évolutionnaires pour amélioration de la résolution des programmes de contraintes.

Un autre point d'amélioration de notre approche est la prise en compte de la dimension temporelle dans la spécification des exigences. Par exemple, par extension du langage REFAS ou en envisageant l'intégration de la logique temporelle telle que LTL dans notre méthodologie de spécification d'exigences.

Finalement, pour une industrialisation du prototype RMAS, nous envisageons d'optimiser le code du prototype et de faire des recherches approfondies sur le passage à l'échelle et sur l'efficacité des remédiations en temps réel.

RÉFÉRENCES BIBLIOGRAPHIQUES

Andersen, E. (2003). Be prepared for the unforeseen. Journal of Contingencies and Crisis Management, 11(3), 129-131.

Andersson, J., De Lemos, R., Malek, S., & Weyns, D. (2009). Modeling dimensions of self-adaptive software systems. In Software engineering for self-adaptive systems (pp. 27-47). Springer, Berlin, Heidelberg.

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.

Benavides, D., Segura, S., & Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later : A literature review. Information Systems, 35(6), 615-636.

Benkomo : Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., & Letier, E. (2010, May). Requirements reflection : requirements as runtime entities. In Software Engineering, 2010 ACM/IEEE 32nd International Conference on (Vol. 2, pp. 199-202). IEEE.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., & Schnoebelen, P. (2013). Systems and software verification : model-checking techniques and tools. Springer Science & Business Media.

Biere, A., Cimatti, A., Clarke, E. M., Strichman, O., & Zhu, Y. (2003). Bounded model checking. Advances in computers, 58(11), 117-148.

Borgida, A., Dalpiaz, F., Horkoff, J., & Mylopoulos, J. (2013, May). Requirements models for design- and runtime : A position paper. In Proceedings of the 5th International Workshop on Modeling in Software Engineering (pp. 62-68). IEEE Press.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos : An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems, 8(3), 203-236.

Brun, Y., Serugendo, G. D. M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., ... & Shaw, M. (2009). Engineering self-adaptive systems through feedback loops. In Software engineering for self-adaptive systems (pp. 48-70). Springer Berlin Heidelberg.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. Computers, IEEE Transactions on, 100(8), 677-691.

C. Forgy. Rete : A fast algorithm for the many pattern/- many object pattern match problem. Artificial Intelli- gence, 19:17–37, 1982.

Cheng, B., Lemos, R.d., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun,Y., Cukic, B., Serugendo, G.M., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Muller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J. (2009). Software engineering for Self-Adaptive systems : A research roadmap. In : Software Engineering for Self-Adaptive Systems, LNCS Hot Topics (p. 1–26).

Clarke, E. M., Grumberg, O., & Peled, D. (1999). Model checking. MIT press.

Cohen, D., Feather, M. S., Narayanaswamy, K., & Fickas, S. S. (1997, May). Automatic monitoring of software requirements. In Proceedings of the 19th international conference on Software engineering (pp. 602-603). ACM.

Cruzes, D. S., & Dyba, T. (2011, September). Recommended steps for thematic synthesis in software engineering. In 2011 International Symposium on Empirical Software Engineering and Measurement (pp. 275-284). IEEE.

Dardenne, A., Van Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. Science of computer programming, 20(1-2), 3-50.

De Rocquigny, E., Devictor, N., & Tarantola, S. (Eds.). (2008). Uncertainty in industrial practice : a guide to quantitative uncertainty management. John Wiley & Sons.

Devoper, C. & Quintin, J. J. (1992). Learner Control vs. Computer Control in a professional Training Context. In M. Giardina (Eds.), Interactive Multimedia learning Environments (Vol. 93, pp. 234-247). Berlin : Springer Verlag.

- Dounas, L., Salinesi, C., & El Beqqali, O. (2019). Requirements monitoring and diagnosis for improving adaptive e-learning systems design. *Journal of Information Technology Education : Research, 18*, 161-184.

Egyed, A. (2006, May). Instant consistency checking for the UML. In Proceedings of the 28th international conference on Software engineering (pp. 381-390). ACM.

Elaasar, M., & Briand, L. (2004). An overview of UML consistency management. Carleton University, Canada, Technical Report SCE-04-18.

Esfahani, N., & Malek, S. (2013). Uncertainty in self-adaptive software systems. In Software Engineering for Self-Adaptive Systems II (pp. 214-238). Springer, Berlin, Heidelberg.

E. Yu, (1995). Modelling Strategic Relationships for Process Reengineering, Ph.D. thesis, also Tech. Report DKBS-TR-94- 6, Dept. of Computer Science, University of Toronto.

Faber, A., Matthes, F., & Michel, F. (2016). Digital Mobility Platforms and Ecosystems

GAO. 1992. Mission critical systems : Defense attempting to address major software chal- lenges. ech. Rep. GAO/IMTEC-93-13, U.S. General Accounting Office, Washington, D.C. Dec.

Giorgini, P., Mylopoulos, J., Nicchiarelli, E., & Sebastiani, R. (2002, October). Reasoning with goal models. In International Conference on Conceptual Modeling (pp. 167-181). Springer, Berlin, Heidelberg.

Gogolla, M., Bu¨ttner, F., Richters, M. : USE : A UML-Based Specification Environ- ment for Validating UML and OCL. Science of Computer Programming 69, 27–34 (2007).

Gu, W., Vetter, J., & Schwan, K. (1994). An annotated bibliography of interactive program steering. ACM Sigplan Notices, 29(9), 140-148.

Heitmeyer, C. L., Jeffords, R. D., & Labaw, B. G. (1996). Automated consistency checking of requirements specifications. ACM Transactions on Software Engineering and Methodology (TOSEM), 5(3), 231-261.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS Q 28 (1) : 75–105.

Honey, P., & Mumford, A. (1992). The manual of learning styles.

Juszczyk, L., & Dustdar, S. (2010, July). Script-based generation of dynamic testbeds for soa. In 2010 IEEE International Conference on Web Services (pp. 195-202). IEEE.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. Information and software technology, 51(1), 7-15.

Laddaga, R. (1999). Creating robust software through self-adaptation. IEEE Intelligent Systems and their Applications, 14, 26-29.

Laddaga, R., & Robertson, P. (2004, May). Self adaptive software : A position paper. In SELF-STAR : International Workshop on Self-* Properties in Complex Information Systems(Vol. 31, p. 19).

Laddaga, R. (2006, September). Self adaptive software problems and projects. In Software Evolvability, 2006. SE'06. Second International IEEE Workshop on (pp. 3-10). IEEE.

Larman, C. (2004). Agile and iterative development : a manager's guide. Addison-Wesley Professional.

Leucker, M., & Schallhart, C. (2009). A brief account of runtime verification. The Journal of Logic and Algebraic Programming, 78(5), 293-303.

Maes, P. (1987, December). Concepts and experiments in computational reflection. In ACM Sigplan Notices (Vol. 22, No. 12, pp. 147-155). ACM.

Manual, H. C. (2000). Transportation research board. National Research Council, Washington, DC, 113, 10.

March, S.T. & Smith, G.F. (1995), « Design and Natural Science Research on Information Technology », Decision Support Systems, vol. 15, p. 251-266.

Mazo, R., Salinesi, C., Diaz, D., & Lora-Michiels, A. (2011, June). Transforming attribute and clone-enabled feature models into constraint programs over finite domains. In 6th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE).

Mazo, R. (2011). A generic approach for automated verification of product line models (Doctoral dissertation, Université Panthéon-Sorbonne-Paris I).

Mazo, R., Salinesi, C., Diaz, D., Djebbi, O., & Lora-Michiels, A. (2012). Constraints: The heart of domain and application engineering in the product lines engineering strategy. International Journal of Information System Modeling and Design (IJISMD), 3(2), 33-68.

McManus, H., & Hastings, D. (2005, July). A framework for understanding uncertainty and its mitigation and exploitation in complex systems. In INCOSE International Symposium (Vol. 15, No. 1, pp. 484-503).

McMillan, K. L. (1992). Symbolic model checking : an approach to the state space explosion problem (Doctoral dissertation, PhD thesis, Carnegie Mellon University.

Mens, T. (2008). Introduction and Roadmap : History and Challenges of Software Evolution, chapter 1. Software Evolution.

Metzger, A., Pohl, K., Heymans, P., Schobbens, P. Y., & Saval, G. (2007, October). Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In Requirements Engineering Conference, 2007. RE'07. 15th IEEE International (pp. 243-253). IEEE.

Muir, B. M. (1994). Trust in automation : Part I. Theoretical issues in the study of trust and human intervention in automated systems. Ergonomics, 37(11), 1905-1922.

Muñoz-Fernández, J. C., Tamura, G., Mazo, R., & Salinesi, C. (2015). Towards a requirements specification multi-view framework for self-adaptive systems. CLEI electronic journal, 18(2), 6-6.

Nuseibeh, B., Easterbrook, S., and Russo, A., 2000. "Leveraging Inconsistency in Software Development". IEEE Computer, April.

Papanikolaou, K. A., Grigoriadou, M., Kornilakis, H., & Magoulas, G. D. (2003). Personalizing the Interaction in a Web-based Educational Hypermedia System : the case of INSPIRE. User modeling and user-adapted interaction, 13(3), 213-267.

Paschke, A., & Kozlenkov, A. (2009, November). Rule-based event processing and reaction rules. In International Workshop on Rules and Rule Markup Languages for the Semantic Web (pp. 53-66). Springer, Berlin, Heidelberg.

Paton, Norman W.; Díaz, Oscar (1999). "Active database systems" (PDF). ACM Computing Surveys. 31: 63

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. (2007), « A Design Science Research Methodology for Information Systems Research », Journal of Management Information Systems, Vol. 24, n°3, p. 45-77.

Pohl, K., Böckle, G., & van Der Linden, F. J. (2005). Software product line engineering: foundations, principles and techniques. Springer Science & Business Media.

Price, W. J. (1989). A benchmark tutorial. IEEE micro, 9(5), 28-43.

Ramirez, A. J., Jensen, A. C., & Cheng, B. H. (2012, June). A taxonomy of uncertainty for dynamically adaptive systems. In Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (pp. 99-108). IEEE Press.

Robinson, W. N. (2006). A requirements monitoring framework for enterprise systems. Requirements engineering, 11(1), 17-41.

Robinson, W. N. (2010). A roadmap for comprehensive requirements modeling. Computer, 43(5), 64-72.

Sardina, S., de Silva, L., & Padgham, L. (2006, May). Hierarchical planning in BDI agent programming languages: A formal approach. In Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (pp. 1001-1008). ACM.

Sachs, K. (2011). Performance modeling and benchmarking of event-based systems. Sierke.

Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. ACM transactions on autonomous and adaptive systems (TAAS), 4(2), 14.

Salinesi, C., Mazo, R., Diaz, D., & Djebbi, O. (2010, September). Solving Integer Constraint in Reuse Based Requirements Engineering. In 18th IEEE International Conference on Requirements Engineering (RE'10) (p. 50).

Salinesi, C., Mazo, R., Djebbi, O., Diaz, D., & Lora-Michiels, A. (2011, May). Constraints: The core of product line engineering. In Research Challenges in Information Science (RCIS), 2011 Fifth International Conference on (pp. 1-10). IEEE.

Sawyer, P., Bencomo, N., Whittle, J., Letier, E., & Finkelstein, A. (2010, September). Requirements-aware systems: A research agenda for re for self-adaptive systems. In Requirements Engineering Conference (RE), 2010 18th IEEE International (pp. 95-103). IEEE.

Sawyer, P., Mazo, R., Diaz, D., Salinesi, C., & Hughes, D. (2012). Using constraint programming to manage configurations in self-adaptive systems. Computer, 45(10), 56-63.

S. Fickas and M.S. Feather. Requirements monitoring in dynamic environments. In RE Conf., 1995.

Shanahan, M. (1999). The event calculus explained. In Artificial intelligence today (pp. 409-430). Springer, Berlin, Heidelberg.

Shell, T. (2001). System function implementation and behavioral modeling: a systems theoretic approach. Systems Engineering, 4(1), 58-75.

Smith, B. C. (1982). Reflection and semantics in a procedural language (Ph. D. thesis). Technical Report.

Taillandier, P., Therond, O., & Gaudou, B. (2012). A new BDI agent architecture based on the belief theory. Application to the modelling of cropping plan decision-making. In International environmental modelling and software society (iEMSs).

Trinidad, P.; Benavides, D.; Durán, A.; Ruiz-Cortés, A. & Toro, M. (2008). Automated error analysis for the agilization of feature modeling. Journal of Systems & Software, 81(6) pp. 883-896, Elsevier.

Sebastiani, R., Giorgini, P., Mylopoulos, J., 2004. Simple and minimum-cost satisfia- bility for goal models. In: Proceedings of the 16th International Conference on Advanced Information Systems Engineering, pp. 20–35.

Souza, V. E. S., & Mylopoulos, J. (2009, November). Monitoring and diagnosing malicious attacks with autonomic software. In International Conference on Conceptual Modeling (pp. 84-98). Springer, Berlin, Heidelberg.

Specht, M. (1998). Empirical Evaluation of Adaptive Annotation in Hypermedia.

Van Lamsweerde, A. (1991). The KAOS project: Knowledge acquisition in automated specification of software. In Proc. of the AAAI Spring Symposium Series, Design of Composite Systems, 1991 (pp. 59-62).

V. Battaglia, Y. Lagarde (2014). Impact du cycle de feux sur la capacité, l'acceptabilité de l'attente aux feux et la sécurité (rapport d'étude). JOURNÉES TRANSPORTS & DÉPLACEMENTS DU RST-2015.

Vierhauser, M., Grünbacher, P., Egyed, A., Rabiser, R., & Heider, W. (2010, September). Flexible and scalable consistency checking on product line variability models. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 63-72). ACM.

Wallace, M. (1996). Practical applications of constraint programming. Constraints, 1(1), 139-168.

Wang, Y., Mcilraith, S. A., Yu, Y., & Mylopoulos, J. (2009). Monitoring and diagnosing software requirements. Automated Software Engineering, 16(1), 3.

Waller, J., & Hasselbring, W. (2013). A benchmark engineering methodology to measure the overhead of application-level monitoring. CEUR Workshop Proceedings.

Wehbi, B., de Oca, E. M., & Bourdelles, M. (2012, April). Événements-based security monitoring using MMT tool. In Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on (pp. 860-863).

Welsh, K., & Sawyer, P. (2009, June). Requirements tracing to support change in dynamically adaptive systems. In International Working Conference on Requirements Engineering: Foundation for Software Quality (pp. 59-73). Springer, Berlin, Heidelberg.

Welsh, K., Sawyer, P., & Bencomo, N. (2011, November). Towards requirements aware systems: Run-time resolution of design-time assumptions. In Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on (pp. 560-563). IEEE.

White, J., Schmidt, D. C., Benavides, D., Trinidad, P., & Ruiz–Cortés, A. (2008, September). Automated diagnosis of product-line configuration errors in feature models. In Software Product Line Conference, 2008. SPLC'08. 12th International (pp. 225-234). IEEE.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H., & Bruel, J. M. (2009, August). Relax: Incorporating uncertainty into the specification of self-adaptive systems. In Requirements Engineering Conference, 2009. RE'09. 17th IEEE International (pp. 79-88). IEEE.

Xiao, J. (2009). Gestion des incertitudes dans le processus de développement de systèmes complexes (Doctoral dissertation).

Zhang, W., Zhao, H., & Mei, H. (2004, November). A propositional logic-based method for verification of feature models. In International Conference on Formal Engineering Methods (pp. 115-130). Springer, Berlin, Heidelberg.

Zowghi, D., & Gervasi, V. (2003). On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software technology, 45(14), 993-1009.

LISTE DES ETUDES PRIMAIRES DU RS SUR LE MONITORING DES EXIGENCES A L'EXECUTION

[Ref1] Welsh, K., Bencomo, N., Sawyer, P., & Whittle, J. (s. d.). Self-explanation in Adaptive Systems based on Runtime Goal-based Models, 24.

[Ref2] Loukil, S., Kallel, S., & Jmaiel, M. (2017). An approach based on runtime models for developing dynamically adaptive systems. Future Generation Computer Systems, 68, 365 375.

[Ref3] Salifu, M., Yu, Y., Bandara, A. K., & Nuseibeh, B. (2012). Analysing monitoring and switching problems for adaptive systems. Journal of Systems and Software, 85(12), 2829 2839.

[Ref4] Khaxar, Mohse, Jalili, S., Khakpour, N., & Jokhio, M. S. (2009). Monitoring safety properties of composite web services at runtime using CSP. In 2009 13th Enterprise Distributed Object Computing Conference Workshops (p. 107 113).

[Ref5] Robertson, P., & Laddaga, R. (2005). Model Based Diagnosis and Contexts in Self Adaptive Software.

[Ref6] Filieri, A., Ghezzi, C., & Tamburrelli, G. (2012). A formal approach to adaptive software: continuous assurance of non-functional requirements. Formal Aspects of Computing, 24(2), 163 186.

[Ref7] Ramirez, A. J., Cheng, B. H. C., Bencomo, N., & Sawyer, P. (2012). Relaxing Claims: Coping with Uncertainty While Evaluating Assumptions at Run Time.

[Ref8] Cámara, J., Moreno, G. A., & Garlan, D. (2014). Stochastic game analysis and latency awareness for proactive self-adaptation. In Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS 2014 (p. 155 164).

[Ref9] Fitzgerald, C., Klöpper, B., & Honiden, S. (2011). Utility-based self-adaption with environment specific quality models. In Adaptive and Intelligent Systems (pp. 107-118). Springer, Berlin, Heidelberg.

[Ref10] Hielscher, J., Kazhamiakin, R., Metzger, A., & Pistore, M. (2008). A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing. In P. Mähönen, K. Pohl, & T. Priol (Éd.), Towards a Service-Based Internet (Vol. 5377, p. 122 133).

[Ref11] Chan, K., Poernomo, I., Schmidt, H., & Jayaputera, J. (2005). A model-oriented framework for runtime monitoring of nonfunctional properties. In Quality of Software Architectures and Software Quality (pp. 38-52). Springer, Berlin, Heidelberg.

[Ref12] Frischbier, S., Pietzuch, P., & Buchmann, A. (2014). Managing Expectations: Runtime Negotiation of Information Quality Requirements in Event-Based Systems. In X. Franch, A. K. Ghose, G. A. Lewis, & S. Bhiri (Éd.), Service-Oriented Computing (Vol. 8831, p. 199 213).

[Ref13] Psaier, H., Juszczyk, L., Skopik, F., Schall, D., & Dustdar, S. (s. d.). Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems, 10.

[Ref14] Hamann, L., Hofrichter, O., & Gogolla, M. (2012). OCL-Based Runtime Monitoring of Applications with Protocol State Machines.

[Ref15] Bertolino, A., Calabrò, A., Lonetti, F., Di Marco, A., & Sabetta, A. (2011). Towards a Model-Driven Infrastructure for Runtime Monitoring. In E. A. Troubitsyna (Éd.), Software Engineering for Resilient Systems (Vol. 6968, p. 130 144).

[Ref16] Aktug, I., Dam, M., & Gurov, D. (2009). Provably correct runtime monitoring. The Journal of Logic and Algebraic Programming, 78(5), 304 339.

[Ref17] Skene, J., & Emmerich, W. (2005). Engineering Runtime Requirements-Monitoring Systems Using MDA Technologies. In R. De Nicola & D. Sangiorgi (Éd.), Trustworthy Global Computing (Vol. 3705, p. 319 333).

[Ref18] Spanoudakis, G., Kloukinas, C., & Mahbub, K. (s. d.). The Runtime Monitoring Framework of SERENITY, 25.

[Ref19] F. Jahanian and A. Goyal. A formalism for monitoring real-time constraints at run-time. In 20th Int. Symp. On Fault-Tolerant Computing Systems (FTCS-20), pages 148–55, 1990.

[Ref20] Bodden, E., & Vangheluwe, H. (2008). Transforming Timeline Specifications into Automata for Runtime Monitoring. In A. Schürr, M. Nagl, & A. Zündorf (Éd.), Applications of Graph Transformations with Industrial Relevance (Vol. 5088, p. 249 264).

[Ref21] Ghezzi, C., Mocci, A., & Sangiorgio, M. (2012). Runtime Monitoring of Functional Component Changes with Behavior Models. In J. Kienzle (Éd.), Models in Software Engineering (Vol. 7167, p. 152 166).

[Ref22] Chowdhury, O., Jia, L., Garg, D., & Datta, A. (s. d.). Temporal Mode-Checking for Runtime Monitoring of Privacy Policies, 19.

[Ref23] Kähkönen, K., Lampinen, J., Heljanko, K., & Niemelä, I. (2009). The LIME Interface Specification Language and Runtime Monitoring Tool. In S. Bensalem & D. A. Peled (Éd.), Runtime Verification (Vol. 5779, p. 93 100).

[Ref24] Havelund, K. (2015). Rule-based runtime verification revisited. International Journal on Software Tools for Technology Transfer, 17(2), 143 170.

[Ref25] Meredith, P. O., Jin, D., Griffith, D., Chen, F., & Roşu, G. (2012). An overview of the MOP runtime verification framework. International Journal on Software Tools for Technology Transfer, 14(3), 249 289.

[Ref26] Souza, V. E. S., & Mylopoulos, J. (2009). Monitoring and Diagnosing Malicious Attacks with Autonomic Software. In A. H. F. Laender, S. Castano, U. Dayal, F. Casati, & J. P. M. de Oliveira (Éd.), Conceptual Modeling - ER 2009 (Vol. 5829, p. 84 98).

[Ref27] Huang, T., Wu, G.-Q., & Wei, J. (2009). Runtime Monitoring CompositeWeb Sewssssss » »rvices Through Stateful Aspect Extension. Journal of Computer Science and Technology, 24(2), 294 308.

[Ref28] Simmonds, J., Ben-David, S., & Chechik, M. (2013). Monitoring and recovery for web service applications. Computing, 95(3), 223 267.

[Ref29] Gerasimou, S., Calinescu, R., & Tamburrelli, G. (2018). Synthesis of probabilistic models for quality-of-service software engineering. Automated Software Engineering, 25(4), 785 831.

[Ref30] Rabiser, R., Thanhofer-Pilisch, J., Vierhauser, M., Grünbacher, P., & Egyed, A. (2018). Developing and evolving a DSL-based approach for runtime monitoring of systems of systems. Automated Software Engineering, 25(4), 875 915

[Ref31] Serral, E., Sernani, P., & Dalpiaz, F. (2018). Personalized adaptation in pervasive systems via non-functional requirements. Journal of Ambient Intelligence and Humanized Computing, 9(6), 1729.

[Ref32] Chen, X., Liu, J., & Liu, Z. (2013). Requirements monitoring for Internetware : an interaction based approach. Science China Information Sciences, 56(8), 1 15.

[Ref33] Robinson, W. N. (2006). A requirements monitoring framework for enterprise systems. Requirements Engineering, 11(1), 17 41.

[Ref34] MacNamee, C., Heffernan, D., & Fogarty, P. (2014). Runtime verification monitoring for automotive embedded systems using the ISO 26262 Functional Safety Standard as a guide for the definition of the monitored properties.

[Ref35] Perez, B., & Correal, D. (2011). A Model Driven Approach to the Analysis of Quality Scenarios within Self-Adaptable SOA Systems. Electronic Notes in Theoretical Computer Science, 281, 113 126.

[Ref36] Horcas, J.-M., Pinto, M., Fuentes, L., Mallouli, W., & Montes de Oca, E. (2016). An approach for deploying and monitoring dynamic security policies. Computers & Security, 58, 20 38.

[Ref37] Knauss, A., Damian, D., Franch, X., Rook, A., Müller, H. A., & Thomo, A. (2016). ACon : A learning-based approach to deal with uncertainty in contextual requirements at runtime. Information and Software Technology, 70, 85 99.

[Ref38] Lee, E., Kim, Y.-G., Seo, Y.-D., Seol, K., & Baik, D.-K. (2018). RINGA : Design and verification of finite state machine for self-adaptive software at runtime. Information and Software Technology, 93, 200 222.

[Ref39] D'Angelo, B., Sankaranarayanan, S., Sanchez, C., Robinson, W., Finkbeiner, B., Sipma, H. B., … Manna, Z. (2005). LOLA : Runtime Monitoring of Synchronous Systems. In 12th International Symposium on Temporal Representation and Reasoning (TIME'05) (p. 166 174).

[Ref40] van Hoorn, A., Rohr, M., Hasselbring, W., Waller, J., Ehlers, J., Frey, S., & Kieselhorst, D. (s. d.). Continuous Monitoring of Software Services : Design and Application of the Kieker Framework, 27.

[Ref41] Comuzzi, M., & Spanoudakis, G. (2010). Dynamic set-up of monitoring infrastructures for service based systems. In Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10 (p. 2414). Sierre, Switzerland.

[Ref42] Qureshi, N. A., Perini, A., Ernst, N. A., & Mylopoulos, J. (2010). Towards a continuous requirements engineering framework for self-adaptive systems. In 2010 First International Workshop on Requirements@Run.Time (p. 9 16).

[Ref43] Baresi, L., & Pasquale, L. (2010). Adaptive Goals for Self-Adaptive Service Compositions. In 2010 IEEE International Conference on Web Services (p. 353 360). Miami, FL, USA : IEEE.

[Ref44] Souza, V. E. S., & Mylopoulos, J. (2009, November). Monitoring and diagnosing malicious attacks with autonomic software. In International Conference on Conceptual Modeling (pp. 84-98). Springer, Berlin, Heidelberg.

[Ref45] Wang, Y., McIlraith, S. A., Yu, Y., & Mylopoulos, J. (2009). Monitoring and diagnosing software requirements. Automated Software Engineering, 16(1), 3 35.

[Ref46] Dalpiaz, F., Giorgini, P., & Mylopoulos, J. (2013). Adaptive socio-technical systems : a requirements-based approach. Requirements Engineering, 18(1), 1 24.

[Ref47] Peng, X., Chen, B., Yu, Y., & Zhao, W. (2012). Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop. Journal of Systems and Software, 85(12), 2707 2719.

[Ref48] Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., & Tamburrelli, G. (s. d.). Dynamic QoS Management and Optimisation in Service-Based Systems, 26.

[Ref49] Fickas, S., & Feather, M. S. (1995). Requirements monitoring in dynamic environments. In Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95) (p. 140 147).

[Ref50] Ali, R., Dalpiaz, F., & Giorgini, P. (2010). À goal-based framework for contextual requirements modeling and analysis. Requirements Engineering, 15(4), 439 458.

ANNEXE

I. Revue systématique sur le monitoring des exigences à l'exécution

Dans cette annexe, nous présentons la méthode de recherche et le processus systématique que nous avons suivi pour mener à bien la revue systématique sur le monitoring des exigences à l'exécution.

La procédure de cette étude a suivi les directives de la méthode systématique proposée par Kitchenham (2004). La principale motivation pour une RS au lieu d'une revue de littérature ad hoc est qu'il nous a permis une assurance contre des résultats de revue biaisés, toutefois les biais dans les études primaires identifiées dans la RS restent inévitables, et finalement pour un souci de rendre la portée de la revue traçable.

Ceci étant, nous avons élaboré un protocole de révision comprenant : les questions de recherche, une stratégie de recherche, des critères de sélection et de qualité, ainsi que des informations sur l'extraction et la synthèse des données.

1.1. Etapes de la revue

a. Questions de recherche

Les questions de recherche abordées par cette revue sont les suivantes :

– QR1 : Comment le monitoring des exigences à l'exécution peut être défini et caractérisé ?

– QR2 : quels sont les outils supportant l'analyse des exigences à l'exécution?

– QR3 : Quels sont les domaines d'application des approches identifiées de monitoring des exigences à l'exécution?

– QR4 : Quelles sont les méthodes de spécification qui ont été bien appliquées et ont été sujettes à une évaluation rigoureuse ?

– QR5 : Quels sont les points faibles et les points d'amélioration possibles dans la recherche sur le monitoring des exigences à l'exécution?

Pour répondre à la QR1, Nous avons développé une taxonomie (voir section 2.1) à partir de l'analyse des études primaires. La taxonomie est ensuite utilisée pour décrire les approches de monitoring des exigences à l'exécution. Les informations correspondantes sont extraites des études primaires (voir les Tableaux A.3, A.4, A.5). Pour répondre à la QR2, QR3 les informations correspondantes sont directement extraites des études primaires (voir Tableau A.3 et Tableau A.5). Pour répondre à QR3 les articles ont été lu en complet et ont évalué en fonction des critères de qualité d'évaluation (Voir Tableau A.6). Ces critères ont été extraits du programme d'évaluation critique (CASP). Pour répondre à QR4, nous avons effectué une analyse croisée des différentes dimensions de la taxonomie à partir des données extraites, afin de dégager des lacunes et des orientations pour des recherches ultérieures.

b. Processus de recherche

L'objectif du processus de recherche est d'identifier les études primaires en se basant sur des stratégies de recherche. La définition des stratégies de recherche comprend la définition des chaînes utilisées pour la recherche, les sources de recherche, et la définition de la procédure de sélection des études primaires :

Pour cette étude, nous avons effectué une recherche automatique entre le 19 et le 20 juillet 2018, en utilisant la requête de recherche ‘ requirements AND monitoring AND adapt’.

Nous avons choisi comme sources de recherche les trois bases de données électroniques suivantes :

- IEEE Xplore
- Science Direct
- Springer Link.

Ces bases de données ont été considérées car elles fournissent les publications les plus importantes en informatique.

Les critères d'inclusion suivants ont été utilisés pour sélectionner les études de recherche primaires : nous avons limité notre recherche aux articles journaux publiés entre 1994 et 2018 et rédigé en anglais. Nous avons choisi l’année 1994 car les principales publications dans le monitoring des exigences sont apparues après cette année, par exemple, celle de Fickas et Feather (1995). Nous avons exclu les articles ne traitant pas le monitoring en temps réel pour les systèmes auto-adaptatifs ou les systèmes dynamiques comme les systèmes basés sur l’architecture. Les articles présentés sous forme de revue de littérature ont été également exclus. Dans le cas des publications en double, la plus récente publication est utilisée.

Ceci étant, la sélection des études incluses a été réalisée en trois étapes :

- ✓ Dans un premier temps, une recherche automatique a été effectuée en cherchant la requête de recherche dans le titre et l’abstract des articles.
- ✓ Ensuite, nous avons lu l’abstract de chaque article pour exclure les études basées sur les critères d'exclusion et éliminer les études en double. À cette phase nous avons sélectionné 188 articles.
- ✓ Finalement, les articles restants ont été filtrés intégralement pour identifier les études primaires.

Un aperçu des résultats de la recherche est présenté dans le tableau A.1.

Tableau A. 1 : Processus de recherche

	Recherche automatique	**Après lecture d'abstract**	**% de sélection**
IEEE	396	46	11,6%
Springer	3693	123	3,27%
Elsevier	467	19	4,08 %
∑	4556	188	4,13%

La recherche automatique a abouti à 4556 publications. Après l'exclusion des articles ne respectant pas les critères de sélection, ce qui a donné lieu à 188 articles, nous avons remarqué que nous avons manqué des publications importantes que nous connaissions déjà, comme l'article de Fickas (1995) et celui de Sawyer et al. (2010) car ces publications sont des articles de conférences. Nous avons alors effectué une deuxième recherche manuelle pour inclure les publications les plus connues et citées dans le domaine de monitoring des exigences en temps réel. Nous avons alors ajouté 16 articles, ce qui a abouti à 204 articles. Finalement, après une première lecture intégrale du texte des articles, Le résultat final a donné 50 études primaires. Pour chaque étude les informations dans le Tableau A.2 ont été collectées.

Tableau A. 2 : Description des informations du formulaire d'extraction

Type d'information	**Description**
Identifiant de l'étude	Identifiant de la forme de Ref +chiffre (Ref1, Ref2…)
Année de publication.	-
Domaine d'application	-
Méthodologie de recherche	étude théorique, enquête, design de solution, étude empirique
Type de monitoring	hors-ligne, en ligne (synchrone, asynchrone)
Méthode de spécification des exigences	Notations et langages utilisés pour définir les exigences.
Implémentation du monitoring	cela consiste aux techniques d'analyse des propriétés spécifiées.
Méthode d'instrumentation.	Cela inclut les couches d'abstraction du système considérées et les technologies pour la réalisation du monitoring, par exemple AOP, JBoss…
Support pour l'incertitude	Est-ce que l'approche aborde l'incertitude ? (oui/non). Si oui, quel type d'incertitude ? (contexte, exigence, décision…)
Description de la validation de l'étude	- Type de validation (étude de cas, expérimentation, exemple… etc.) - Critères de validation (temps d'exécution, passage à l'échelle… etc.)

Toutes ces données ont été extraites et vérifiées encore une fois après l'extraction. Là où des divergences sont apparues, nous avons discuté les points de désaccord jusqu'à ce que nous parvenions à un accord.

c. Évaluation de qualité

Nous avons évalué chaque étude selon les six questions suivantes :

Q1. Les objectifs de la recherche ont été clairement annoncés ?

Q2. Le modèle de recherche était-il approprié pour répondre aux objectifs de la recherche ?

Q3. Le chercheur a-t-il présenté une évaluation ou une analyse de ses modèles via l'expérimentation ? L'analyse des données était-elle suffisamment rigoureuse ?

Q4. Les limites de l'étude sont-elles explicitement discutées ?

Q5. L'étude a-t-elle fourni suffisamment de détails pour permettre la réplication et la validation de la solution ? (Ex : fournir un rapport complet et mettre à disposition toutes les données et le code requis)

Q6. L'étude est-elle utile pour la recherche et/ou la pratique ?

En effet, nous avons noté chaque question sur une échelle de Oui (1), Non (0), Partiellement (0,5), et additionné les scores. Les 6 questions concernent notamment la clarté de l'étude (Q1), la rigueur dans la conception et la mise en œuvre et la validation (Q2) - (Q3), la validité (Q4) - (Q5) et la pertinence pratique (Q6).

1.2. Extraction des données

Pour chaque étude primaire sélectionnée, l'article a été lu en totalité pour extraire les informations pertinentes répondant aux questions de recherche. L'extraction et l'analyse des données ont été motivées par la recherche d'information permettant de décrire les différentes dimensions de la taxonomie définie dans le chapitre 2, concernant notamment le but de monitoring, les propriétés à surveiller, les méthodes de spécification et d'implémentation, le support d'incertitude, et les techniques d'instrumentation (les couches d'abstraction ainsi que les technologies d'instrumentation). Pour chaque étude, les données ont été extraites et enregistrées dans un tableur. Les données extraites des études primaires en fonction des dimensions de la taxonomie sont présentées dans les tableaux A.3, A.4 A.5 et A.6. Notez qu'une étude peut figurer dans plusieurs emplacements d'une même dimension si elle considère plusieurs aspects à la fois.

Tableau A. 3 Description des études primaires en réponse aux questions quoi de la taxonomie

Etudes primaires	**Type de propriétés**	**Domaine d'application**	**Couche d'abstraction du système**	**Technologie d'instrumentation**

[Ref1]	exactitude	SAS	-	-
[Ref2]	cohérence & exactitude	DAS	métier	middleware/aspectJ
[Ref3]	exactitude	SAS	métier (composant)	-
[Ref4]	sûreté de fonctionnement	SBS	métier (processus)	interception des événements
[Ref5]	exactitude	vision par ordinateur	-	-
[Ref6]	fiabilité & performance	SBS	application	plugins
[Ref7]	exactitude	DAS	infrastructure	capteurs
[Ref8]	performance (latency)	SAS	application	-
[Ref9]	performance (qualité des configurations)	SAS	-	-
[Ref10]	cohérence	SBS	-	-
[Ref11]	sûreté de fonctionnement & performance & sécurité	systèmes orientés composant	application	Windows Management Instrumentation (WMI)
[Ref12]	sûreté de fonctionnement & performance & sécurité	systèmes orientés événements (EBS)	application	Ganglia
[Ref13]	sûreté de fonctionnement (confiance humaine)	SOA collaborative	application	intercepteurs
[Ref14]	sûreté de fonctionnement	programmes Java	application (machine virtuelle)	Java Platform Debugger Architecture (JVM TI interfaces)
[Ref15]	performance	-	infrastructure	Glimpse
[Ref16]	sûreté de fonctionnement & sécurité	Machine virtuelle Java	application	Bytecode java

[Ref17]	exactitude	SBS	application	intercepteur coté serveur (JBOSS)
[Ref18]	sûreté de fonctionnement & sécurité	Systèmes complexes	application	interception
[Ref19]	sûreté de fonctionnement (contraintes de temps)	Systèmes distribués temps réel	application	moniteurs locaux
[Ref20]	sûreté de fonctionnement & sécurité	Programmes Java	application	aspectJ
[Ref21]	exactitude & fiabilité	Systèmes orientés composant	application	moniteurs externes
[Ref22]	sécurité (les politique de confidentialité : HIPAA & GLBA)	systèmes critiques de sécurité	application	-
[Ref23]	fiabilité	Programmes java	application	aspectJ
[Ref24]	exactitude	Programmes SCALA	application	Langage dédié interne
[Ref25]	fiabilité	-	application	aspectJ
[Ref26]	exactitude	SAS	application	Eclipse TPTP
[Ref27]	fiabilité	SOA-BPEL	métier	AspectJ
[Ref28]	fiabilité	SOA-BPEL	application	interception
[Ref29]	performance & sûreté de fonctionnement	SAS	infrastructure	capteurs
[Ref30]	exactitude	SOS	infrastructure	sondes
[Ref31]	exactitude	Systèmes pervasifs	application	JMX
[Ref32]	exactitude	Internet computing	application	Microsoft EIF
[Ref33]	exactitude	-	application	Microsoft EIF
[Ref34]	fiabilité (ISO 26262 Standard)	Systèmes embarqués automobiles	infrastructure (hardware)	moniteurs de hardware
[Ref35]	fiabilité	SBS	métier	-
[Ref36]	politiques de sécurité	Systèmes	application	aspectJ

		pervasifs		
[Ref37]	exactitude	Systèmes pervasifs	infrastructure	capteurs
[Ref38]	exactitude	Systèmes adaptatifs	-	capteurs
[Ref39]	fiabilité & vivacité	circuits et systèmes embarqués	-	-
[Ref40]	performance & sécurité	systèmes orientés composants	application	aspectJ
[Ref41]	fiabilité & maintenabilité	SBS	infrastructure	bus d'événement
[Ref42]	exactitude	SBA	utilisateur	-
[Ref43]	exactitude	SBS	-	collecteurs de données
[Ref44]	sécurité	Attaques malveillantes	-	instrumentation du code source
[Ref45]	exactitude	Attaques malveillantes	-	instrumentation du code source
[Ref46]	exactitude	systèmes sociotechniques	infrastructure	capteurs
[Ref47]	exactitude	-	application	interception
[Ref48]	performance & fiabilité	SBS	infrastructure et application	web service invocation Framework (WSIF)
[Ref49]	exactitude	DAS	application	AI-like Daemons
[Ref50]	exactitude	SBS	-	-

Tableau A. 4: Les langages de spécification des exigences adoptés dans les études primaires

Méthode de modélisation	**Langage utilisé**
Modélisation orientée but	Tropos étendu[Ref50][Ref46], KAOS[Ref49][Ref33], modèle de buts étendu [Ref47], modèle de buts[Ref45][Ref26][Ref9],modèles de buts et anti-buts[Ref44], KAOS+logique floue[Ref43],Techne[Ref42], i*[Ref7][Ref1]
Programmation par contraintes	SAT [Ref50] [Ref47] [Ref46] [Ref45] [Ref44] [Ref3]
Event Calculus	[Ref41][18]
Logique temporelle	PCTL[Ref11][Ref48][Ref29][Ref6],

	LTL[Ref25][Ref43][Ref36][Ref29] [Ref25][Ref23],pTLTL[Ref34][Ref25][Ref23], CSL[Ref29][Ref6], RTL[Ref19], rPATL[Ref8], MFOTL[Ref22].
Systèmes de transition d'états	DTMC [Ref48][Ref6][Ref29], Markov chain profile[Ref40], Processus de décision markovien [Ref29], HMM[Ref5], CTMC[Ref6][Ref29], FSM[Ref38][Ref25], Langage dédié[Ref32], FSA[Ref27], modèles de comportements [Ref21], Automate fini[Ref20], système de transition d'états étiqueté[Ref4], NFA[Ref23][Ref28]
Machine learning	Classificateur à base de règles [Ref37]
UML	CIM[Ref14],[Ref11], UML+ Problème de cadre[Ref3], diagramme de séquence[Ref28][Ref40], MDA (MOF+OCL)[Ref17], OCL[Ref14][Ref2], OCL_{TM}[Ref26]
Modélisation orientée processus	WS-BPEL [Ref28] [Ref4] [Ref27], SLAng SLA [Ref17], WSDL [Ref13], AADL [Ref2]
Langage dédié	langage dédié [Ref30], langage dédié interne pour Scala [Ref24], langage dédié [Ref32], langage dédié [Ref1], formalisme 'Lola' [Ref39].
Autres	OrBAC[Ref36], archivol[Ref35], Modèle de tâche+NFR [Ref31], Méthode B[Ref22] LIME ISL[Ref23], Timeline[Ref20],ConSpec[Ref16], SMG[Ref8], XML [Ref41], ERE[Ref25],CFG[Ref25], graphe de dépendance[Ref40], arbre d'appels dynamique[Ref40], AHP [Ref31], PMM[Ref15], Théorie d'utilité[Ref12], RELAX[Ref7], SADL[Ref35], programmation par contrat[Ref11], IF/THEN [10][13], Event/Action[33], FLEA[49].

Tableau A. 5 : Description des études primaires en réponse aux questions pourquoi, comment, quand et où de la taxonomie

Etudes primaires	But du monitoring	Techniques / outils d'analyse de propriétés	Déclencheur	Mode (location) du monitoring	Support d'incertitude
[Ref1]	conscience des exigences	système de raisonnement procédural / BDI agents	changement de contexte	synchrone (interne)	interne (utilisation des claims comme marqueurs d'incertitude)
[Ref2]	reconfiguration	SAT / solveur SAT4J	reconfiguration	synchrone (externe)	Interne (configurations imprévues sont

					stockées dans une BD)
[Ref3]	détection des défaillances & reconfiguration	SAT / solveur SAT	changement de contexte	hors ligne/asynchrone (externe)	externe (sensibilité au contexte via des dépendances contextuelles)
[Ref4]	détection des défaillances	Model checking symbolique / techniques d'abstraction et de raffinement	évènements	asynchrone (externe)	-
[Ref5]	diagnostic	-	changement de contexte	-	-
[Ref6]	détection des défaillances	Model checking / le model checker PRISM	changement de contexte	synchrone (interne)	interne (utilisation des estimateurs bayésiens)
[Ref7]	reconfiguration	Moteur de règles floue / fonctions d'appartenances	changement de contexte	synchrone (interne)	interne (Relaxation des exigences)
[Ref8]	détection des défaillances & adaptation	Model checking / le model checker PRISM-games.	reconfiguration	synchrone (interne)	-
[Ref9]	reconfiguration	Méthodes mathématiques / Théorie d'utilité	reconfiguration	asynchrone (externe)	-
[Ref10]	test en ligne & détection des défaillances	Test en ligne/ analyse des valeurs limites	périodique	asynchrone (externe)	-
[Ref11]	mise en vigueur des contrats	Traitement d'évènements à base de connaissance	évènements	asynchrone (externe)	-
[Ref12]	mise en vigueur des contrats de services	Méthodes mathématiques / Théorie d'utilité	événements	asynchrone (externe)	-
[Ref13]	sélection et découverte des services	Test en ligne / le générateur de test Genesis2 **(**Juszczyk & Dustdar, 2010)	changement dans la structure du réseau	asynchrone (externe)	-
[Ref14]	détection des défaillances	Model checking / le model checker USE (Gogolla et	sur demande	asynchrone (externe)	-

		al., 2007)			
[Ref15]	reconfiguration des moniteurs	Traitement d'événements complexe / l'outil drools Fusion	événements	asynchrone (externe)	-
[Ref16]	mise en vigueur des politiques de sécurité	Vérification déductive / le générateur des conditions de vérification « faible précondition » + vérificateur formel par SMT	appels à des méthodes	synchrone (interne)	-
[Ref17]	détection des défaillances	Programmation par contraintes / OCL2 (Akehurst et al., 2003).	périodique	asynchrone (interne)	-
[Ref18]	détection des défaillances	Traitement d'évènements à base de connaissance temporelle	événements	asynchrone (externe)	-
[Ref19]	détection des défaillances	Implémentation personnalisée / techniques de recherche du plus court chemin	évènements	synchrone (externe)	-
[Ref20]	détection des défaillances	Model checking à l'aide d'automate	sur demande	synchrone (interne)	-
[Ref21]	détection des défaillances	Model-checking symbolique/techniques d'abstraction	appels à des méthodes (opérations d'un composant)	asynchrone (externe)	-
[Ref22]	détection des défaillances	Model checking symbolique BMC	événements	asynchrone (externe)	-
[Ref23]	détection des défaillances	Model checking à l'aide d'automate / AspectJ & Scheck	appels à des méthodes	synchrone (interne)	-
[Ref24]	détection des défaillances	Système de production/ l'algorithme RETE (Forgy, 1982)	événements	synchrone (interne)	-

[Ref25]	détection de défaillances	Implémentation personnalisée / model checker pour les logiques LTL/PTLTL/ERE/ CFG	événements	hors ligne/asynchrone/synchrone (interne/externe)	-
[Ref26]	évolution des exigences	Implémentation personnalisée / formalisme ReqMon	événements	asynchrone (externe)	-
[Ref27]	détection des défaillances & reconfiguration	Model-checking à l'aide d'automate/Aspect J	événements (traceCuts)	synchrone (interne)	-
[Ref28]	détection des défaillances & reconfiguration	Vérification déductive / solveur SMT MathSAT	sur demande	asynchrone (externe)	-
[Ref29]	détection des défaillances & reconfiguration	Model checking probabiliste / model checker PRISM	périodique/après un changement de contexte	asynchrone (externe)	-
[Ref30]	détection des défaillances	Implémentation personnalisée / Inférence de type + ICC (Egyed, 2006)	événements	asynchrone (externe)	interne (support de la définition dynamique des contraintes)
[Ref31]	détection des défaillances & adaptation	Traitement d'évènements à base de connaissance temporelle / moteur d'inférence JENA + web sémantique	changement de contexte	asynchrone (externe)	-
[Ref32]	détection des défaillances	Model-checking symbolique / techniques d'abstraction	changement de contexte	asynchrone (externe)	-
[Ref33]	détection des défaillances	BD active	événements	asynchrone (externe)	-
[Ref34]	détection des défaillances	Implémentation personnalisée / formalisme PathExplorer (Havelund & Roşu, 2002)	événements	asynchrone (externe)	-
[Ref35]	composition des services	Moteur de règle déterministe	les interactions entre services	asynchrone (externe)	-
[Ref36]	adaptation des exigences & détection des vulnérabilités	Implémentation personnalisée / outil de monitoring	événements	asynchrone (externe)	-

		Montimage (Wehbi et al., 2012)			
[Ref37]	adaptation des exigences	Moteur d'inférence déterministe	changement de contexte	synchrone (interne)	externe (incertitude concernant la capacité des capteurs)
[Ref38]	détection des défaillances	Model-checking symbolique / techniques d'abstraction	changement de contexte	synchrone (interne)	-
[Ref39]	détection des défaillances	Implémentation personnalisée basé e sur le langage dédié LOLA	flux entrant	hors ligne asynchrone (externe)	-
[Ref40]	analyse de performance & détection d'intrusion	Implémentation personnalisée basée sur l'analyse des traces	événements	synchrone (interne)	-
[Ref41]	mise en vigueur des contrats SLA	Moteur d'inférence déterministe	appel de service	asynchrone (externe)	-
[Ref42]	conscience des exigences	Système de raisonnement procédural/ BDI agent	changement de contexte	synchrone (interne)	interne ('goals' adaptatifs)
[Ref43]	adaptation des exigences	BD active / LTL model checker	changement de contexte	asynchrone (externe)	interne (fuzzy 'goals')
[Ref44]	diagnostic	SAT / solveur SAT	changement de contexte	asynchrone (externe)	-
[Ref45]	diagnostic	SAT / solveur SAT	changement de contexte	asynchrone (externe)	-
[Ref46]	détection des défaillances & reconfiguration	Système de raisonnement procédural / BDI agents	changement de contexte	asynchrone (externe)	-
[Ref47]	adaptation des exigences	SAT / solveur SAT	changement de contexte	synchrone (interne)	interne (incertitude concernant les préférences des SGs)
[Ref48]	sélection et paramétrage des services	Model checking / model checker PRISM	périodique/après un changement de contexte	asynchrone (externe)	interne (incertitude concernant paramètres du modèle des exigences)
[Ref49]	détection des défaillances & remédiation	BD active	événements	asynchrone (externe)	-
[Ref50]	adaptation des exigences	SAT / solveur SAT	changement de contexte	asynchrone (externe)	Incertitude externe

Tableau A. 6 : Détails des scores de qualité des études primaires pour chaque question de qualité

Etudes primaires	Q1	Q2	Q3	Q4	Q5	Q6	Score total
[Ref1]	1	1	0	1	0	1	4
[Ref2]	1	1	0	0	0	1	3
[Ref3]	1	1	0.5	0	0	1	3
[Ref4]	1	1	0.5	0	0	1	3
[Ref5]	1	1	0	0	0	1	3
[Ref6]	1	1	0.5	1	1	1	5
[Ref7]	1	1	1	0	0	1	4
[Ref8]	1	1	1	1	0	1	5
[Ref9]	1	1	0.5	0.5	1	1	4
[Ref10]	1	1	0	0	0	1	3
[Ref11]	1	1	0	1	0	1	4
[Ref12]	1	1	0	0	0	1	3
[Ref13]	1	1	1	0	0	1	4
[Ref14]	1	1	0	0	0	1	3
[Ref15]	1	1	0	0	0	1	3
[Ref16]	1	1	0	0	0	1	3
[Ref17]	1	1	0.5	1	0	1	4
[Ref18]	1	1	1	1	0	1	5
[Ref19]	1	1	0.5	0	0	1	3
[Ref20]	1	0.5	0	1	0	1	3
[Ref21]	1	1	0	1	0	1	4
[Ref22]	1	1	0.5	0	0	1	3
[Ref23]	1	1	0	0	0	1	3
[Ref24]	1	1	1	1	0.5	1	5
[Ref25]	1	1	1	1	0.5	1	5
[Ref26]	1	1	1	1	1	1	6
[Ref27]	1	1	0.5	0.5	0	1	3
[Ref28]	1	1	1	0.5	0.5	1	4
[Ref29]	1	1	1	1	1	1	6
[Ref30]	1	1	1	1	0.5	1	5
[Ref31]	1	1	0.5	0	0	1	3
[Ref32]	1	1	0	0.5	0	1	3
[Ref33]	1	1	0.5	0	0.5	1	3
[Ref34]	1	1	0	0.5	0	1	3
[Ref35]	1	1	0.5	0	0	1	3
[Ref36]	1	1	0.5	1	0	1	4
[Ref37]	1	1	0.5	1	0	1	4
[Ref38]	1	1	0.5	0	0	1	3
[Ref39]	1	1	0.5	0	0	1	3
[Ref40]	1	1	0.5	0	0	1	3
[Ref41]	1	1	0	0	0	1	3
[Ref42]	1	1	0	0	0	1	3

[Ref43]	1	1	0	0	0	1	3
[Ref44]	1	1	0.5	0	0.5	1	3
[Ref45]	1	1	1	1	0.5	1	5
[Ref46]	1	1	1	1	0.5	1	5
[Ref47]	1	1	1	1	0.5	1	5
[Ref48]	1	1	1	1	0.5	1	5
[Ref49]	1	1	0	0	0	1	3
[Ref50]	1	1	0.5	1	0.5	1	4

1.3. Synthèse et analyse de données

Cette section présente une synthèse des données extraites des études primaires qui a pour objectif d'examiner les approches de monitoring des exigences à l'exécution. Une analyse quantitative des résultats a été également effectuée. Au cours de cette étape, nous avons analysé les résultats et extrait les statistiques. Notre approche de synthèse de données repose une synthèse narrative ainsi que sur les étapes d'analyse thématique recommandées dans (Cruzes et al., Septembre 2011), à savoir l'extraction de données, le codage de données, la traduction des codes en thèmes, la création d'un modèle de thèmes d'ordre supérieur et enfin l'évaluation de la fiabilité de la synthèse.

a. Analyse démographique

La Figure A.1 montre la répartition des études primaires dans le temps. Il est à noter que l'intérêt pour le monitoring des exigences en temps réel remonte à l'an 1994 avec l'article de Fickas (1994), qui est à l'origine du terme « requirements monitoring » (le monitoring des exigences) et l'un des pionniers dans ce domaine, suivi de quelques études au cours de la prochaine décennie. Le nombre des études a remarquablement augmenté depuis 2009 pour atteindre le maximum en 2012.

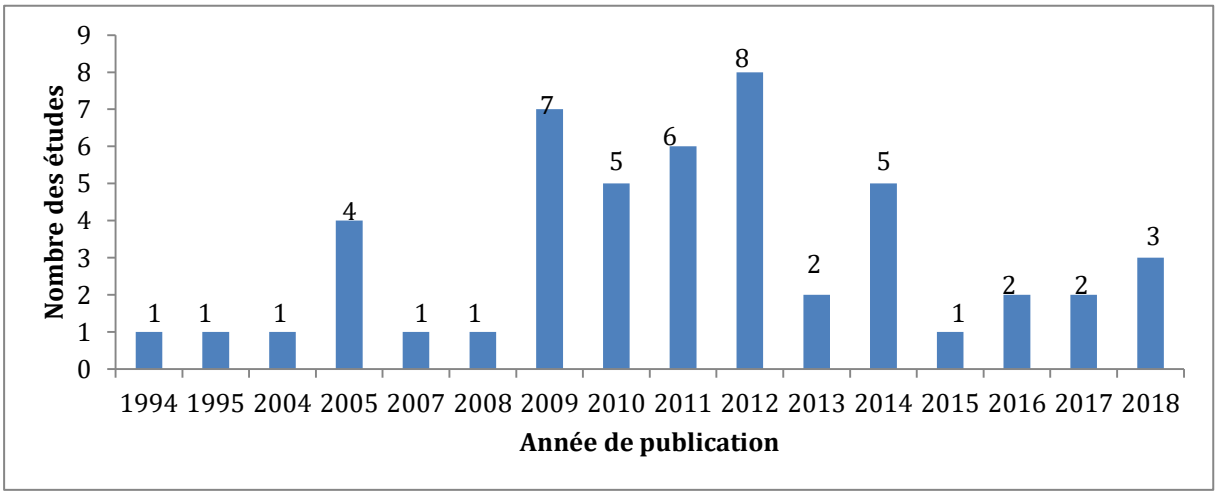


Figure A. 1 : Nombre d'études par année de publication

D'autre part, la Figure A.2 montre le score de qualité des études primaires dans le temps, selon la notation du programme d'évaluation critique (CASP). Nous pouvons constater que la qualité des études s'est améliorée depuis l'an 2009. Une bonne qualité des études (score 6) a été repérée

principalement dans les années 2012 et 2018. En plus, une analyse approfondie des différentes questions de qualité (voir Figure A.3) montre que la plupart des études primaires ont reçu un faible score au niveau des questions Q3, Q4 et Q5, elles sont liées spécialement à un manque de validation empirique dans la plupart des études, et à une crise de reproductibilité.

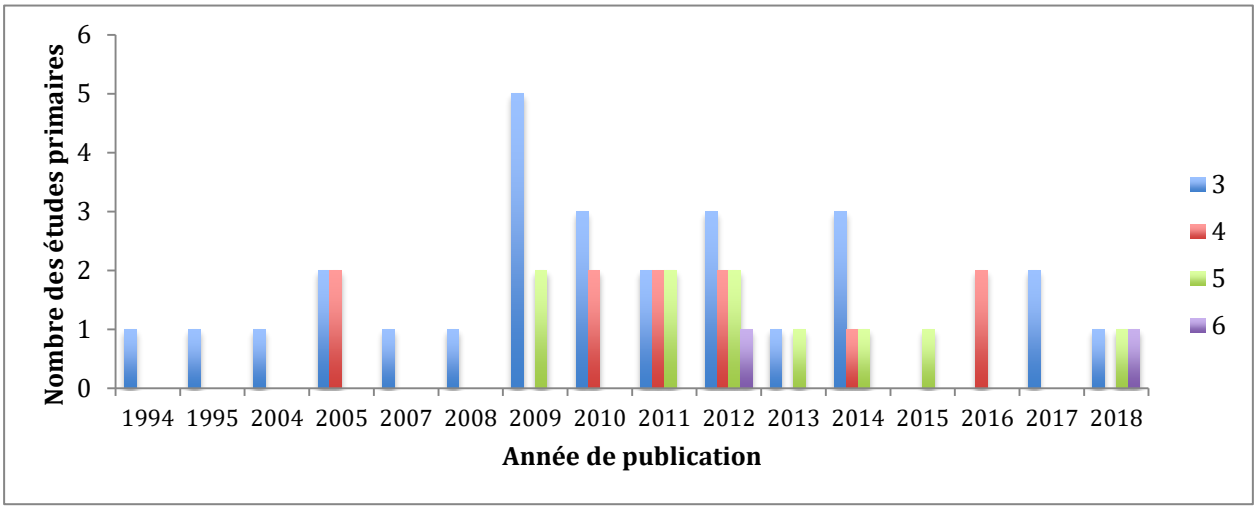


Figure A. 2 : distribution de la qualité des études primaires dans le temps

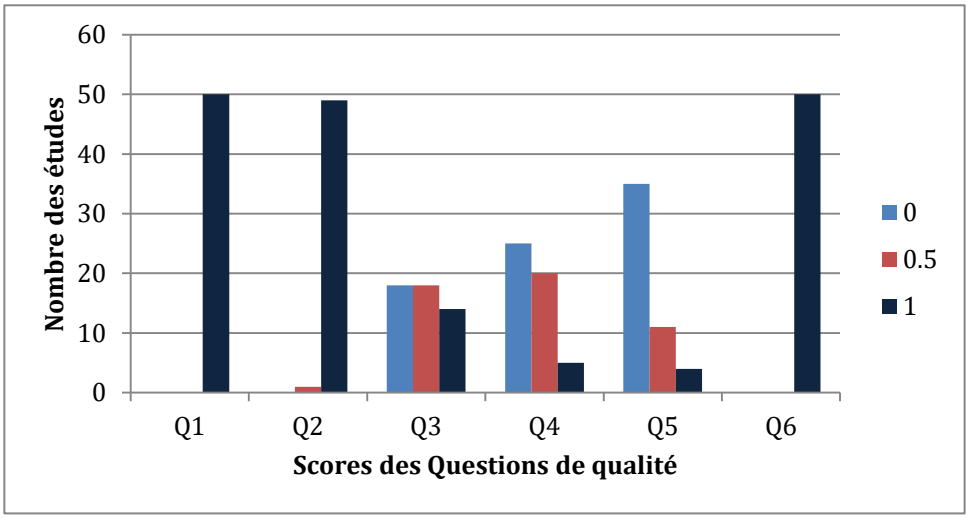


Figure A. 3 : notation des études primaires selon les six questions de qualité (CASP)

b. L'analyse quantitative selon les dimensions de la taxonomie

Au cours de l'analyse, nous avons classé des études primaires dans la catégorie NC (non communiqué) lorsque aucune information n'est fournie quant aux certaines dimensions de la taxonomie.

Dimension Spécification des exigences : cette dimension contient les deux éléments type de propriétés et langage de spécification :

- Type de propriétés : la Figure A.4 montre la distribution des études primaires selon les propriétés surveillées par le monitoring. La majorité des études ont porté sur la sûreté de fonctionnement (21

études) (vivacité, la fiabilité et la maintenabilité) et la propriété d'exactitude (22 études). Les propriétés de sécurité et de performance ont reçu moins d'attention.

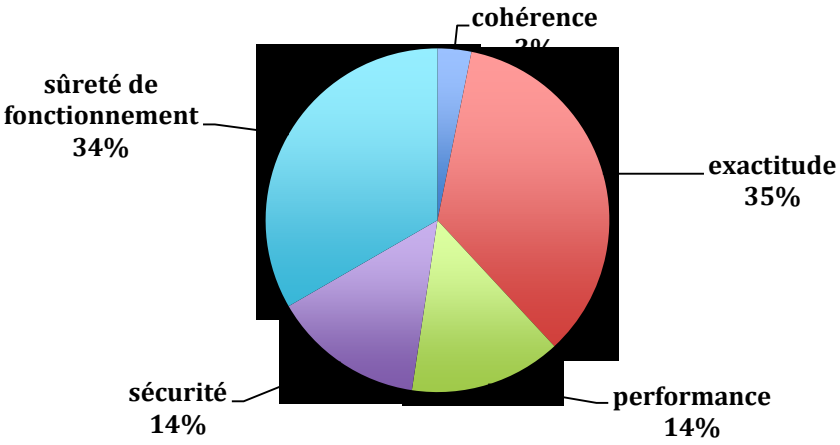


Figure A. 4 : Distribution des études primaires selon les types de propriétés.

- Langage de spécification : La répartition des études primaires selon les méthodes de spécification des propriétés est illustrée par la Figure A.5. La catégorie « Autres » inclut particulièrement les expressions régulières (ERE), XML et les graphes de contraintes. De la Figure, on peut constater un intérêt significatif pour les systèmes de transition d'états (14 études), notamment avec les chaines de Markov, les FSM et les automates finis, les méthodes de modélisation orientée but, notamment avec les méthodologies TROPOS et KAOS (12 études), la logique temporelle, principalement avec les formalismes LTL et PCTL (12 études) et le langage UML (10 études). La programmation par contraintes en particulier le problème SAT (6 études) et la modélisation orientée processus, notamment avec les langages WS-BPEL et WSDL (6 études) ont été également considérées.

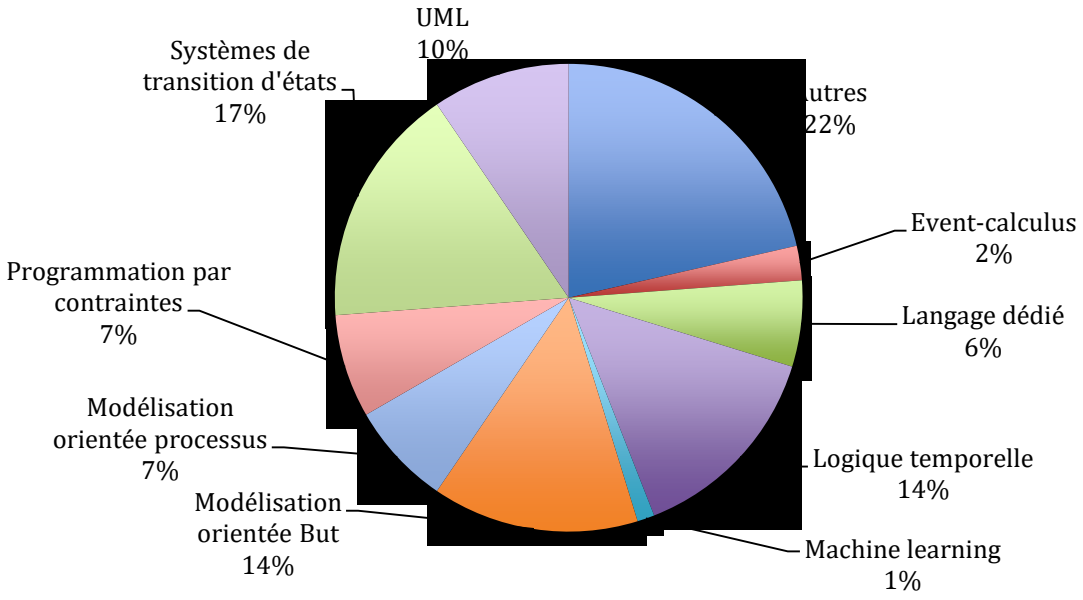


Figure A. 5 : Distribution des études primaires selon les méthodes de modélisation

D’autre part, une corrélation entre les méthodes de modélisation et le score de qualité (voir Figure A.6) montre que ces méthodes ont reçu les meilleurs scores de qualité.

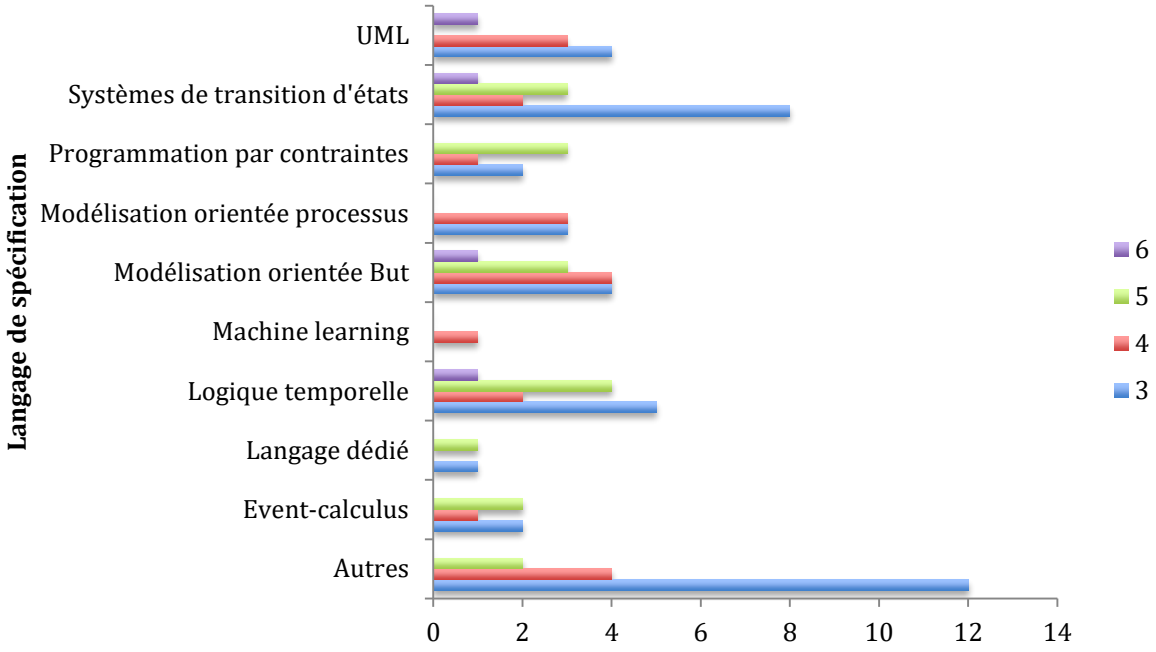


Figure A. 6 : Fréquence des scores de qualité obtenus selon les méthodes de spécification

Dimension But du monitoring : La figure A.7 montre que plus que deux tiers des études (32 études) ont visé la détection des défaillances, avec seulement 3 de ces études qui ont supporté le diagnostic de ces défaillances. Nous pouvons conclure qu’il y a des lacunes de recherche en ce sens. D’autre part, le support de l’adaptation/la reconfiguration, la conscience des exigences ont également suscité beaucoup d’intérêt de la part des chercheurs. Quant à l’analyse de performance, ce but a reçu moins d’attention avec seulement une étude trouvée.

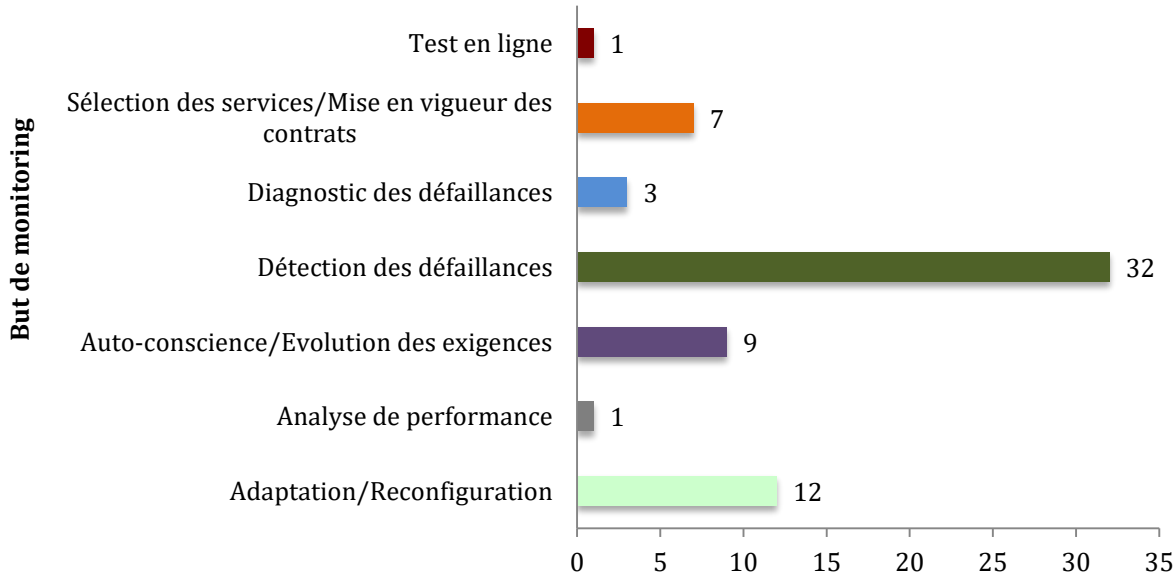


Figure A. 7 : Distribution des études primaires selon le but de monitoring

Dimension Approche : cette dimension est caractérisée par (i) les techniques d'implémentation (ii) le déclencheur du monitoring, l'emplacement du code du monitoring et le support d'incertitude

- Implémentation du monitoring : La Figure A.8 montre que l'implémentation personnalisée et la modélisation par SAT et les différents types de model-checking restent les méthodes les plus utilisées pour l'analyse des propriétés. De même, les bases de données actives et temporelles (CEP-TK) ont reçu beaucoup d'attention. Par contre, la logique floue et la programmation par contrainte n'ont pas été explorées pleinement dans ces études.

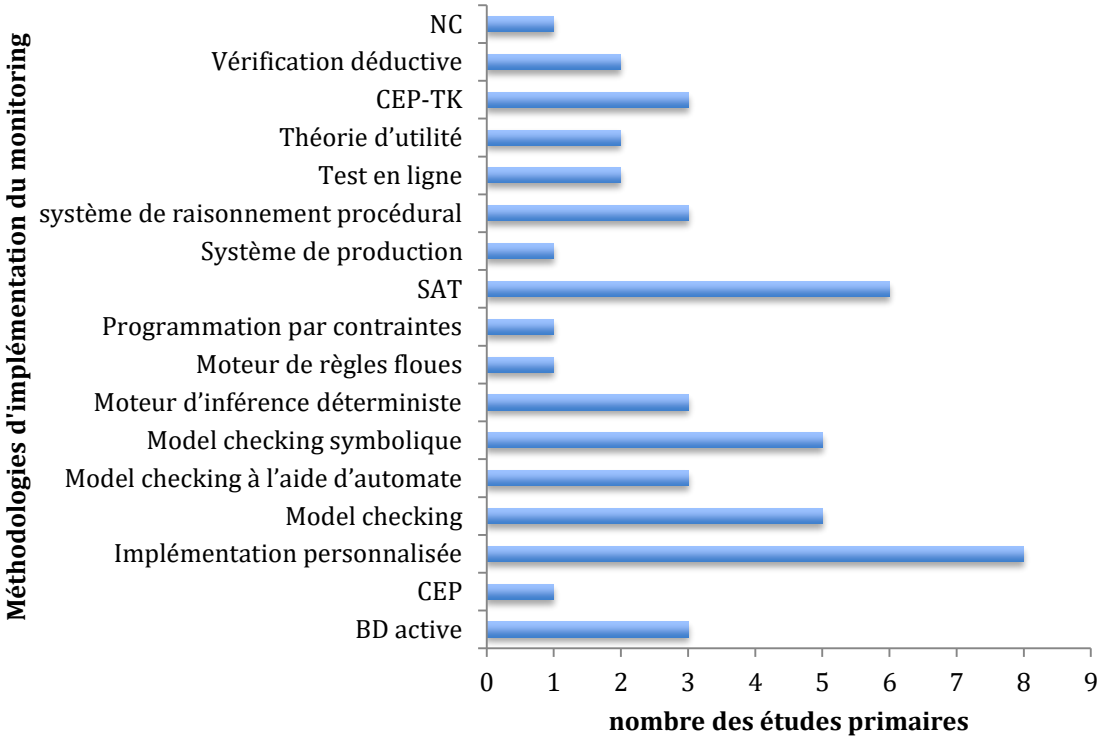


Figure A. 8 : Distribution des études primaires selon l'implémentation du monitoring

- Déclencheur : La Figure A.9 montre que les approches orientées évènements (19 études) sont les plus utilisées ainsi que les approches portant sur des changements des conditions d'environnement (19 études).

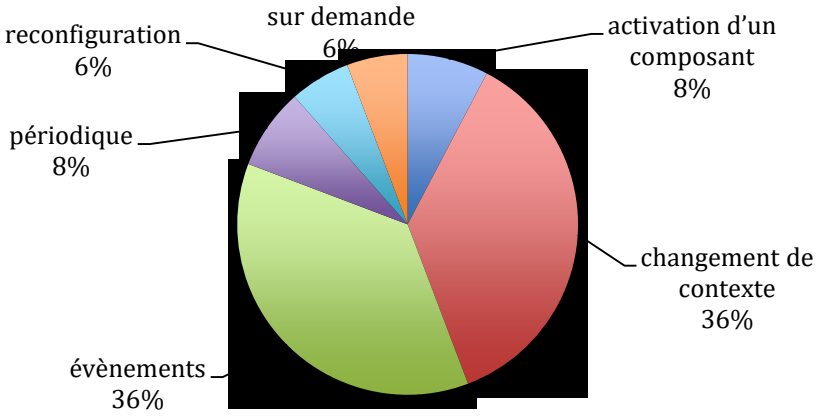


Figure A. 9 : Distribution des études primaires selon le déclencheur de monitoring

- Support d'incertitude : La Figure A.10 décrit la répartition des études primaires selon s'ils supportent l'incertitude (interne ou externe) ou non. Les résultats montrent que la majorité des approches (39 études) ne considèrent pas l'incertitude dans leurs démarches. Par ailleurs, nous avons constaté que la plupart des études (18 études) supportant l'incertitude, ont considéré principalement l'incertitude interne liée à la modélisation des exigences.

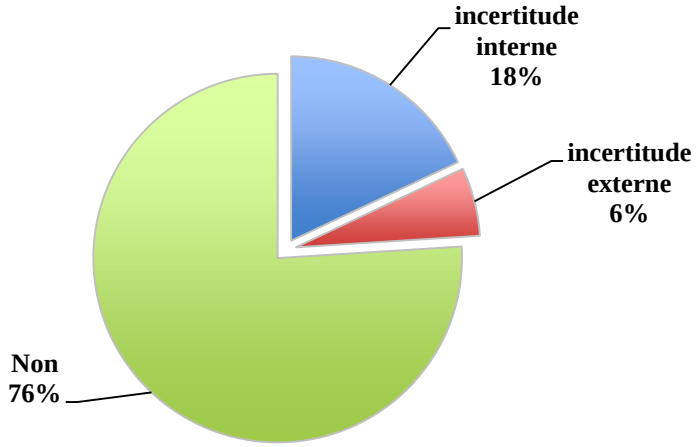


Figure A. 10 : Distribution des études primaires selon leur support à l'incertitude

Dimension instrumentation : La Figure A.11 montre la répartition des études primaires selon les solutions d'instrumentation mises en œuvre pour la collecte des données, considérées du point de vue des différentes couches d'abstraction du système cible, allant du plus bas— le plus proche des composants électroniques, au plus proche de l'utilisateur. Les résultats montrent qu'un nombre significatif d'études considère la couche application (26 études), suivi d'un nombre moins important pour les couches infrastructure (9 études) et métier (5 études). La couche utilisateur est largement ignorée par rapport aux autres couches (1 étude). Le reste (10 études) concerne les études théoriques qui n'ont pas implémenté leurs approches.

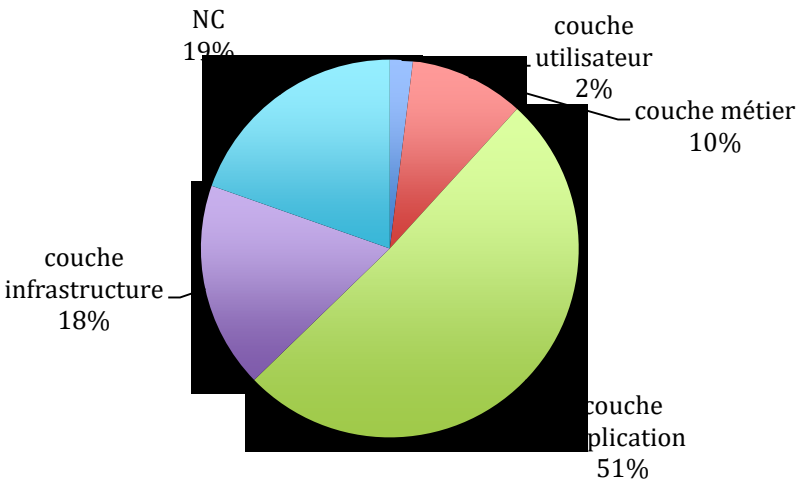


Figure A. 11 : Distribution des études primaires selon les couches d'abstraction

Dimension Mode du monitoring : La Figure A.12 montre que la plupart des études (40 études) ont adopté une solution de monitoring de type asynchrone pour ne pas alourdir le système cible. Contre 15 études qui ont mis en œuvre un monitoring synchrone.

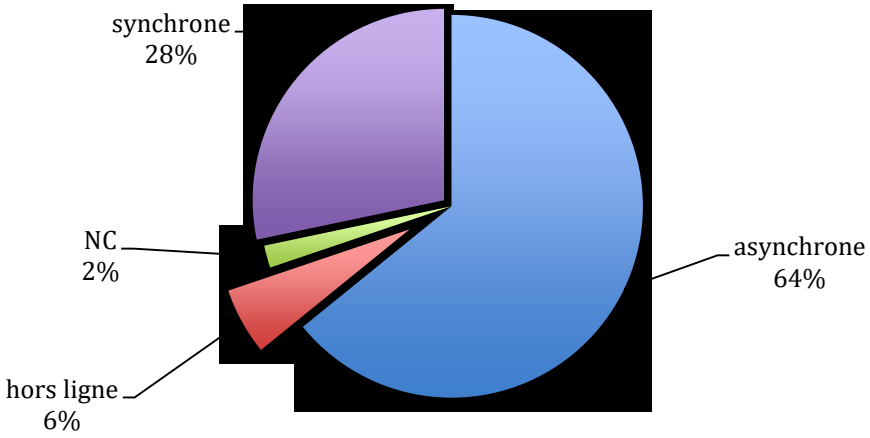


Figure A. 12 : Distribution des études primaires selon le mode de monitoring