



UNIVERSITÉ LARBI TÉBESSI, TÉBESSA

Faculté des Sciences Exactes et Sciences de la Nature et de la Vie

Département des Mathématiques et Informatique

Laboratoire des Mathématiques, Informatique et Systèmes (LAMIS)

THÈSE

En vue de l'obtention du diplôme de

Doctorat LMD

Discipline: Informatique

Spécialité : Système d'information coopératif

Présentée et soutenue par

Ayoub YAHIAOUI

Titre :

**Ingénierie des exigences pour les systèmes
auto-adaptatifs**

devant le jury ci-dessous :

M. Makhlouf Dardour	MCA	Université Larbi Tébessi	Président
M. Djamel Benmerzoug	MCA	Université de Constantine 2	Examineur
M. Karim Zarour	MCA	Université de Constantine 2	Examineur
M. Abdeljalil Gattal	MCA	Université Larbi Tébessi	Examineur
M. Louardi Bradji	MCA	Université Larbi Tébessi	Examineur
M. Hakim Bendjenna	Prof.	Université Larbi Tébessi	Encadreur
M. Mohammed Amroune	MCA	Université Larbi Tébessi	Co-encadreur

Le 22/01/2019



Laboratoire des Mathématiques, Informatique et
Systèmes (LAMIS)

Ingénierie des exigences pour les systèmes auto-adaptatifs

Ayoub YAHIAOUI

UNIVERSITÉ LARBI TÉBESSI, TÉBESSA

ملخص

تصبح التطبيقات تدريجيا أكبر ، أقل تجانسا ، وأكثر تعقيدا ، هناك بعض المهام حيث لا يمكن تعريف البيانات الخاصة بالمحيط، خلال مرحلة التصميم. في ضوء ذلك ، يصبح من الضروري أن تتكيف هذه الأنظمة تلقائيًا مع التغيرات في البيئة. نحن نسمي هذه الأنظمة "أنظمة التكيف الذاتي (SAA)". يقدم التكيف الذاتي نهجًا واعدًا لإدارة تعقيد الأنظمة الحديثة. التطبيقات التي تتطلب قدرات من هذا النوع، نذكر على سبيل المثال ، أنظمة البنية التحتية الذكية ، وشبكات الاستشعار ، والأنظمة المدمجة. التغيرات في العوامل البيئية مثل التفاعلات البشرية (المدخلات الغير المخطط لها) تجعل التحليل صعبًا لجميع الحالات التي سيكون فيها النظام خلال فترة حياته. لذلك ، يجب أن تكون أنظمة التكيف الذاتي قادرة على التكيف مع مجموعة من السياقات البيئية ، ولكن الطبيعة الدقيقة لهذه السياقات تظل مبهمه. يتمثل التحدي العام في تطوير أنظمة التكيف الذاتي، في كيفية التعبير عن المواصفات من أجل جعل أنظمة التكيف الذاتي قادرة على التعامل مع المشاكل التي تطرحها مجالات التطبيق ، بما في ذلك الغموض السلوكي. أنظمة التكيف الذاتي تحتاج إلى مواصفات مرنة ومنهجية في نفس الوقت. تقدم هذه الأطروحة لغة جديدة تسمى PSAS ، لمواصفات المتطلبات ، وهي تعتمد على أنماط مواصفات أنظمة التكيف الذاتي لتسهيل التعبير عن قيود الوقت للتكيف الذاتي. سيتم اقتراح أداة دعم لتسهيل مرحلة المواصفات. يتم تحسين فعالية لغة PSAS من خلال حالات في العالم الحقيقي ، وكذلك من خلال تقييم تجريبي لثلاث دراسات حالة ، ومقارنتها بنهج مماثلة من ناحية الوضوح وسهولة الاستخدام.

Résumé

Au fur et à mesure que les applications deviennent de plus en plus volumineuses, encore plus hétérogènes et complexes, il existe certaines tâches où les données contextuelles ne peuvent pas être définies durant la phase de conception. A la lumière de cela, il devient vital pour ces systèmes de s'adapter automatiquement aux changements qui se produisent dans l'environnement. Nous appelons ces systèmes des « Systèmes Auto-adaptatif (SAA) ». L'auto-adaptation présente une approche prometteuse pour la gestion de la complexité des systèmes actuels. Les applications nécessitant des capacités de type SAA sont par exemple des systèmes d'infrastructure intelligents, les réseaux de capteurs et les systèmes embarqués. Les changements de facteurs de l'environnement comme les interactions humaines (entrées imprévues) rendent l'analyse difficile de tous les états dans lesquels le système sera pendant sa durée de vie. Par conséquent, un SAA doit être capable de s'adapter à un ensemble de contextes environnementaux, mais la nature exacte de ces contextes reste vaguement comprise. Un défi global dans le développement des SAAs est la façon d'exprimer une spécification afin de rendre le SAA capable de gérer les problèmes posés par les domaines d'application, y compris les incertitudes comportementales. Les SAAs ont besoin de spécifications flexibles tout en étant formelles. Cette thèse présente un nouveau langage appelé PSAS, pour la spécification des exigences, il est basé sur des patrons spécification pour les SAAs afin de faciliter l'expression des contraintes temporelles d'auto-adaptation. Un outil de support sera proposé pour faciliter la phase spécification. L'efficacité du langage PSAS est renforcée via des instances du monde réel, ainsi que par une évaluation empirique de trois cas d'études, et une comparaison avec des approches similaires en matière de clarté et de facilité d'usage.

Abstract

As applications become larger, more heterogeneous, and more complex, there are some tasks where contextual data cannot be defined during the design phase. In light of this, it becomes vital for these systems to automatically adapt to changes in the environment. We call these systems “self-adaptive systems (SAS)”. Self-adaptation presents a promising approach for managing the complexity of today's systems. Applications requiring SAS-type capabilities are, for example, intelligent infrastructure systems, sensor networks, and embedded systems. Changes in environmental factors such as human interactions (unplanned inputs) make the analysis difficult for all states in which the system will be during its lifetime. Therefore, an SAS must be able to adapt to a set of environmental contexts, but the exact nature of these contexts remains vaguely understood. A global challenge in SAS development is how to express a specification in order to make the SAS capable of handling the problems posed by the application domains, including behavioral uncertainties. SASs need flexible specifications while being formal. This thesis presents a new language called PSAS, for requirements specification, it is based on specification patterns for SAS to facilitate the expression of time constraints of self-adaptation. A support tool will be proposed to facilitate the specification phase. The effectiveness of the PSAS language is enhanced through real-world instances, as well as through an empirical evaluation of three case studies, and a comparison with similar approaches to clarity and ease of use.

Remerciement

L'obtention de mon doctorat est probablement l'activité la plus difficile de mes 28 premières années de vie. Les meilleurs et les pires moments de mon parcours de doctorat ont été partagés avec de nombreuses personnes. Ce fut un grand privilège de passer plusieurs années au sein du laboratoire LAMIS à l'Université de Tebessa et ses membres me resteront toujours chers.

Tout d'abord, je voudrais exprimer ma sincère gratitude à mon conseiller, le professeur Hakim BENDJENNA, pour le soutien continu apporté au cours de mes études de doctorat et de mes recherches, pour sa patience, sa motivation, son enthousiasme et son immense connaissance. Je n'aurais pas pu imaginer avoir un meilleur conseiller et mentor pour mon doctorat. Je voudrais également exprimer ma sincère gratitude à mon co-encadreur, Mohammed AMROUNE. Leurs conseils m'ont aidé dans tout le temps de la recherche et l'écriture de cette thèse.

Je remercie particulièrement mon co-superviseur, Philippe ROOSE, pour ses conseils, commentaires afin de raffiner le travail présent dans cette thèse.

Outre mes conseillers, je voudrais remercier le comité de thèse: Dr. Makhoul DARDOUR, Dr. Djamel BENMERZOU, Dr. Karim ZAROOR, Dr Gattal Abdeljalil, Dr. Louardi BRADJI, pour leurs encouragements, leurs commentaires éclairés et leurs questions. Réviser une thèse n'est pas une tâche facile et je leur suis reconnaissant de leurs commentaires réfléchis et détaillés.

Je remercie également MESRS (Ministère de l'enseignement supérieur et de la recherche scientifique) du gouvernement Algérien, qui a financé mon stage de finalisation de thèse au LIUPPA / Université de Pau et des Pays de l'Adour en France.

Je voudrais remercier mes parents, leur inspiration a été ma force motrice. Je leur dois tout et j'aimerais pouvoir leur montrer à quel point je les apprécie.

Je voudrais remercier ma famille et mes amis: mes frères, en particulier Abdeldjalil qui m'a aidé à la rédaction de ce mémoire, et mes sœurs qui étaient toujours là pour m'encourager et me motiver.

Table des matières

INTRODUCTION GENERALE.....	11
1.1 MOTIVATION.....	11
1.2 LA PROBLEMATIQUE DE LA RECHERCHE.....	12
1.3 OBJECTIFS DE LA THESE.....	13
1.4 METHODOLOGIE.....	13
1.5 ORGANISATION DE LA THESE.....	14
L'INGENIERIES DES EXIGENCES.....	16
2.1 INTRODUCTION.....	16
2.2 LE GENIE LOGICIEL.....	16
2.2.1 <i>Qu'est-ce que le génie logiciel.....</i>	<i>16</i>
2.2.2 <i>Origine de l'ingénierie logicielle.....</i>	<i>17</i>
2.2.3 <i>Activités de génie logiciel.....</i>	<i>17</i>
2.3 EXIGENCES ET SPECIFICATIONS LOGICIEL.....	17
2.3.1 <i>Qu'est-ce que l'ingénierie des exigences.....</i>	<i>17</i>
2.3.2 <i>Documents d'exigences.....</i>	<i>18</i>
2.3.3 <i>Types d'exigence.....</i>	<i>18</i>
2.3.4 <i>Exigences fonctionnelles et non fonctionnelles.....</i>	<i>19</i>
2.3.5 <i>Elicitation des exigences.....</i>	<i>20</i>
2.3.6 <i>Exigences et qualité.....</i>	<i>24</i>
2.3.7 <i>Exigences et le cycle de vie de logiciel.....</i>	<i>25</i>
2.3.8 <i>Traçabilité des exigences.....</i>	<i>28</i>
2.3.9 <i>Exigences et modélisation.....</i>	<i>32</i>
2.3.10 <i>Exigences et tests.....</i>	<i>34</i>
2.3.11 <i>Exigences dans les domaines du problème et de la solution.....</i>	<i>35</i>
2.4 MODELISATION DU SYSTEME POUR L'INGENIERIE DES EXIGENCES.....	38
2.5 SPECIFICATION ET REVISION DES EXIGENCES.....	39
2.5.1 <i>Exigences pour les exigences.....</i>	<i>40</i>
2.5.2 <i>Documents de besoins structurants.....</i>	<i>41</i>
2.5.3 <i>Exigences clés.....</i>	<i>42</i>
2.5.4 <i>Utilisation des attributs.....</i>	<i>43</i>
2.6 LA LANGUE DES EXIGENCES.....	44
2.7 CONCLUSION.....	45
LA SPECIFICATION DES EXIGENCES POUR LES SYSTEMES AUTO-ADAPTATIFS.....	46
3.1 INTRODUCTION.....	46
3.2 LES SYSTEMES AUTO-ADAPTATIFS.....	46
3.2.1 <i>Perspectives architecturales.....</i>	<i>47</i>
3.2.2 <i>Technologies habilitantes.....</i>	<i>50</i>
3.2.3 <i>Cadre d'adaptation.....</i>	<i>55</i>
3.3 TRAVAUX SUR L'INGENIERIE DES EXIGENCES POUR L'AUTO-ADAPTATION.....	56
3.4 TRAVAUX SUR LES PATRONS DE SPECIFICATION.....	62
3.5 CONCLUSION.....	65
ELEMENTS DE BASE.....	66

4.1	PATRON DE SPECIFICATION	66
4.2	PORTEE DE PATRON	66
4.3	LOGIQUES TEMPORELLES	68
4.3.1	<i>Logique temporelle linéaire (LTL)</i>	68
4.3.2	<i>Logique temporelle métrique (MTL)</i>	68
4.3.3	<i>Logique temporelle métrique floue (FMTL)</i>	69
4.3.4	<i>Grammaire anglaise structurée</i>	76
4.4	CONCLUSION	77
PSAS : LES PATRONS DE SPECIFICATION POUR L'AUTO-ADAPTATION.....		78
5.1	INTRODUCTION	78
5.2	CATALOGUE DE PATRON AUTO-ADAPTATIF	78
5.2.1	<i>Patrons auto-adaptatifs d'occurrence</i>	78
5.2.2	<i>Patrons auto-adaptatifs d'ordre</i>	79
5.3	SYNTAXE DU LANGAGE.....	80
5.4	SEMANTIQUE DU LANGAGE	82
5.4.1	<i>Famille d'occurrence auto-adaptative</i>	84
5.4.2	<i>Famille d'ordre auto-adaptatif</i>	88
5.5	LA VERIFICATION AUTOMATIQUE.....	90
5.6	CONCLUSION	91
PSAS : TESTS ET MISE EN ŒUVRE		92
6.1	INTRODUCTION	92
6.2	PSAS-TOOL	92
6.2.1	<i>Présentation de PSAS-tool</i>	93
6.2.2	<i>Exemples de spécifications PSAS</i>	94
6.3	EVALUATION EMPIRIQUE.....	95
6.3.1	<i>Scénario du protocole de communication MQTT</i>	96
6.3.2	<i>Autres scénarios : AAL et ADS</i>	98
6.4	DISCUSSION	101
6.5	CONCLUSION	103
CONCLUSIONS GENERALES ET TRAVAUX FUTURS.....		104
	NOTRE CONTRIBUTION	104
	PERSPECTIVES	105
PRODUCTION SCIENTIFIQUE.....		107
BIBLIOGRAPHIE.....		108
ACRONYMES.....		116

Liste des figures

Fig. 1 Processus de l'ingénierie des exigences (Sommerville, 2010)	18
Fig. 2. Raisons de l'échec du projet (%) (Group, 2009)	23
Fig. 3. Facteurs de succès du projet (%) (Group, 2009)	23
Fig. 4. Ingénierie des exigences en couches (Easterbrook, 2001).	25
Fig. 5. Exigences dans le modèle en V (Easterbrook, 2001).....	25
Fig. 6. Ingénierie des exigences de l'entreprise (Elizabeth, Ken, & Jeremy, 2004)	26
Fig. 7. Risque de traçabilité dans la gestion du changement (Dick, Hull, & Jackson, 2017)	27
Fig. 8. Traçabilité des exigences (Gotel & Finkelstein, 1994).	29
Fig. 9. Analyse d'impact et de dérivation (Elizabeth, Ken, & Jeremy, 2004).	31
Fig. 10 Analyse de couverture (Dick, Hull, & Jackson, 2017).....	32
Fig. 11 Stratégie de qualification (Elizabeth, Ken, & Jeremy, 2004)	35
Fig. 12 Attributs des exigences (Elizabeth, Ken, & Jeremy, 2004)	43
Fig. 13 Processus d'auto-adaptation (Kephart & Chess, 2003).....	49
Fig. 14 Portée du patron : présente la chronologie autorisée d'un motif spécifique, délimitée par les événements A et B (Dwyer, Avrunin, & Corbett, 1999).....	67
Fig. 15 Distribution de possibilité trapézoïdale.....	70
Fig. 16 Période éventuellement/nécessairement après/avant une date inconnue	71
Fig. 17 à gauche : $\pi e, \pi f$ et la ligne en gras montrée E, F , à droite : $E, F =]E, +\infty \cap -\infty, F[$, dans ce cas E, F est vide.	72
Fig. 18 à gauche : La ligne en gras montrée $F, E = F, +\infty \cap -\infty, E$, à droite : $F, E =]F, +\infty \cap -\infty, E[$, dans ce cas F, E est vide.	73
Fig. 19 Exemple d'une exclusion mutuelle dans laquelle une ressource commune Pr est partagée par deux processus.....	74
Fig. 20 La partie de (5, 8, 10, 14) avant $\tau = 10$ indiquée par la zone en dégradé	75
Fig. 21 Vue d'ensemble de l'outil PSAS-tool	92

Fig. 22 La sélection de patron et la création de propriété.....	93
Fig. 23 Spécification en terme d'anglais structuré et FMTL.....	94
Fig. 24 Aperçu sur le protocole MQTT	96
Fig. 25 Traitement des exigences avec PSAS.....	101
Fig. 26 Nombre d'opérateurs et le nombre de cas d'étude (PSAS et autres)	102

Liste des tableaux

Tableau 1. Points d'intérêt de la traçabilité	28
Tableau 2. Types d'analyse de traçabilité (Dick, Hull, & Jackson, 2017).....	30
Tableau 3 Problèmes et espaces de solution (Elizabeth, Ken, & Jeremy, 2004).....	36
Tableau 4 Capacités requises pour les besoins (Dick, Hull, & Jackson, 2017).....	40
Tableau 5 Liste des concepts communs utilisés dans la programmation orientée aspect.....	52
Tableau 6. Ingénieries des exigences pour le systèmes auto-adaptatif et patrons de spécification.	63
Tableau 7 Vue générale sur les langages d'auto-adaptation.....	64
Tableau 8 Portée du patron	67
Tableau 9 Génération de patron.....	80
Tableau 10 Anglais structuré pour les patrons auto-adaptatifs.	80
Tableau 11 Définitions de temps et de quantité.	82
Tableau 12 Spécifications du réfrigérateur intelligent (RELAX vs PSAS).....	98
Tableau 13 Spécifications ADS.....	99

Introduction générale

1.1 Motivation

La grande explosion de l'information, l'intégration de la technologie et l'évolution continue des systèmes complexes (logiciels techniquement exigeants) vers les systèmes à grande échelle nécessitent de nouvelles approches pour la conception, l'exécution et la gestion des systèmes (De Lemos R. , et al., 2009). Le résultat de cette évolution est que le logiciel doit devenir plus polyvalent, autoréparable, configurable, personnalisable, récupérable, flexible, résilient et économe en énergie en s'adaptant aux changements intervenant dans les contextes, les environnements ou les caractéristiques du système. Par conséquent, les systèmes qui sont capables d'ajuster leur comportement en fonction de leur perception de l'environnement et du système lui-même sont devenus un sujet de recherche important.

Habituellement, le logiciel est l'élément clé dans tous les travaux innombrables pour explorer le comportement auto-adaptatif. C'est le cas de la recherche dans divers domaines d'application tels que les systèmes embarqués, les robots mobiles et autonomes, les réseaux mobiles ad hoc, les applications peer-to-peer, les systèmes multi-agents et les réseaux de capteurs. De nombreux domaines de recherche ont également porté sur les aspects de l'auto-adaptation de leur point de vue, tels que les systèmes distribués, les systèmes tolérants aux pannes, l'intelligence artificielle, la gestion intégrée, les systèmes basés sur les connaissances, la théorie de contrôle, la robotique, etc. Cependant, la réalisation correcte de la fonctionnalité d'auto-adaptation reste un défi important qu'ont récemment tenté de construire des systèmes auto-adaptatifs dans des domaines d'application spécifiques. De plus, peu d'efforts ont été faits pour développer des approches de génie logiciel appropriées pour la mise en œuvre de l'auto-adaptation.

Un système auto-adaptatif (SAA) est capable de changer son comportement en réponse à des changements dans son contexte. Un système auto-adaptatif doit également surveiller en permanence les changements environnementaux et réagir en conséquence. Évidemment, le système ne peut pas tout surveiller. On peut supposer que le système doit toujours garder un ensemble d'objectifs de haut niveau qui doivent être satisfaits indépendamment des conditions environnementales. Mais les objectifs non-cruciaux peuvent être assouplis, permettant au système une certaine flexibilité pendant ou après l'adaptation. Les principales considérations pour la construction de systèmes auto-adaptatifs sont: Les aspects environnementaux doivent être surveillés par le système auto-adaptatif, et la réaction du système s'il ne détecte pas l'état optimal de l'environnement.

L'ingénierie des exigences (IE) se concentre sur ce qu'un système doit faire et sous quelle restriction il doit le faire. L'ingénierie des exigences pour les systèmes auto-adaptatifs doit donc être sensible aux adaptations possibles et cela limite le développement de ce type d'adaptations.

Les questions à traiter comprennent les aspects de l'environnement pertinents à l'adaptation, les exigences permises de varier ou d'évoluer au moment de l'exécution, et celles-ci doivent toujours être maintenues. L'ingénierie des exigences pour les systèmes auto-adaptatifs doit faire face à l'incertitude car les informations sur les contextes d'exécution futurs sont incomplètes et les exigences du comportement du système peuvent donc nécessiter une modification au moment de l'exécution pour faire face aux changements environnementaux.

L'ingénierie des exigences pour les systèmes auto-adaptatifs s'avère être un domaine d'étude largement ouvert (De Lemos R. , et al., 2009). Les approches actuelles pour spécifier les exigences ne sont pas bien adaptées à l'incertitude (De Lemos R. , et al., 2009), qui est une préoccupation essentielle pour les systèmes auto-adaptatifs. Naturellement, le langage naturel est l'approche la plus couramment utilisée pour spécifier les exigences de l'industrie.

Généralement, les documents d'exigences sont formés en utilisant des affirmations telles que « le système doit faire ceci ... » Pour les systèmes auto-adaptatifs, le verbe «doit» doit être relaxé, par exemple « le système peut le faire ... ou il peut le faire ... » ou « si le système ne peut pas le faire ... alors il devrait éventuellement le faire ... » . En conséquence, un nouveau vocabulaire des exigences pour les systèmes auto-adaptatifs est nécessaire, ce qui permet l'expression de la flexibilité dans les documents d'exigences, afin de faire face à l'incertitude.

1.2 La problématique de la recherche

La demande de systèmes auto-adaptatifs ne cesse de croître, couvrant les activités d'ingénierie des exigences au cours du processus de développement et du temps d'exécution. Afin d'assurer le comportement adaptatif, il est également important de surveiller la traçabilité du système en cours d'exécution et son engagement à répondre aux exigences. En outre, il est important de reconnaître et de prendre en charge l'évolution des exigences au moment de l'exécution. Compte tenu de la complexité croissante des applications, celles-ci nécessitent une adaptation, les objets que les développeurs manipulent et analysent doivent être plus abstraits.

Les spécifications formelles, modèles graphiques, politiques, etc, peuvent être utilisés comme base pour l'évolution des systèmes adaptatifs par rapport au code source, l'artefact traditionnel qui est manipulé après le déploiement du système, la capacité de maintenir la traçabilité parmi les artefacts pertinents (y compris le code), la capacité à maintenir les contraintes d'assurance pendant et après l'auto-adaptation, et la capacité du système à être adapté et à maintenir la traçabilité du système initial. L'ingénierie des exigences doit être proactive pour relever ces défis afin de garantir que les capacités d'auto-adaptation sont fournies aux concepteurs de systèmes auto-adaptatifs.

1.3 Objectifs de la thèse

L'objectif principal de cette thèse est de fournir un langage permettant d'identifier le degré de flexibilité lors de la spécification des exigences pour les systèmes auto-adaptatifs et de fournir un mécanisme compatible pour la vérification des exigences de ce type de système, ce mécanisme implémente la flexibilité liée aux exigences. Nous pensons que deux points doivent être traités. D'une part, les exigences auto-adaptatives devraient tenir compte du risque d'incertitude. D'autre part, la vérification de ces exigences doit être effectuée le plus tôt possible, avant même que le développement ne commence. Afin de gérer l'incertitude dans les systèmes auto-adaptatifs, les langages d'ingénierie des exigences devraient identifier explicitement les points de flexibilité dans les exigences (Whittle J. a.-M., 2009).

1.4 Méthodologie

Notre proposition est basée sur des patrons de spécifications (Dwyer, Avrunin, & Corbett, 1999) qui est une méthodologie pour spécifier les exigences. Nous proposons un catalogue de modèles qui supportent l'auto-adaptation dans la spécification, et qui inclut la relaxation dans le patron de spécification afin de gérer l'incertitude. Pour les systèmes auto-adaptatifs, les exigences assouplies sont importantes car elles sont associées aux caractéristiques d'adaptabilité de ces systèmes. Nous proposons ensuite une cartographie pour chaque patron afin d'apporter une signification formelle à notre langage.

Le choix de la logique floue est motivé par le fait qu'elle peut traiter des problèmes posés par des adaptations imprévues, c'est-à-dire en spécifiant de façon déclarative les formes dont une exigence peut être assouplie. L'exigence n'exige pas que toutes les autres adaptations soient spécifiées. Cette élasticité rend le choix de conception ouvert à la réalisation de l'adaptation, favorisant ainsi la conception en utilisant des algorithmes de théorie de contrôle, des règles d'adaptation, des algorithmes de planification, etc. Ce travail est un cadre d'auto-adaptation dans la phase initiale du cycle de vie de développement de systèmes auto-adaptatifs.

Notre proposition comprend un processus et un outil de soutien pour répondre aux préoccupations d'auto-adaptation énumérées ci-dessous:

- Pour spécifier une exigence, nous proposons un ensemble de patrons auto-adaptatifs basés sur des catalogues existants.
- Nous proposons une grammaire pour gérer la génération des patrons. La grammaire est une grammaire anglaise structurée, ce qui rend le langage moins ambigu par rapport aux ingénieurs avec peu d'expérience.

- Nous proposons une cartographie pour chaque patron, en termes de logique temporelle floue. La flexibilité introduite dans le patron est traduit par les contraintes temporelle floue.
- Afin de simplifier le travail de l'ingénieur des exigences, nous proposons un outil de spécification.
- Nous comparons notre approche avec des travaux connexes en matière de richesse de vocabulaire et simplicité d'usage dans divers domaines.

1.5 Organisation de la thèse

La thèse est organisée comme suit. Dans le chapitre 2, nous présentons l'ingénierie des exigences (IE) avec ses concepts de base. Dans le chapitre 3, nous donnons une description du contexte principal de notre travail, c'est-à-dire c'est quoi un SAA et comment il diffère des autres systèmes, puis nous donnons quelques exemples de SAA. Ce chapitre présente ainsi une description de l'état de l'art des différentes approches, c'est-à-dire IE pour SAA et la vérification des propriétés à l'aide de méthodes formelles. Comme cette thèse est centrée sur la définition d'une approche formelle pour l'IE de SAA, les techniques basées sur la logique temporelle sont détaillées car nous basons notre proposition sur l'utilisation de ces techniques pour définir et spécifier les exigences de SAA. Nous présentons ensuite, les raisons qui nous ont motivés à proposer cette approche en montrant quelques problèmes que nous avons identifiés avec les approches existantes. Nous expliquons ensuite la nécessité d'une approche intégrée qui prenne en compte les différentes fonctionnalités que nous proposons et de fournir un environnement d'outillage intégré pour traiter les problèmes identifiés.

Dans le chapitre 4, nous décrivons les patrons de spécification (PS), qui sont les éléments de base de notre approche ainsi que la logique métrique temporelle floue (FMTL). PS et FMTL nous servent de point de départ pour proposer notre approche intégrée.

<i>Introduction générale</i>	<i>Parties</i>					<i>Conclusions et travaux futurs</i>
	<i>Contexte</i>			<i>Contribution</i>		
	<i>2. Ingénierie des exigences</i>	<i>3. Ingénierie des exigences pour les systèmes auto-adaptatifs</i>	<i>4. Eléments de base</i>	<i>5. Patrons de spécification pour l'auto-adaptation: langage PSAS</i>	<i>6. Validation</i>	
	<i>Chapitres</i>					

Dans le chapitre 5, nous présentons PSAS qui est un langage de spécification des exigences. nous donnons une vue d'ensemble de notre approche proposée, définissant les concepts proposés tels que les deux familles de patrons de PSAS. D'autre part nous montrons comment ces patrons vont être générés à travers la grammaire structurée proposée, nous donnons ainsi la forme formelle de ces patrons de spécification. Afin de valider l'approche proposée, dans le chapitre 6, nous donnons une description de l'outil PSAS-tool que nous avons développés pour la spécification des propriétés SAA, nous donnons ensuite une description des deux études de cas, à savoir une maison intelligente et un système d'ambulance, et la spécification de certaines de leurs exigences en utilisant PSAS. Enfin une conclusion et perspectives terminent ce mémoire de thèse.

Chapitre 2

L'ingénieries des exigences

2.1 Introduction

Dans ce chapitre, nous donnons une description de l'ingénierie des exigences et sa position dans le cycle de développement logiciel, ainsi que son importance par rapport aux autres étapes de conception. D'autre part, les aspects fonctionnels et non-fonctionnels liés à IE.

2.2 Le Génie logiciel

Les systèmes logiciels sont abstraits et intangibles. Ils ne sont pas limités par les propriétés des matériaux, régis par des lois physiques ou par des procédés de fabrication. Cela simplifie le génie logiciel, car il n'y a pas de limites naturelles au potentiel du logiciel. Cependant, en raison de l'absence de contraintes physiques, les systèmes logiciels peuvent rapidement devenir extrêmement complexes, difficiles à comprendre et coûteux à modifier. C'est pourquoi ils ont besoin de processus et de techniques d'ingénierie spécifiques pour prendre en compte ces considérations.

2.2.1 Qu'est-ce que le génie logiciel

Le génie logiciel (GL) est une discipline qui s'occupe de tous les aspects de la production de logiciels depuis les premières étapes de la spécification du système jusqu'à la maintenance du système après son utilisation. Voici les définitions du GL :

- L'application d'une approche systématique, disciplinée et quantifiable au développement, à l'exploitation et à la maintenance des logiciels; c'est-à-dire l'application de l'ingénierie au logiciel (IEEE, 1990).
- L'établissement et l'utilisation de principes d'ingénierie solides afin d'obtenir des logiciels économiquement fiables et efficaces sur des machines réels (Naur & Randell, 1968).

Un bon logiciel doit fournir les fonctionnalités et les performances requises à l'utilisateur et doit être maintenable, fiable et utilisable.

2.2.2 Origine de l'ingénierie logicielle

La notion de GL a été proposée pour la première fois en 1968 lors d'une conférence organisée pour discuter de ce qu'on appelait alors la crise du logiciel (Naur & Randell, 1968). Il est apparu clairement que les approches individuelles du développement de programmes n'étaient pas adaptées aux grands systèmes logiciels complexes. Ceux-ci n'étaient pas fiables, coûtaient plus cher que prévu et étaient livrés en retard. Tout au long des années 1970 et 1980, une variété de nouvelles techniques et méthodes de génie logiciel ont été développées, telles que la programmation structurée et le développement orienté-objet. Des outils et des notations standard ont été développés et sont maintenant largement utilisés.

2.2.3 Activités de génie logiciel

L'approche systématique utilisée dans Le GL est parfois appelée un processus logiciel. Un processus logiciel est une séquence d'activités qui mène à la production d'un produit logiciel (Sommerville, 2010). Quatre activités fondamentales sont plus ou moins communes à tous les processus logiciels. nous le décrirons dans les sous-sections suivantes.

2.3 Exigences et spécifications logiciel

2.3.1 Qu'est-ce que l'ingénierie des exigences

La spécification logicielle ou ingénierie des exigences (IE) est le processus de compréhension et de définition des services requis du système et d'identification des contraintes sur le fonctionnement et le développement du système. Le IE est une étape particulièrement critique du processus logiciel car les erreurs à ce stade conduisent inévitablement à des problèmes ultérieurs dans la conception et la mise en œuvre du système. Le processus de IE de la Fig. 1 vise à produire un document d'exigences convenu qui spécifie un système répondant aux exigences des parties prenantes. Les exigences sont généralement présentées à deux niveaux de détail. Les utilisateurs finaux et les clients ont besoin d'une déclaration de haut niveau d'exigences; Les développeurs du système ont besoin d'une spécification plus détaillée.

Les exigences sont des descriptions des services qu'un logiciel doit fournir et des contraintes sous lesquelles il doit fonctionner. Les exigences peuvent aller d'énoncés abstraits de services de haut niveau ou de contraintes système à des spécifications fonctionnelles mathématiques détaillées.

L'ingénierie des exigences est le processus d'établissement des services que le client requiert du système et des contraintes sous lesquelles il doit être développé et exploité. Les exigences peuvent remplir une double fonction:

- En tant que base d'une offre pour un contrat.
- En tant que base du contrat lui-même.

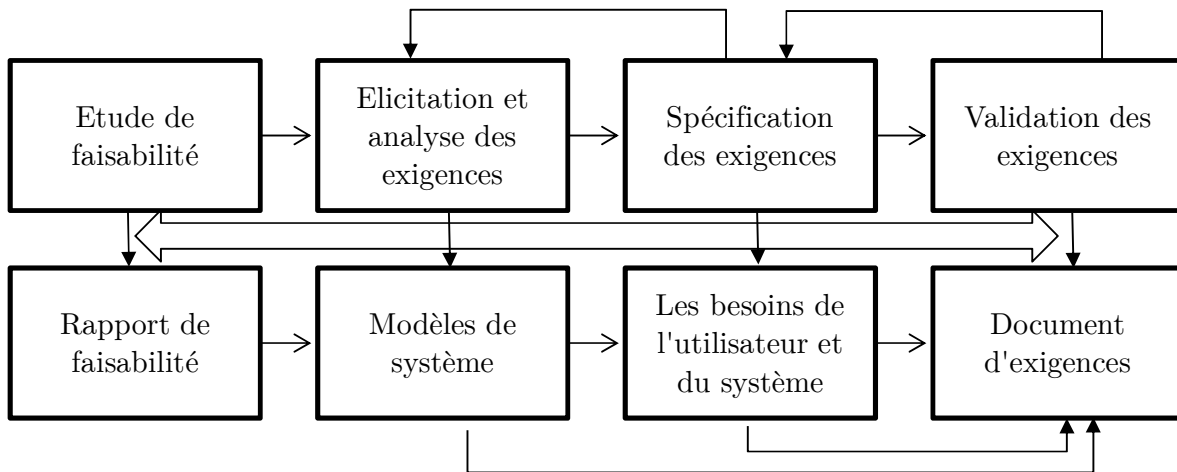


Fig. 1 Processus de l'ingénierie des exigences (Sommerville, 2010)

2.3.2 Documents d'exigences

Si une entreprise souhaite passer un contrat pour un grand projet de développement logiciel, elle doit définir ses besoins de manière suffisamment abstraite pour qu'une solution ne soit pas prédéfinie. Les exigences doivent être rédigées de manière à ce que plusieurs contractants puissent soumissionner pour le contrat, en offrant, peut-être, différentes façons de répondre aux besoins de l'organisation cliente. Une fois le contrat attribué, le contractant doit rédiger une définition système plus détaillée pour le client, afin que celui-ci comprenne et puisse valider le fonctionnement du logiciel. Ces deux documents peuvent être appelés le document de configuration du système.

2.3.3 Types d'exigence

- Besoins des utilisateurs : Des instructions en langage naturel ainsi que des diagrammes des services fournis par les systèmes et de leurs contraintes opérationnelles. Écrit pour les clients.
- Besoins du système : Un document structuré contenant des descriptions détaillées des services du système. Écrit comme un contrat entre le client et l'entrepreneur.
- Spécification du logiciel : Une description logicielle détaillée pouvant servir de base à une conception ou à une mise en œuvre. Écrit pour les développeurs.

Exemple de définition et spécifications

- Définition des besoins de l'utilisateur : Le logiciel doit fournir un moyen de représenter et d'accéder aux fichiers externes créés par d'autres outils.
- Spécifications système requises:
 - (a) L'utilisateur doit disposer de fonctionnalités lui permettant de définir le type de fichiers externes.
 - (b) Chaque type de fichier externe peut avoir un outil associé qui peut être appliqué au fichier.
 - (c) Chaque type de fichier externe peut être représenté par une icône spécifique sur l'affichage de l'utilisateur.
 - (d) Des facilités devraient être prévues pour que l'icône représentant un fichier externe soit définie par l'utilisateur
 - (e) Lorsqu'un utilisateur sélectionne une icône représentant un fichier externe, cette sélection a pour effet d'appliquer l'outil associé au type de fichier externe au fichier représenté par l'icône sélectionnée.

2.3.4 Exigences fonctionnelles et non fonctionnelles

a) Exigences fonctionnelles

Décrire la fonctionnalité ou les services du système. Dépend du type de logiciel, des utilisateurs attendus et du type de système sur lequel le logiciel est utilisé. Les besoins des utilisateurs fonctionnels peuvent être des déclarations de haut niveau sur ce que le système doit faire. les exigences système fonctionnelles doivent décrire les services du système en détail.

Exemples

- L'utilisateur doit pouvoir effectuer une recherche dans l'ensemble de l'ensemble initial de bases de données ou en sélectionner un sous-ensemble.
- Le système doit fournir aux utilisateurs appropriés des visualiseurs pour lire les documents dans le magasin de documents.
- Chaque commande se voit attribuer un identifiant unique (ID Commande) que l'utilisateur pourra copier dans la zone de stockage permanent du compte.

b) Exigences non fonctionnelles

- Exigences du produit
 - Exigences spécifiant que le produit livré doit se comporter de manière particulière, par exemple vitesse d'exécution, fiabilité etc.
- Exigences organisationnelles
 - Les exigences qui sont une conséquence des politiques et procédures organisationnelles, par exemple normes de processus utilisées, exigences de mise en œuvre, etc.
- Exigences externes
 - les exigences découlant de facteurs extérieurs au système et à son processus de développement, par exemple: exigences d'interopérabilité, exigences législatives, etc.

2.3.5 Elicitation des exigences

L'élicitation regroupe toutes les exigences du système des parties prenantes et englobe toutes les activités impliquées dans la découverte des exigences d'un système. Les développeurs et les ingénieurs système travaillent en étroite collaboration avec les clients et les utilisateurs finaux pour mieux cerner le problème à résoudre et pour combler l'écart entre les parties prenantes et les développeurs (Bendjenna, 2010). Les techniques d'élicitation facilitent ce processus en permettant de :

- Savoir plus sur le problème à résoudre.
- Décrire les fonctionnalités du système et les attributs non fonctionnels.
- Améliorer les performances du système.
- Surmonter les contraintes matérielles.
- Combler l'écart entre les parties prenantes et les développeurs.

L'évolution rapide de la technologie et l'intensification de la concurrence exercent une pression toujours plus forte sur le processus de développement. L'ingénierie des exigences est au cœur de la capacité d'une organisation à guider le navire et à suivre la complexité croissante.

Le logiciel est actuellement la force dominante pour changer ou créer de nouveaux produits. La tendance est déterminée par trois facteurs clés (Elizabeth, Ken, & Jeremy, 2004) :

1. Complexité arbitraire. Les systèmes les plus complexes tendent à être ceux avec des logiciels, souvent intégrés dans les composants les plus profonds du système. La complexité de ces produits n'est limitée que par l'imagination de ceux qui les conçoivent.
2. Distribution instantanée. Aujourd'hui, une entreprise peut penser à un nouveau produit, l'implémenter dans un logiciel et le distribuer rapidement dans le monde entier. Par exemple, un constructeur automobile peut améliorer le logiciel de son système de diagnostic et le transmettre électroniquement dans le monde entier à des dizaines de milliers de salles d'exposition en une journée.
3. Composants « Prêt à l'emploi ». Les systèmes sont désormais construits à partir de la technologie achetée et de composants prêts à l'emploi avec une réduction correspondante du cycle de développement du produit.

L'impact net de ces tendances est une intensité soudaine de la concurrence et la capacité de monopoliser les avantages de la nouvelle technologie sans avoir besoin de grandes usines. Le résultat est la pression pour réduire le cycle de développement et le temps de déployer la technologie. Cependant, le « time-to-market » ne suffit pas. Le véritable objectif est le « time-to-market avec le **bon produit** ». La définition des exigences nous permet de convenir et de visualiser le « bon produit ». En tant qu'élément essentiel du processus d'ingénierie des systèmes, l'ingénierie des exigences définit d'abord l'étendue du problème, puis relie toutes les informations de développement ultérieures. C'est seulement de cette manière que nous pouvons espérer contrôler et diriger l'activité du projet; gérer le développement d'une solution à la fois appropriée et rentable.

Les exigences sont la base de chaque projet, définissant ce que les parties prenantes - utilisateurs, clients, fournisseurs, développeurs, entreprises - d'un nouveau système potentiel ont besoin et ce que le système doit faire pour répondre à ce besoin. Pour être bien compris de tous, ils sont généralement exprimés en langage naturel et le défi consiste à saisir le besoin ou le problème d'une manière complète et non ambiguë sans recourir à un langage spécialisé difficile à comprendre (Cheng, et al., 2009). Une fois communiquées et acceptées, les exigences guident l'activité du projet. Cependant, les besoins des parties prenantes peuvent être nombreux et variés et peuvent même entrer en conflit. Ces besoins peuvent ne pas être clairement définis au départ, peuvent être contraints par des facteurs indépendants de leur volonté ou peuvent être influencés par d'autres objectifs qui changent au fil du temps. Sans une base d'exigences relativement stable, un projet de développement ne peut qu'échouer. C'est comme entreprendre un voyage dans le désert

sans aucune idée de la destination et sans une carte de navigation. Les exigences fournissent à la fois la « carte de navigation » et les moyens de direction à la destination sélectionnée.

Les exigences convenues servent de base pour planifier le développement d'un système et son acceptation à la fin. Ils sont essentiels lorsque des compromis judicieux et éclairés doivent être faits et qu'ils sont également vitaux lorsque des changements sont nécessaires au cours du processus de développement. Comment évaluer l'impact d'un changement sans un modèle suffisamment détaillé du système précédent? Si non, que devrions-nous retourner si le changement doit être annulé ?

Même si le problème à résoudre et les solutions potentielles sont définies, il faut évaluer les risques à ne pas fournir une solution satisfaisante. Peu de créateurs ou d'intervenants appuieront le développement de produits ou de systèmes sans une stratégie de gestion des risques convaincante. Les exigences permettent la gestion des risques le plus tôt possible dans le développement. Les risques des exigences peuvent être surveillés, leur impact évalué et les effets des plans d'atténuation et de repli bien compris avant que des coûts de développement significatifs aient été encourus.

Les exigences sont donc la base pour (Cheng, et al., 2009) :

- a. La planification du projet.
- b. La gestion des risques.
- c. Les tests d'acceptation.
- d. Le « Trade-off » : Renoncer à la fonctionnalité pour en obtenir une autre.
- e. Le contrôle des changements.

Les raisons les plus courantes des échecs d'un projet ne sont pas techniques et la Fig. 2 identifie les principales raisons pour lesquelles les projets échouent. Les données tirées d'enquêtes menées par le groupe Standish (Group, 2009), et montrent le pourcentage de projets qui ont indiqué diverses raisons de l'échec du projet. Ceux soulignés sont directement liés aux exigences.

Les problèmes se répartissent en trois catégories principales:

1. Exigences - soit mal organisées, mal exprimées, faiblement liées aux parties prenantes, changeant trop rapidement ou inutiles; Attentes irréalistes.
2. Problèmes de gestion des ressources - manque d'argent et manque de soutien ou d'incapacité à imposer une discipline et une planification appropriées; beaucoup d'entre eux proviennent d'un mauvais contrôle des exigences.

3. Politique - qui contribue aux deux premiers problèmes.

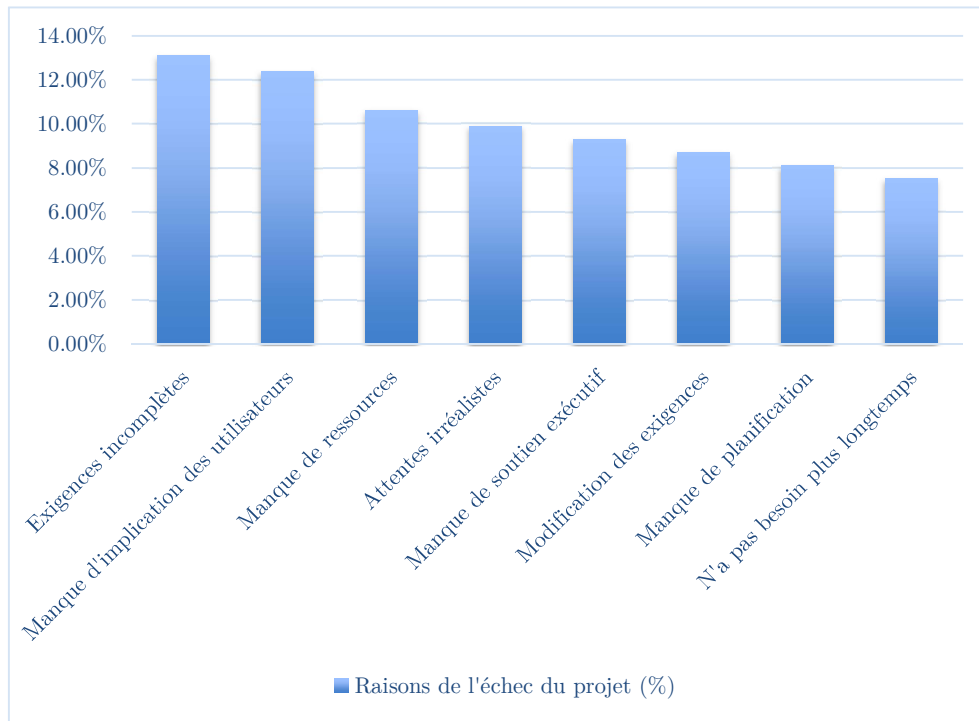


Fig. 2. Raisons de l'échec du projet (%) (Group, 2009)

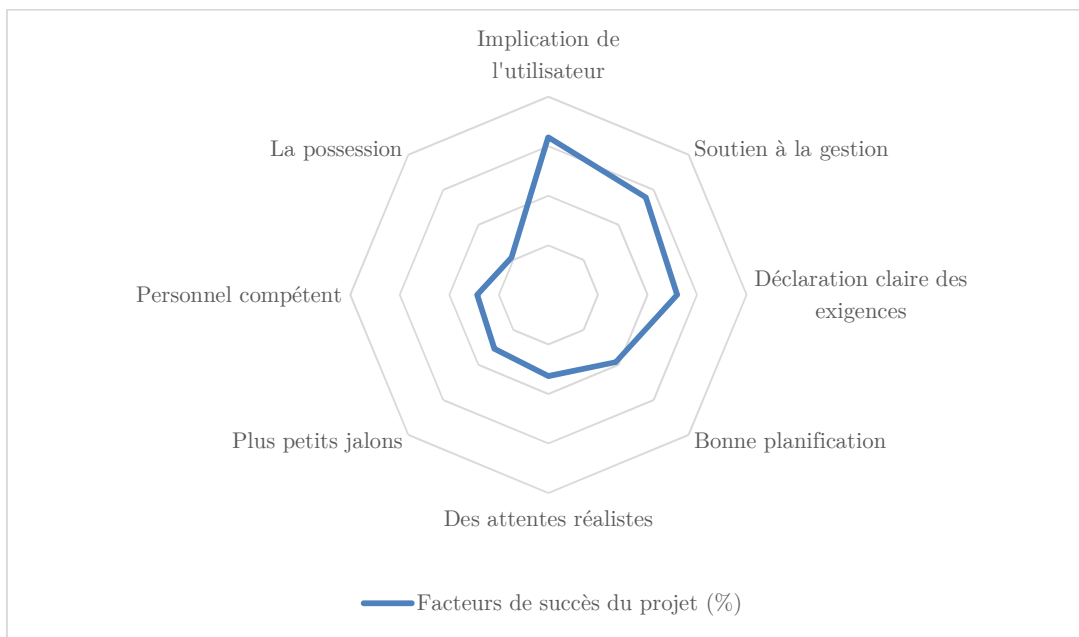


Fig. 3. Facteurs de succès du projet (%) (Group, 2009)

Les facteurs de succès du projet ne sont pas tout à fait l'inverse des facteurs de défaillance, mais comme le montre la Fig. 3. Le soutien de la direction et une bonne planification sont clairement considérés comme importants ici - plus le projet est grand et plus son calendrier est long, plus les chances d'échec sont grandes (rapport du groupe Standish).

2.3.6 Exigences et qualité

Les conséquences de ne pas avoir d'exigences sont nombreuses et variées. Il existe de nombreuses preuves de systèmes qui ont échoué parce que les exigences n'étaient pas correctement organisées. Même si le système peut sembler fonctionner, si ce n'est pas le système que les utilisateurs veulent ou ont besoin, il sera inutile.

Il est intéressant de considérer la relation entre les exigences et la qualité. Le terme « qualité » peut être compris de diverses manières. Lorsqu'on l'interroge sur les voitures de qualité, on peut citer Rolls Royce, Mercedes ou BMW. Cette confusion inhérente entre « qualité » et « luxe » est révélée si l'on considère le choix de la meilleure voiture pour le rallye annuel du RAC. Ni Rolls Royce, ni Mercedes ni BMW ne sont choisis, car ils ne présentent pas le bon rapport poids/puissance, la garde au sol et la robustesse. L'histoire récente montre que la meilleure voiture de sa catégorie est une Škoda - pas une voiture de luxe, mais la bonne voiture pour le travail (Elizabeth, Ken, & Jeremy, 2004).

La qualité est donc «l'aptitude à l'emploi» ou la conformité aux exigences - elle fournit quelque chose qui satisfait le client et, ce faisant, garantit la prise en compte des besoins de toutes les parties prenantes.

L'ingénierie des exigences agit comme un complément à d'autres considérations de gestion, telles que les coûts et le calendrier, en mettant l'accent sur la qualité de la prestation. Chaque décision de gestion est un compromis entre coût, échéancier et qualité, trois axes interdépendants.

Puisque l'ingénierie des exigences est une discipline qui s'applique dès le début du cycle de vie du développement, l'effet de levier sur la qualité qui peut être exercé par une bonne gestion des exigences est proportionnellement plus grand. Relativement peu d'efforts sont déployés dans les premiers stades de développement et peuvent porter leurs fruits dans les dernières étapes, en ce sens, le fait de le faire correctement dès le départ peut permettre d'économiser énormément d'efforts qui auraient été nécessaires pour réparer les choses plus tard. Améliorer les exigences signifie améliorer la qualité du produit.

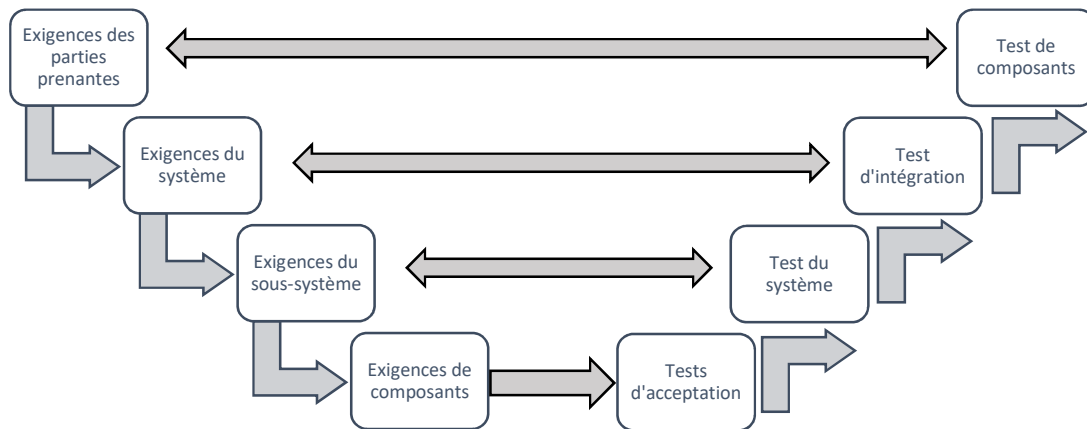


Fig. 4. Ingénierie des exigences en couches (Easterbrook, 2001).

2.3.7 Exigences et le cycle de vie de logiciel

Il y a une idée fausse commune que l'ingénierie des exigences est juste une phase unique qui est réalisée et complétée au début du développement du produit. L'ingénierie des exigences a un rôle essentiel à jouer à chaque étape du développement.

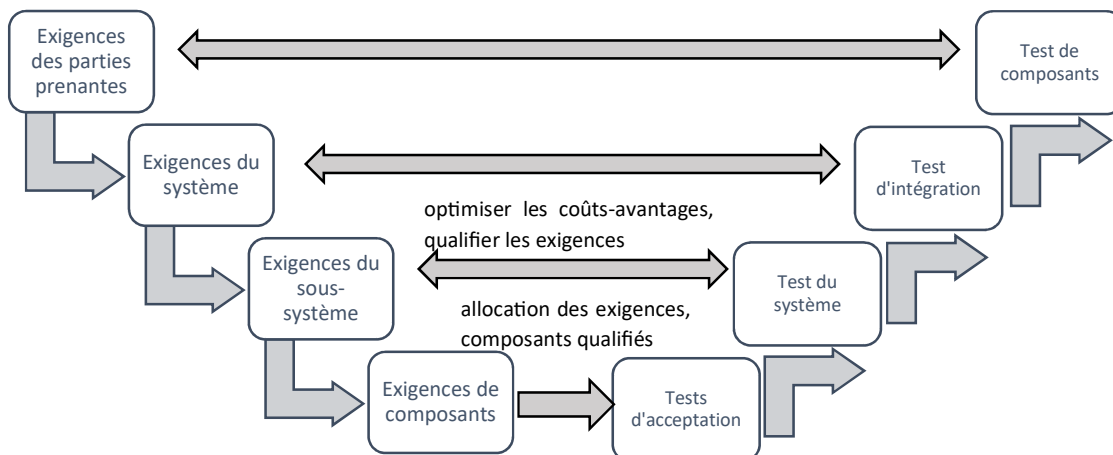


Fig. 5. Exigences dans le modèle en V (Easterbrook, 2001)

En guise d'approche initiale, pensez à l'une des toutes dernières activités du processus de développement: les tests d'acceptation. Ainsi, on peut voir immédiatement que les exigences développées au départ sont encore utilisées dans les dernières étapes du développement.

Le modèle V classique, qui est utilisé pour décrire les différentes étapes du développement a son fondement dans cette relation entre les tests et les exigences. La Fig. 4 montre cette relation à chaque étape du développement.

Le modèle V considère également le développement en termes de couches où chaque couche répond aux préoccupations propres au stade de développement correspondant. Bien que des processus légèrement différents puissent être utilisés à chaque niveau, le modèle de base d'utilisation des exigences est le même. La Fig. 5 montre la principale préoccupation de l'ingénierie des exigences à chaque couche.

Un autre rôle que les exigences peuvent jouer dans une organisation est d'agir comme un moyen de communication entre les projets. C'est une bonne idée, car de nombreuses organisations souhaitent:

- Maximiser la réutilisation des artefacts à travers les projets.
- Gérer les familles de produits similaires.
- Utiliser la gestion de programme pour coordonner les activités.
- Optimiser le processus en tirant parti des expériences d'autres projets.

Un bon ensemble d'exigences des parties prenantes peut fournir une description concise et non technique de ce qui est développé à un niveau accessible à la haute direction. De même, les exigences du système peuvent constituer un excellent résumé technique d'un projet de

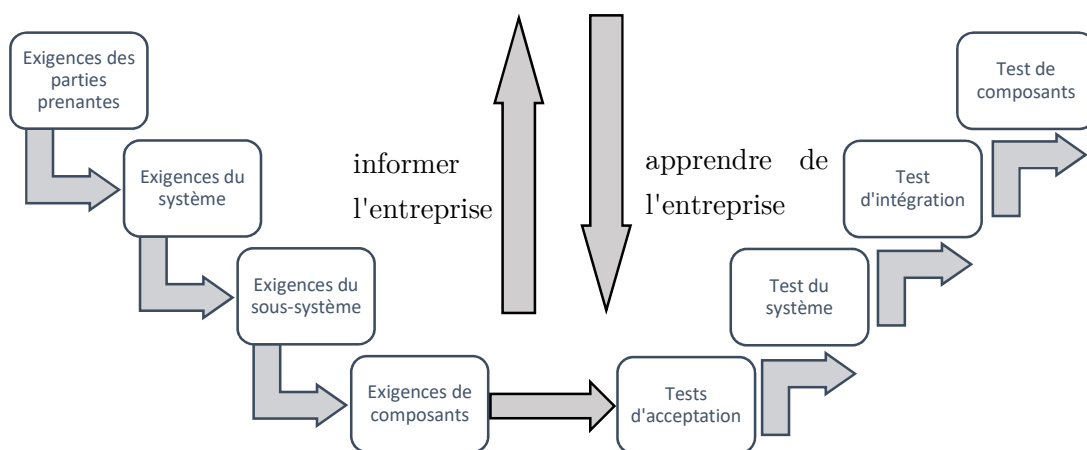


Fig. 6. Ingénierie des exigences de l'entreprise (Elizabeth, Ken, & Jeremy, 2004)

développement. Ces descriptions peuvent servir de base à la comparaison avec d'autres activités. Ceci est illustré dans la Fig. 6.

Si les exigences doivent jouer un rôle si central dans le développement des systèmes, elles doivent être maintenues. La modification de la conception d'un produit sans avoir également mis à jour les exigences pour refléter ce changement, va impliquer d'énormes problèmes pour les étapes ultérieures du développement. L'ingénierie des exigences est donc fortement liée à la gestion du changement.

Que le changement provienne d'un projet, par exemple, des problèmes techniques découlant des détails de la conception ou de l'extérieur, comme l'évolution des besoins des parties prenantes, l'impact de ce changement sur la qualité, les coûts et le calendrier doit être évalué. Cette évaluation constitue la base sur laquelle:

- accepter ou rejeter le changement (lorsque c'est un choix).
- négocier le coût du changement avec le client/les fournisseurs.
- organiser le travail de réaménagement.

Le concept clé permettant ce type d'analyse d'impact est la traçabilité des exigences. Il suffit de dire que la gestion des changements fait partie intégrante du processus d'ingénierie des exigences. Ce rôle est illustré à la Fig. 7.

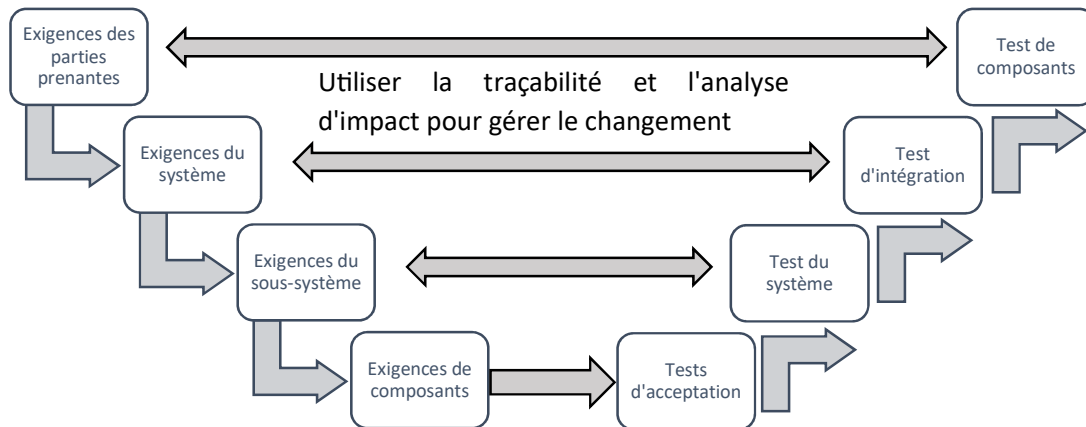


Fig. 7. Risque de traçabilité dans la gestion du changement (Dick, Hull, & Jackson, 2017)

Indépendamment de la gestion du changement, la capacité d'un gestionnaire à contrôler un projet est considérablement améliorée par une bonne ingénierie des exigences. Sans exigences, les gestionnaires de projet n'ont aucun moyen de savoir si le projet va bien, ou même s'il va dans la bonne direction. Quand il s'agit de changements, il n'y a rien contre lequel le changement puisse être jugé. Qui plus est, quand ils interviennent, leur seule approche est d'ordre technique,

inadaptée à leur rôle, et qui interfère avec le rôle technique joué par les ingénieurs. Les exigences bien exprimées au niveau approprié donnent aux gestionnaires la juste vision du projet pour être en mesure de remplir leur rôle.

2.3.8 Traçabilité des exigences

Dans le contexte de l'ingénierie des exigences, la traçabilité consiste à comprendre comment les exigences de haut niveau - objectifs, buts, attentes, besoins - sont transformées en exigences de bas niveau. Elle est donc principalement concerné par les relations entre les couches d'information. Le tableau ci-dessous montre les points d'intérêt de la traçabilité par rapport au plan commercial et contexte d'ingénierie (Gotel & Finkelstein, 1994).

Tableau 1. Points d'intérêt de la traçabilité

Contexte	Points d'intérêt
Commercial	<p>La vision d'entreprise <i>est interprété comme ...</i></p> <p>Les objectifs d'affaires <i>sont mis en œuvre comme ...</i></p> <p>L'organisation et le processus d'affaires.</p>
Ingénierie	<p>Les exigences des parties prenantes <i>sont satisfaites</i></p> <p>Les exigences système <i>sont partitionnées en ...</i></p> <p>Les sous-systèmes sont mis en œuvre comme ...</p> <p>Composants.</p>

L'utilisation de la traçabilité peut contribuer aux avantages suivants:

- Une plus grande confiance dans la réalisation des objectifs. L'établissement et la formalisation de la traçabilité engendrent une réflexion plus approfondie sur la manière dont les objectifs sont atteints.
- Capacité d'évaluer l'impact du changement. Différentes formes d'analyse d'impact deviennent possibles en présence d'informations de traçabilité.
- Amélioration de la responsabilité des organisations subordonnées. Une plus grande clarté de la façon dont les fournisseurs contribuent à l'ensemble.

- Capacité de suivre les progrès. Il est difficile de mesurer les progrès lorsque tout ce que vous faites est de créer et de réviser des documents. Les processus entourant la traçabilité permettent des mesures précises des progrès dans les premières étapes.
- Capacité à équilibrer les coûts et les avantages. Relier les composants du produit aux exigences permet d'évaluer les avantages par rapport aux coûts.

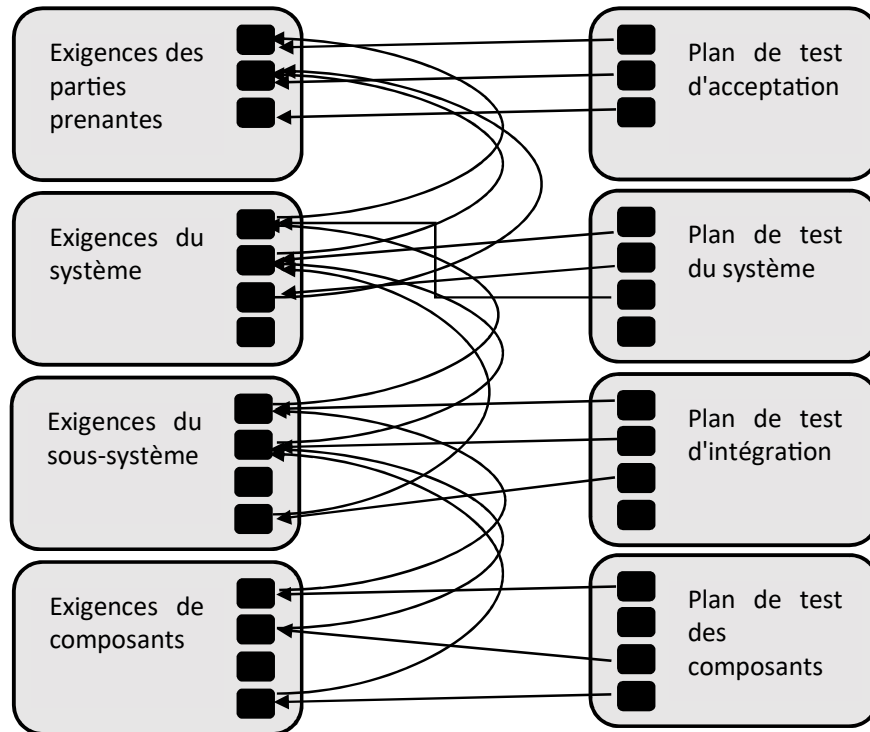


Fig. 8. Traçabilité des exigences (Gotel & Finkelstein, 1994).

Les relations de traçabilité sont généralement plusieurs à plusieurs, c'est-à-dire qu'une exigence de niveau inférieur peut être liée à plusieurs exigences de niveau supérieur et vice versa. La manière la plus simple de mettre en œuvre une forme de traçabilité consiste à lier des instructions d'exigences dans une couche avec des instructions dans une autre. Les outils de gestion des exigences permettent généralement une telle liaison par glisser-déposer entre les paragraphes de documents. Les liens sont plutôt comme des hyperliens dans les pages Web, mais devraient idéalement être traversables dans les deux sens. La Fig. 8 montre la traçabilité vers le bas à travers les couches d'exigences et à travers les informations de test. La direction des flèches suit une convention particulière: l'information revient à l'information à laquelle elle répond. Il y a plusieurs raisons à cette convention (Dick, Hull, & Jackson, 2017):

- Cela correspond généralement à l'ordre chronologique dans lequel les informations sont créées: toujours revenir à l'ancienne.
- Il correspond généralement aux droits d'accès en raison de la propriété: l'un possède les liens sortants d'un document, un autre possède les liens entrants.

Tableau 2. Types d'analyse de traçabilité (Dick, Hull, & Jackson, 2017).

Type d'analyse	La description	Processus pris en charge
Analyse d'impact	Après les liens entrants, en réponse à la question: « Et si cela devait changer? »	Gestion du changement
Analyse de dérivation	Suite à des liens sortants, en réponse à la question: « Pourquoi est-ce ici? »	L'analyse coûts-avantages
Analyse de couverture	Comptage des instructions qui ont des liens, dans répondez à la question: « Ai-je tout couvert? »	Ingénierie générale Rapports de gestion

Différentes formes d'analyse de traçabilité peuvent être utilisées pour prendre en charge les processus d'ingénierie des exigences, présentés dans le Tableau 2.

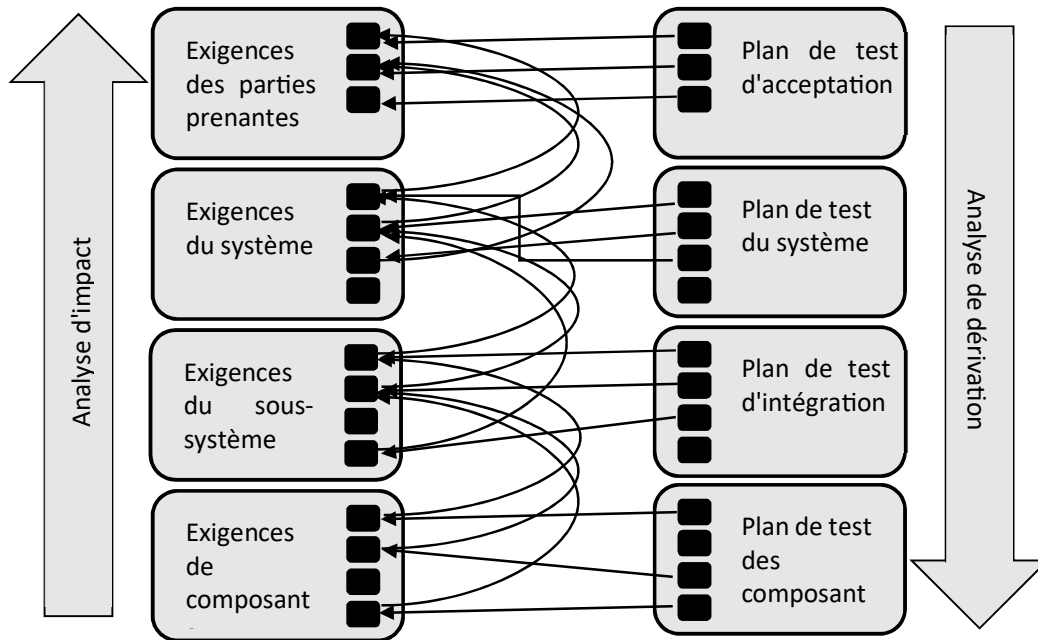


Fig. 9. Analyse d'impact et de dérivation (Elizabeth, Ken, & Jeremy, 2004).

L'analyse d'impact est utilisée pour déterminer quels autres artefacts dans le développement pourraient être affectés si un artefact sélectionné change. Ceci est illustré à la Fig. 9, l'impact est potentiel. L'analyse créative doit être effectuée, dans le cas échéant, par un ingénieur pour déterminer la nature exacte de l'impact.

L'analyse de dérivation fonctionne dans la direction opposée à l'analyse d'impact. Un artefact de faible niveau - comme une exigence, un élément de conception ou un test - est sélectionné et les liens de traçabilité sont utilisés pour déterminer quelles exigences de niveau supérieur lui ont donné lieu. Les éléments de la conception qui ne remontent pas à ce niveau ajoutent potentiellement des coûts sans bénéfice.

Enfin, l'analyse de la couverture peut être utilisée pour déterminer si toutes les exigences se retrouvent vers le bas, vers les couches inférieures et vers les tests. L'absence d'une telle trace est une indication assez certaine que l'exigence ne sera pas satisfaite ou testée. La présence d'un lien ne garantit pas, bien sûr, que l'exigence sera satisfaite ce qui nécessite à nouveau, un jugement d'ingénierie créatif.

La couverture peut également servir de mesure du progrès: jusqu'où les ingénieurs systèmes ont-ils répondu aux exigences des parties prenantes ? Supposons que la tâche d'écrire les exigences des systèmes en réponse aux exigences des parties prenantes soit donnée aux ingénieurs. Lorsqu'ils écrivent les exigences du système, ils les relient aux exigences des parties prenantes auxquelles ils répondent. En le faisant au fur et à mesure, la création de la traçabilité est très peu coûteuse, il est beaucoup plus difficile d'établir la traçabilité après que les deux documents ont été écrits.

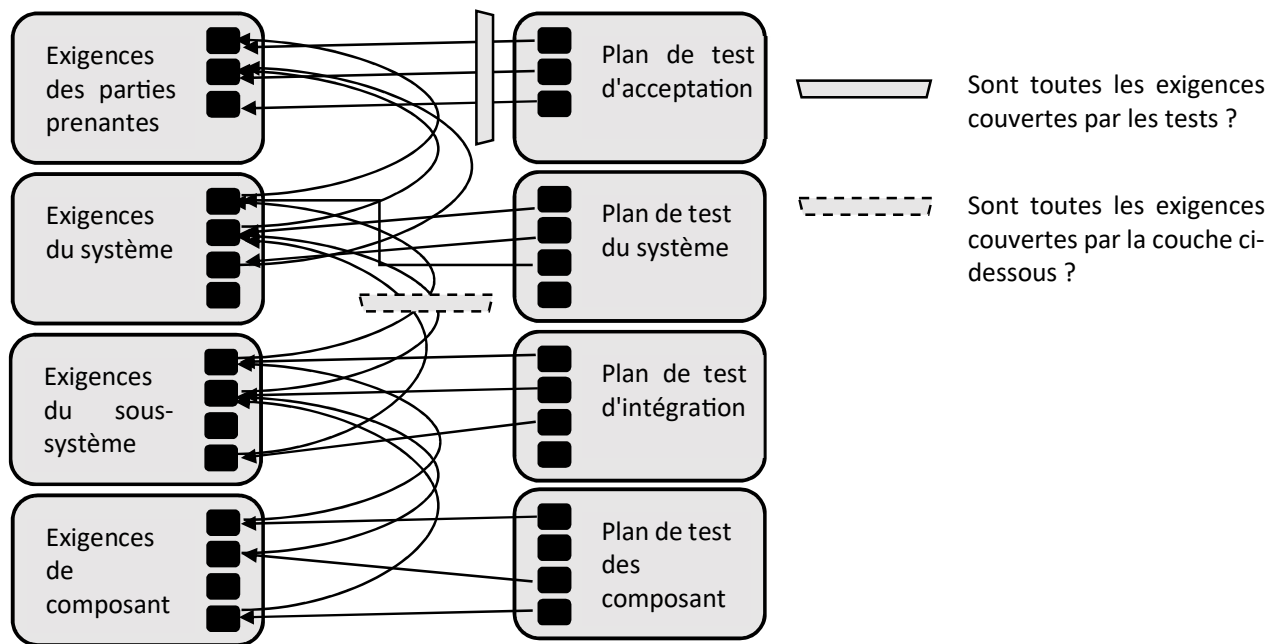


Fig. 10 Analyse de couverture (Dick, Hull, & Jackson, 2017)

À n'importe quel stade de la tâche, les progrès des ingénieurs peuvent être mesurés en termes de pourcentage des besoins des parties prenantes qui ont été couverts jusqu'à présent. C'est un outil de gestion très utile durant les premières étapes du développement. Le même principe peut être utilisé pour mesurer les progrès dans les tests de planification. Quel pourcentage des exigences ont des tests définis jusqu'à présent ? Ces deux dimensions de la couverture sont résumées à la Fig. 10. En raison des types d'analyses qui peuvent être effectuées, la traçabilité est un concept simple qui est au cœur du processus d'ingénierie des exigences.

2.3.9 Exigences et modélisation

Il est important de comprendre la relation entre la gestion des exigences et la modélisation du système. Ce sont des activités de soutien mutuel qui ne devraient pas être assimilées. Certaines personnes parlent de la modélisation des exigences. C'est un abus de langage (Elizabeth, Ken, & Jeremy, 2004). On modélise la conception du système, pas les exigences. La modélisation prend en charge l'activité de conception et constitue l'essentiel de la création. Il aide l'ingénieur à

comprendre suffisamment le système pour décomposer les exigences à un niveau particulier dans le niveau suivant. Les exigences elles-mêmes sont un aperçu complet de ce qui est requis à chaque niveau dans les niveaux de détail croissants.

Un modèle particulier ne dit jamais tout sur un système, si c'était le cas, ce ne serait pas un seul modèle. Pour cette raison, plusieurs modèles de systèmes différents, éventuellement interdépendants, sont souvent utilisés pour couvrir une variété d'aspects différents. Il est laissé à l'expression des exigences - généralement sous forme textuelle - pour couvrir les aspects non modélisés.

Un modèle est une abstraction d'un système qui se concentre délibérément sur certains aspects d'un système à l'exclusion des autres. L'abstraction est, dans ce sens, l'évitement de la distraction - en ignorant les détails qui, bien qu'importants, ne sont pas pertinents pour un modèle particulier. L'avantage est que de plus petites quantités d'informations connexes peuvent être collectées, traitées, organisées et analysées, en appliquant diverses techniques spécifiques pertinentes aux aspects étudiés (Elizabeth, Ken, & Jeremy, 2004).

Lorsqu'une grande quantité d'informations complexes doit être gérée, la modélisation fournit un moyen de zoomer, de rassembler des sous-ensembles de données dans un but particulier et d'effectuer un zoom arrière pour apprécier l'ensemble. Elle aide à maintenir une compréhension à l'échelle du système en se concentrant sur de petites quantités d'informations à la fois. Les exigences et la modélisation du système jouent des rôles interdépendants. Les modèles aident l'ingénieur des exigences à analyser les exigences à un niveau particulier afin de:

- communiquer avec le client et améliorer la compréhension mutuelle du système à développer.
- analyser le système pour déterminer la présence des propriétés émergentes souhaitées (et l'absence de propriétés indésirables).
- déterminer comment satisfaire les exigences en dérivant de nouvelles exigences dans la couche ci-dessous.

La nature des modèles utilisés variera d'une couche à l'autre. Dans la couche supérieure, des modèles d'utilisation tels que les « scénarios de parties prenantes » sont utilisés pour dériver la première déclaration des exigences des parties prenantes. Par la suite, différents types de modèles fonctionnels peuvent être utilisés pour dériver les exigences du système des exigences des parties prenantes. Pour les logiciels, ces modèles peuvent inclure des diagrammes de classes UML, des diagrammes de séquences de messages et des diagrammes d'états.

Passant des exigences du système à l'architecture, les préoccupations se concentrent sur divers aspects de la performance. Plusieurs modèles peuvent être utilisés pour donner l'assurance que l'architecture sélectionnée peut répondre à la fois aux exigences fonctionnelles et non fonctionnelles. Ici, les modèles peuvent inclure la théorie de la file d'attente utilisée pour évaluer la performance, les souffleries pour évaluer l'aérodynamique et la modélisation des horaires pour évaluer la viabilité des temps de parcours.

Comme il ressort de ces exemples, la nature des modèles varie également d'une application à l'autre. La modélisation des horaires peut être appropriée pour la conception des systèmes ferroviaires, mais pas pour la conception des avions, où la modélisation de l'aérodynamique est plutôt plus appropriée (l'aérodynamique peut aussi être importante pour les trains à grande vitesse). Elle peut être utilisée dans les systèmes de communication, mais les applications riches en données trouveront plus appropriée la modélisation centrée sur les données telle que le diagramme d'entité-relation. Alors que les modèles peuvent varier, les principes de gestion des exigences restent génériques pour toutes les applications (Elizabeth, Ken, & Jeremy, 2004).

2.3.10 Exigences et tests

Comme cela a été discuté ci-dessus, les tests sont étroitement liés aux exigences à tous les niveaux. Dans son sens le plus large, le test est une activité qui permet de détecter ou d'éviter les défauts du système, lorsqu'un défaut est un écart par rapport aux exigences, examens, inspections, analyses par modélisation en plus des tests classiques des composants, sous-systèmes et systèmes réalisés. En raison de la diversité des activités de test, le terme qualification est utilisé pour désigner toutes ces activités (Sommerville, 2010).

La qualification devrait commencer le plus tôt possible, car attendre que le système soit presque terminé avant d'effectuer tout type de test peut entraîner des modifications et des reconstructions très coûteuses. Les premiers types d'actions de qualification ont lieu au cours de la conception du système et comprennent des révisions des exigences, des inspections de conception et diverses formes d'analyse effectuées sur les modèles de système.

La Fig. 11 présente la stratégie de qualification le long des étapes de conception du système. Les actions de qualification précoces se rapportent à côté du modèle.

Une exigence d'un seul intervenant donnera généralement lieu à une multitude d'activités de qualification à divers stades de développement. Lorsqu'une exigence est satisfaite par des propriétés émergentes utiles, la qualification des composants seule est insuffisante; les tests doivent être effectués au niveau où les propriétés émergentes sont manifestes.

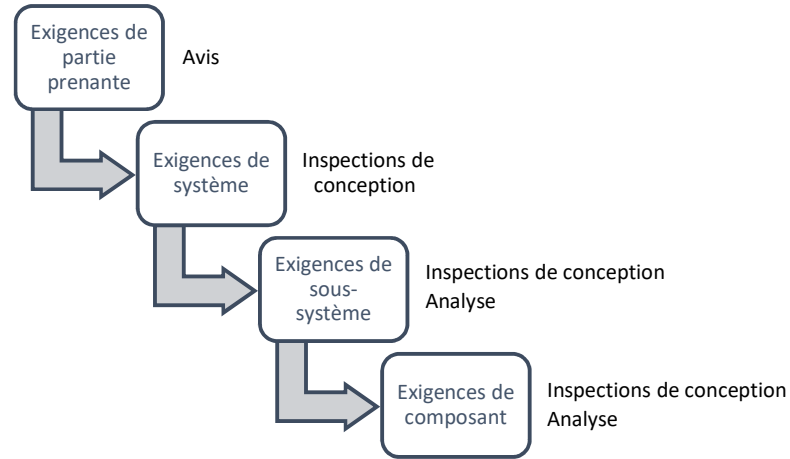


Fig. 11 Stratégie de qualification (Elizabeth, Ken, & Jeremy, 2004)

2.3.11 Exigences dans les domaines du problème et de la solution

L'ingénierie des systèmes est concernée par le développement et la gestion de solutions efficaces aux problèmes. Comme cela a été discuté, il s'agit d'un processus par étapes, vital pour les entreprises, leur permettant de produire le bon produit dans des délais et des coûts acceptables.

Au début du processus, la définition des exigences pour le produit à construire est d'une importance primordiale. Du point de vue de la gestion et de l'ingénierie, une distinction claire doit être faite entre le « domaine du problème » et le « domaine de la solution ». Les stades de développement associés aux plus hauts niveaux de description du système - énoncé des besoins, modélisation de l'utilisation et exigences des parties prenantes - doivent être fermement ancrés dans le domaine du problème, tandis que les couches suivantes, en commençant par les exigences du système, fonctionnent dans le domaine de la solution. Le Tableau 3 illustre la limite idéale entre les domaines de problème et de solution et les rôles joués par les couches de besoins supérieures.

Tableau 3 Problèmes et espaces de solution (Elizabeth, Ken, & Jeremy, 2004)

Couche des exigences	Domaine	Vue	Rôle
Exigences des parties prenantes	Domaine du problème	Point de vue des parties prenantes	Indiquez ce que les parties prenantes veulent réaliser en utilisant le système. Évitez toute référence à une solution particulière
Configuration requise	Domaine de la solution	Opinion de l'analyste	Indiquer abstraitement comment le système répondra aux exigences des parties prenantes. Évitez toute référence à un design particulier
Conception architecturale	Domaine de la solution	Le point de vue du designer	Indiquer comment la conception spécifique répondra aux exigences du système

Il y a un principe important d'abstraction en jeu ici. L'énoncé initial des capacités ne doit pas indiquer plus que nécessaire pour définir le problème et éviter toute référence à des solutions particulières. Cela donne la liberté aux ingénieurs systèmes de remplir leur rôle, qui est de concevoir la meilleure solution sans idées préconçues.

La modélisation aide à la dérivation de la prochaine couche d'exigences et tend à envisager des solutions possibles, même à un niveau élevé. Pour éviter les biais de solution inappropriés, plutôt que de se concentrer sur le système en question, la modélisation précoce devrait se concentrer sur le système englobant immédiatement. Par exemple, si un système radio est développé pour un voilier, alors la modélisation précoce devrait se concentrer sur le vaisseau et pas tellement sur la radio. Cela conduit à une déclaration du problème à résoudre dans le contexte de la solution englobante.

Le même principe s'applique aux ingénieurs systèmes: ils doivent laisser aux concepteurs la liberté de jouer leur rôle, celui de concevoir une solution abstraite. Les éléments de solution introduits par la modélisation fonctionnelle restent à un niveau élevé, laissant le détail à définir dans les étapes ultérieures.

Par exemple, dans un système de contrôle du trafic:

- Les parties prenantes peuvent exprimer le problème en termes de maximisation de la circulation tout en minimisant le risque d'accidents à un carrefour et en minimisant les coûts de maintenance.
- Les ingénieurs systèmes peuvent considérer diverses solutions, telles que les feux de circulation ou les carrefours giratoires, et un pont comme l'approche qui résout le mieux le problème en tenant compte des contraintes de développement et de coûts de maintenance.
- Les concepteurs se sont alors mis à concevoir le pont dans les limites des contraintes physiques présentées par l'environnement physique.

Il arrive fréquemment que les parties prenantes expriment le problème en termes de solution préconçue. Il devient alors le travail des ingénieurs des exigences de déterminer s'il y a une bonne raison pour mandater une solution particulière ou si c'est une contrainte inutile. Par exemple, le client commence par essayer d'obtenir des feux de circulation; le fournisseur pose des questions qui permettent de comprendre les objectifs sous-jacents - optimiser le flux de trafic et minimiser les risques pour les conducteurs et les piétons - ce qui permet d'exprimer le problème indépendamment de la solution; les raisons du choix de la solution sont maintenant mieux comprises et peut-être confirmées par une modélisation appropriée, conduisant à une spécification précise et bien informée de la solution abstraite.

Lorsqu'il s'agit de se procurer des systèmes, il faut faire un jugement quant à savoir s'il faut se procurer contre le domaine du problème (exigences des intervenants) ou contre le domaine de la solution abstraite (exigences du système). Souvent, la nature de la solution est connue à l'avance et il est logique de se prémunir contre les exigences du système encadrées en termes de cette solution. Cependant, même si l'acquisition se fait contre une solution particulière, la discipline de capturer une déclaration du problème pur avant une solution offre encore des avantages importants (Dick, Hull, & Jackson, 2017).

Sans une distinction claire entre le problème et la solution, ce qui suit peut résulter (Dick, Hull, & Jackson, 2017):

- Le manque de compréhension du vrai problème.
- L'incapacité de définir le système et de comprendre les fonctions à inclure.
- La domination du débat sur le système par les développeurs et les fournisseurs, car les seules descriptions du système sont exprimées en termes de solutions.
- L'incapacité de trouver des solutions optimales en raison du manque de liberté de conception.

2.4 Modélisation du système pour l'ingénierie des exigences

La modélisation du système soutient le processus d'analyse et de conception en introduisant un certain degré de formalité dans la façon dont les systèmes sont définis. Au cours du développement du système, il arrive souvent que des images soient utilisées pour aider à visualiser certains aspects du développement. La modélisation fournit un moyen de formaliser ces représentations, à travers des diagrammes, non seulement en définissant une syntaxe standard, mais aussi en fournissant un moyen de comprendre et de communiquer les idées associées au développement du système.

L'art de la modélisation est sans doute l'aspect le plus créatif du travail de l'ingénieur système. Il n'y a pas de « bonne » solution et les modèles évolueront à travers les différentes étapes du développement du système. Les modèles sont le plus souvent représentés visuellement et l'information est donc représentée par des diagrammes connectés. De nouvelles méthodes telles que l'orienté objet ont fait progresser le concept de modélisation; Cependant, la plupart des approches sont également basées sur les principes utilisés et testés au fil du temps.

Un bon modèle est celui qui est facilement communiqué. Ils doivent être utilisés pour la communication au sein d'une équipe de développement, mais aussi pour une organisation dans son ensemble, y compris les parties prenantes. Les utilisations d'un modèle peuvent être diverses et couvrir un large spectre. Il peut s'agir de modéliser les activités d'une organisation entière ou de modéliser une exigence fonctionnelle spécifique d'un système. La modélisation présente les avantages suivants (Shaw & Garlan, 1996):

- Encourage l'utilisation d'un vocabulaire défini avec précision dans l'ensemble du système.
- Permet de visualiser les spécifications et la conception du système dans des diagrammes.
- Permet l'examen de plusieurs aspects et vues en interaction d'un système.
- Supporte l'analyse des systèmes à travers une discipline définie.

- Permet la validation de certains aspects de la conception du système.
- Permet un affinement progressif vers la conception détaillée, permettant la génération de cas de test et la génération de code.
- Encourage la communication entre les différentes organisations en utilisant des notations standard communes.

Une grande partie de la créativité et de l'art de l'ingénieur système est exprimée dans l'utilisation de techniques de modélisation.

2.5 Spécification et révision des exigences

L'ingénierie des exigences est un processus technique. L'écriture des exigences n'est donc pas comme les autres types d'écriture. Ce n'est certainement pas comme écrire un roman ou un livre, ce n'est même pas comme le genre de « rédaction technique » vu dans les manuels d'instructions et les guides de l'utilisateur.

Le but est de présenter les aspects des exigences d'écriture communs à chaque couche de développement. Partout où le processus générique est instancié, certains principes et techniques sont constants dans leur application à l'expression et à la structuration des exigences.

En écrivant un document d'exigences, deux aspects doivent être soigneusement équilibrés (Dick, Hull, & Jackson, 2017) :

- la nécessité de rendre le document d'exigences lisible;
- la nécessité de traiter l'ensemble des exigences.

Le premier concerne la structure du document, comment il est organisé et comment son déroulement aide l'examineur à replacer les énoncés d'exigences individuels dans leur contexte. La seconde se concentre sur les qualités des énoncés de besoins individuels, le langage utilisé pour promouvoir la clarté et la précision et comment ils sont divisés en éléments uniques traçables.

L'ingénieur des exigences expérimenté se rend compte qu'un traitement de texte seul n'est pas suffisant pour gérer un ensemble d'exigences, pour que les déclarations individuelles doivent être identifiés, classés et tracés. Un problème classique, par exemple, est l'utilisation de numéros de paragraphe pour identifier les exigences: insérer un nouveau au milieu, et soudainement tous les identifiants d'exigence suivants ont changé.

De même, ceux qui ont simplement essayé de gérer leurs besoins dans une base de données se rendent rapidement compte que les tables pleines de déclarations individuelles sont ingérables.

Malgré la possibilité d'identifier, classer et trier les exigences, les informations contextuelles vitales fournies par le document ont été perdues; les déclarations individuelles perdent leur sens lorsqu'elles sont séparées de leur place dans le tout (Dick, Hull, & Jackson, 2017).

Donc les deux aspects - document et individualité - doivent être maintenus.

La rédaction et l'examen des exigences (ou de tout autre type de document, d'ailleurs) devraient aller de pair, en ce sens que les critères d'écriture d'une bonne exigence sont exactement ceux sur lesquels l'exigence doit être examinée.

2.5.1 Exigences pour les exigences

Avant de discuter de la façon dont les documents d'exigences et les énoncés doivent être rédigés, il est préférable de passer en revue certains des objectifs et du but de la rédaction des exigences en premier lieu. Cela aidera à comprendre pourquoi certains principes sont suggérés. Le point de départ est l'identification des parties prenantes.

Le Tableau 4 énumère les capacités requises par les différentes parties prenantes qui se rapportent à la façon dont les documents d'exigences et les énoncés sont écrits. Ce sont les choses de base que l'on doit être en mesure de faire pour - et avec - les exigences, y compris l'identification, la classification, l'élaboration, le suivi de l'état, le traçage, la mise en contexte et la récupération. Les exigences exprimées et organisées ont une grande influence sur la « faisabilité » des ensembles d'exigences.

Tableau 4 Capacités requises pour les besoins (Dick, Hull, & Jackson, 2017)

Capacité
Capacité unique d'identifier chaque énoncé de besoin
Possibilité de classer chaque énoncé de besoin de plusieurs manières, telles que: <ul style="list-style-type: none"> ➤ Par importance ➤ Par type (par exemple fonctionnel, performance, contrainte, sécurité) ➤ Par urgence (quand il faut le fournir)
Possibilité de suivre le statut de chaque énoncé des besoins, en prenant en charge plusieurs processus, tels que: <ul style="list-style-type: none"> ➤ Statut de révision ➤ État de satisfaction ➤ Statut de qualification

<p>Capacité à élaborer une exigence de plusieurs manières, par exemple en fournissant:</p> <ul style="list-style-type: none"> ➤ Information de performance ➤ Quantification ➤ Critères de test ➤ Raisonnement ➤ Commentaires
<p>Possibilité de visualiser un énoncé de besoin dans le contexte du document, c'est-à-dire parallèlement à ses énoncés environnants</p>
<p>Capacité à naviguer dans un document d'exigences pour trouver des exigences en fonction d'une classification ou d'un contexte particulier</p>
<p>Capacité de remonter à n'importe quel énoncé de besoin individuel</p>

2.5.2 Documents de besoins structurants

La documentation sur les exigences peut être très volumineuse. Sur le papier, les exigences complètes du sous-système pour un porte-avions, par exemple, peuvent remplir plusieurs classeurs. Il n'est pas inconnu que les réponses des fournisseurs aux grands systèmes soient livrées dans les camions. Dans de telles situations, disposer d'une structure bien comprise et clairement documentée pour l'ensemble des exigences est essentiel à la gestion efficace de la complexité. L'organisation des exigences dans la bonne structure peut aider à (Pohl, 2010):

- minimiser le nombre d'exigences;
- comprendre de grandes quantités d'informations;
- trouver des ensembles d'exigences relatives à des sujets particuliers;
- détecter les omissions et les duplications;
- éliminer les conflits entre les exigences;
- rejeter les mauvaises exigences;
- évaluer les exigences;
- réutiliser les exigences pour tous les projets.

Les documents sont généralement hiérarchiques, avec des sections et des sous-sections à plusieurs niveaux. Les hiérarchies sont des structures utiles pour la classification, et une façon de structurer un document d'exigences consiste à utiliser la structure d'en-tête de section pour catégoriser les instructions d'exigences. Dans un tel régime, la position qu'une déclaration d'exigence a dans le

document représente sa classification primaire. Des classifications secondaires peuvent être données via des liens vers d'autres sections ou en utilisant des attributs (Pohl, 2010).

Les modèles de système utilisent fréquemment des hiérarchies dans l'analyse d'un système. Les exemples sont:

- La décomposition de l'objectif ou de la capacité comme dans les scénarios d'intervenants.
- La décomposition fonctionnelle comme dans les diagrammes de flux de données.
- La décomposition de l'état comme dans les diagrammes d'états.

Lorsque les exigences sont dérivées de ces modèles, l'une des hiérarchies résultantes peut être utilisée dans le cadre de la structure d'en-tête du document d'exigences.

En plus des énoncés d'exigences eux-mêmes, les documents d'exigences peuvent contenir une variété de textes techniques et non techniques, ce qui favorise la compréhension des exigences. Ceux-ci peuvent être des choses telles que:

- Les informations de base qui placent les exigences dans leur contexte.
- Le contexte externe décrivant le système englobant, souvent appelé « connaissance du domaine ».
- La définition de la portée des exigences (ce qui est dans et ce qui est exclu).
- Les définitions des termes utilisés dans les énoncés d'exigences.
- Un texte descriptif qui relie différentes sections du document.
- Les descriptions des parties prenantes.
- Résumé des modèles utilisés pour établir les exigences.
- Des références à d'autres documents.

2.5.3 Exigences clés

De nombreuses organisations utilisent le concept des «exigences clés», en particulier au niveau des parties prenantes. Souvent appelés EUC (Exigence Utilisateur Clé) ou IPC (Indicateur de Performance Clé). Ces exigences sont un petit sous-ensemble extrait de l'ensemble qui capture l'essentiel du système. La philosophie qui guide la sélection des exigences clés est : « Nous ne devons pas penser aux choses que nous pourrions faire avec, mais seulement les choses que nous ne pouvons pas faire sans ». Chaque exigence clé devrait solliciter une réponse négative à la

question : Si la solution ne me fournissait pas cette capacité, l'achèterais-je encore ? Ou, si au niveau du système : Si le système ne le faisait pas, le voudrais-je encore ? (Elizabeth, Ken, & Jeremy, 2004)

De cette manière, les exigences clés deviennent celles qui sont absolument obligatoires. Bien sûr, tout est négociable, mais la négociation des exigences clés engendrerait toujours un examen très attentif. Le cas échéant, chaque exigence clé doit être quantifiée avec les attributs de performance. Cela permet de les utiliser comme IPC, d'évaluer les propositions alternatives par rapport aux exigences ou de les utiliser comme un résumé des statistiques de l'état civil sur l'avancement du projet.

2.5.4 Utilisation des attributs

Un simple énoncé textuel n'est pas suffisant pour définir une exigence; il existe d'autres informations de classification et d'état que chaque exigence porte.



Fig. 12 Attributs des exigences (Elizabeth, Ken, & Jeremy, 2004)

Plutôt que d'encombrer le texte d'une exigence, des informations supplémentaires doivent être placées dans les «attributs» attachés à l'exigence. Les attributs permettent de structurer les informations associées à une seule exigence pour faciliter le traitement, le filtrage, le tri, etc. permettant de trier ou de sélectionner les exigences pour une action ultérieure, et de contrôler le processus de développement des exigences lui-même. La Fig. 12 montre un exemple d'exigence avec un certain nombre d'attributs.

Les attributs particuliers utilisés dépendront des processus exacts qui doivent être pris en charge. Certains attributs sont entièrement automatiques, par ex. dates et numéros, certains proviennent d'utilisateurs, par ex. priorité, et d'autres attributs sont des indicateurs, qui sont définis après le travail d'analyse, par ex. vérifiabilité.

2.6 La langue des exigences

L'utilisation d'un langage cohérent facilite l'identification de différents types d'exigences. Un simple exemple de ceci est l'utilisation de « **doit** » comme mot clé pour indiquer la présence d'une exigence dans le texte. Certaines approches vont jusqu'à utiliser « **doit** », « **devrait** » et « **peut** » pour indiquer différentes priorités d'exigence.

La langue utilisée variera en fonction du niveau d'exigence exprimé. La principale différence est entre les exigences des parties prenantes qui se situent dans le domaine du problème et les exigences du système qui se trouvent dans le domaine de la solution (Ryan, 1993).

Les exigences des parties prenantes concernent principalement la capacité et les contraintes de capacité. Une déclaration de capacité doit exprimer une capacité (unique) requise par un ou plusieurs types d'intervenants identifiés (ou groupes d'utilisateurs). dans le texte de l'exigence. Une exigence de capacité typique prend la forme suivante:

<type de partie prenante> **doit être en mesure de** <capacité>.

Lorsque certains aspects de la performance ou de la contrainte sont associés uniquement à l'exigence, ils peuvent également être indiqués dans le texte, par exemple en donnant le formulaire:

<type de partie prenante> **doit être en mesure de** <capacité> **dans** <performance> **de** <event> **tandis que** <condition opérationnelle>.

Par exemple, l'exigence de capacité suivante a une performance et une contrainte attachées:

- L'opérateur d'armes doit être en mesure de tirer un missile dans les 4 secondes suivant l'observation radar dans des conditions climatiques difficiles.

Moins souvent, un seul attribut de performance est associé à plusieurs capacités. Par exemple, plusieurs capacités peuvent devoir être fournies avec un temps défini. En pratique, ces capacités sont généralement des subdivisions d'une capacité de haut niveau auxquelles l'attribut de performance doit être attaché.

Cependant, il arrive souvent que les contraintes doivent être exprimées séparément des capacités, soit parce qu'elles s'appliquent à l'ensemble du système, soit parce qu'elles s'appliquent à des

capacités diverses. En règle générale, les contraintes imposées aux parties prenantes sont fondées soit sur des performances minimales acceptables, soit sur la nécessité d'interagir avec des systèmes externes (y compris les systèmes juridiques et sociaux). Une exigence de contrainte typique prend la forme suivante (Elizabeth, Ken, & Jeremy, 2004):

<partie prenante> **ne doit pas** <être enfreinte à la loi applicable>.

- Le conducteur de l'ambulance ne doit pas être enfreint aux règlements de la route nationale.

Comme ils se situent dans le domaine de la solution, le langage des exigences du système est légèrement différent. Ici, l'accent est mis sur la fonction et les contraintes du système. La langue dépend des types de contraintes ou de performances associées à l'exigence. Voici un exemple de fonction avec une performance de capacité:

Le <système> **doit** <fonction> **pas moins que** <quantité> <objet> **tandis que** <condition opérationnelle>.

- Le système de communication doit maintenir le contact téléphonique avec pas moins de 10 appelants en l'absence d'alimentation externe.

En voici un autre qui exprime une contrainte de périodicité :

Le <système> **doit** <fonction> <objet> **chaque** <performance> <unités>.

- La machine à café doit produire une boisson chaude toutes les 10 secondes.

2.7 Conclusion

Les exigences sont essentielles au développement de système. Ils influencent l'ensemble du développement du début à la fin et du haut en bas. Sans ingénierie des exigences efficace, les projets de développement de grand ampleur sont comme des navires dans une tempête ! Par-dessus tout, avec une bonne gestion des exigences, entendre la voix des utilisateurs et des clients cesse d'être un jeu de murmures chinois et devient une question de lignes de communication claires tout au long du processus de développement. Dans le chapitre suivant, nous allons présenter les systèmes auto-adaptatifs ainsi que des travaux dans le domaine d'ingénierie des exigences pour les SAAs.

Chapitre 3

La spécification des exigences pour les systèmes auto-adaptatifs

3.1 Introduction

La tendance de l'évolution et de la maintenance des logiciels évolue vers un support de l'évolution dynamique et de l'exécution (Kephart & Chess, 2003). La modification fréquente des besoins des utilisateurs ou la forte variabilité de la quantité de ressources disponibles ne sont que deux exemples de scénarios dans lesquels un système moderne (ex : système d'information) doit être capable de se reconfigurer et de s'auto-optimiser de façon continue au moment de l'exécution. La communauté des chercheurs a fait de grands progrès dans la complexité de la construction de systèmes logiciels auto-adaptatifs. Cependant, il existe peu de techniques applicables, au niveau de spécification des exigences, pour gérer les concepts liés à l'auto-adaptation tel que l'incertitude (De Lemos R. , et al., 2013). Les chercheurs ont commencé à s'attaquer à ce dernier. Dans ce qui suit, nous présentons les systèmes auto-adaptatifs et leur architecture, technologies et cadre (Framework). Nous présentons aussi, un ensemble des approches sur IE pour les systèmes auto-adaptatifs ainsi que d'autre liées au domaine de patrons de spécification.

3.2 Les Systèmes auto-adaptatifs

Le terme « système auto-adaptatif » ou SAA a été introduit par l'Agence des projets de recherche avancée de la Défense (APRAD) comme un système capable d'évaluer son propre comportement et de changer ce comportement lorsque son évaluation indique qu'il ne réalise pas ce qu'il est destiné à faire, ou quand une meilleure fonctionnalité ou performance est possible (Laddaga, 2000). Une autre définition est donnée par Oreizy et al. (Oreizy, et al., 1999): « Un système auto-adaptatif modifie son propre comportement en réponse aux changements dans son environnement d'exploitation. Par environnement d'exploitation, nous entendons tout ce qui peut être observé par le système, tel que l'entrée de l'utilisateur final, les dispositifs matériels externes et les capteurs, ou l'instrumentation du programme ».

D'après les définitions ci-dessus, on peut conclure qu'un système auto-adaptatif devrait avoir les caractéristiques suivantes:

- La capacité d'observer les changements dans son environnement d'exécution.

- La capacité de détecter et de diagnostiquer les transitions d'état de l'environnement d'exécution et d'évaluer son propre comportement.
- La capacité de modifier son propre comportement afin de s'adapter aux nouveaux changements.
- Le support du comportement dynamique. Son comportement interne et/ou externe devrait pouvoir changer intentionnellement et automatiquement.

3.2.1 Perspectives architecturales

Le système auto-adaptatif (SAA) doit être auto-conscient en ce sens qu'il connaît au moins un sous-ensemble de ses propres éléments structurels et comportementaux. C'est une propriété essentielle d'un SAA, car les modifications de l'état du système doivent être observées par un contrôleur capable de répondre aux changements dans l'environnement d'exécution du système.

La réflexion computationnelle, telle que définie par Maes (Maes, Concepts and experiments in computational reflection, 1987), est une solution pour atteindre la propriété d'auto-conscience (Weyns, 2010). Un système de réflexion est relié de manière causale à une représentation de lui-même. Si le système change, son auto-représentation change en conséquence. Inversement, les changements dans l'auto-représentation entraînent des changements au système sous-jacent (réflexion).

Dans une architecture réflexive, la partie du système qui spécifie la logique métier et interagit avec le domaine d'application s'appelle la couche de base. Il est relié de manière causale à une méta-couche, qui spécifie l'aspect d'auto-représentation du système. Avoir une méta-couche auto-représentative permet au système de raisonner sur sa propre structure et son propre comportement afin de prendre les mesures nécessaires pour son ajustement de comportement.

Un aspect important de l'adaptation est la granularité du changement. Ceci est souvent discuté en termes de deux catégories générales de changements à grains grossiers et à grains fins. Un nombre remarquable d'études bien connues dans la littérature abordent les changements à granularité grossière tels que les changements architecturaux, par exemple (Garlan, Cheng, Huang, Schmerl, & Steenkiste, 2004) (Oreizy, Medvidovic, & Taylor, 1998). Bien que ces changements puissent être très efficaces, le problème est qu'ils sont difficiles à implémenter, à tester et à gérer dans des systèmes distribués complexes à grande échelle. Ce problème est beaucoup plus difficile dans des conditions imprévues. L'adaptation à grain fin a été de plus en plus utilisée, en raison de son coût plus faible, et également en raison de l'avènement des technologies de support (par exemple, la composition d'aspect dynamique). Les modifications

d'adaptation des paramètres sont fines, mais certaines actions de gestion des ressources et la composition des aspects dynamiques entrent également dans cette catégorie.

Les approches d'adaptation diffèrent également dans la façon dont la couche de base et la méta-couche sont spécifiées et communiquent entre elles et avec d'autres parties du système. Fondamentalement, ces approches peuvent être divisées en deux catégories principales: (i) les approches internes, qui traitent et réalisent les besoins d'adaptation comme des propriétés fonctionnelles de premier niveau, et (ii) les approches externes qui séparent les préoccupations d'adaptation des autres préoccupations fonctionnelles du système. Chaque approche a ses propres avantages et inconvénients. Cette section présente les deux approches et présente leurs avantages et leurs inconvénients.

a) Approches internes

Les approches internes entremêlent les spécifications d'application et d'adaptation. Ces approches sont principalement reconnues comme des mécanismes basés sur la programmation pour les adaptations de bas niveau. À l'appui des approches internes, il existe un point de vue des applications adaptatives basées sur des principes de programmation adaptative comme une extension de la programmation orientée objet (Lieberherr, 1994): « Un programme doit être conçu de telle sorte que la représentation d'un objet puisse être modifiée dans certaines limites sans affecter le programme du tout ». Selon ce point de vue, un programme adaptatif sera: « Un modèle de processus générique paramétré par des contraintes de graphes qui définissent des modèles structurels compatibles (customizers) comme des paramètres du modèle de processus. » Ce point de vue est similaire à la réflexion et à la méta-programmation. Les principaux avantages des approches internes sont: (i) la testabilité, due au comportement attendu d'un logiciel adaptatif, (ii) l'applicabilité des méthodes formelles et la vérification du modèle pour augmenter la fiabilité et la robustesse du système, et (iii) une meilleure performance en raison du soutien linguistique de bas niveau.

Les principaux inconvénients des approches internes peuvent être résumés comme suit: (i) soutien limité à l'adaptation dynamique, (ii) manque d'évolutivité pour les systèmes complexes et de grande envergure, (iii) réingénierie coûteuse des exigences d'adaptation et (iv) langages de programmation et outils de développement.

Un autre ensemble intéressant d'approches est celui qui est considéré comme interne au niveau de la conception, mais peut être réalisé comme interne ou externe au niveau du code (Cazzola, 2004). Cependant, en raison des inconvénients mentionnés des approches internes, une plus grande attention est accordée aux approches externes dans la pratique.

b) Approches externes

Les approches externes utilisent un sous-système externe pour contrôler l'adaptation du système logiciel, similaire aux systèmes classiques de contrôle par rétroaction (Brun, 2009), dans lesquels un système peut soit suivre une architecture en boucle ouverte, soit en boucle fermée. Un système en boucle ouverte (c'est-à-dire, en aval) a un modèle précis de son domaine et ajuste la sortie de contrôle en fonction des perturbations de l'environnement (Ogata & Yang, 2002). Le contrôle par rétroaction et ses variations, y compris le contrôle adaptatif, inspirent les concepteurs à utiliser l'architecture en boucle fermée dans les systèmes logiciels. Les principaux avantages des approches externes sont: (i) séparation des exigences fonctionnelles, (ii) séparation de la logique d'adaptation et de la logique applicative, (iii) augmentation de la réutilisabilité, évolutivité et maintenabilité, (iv) support de l'adaptation dynamique, et (v) l'applicabilité de l'intelligence artificielle et des techniques de la théorie du contrôle.

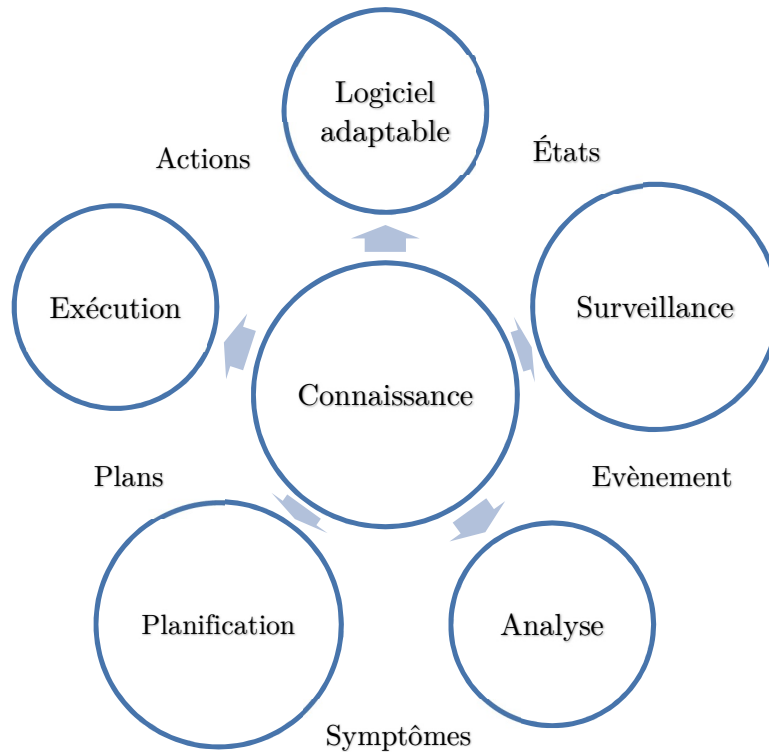


Fig. 13 Processus d'auto-adaptation (Kephart & Chess, 2003)

Le gestionnaire d'adaptation, en tant que contrôleur, exécute en continu quatre processus sous forme de boucle fermée: surveillance, analyse, planification et exécution, comme illustré à la Fig. 13. L'exécution continue du gestionnaire forme une boucle de contrôle, connue sous le nom de MAPE-loop (IBM, 2006). La boucle est complétée par la connexion au logiciel adaptable à travers

les interfaces capteur et effecteur fournies. Voici un résumé de chaque processus décrit dans la Fig. 13 :

- La surveillance concerne la collecte et la corrélation des données provenant des capteurs et les convertit en modèles et symptômes comportementaux. Le processus de surveillance peut être réalisé par corrélation d'événements, vérification de seuil et surveillance de jauge.
- L'analyse est responsable de la détection des symptômes fournis par le processus de surveillance et également de l'historique du système afin de détecter les cas où un changement doit être appliqué.
- La planification détermine ce qui doit être changé et quelle est la meilleure façon de faire un tel changement. La planification est généralement spécifiée comme modèle de décision et soutenue par des algorithmes de prise de décision.
- L'exécution exécute les actions planifiées par le processus de décision. Cela inclut la gestion des actions non primitives via des workflows ou des actions de mappage vers des technologies d'adaptation dynamiques fournies par un logiciel adaptable.

La sous-section suivante présente les techniques, les mécanismes et les technologies qui facilitent le développement de systèmes adaptatifs.

3.2.2 Technologies habilitantes

Il existe plusieurs techniques disponibles pour explorer et détecter des applications logicielles. La journalisation est probablement la technique la plus ancienne et la plus simple pour surveiller les logiciels. Les journaux doivent être filtrés et analysés pour extraire des informations utiles. Pour faciliter ce processus, nous pouvons tirer parti des outils d'analyse de journaux. L'adaptateur de journal générique (AJG) d'IBM et l'analyseur de trace de journal (ATJ) (IBM, *Autonomic computing toolkit: Developer's guide*, 2005) sont des exemples de ces types d'outils. Outre la journalisation, plusieurs normes et cadres d'entreprise existent pour ajouter des fonctionnalités de surveillance dans les logiciels. « Application Response Measurement (ARM) » est l'une de ces normes (ARM: *Application Response Measurement*, 2011). ARM décrit une méthode d'intégration des applications d'entreprise en tant qu'entités gérables. Elle est livrée avec un SDK pour les langages Java et C, ce qui permet aux développeurs d'ajouter des capteurs logiques pour mesurer les paramètres de bout en bout. Les paramètres de mesure incluent la disponibilité des applications, les performances, l'utilisation et le temps de réponse des transactions de bout en bout.

Les profileurs peuvent également assister le processus de surveillance en tant que capteurs d'application ou capteurs de middleware. L'interface Java Virtual Machine Tools (JVMTI) est un exemple de profileur d'application et de middleware pour les applications Java (Sun Microsystems, Java Virtual Machine Tool Interface (JVMTI), 2006). JVMTI permet à un programme de surveiller les applications s'exécutant dans la JVM. Il est également livré avec un ensemble d'API pour la surveillance du niveau d'application. L'instrumentation de l'application pour les capteurs peut être effectuée au moment de la compilation, comme dans gprof (Graham, Kessler, & Mckusick, 1982), ou des codes d'instrumentation peuvent être ajoutés à l'application binaire comme dans ATOM (Lee, 2007) (Srivastava, 1994).

Java Management eXtensions (JMX) est un autre puissant cadre de gestion pour Java, qui offre de puissantes fonctions de détection et d'exécution (Sun Microsystems, Java Management Extensions (JMX) Specification, 2006). La technologie JMX, en tant que cadre de gestion inclus dans le kit de développement Java (JDK), est largement utilisée dans les environnements Java pour la gestion et la surveillance des ressources (par exemple, des périphériques et des applications) (Sun Microsystems, 2006). Dans JMX, chaque ressource est instrumentée par un objet Java appelé Managed Bean (MBean). Les applications externes peuvent accéder aux MBeans via le serveur JMX MBean. MBeans peut être consulté pour observer les changements, et peut être modifié afin de contrôler le comportement de l'application (Asadollahi, 2009).

Contrairement aux capteurs, les approches pour les effecteurs sont moins générales et plus spécifiques à chaque cas. Les effecteurs ont essentiellement la nature d'événements d'action. Le gestionnaire d'adaptation (ou le logiciel lui-même dans le cas de l'approche interne) déclenche des événements d'action, et le logiciel doit suivre certaines infrastructures structurelles et comportementales pour prendre en charge les effecteurs en tant qu'événements d'action.

Déclencher des événements d'action en ayant des paramètres de réglage appropriés dans le logiciel et en définissant leurs valeurs (ex. le projet STAC) qui vise à paramétrer les paramètres comme effecteurs dans les logiciels existants (Brake, 2008) . Cependant, la plupart des solutions proposées pour implémenter des effecteurs tentent de modifier le comportement du logiciel en changeant dynamiquement les données d'application et le flux de contrôle, tissage d'aspect dynamique (Popovici, Gross, & Alonso, 2002) et protocole de méta-objet (Kiczales, Des Rivieres, & Bobrow, The art of the metaobject protocol, 1991)).

a) Programmation orientée aspect

L'orientation d'aspect est utilisée comme une solution capable de développer des systèmes logiciels auto-adaptatifs. La programmation orientée aspect (POA) est une solution viable pour modifier

le code source et les exécutables de l'application selon les besoins (Kiczales, et al., 1997). Ceci est principalement dû au fait que la séparation des préoccupations (adaptation et fonctionnalité du système) est un facteur important dans les systèmes adaptatifs, ce qui est essentiellement l'un des principaux avantages de la conception orientée aspect. Les langages POA existants tels que les cadres AspectJ (Eclipse, 2008)] ou comme JBoss AOP (Khan, 2006), masquent la complexité d'implémentation des transformations de code-octet impliquées. Les concepts POA peuvent être implémenter autrement en utilisant des bibliothèques de manipulation de code-octet de niveau inférieur telles que ASM (Kuleshov, 2007) ou BCEL (Dahm, 2002).

Généralement, les cadres POA définissent deux choses: une façon d'implémenter des problèmes transversaux comme des extraits de code, et une construction programmatique, un langage de programmation ou un ensemble de balises, pour spécifier comment et où appliquer ces extraits de code. La terminologie varie d'une technologie à l'autre mais, en substance, les mêmes concepts s'appliquent. Le Tableau 5 présente une brève introduction à la terminologie POA, couvrant les concepts POA les plus courants.

Tableau 5 Liste des concepts communs utilisés dans la programmation orientée aspect

Concept	La description
Aspect	Une modularisation d'un problème qui concerne plusieurs objets. Les aspects peuvent être implémentés sous forme de classes standard mettant en œuvre des interfaces spécifiques ou sous forme de méthodes simples avec une signature prédéfinie.
Point de jointure	Un point lors de l'exécution d'un programme, tel que le début ou la fin de l'exécution d'une méthode.
Conseil	Action effectuée par un aspect à un point de jointure particulier. Les différents types de conseil incluent autour, avant et après.
Point de coupe	Un prédicat qui correspond aux points de jointure. Un conseil est associé à une expression de point de coupe et s'exécute à tout point de jointure correspondant au point de coupe.
Tissage	Processus consistant à mélanger des aspects avec d'autres types d'applications ou objets pour créer un objet conseillé.

La programmation orientée aspect (POA), et plus particulièrement POA dynamique, facilite l'encapsulation des problèmes d'adaptation sous la forme d'aspects grâce à l'adaptation

dynamique du temps d'exécution. Il permet également de mettre en œuvre des actions d'adaptation fines à un niveau inférieur à celui des composants (Greenwood, 2004) (Salehie M. a., 2009). Une autre approche est JAC (Pawlak, et al., 2004), qui utilise une chaîne d'enveloppement qui peut changer dynamiquement des points de jointure existants ou nouveaux.

Par ailleurs, Les techniques d'IE qui reconnaissent de manière explicite l'importance accordée au même titre pour les préoccupations transversales fonctionnelles et non-fonctionnelles ainsi que pour celles de base (non transversales) sont qualifiées d'approches d'IE orientées aspect (Amroune, 2014).

b) Middleware

Le développement de logiciels adaptables n'est pas une tâche simple, car les comportements autonomes augmentent la complexité du logiciel et peuvent entrer en conflit avec d'autres propriétés non fonctionnelles tel que le coût ou le temps de réalisation. En ce qui concerne l'architecture de référence générique pour les systèmes autonomes proposée dans (Kephart & Chess, 2003), le logiciel adaptable peut se rapporter à quelque chose d'aussi étroit qu'un seul module, ou aussi large que des conteneurs d'application, des intergiciels ou même des systèmes d'exploitation. Par conséquent, nous pouvons considérer un logiciel adaptable comme une application ou un service, en conjonction avec ses couches constitutives. Ce point de vue nous permet de tirer parti des possibilités de détection et de mise en œuvre possibles fournies par l'environnement d'exploitation de l'application. A titre d'exemple, la charge du système peut être surveillée à partir de différentes vues par l'intermédiaire de l'API du système d'exploitation ou de paramètres d'état (par exemple, nombre de sessions actives).

Par conséquent, il existe toute une gamme de mécanismes de détection et de mise en œuvre à partir de capteurs et d'effecteurs de niveau applicatif spécifiques aux cas, de capteurs génériques et d'effecteurs fournis par l'environnement d'exploitation de l'application. Avoir une plate-forme gérable est un grand avantage pour le logiciel adaptable, parce que les mécanismes de détection et d'exécution de haut niveau sont généralement plus robustes, et sont réalisés par des solutions systématiques. Une bonne plate-forme qui prend en charge l'adaptation doit être capable de fournir un ensemble flexible et riche de mécanismes de détection des données et d'exécution pour toutes les couches d'application. Cela peut être réalisé en fournissant des fonctionnalités de gestion, telles que des API, des protocoles et des normes. Ainsi, les plates-formes appropriées pour développer des systèmes gérables devraient avoir:

- des capacités pour observer et modifier les états du système,
- des modules configurables en temps de chargement,

- des modules modifiables dynamiquement (temps d'exécution), et
- des mécanismes pour l'ajustement à un seul point de changement des variables d'état.

Comparés aux mécanismes directs au niveau de l'application, les mécanismes de détection et d'exécution au niveau de la plate-forme peuvent ajouter des limitations indésirables (par exemple, des données de détection moins précises) ou des surcharges supplémentaires sur les performances du système. Cependant, d'une manière générale, l'adaptation au niveau de la plate-forme présente moins de couplage entre les capteurs et les effecteurs et la logique métier de l'application.

c) La prise de décision

La tâche d'analyser et de décider de l'action d'adaptation optimale à exécuter, étant donné l'état et le contexte du système, est un problème de planification (Dowling, 2004): « comment trouver une politique de décision optimale pour effectuer des actions d'adaptation ? modèle de l'état du système, un ensemble d'actions d'adaptation disponibles, et un moyen d'évaluer le résultat des actions d'adaptation ? »

En général, la planification et la prise de décision dans les logiciels auto-adaptatifs sont une forme particulière du problème de sélection d'action (ASP) dans les agents autonomes (Maes, *Situated agents can have goals.*, 1990) et la robotique, en particulier la robotique comportementale (Rosenblatt, 1997). Des similitudes peuvent être énumérées en considérant le dynamisme et l'incertitude de l'environnement, ainsi que la prise de décision en ligne. Cependant, le problème dans le logiciel est généralement plus complexe que dans la robotique. Principalement parce que le comportement du logiciel n'obéit pas aux règles physiques, a plus de variables de contrôle et de perturbations, et plus de complexité.

Plusieurs idées issues de la robotique comportementale semblent prometteuses pour les logiciels autonomes. La majorité des gestionnaires autonomes existants utilisent une approche fondée sur un modèle pour l'adaptation. Ces systèmes utilisent normalement un modèle de fusion de capteurs pour capturer des événements et mettre à jour le modèle. L'idée de la robotique basée sur le comportement consiste à utiliser des modules de réalisation de tâches spécifiques distribués, appelés comportements, et à appliquer une fusion de commandes à la place de la fusion de capteurs (par exemple, architecture de subsomption (Brooks, 1986)). Par ce moyen, il n'est pas nécessaire de développer, maintenir et étendre un modèle monolithique cohérent pour l'élément autonome et son contexte, Salehie et al. (Salehie & Tahvildari, *A weighted voting mechanism for action selection problem in self-adaptive software*, 2007) utilisent un mécanisme de vote pour la prise de décision dans les systèmes autonomes.

Par ailleurs, il y a eu un intérêt accru pour l'utilisation de techniques d'apprentissage automatique pour l'adaptation de logiciels. Dans (Tesauro, 2007), Tesauro présente une approche hybride qui permet à l'apprentissage par renforcement (Sutton, 1998) de s'amorcer à partir des politiques de gestion existantes, ce qui réduit considérablement le temps d'apprentissage et le coût. Il démontre également l'efficacité des apprentissages de renforcement hybride dans le contexte d'un simple prototype de centre de données. Dowling décrit comment l'apprentissage par renforcement peut être utilisé pour coordonner les composantes autonomes dans un environnement distribué (Dowling, 2004). Sa thèse de doctorat introduit K-Component, un cadre pour intégrer un ensemble décentralisé d'agents qui apprennent pour l'adaptation. En outre, Littman et al. ont utilisé l'apprentissage par renforcement pour traiter les propriétés d'auto-guérison dans le domaine des systèmes de réseau (Littman, 2004), affirmant que leur système est capable d'apprendre à restaurer efficacement la connectivité réseau après une défaillance.

3.2.3 Cadre d'adaptation

Il y a plusieurs projets réussis liés aux cadre d'adaptation d'exécution (Salehie & Tahvildari, 2009). Dans cette section, nous présentons des projets, des cadres et des technologies connexes, en mettant brièvement en évidence leurs caractéristiques.

Le projet DiVA (Morin B. a.-M., 2009) considère les phases de conception et d'exécution de l'adaptation. DiVA se concentre sur l'adaptation au niveau architectural et repose sur une solide approche de modélisation en quatre dimensions de SAA (Fleurey, 2009). Au moment du design, un modèle de base et ses variantes d'architecture sont créés. Ces modèles incluent des invariants et des contraintes, qui peuvent être utilisés pour valider les règles d'adaptation. L'un des principaux objectifs de ce travail est de s'attaquer au problème de l'explosion du nombre de configurations possibles du système d'exécution (modes). Les auteurs utilisent la modélisation orientée aspect pour dériver un ensemble de modes en tissant des aspects dans un modèle explicite (Morin B. a.-M., 2009). Enfin, les modes générés sont ensuite utilisés pour adapter le système automatiquement.

Vogel et Giese (Vogel, 2010) proposent une approche orientée modèle qui fournit de multiples modèles d'exécution architecturale à différents niveaux d'abstraction. Par exemple, des modèles plus abstraits sont dérivés de modèles de code source concrets. Les sous-ensembles de ces modèles cibles abstraits se concentrent sur des préoccupations d'adaptation spécifiques pour simplifier les gestionnaires autonomes et soutenir la séparation des préoccupations entre eux. Comme DiVA, cette recherche cible également l'adaptation d'un point de vue architectural. Ces deux approches reposent sur des architectures basées sur les entreprises et les composants, telles que JavaEE, ce qui les rend moins adaptées à l'activation de l'adaptabilité dans les applications traditionnelles et

héritées. La mise en œuvre présentée est basée sur l'infrastructure mKernel (Bruhn, 2008), qui offre un support complet pour la gestion des applications d'entreprise Java.

TRAP/J (Sadjadi, McKinley, Cheng, & Stirewalt, 2004) permet d'ajouter un nouveau comportement adaptable aux applications Java existantes sans nécessiter de modifications du code source d'origine ou de la machine virtuelle. TRAP/J utilise la réflexion comportementale et les techniques POA. Dans cette approche, les développeurs de logiciels choisissent parmi un sous-ensemble de classes existantes et génèrent des aspects et des classes réfléchissantes pour dériver une version adaptable de leur logiciel d'origine. TRAP/J se concentre sur l'approche technologique de la façon de rendre une application Java existante adaptable. Il prend en charge la réification des invocations de méthodes, mais pas la réification des opérations de lecture/écriture sur les champs.

StarMX (Asadollahi, 2009) est un cadre générique de gestion d'adaptation configurable. Il sépare la logique de gestion de la logique d'application en utilisant des interfaces de détection et d'exécution explicites. StarMX n'a aucune dépendance sur les caractéristiques de l'application (par exemple, l'architecture ou l'environnement) ou les problèmes d'adaptation. Conçu pour les systèmes basés sur Java, il intègre la technologie JMX (Java Management Extensions) et est capable de s'intégrer à divers moteurs de politique/règles.

AspectOpenCOM (Grace, Lagaisse, Truyen, & Joosen, 2008) est un cadre de composants basé sur la réflexion. Contrairement aux travaux présentés ci-dessus, cette technologie est axée sur l'adaptation fine des aspects déjà déployés. AspectOpenCOM est implémenté en Java, et les auteurs présentent une étude de cas illustrant les avantages de performance de leur approche fine de l'adaptation dans le contexte de la composition d'aspect. Cette recherche n'est pas liée à des modèles d'exécution, mais fournit des installations de niveau inférieur à utiliser dans l'adaptation d'exécution POA en général.

3.3 Travaux sur l'ingénierie des exigences pour l'auto-adaptation

RAINBOW : Cheng et Garlan (Cheng & Garlan, 2007) ont décrit trois sources spécifiques d'incertitude dans l'auto-adaptation (identification de l'état du problème, sélection de la stratégie et le résultat de la stratégie), en fournissant des directives de haut niveau pour les atténuer dans le cadre de travail de l'arc-en-ciel (Rainbow) (Garlan, Cheng, Huang, Schmerl, & Steenkiste, 2004). L'identification de l'état de problème est liée aux activités de surveillance et d'analyse de la boucle MAPE, tandis que la sélection de la stratégie et le résultat de la stratégie sont liés aux activités de planification et d'exécution, respectivement. En d'autres termes, ils essaient d'atténuer l'incertitude dans les activités de la boucle de contrôle de rétroaction d'adaptation.

Pour atténuer l'incertitude dans l'identification de l'état du problème, ils utilisent la moyenne courante pour surveiller la variabilité et les propriétés stochastiques de l'environnement. Les observations sont ensuite comparées aux descriptions architecturales qui sont complétées par des informations probabilistes pour détecter les tendances du comportement. Une fois le problème détecté, une stratégie est sélectionnée pour résoudre le problème. L'incertitude dans la sélection de la stratégie est atténuée en utilisant le langage STITCH. Ce langage permet de modéliser l'incertitude dans les stratégies. Par conséquent, lorsque RAINBOW tente de sélectionner une stratégie à l'exécution, il peut décider en fonction de la valeur attendue (qui capture l'incertitude) de différentes stratégies. Enfin, une fois qu'une stratégie est sélectionnée et mise en œuvre, elle peut réussir ou échouer. Au lieu de faire face à cette incertitude dans la prochaine boucle d'adaptation, ils considèrent l'incertitude du résultat de la stratégie en spécifiant la durée pendant laquelle RAINBOW devrait mettre en œuvre la stratégie avant de s'engager dans le changement. C'est un autre attribut de l'approche qui peut être modélisé en utilisant le langage STITCH.

En augmentant les modèles architecturaux avec des modèles probabilistes, RAINBOW atténue l'incertitude due à la simplification des hypothèses et du bruit. De plus, en surveillant le système après adaptation, RAINBOW atténue l'incertitude due à la dérive des modèles architecturaux.

RELAX : Whittle et al. ont introduit RELAX (Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2010), un langage formel de spécification des exigences qui s'appuie sur la logique temporelle de ramification floue (Moon, Lee, & Lee, 2004) pour spécifier les exigences incertaines dans les systèmes auto-adaptatifs. RELAX utilise la théorie des possibilités pour gérer l'incertitude des objectifs. RELAX permet l'expression explicite de l'incertitude environnementale et de son effet sur les exigences. Selon l'état de l'environnement, RELAX spécifie les exigences qui peuvent être désactivées ou « soulagées ». À cette fin, RELAX introduit un ensemble d'opérateurs pouvant être utilisés pour définir les exigences. Ces opérateurs définissent également la façon dont l'exigence peut être assouplie lors de l'exécution. De plus, les opérateurs saisissent le type d'incertitude (facteur d'incertitude) qui peut déclencher l'assouplissement des exigences.

Dans une publication ultérieure (Cheng, Sawyer, Bencomo, & Whittle, 2009), Les auteurs ont étendu RELAX avec la modélisation d'objectif pour spécifier l'incertitude dans les objectifs. Ils construisent d'abord le réseau cible puis l'utilisent de manière ascendante pour rechercher des sources d'incertitude, qui sont les éléments du domaine de l'environnement et peuvent compromettre la satisfaction des objectifs. Dans leur approche, ils identifient l'incertitude au moyen d'une variante de la modélisation des menaces, utilisée pour identifier les menaces de sécurité dans un système. Une fois l'incertitude identifiée, son impact est évalué pour concevoir

des tactiques d'atténuation. La tactique ultime pour atténuer l'incertitude (lorsque toutes les autres tactiques échouent) consiste à ajouter de la flexibilité à l'objectif en le « relaxant ».

FLAGS (Baresi, Pasquale, & Spoletini, 2010) utilise également la théorie des possibilités pour atténuer l'incertitude des projets. Semblable à RELAX, «FLAGS» vise à atteindre l'objectif fondamental des systèmes adaptatifs au niveau des exigences: atténuer l'incertitude associée à l'environnement et aux nouveaux besoins commerciaux en intégrant l'adaptabilité dans le système logiciel dès la demande. En d'autres termes, «FLAGS» considère l'auto-adaptation comme un type particulier d'exigence, qui affecte d'autres exigences. Ces exigences spéciales sont appelées objectifs adaptatifs et FLAGS permet de définir les contre-mesures qui doivent être effectuées si certains objectifs ne sont pas atteints comme prévu (en raison de l'incertitude prévue).

FLAGS traite également une autre source d'incertitude, en plus de l'incertitude dans le contexte du logiciel, l'incertitude dans les objectifs eux-mêmes. Comme la satisfaction de certains objectifs ne peut pas être spécifiée par une simple réponse oui-non, FLAGS repose sur des objectifs flous pour lesquels les propriétés ne sont pas complètement connues, dont la spécification intégrale n'est pas disponible et dont les petites violations temporaires sont tolérées. Par conséquent, FLAGS se termine par deux séries d'objectifs, des objectifs précis et des objectifs flous. Il formalise les objectifs précis en utilisant la logique temporelle linéaire (LTL) et les objectifs flous en utilisant un langage temporel flou, qui est finalement unifié avec la spécification LTL. Par conséquent, toutes les exigences logicielles peuvent être spécifiées dans un seul langage cohérent.

RESIST (Cooray, 2010) utilise des informations provenant de plusieurs sources, telles que la surveillance des propriétés logicielles internes et externes, les modifications de la structure du logiciel et les propriétés contextuelles pour fournir en permanence des prévisions de fiabilité affinées au moment de l'exécution. Les prévisions de fiabilité à jour expriment la fiabilité des systèmes dans un futur proche en utilisant les probabilités. Ces prévisions sont ensuite utilisées pour décider de la modification de la configuration du logiciel pour améliorer sa fiabilité de manière proactive. RESIST est destiné aux systèmes logiciels situés, essentiellement mobiles, intégrés et omniprésents. L'incertitude dans ces systèmes est très répandue car ils ont une configuration hautement dynamique, un profil/contexte opérationnel inconnu et des conditions fluctuantes, mais ils sont généralement déployés dans des environnements critiques (par exemple: intervention d'urgence) et des liens stricts d'exigences de mobilité. RESIST atténue l'incertitude due au contexte et simplifie les hypothèses grâce à un apprentissage constant. De plus, de légères modifications de la fiabilité sont modélisées comme des distributions de probabilité indiquant le bruit.

RESIST adopte une approche compositionnelle pour l'estimation de la fiabilité; le processus commence par une analyse au niveau des composants, ce qui permet d'évaluer l'impact des choix d'adaptation sur la fiabilité du système. La fiabilité au niveau des composants est estimée de manière stochastique en utilisant une chaîne de Markov à temps discret et en termes de fraction du temps passé en état de défaillance par le composant. Une fois que la fiabilité de tous les composants est obtenue, un modèle de composition est utilisé pour déterminer la fiabilité des configurations spécifiques du système. RESIST modélise l'incertitude dans l'apprentissage en utilisant les probabilités.

FUSION (Elkhodary, Esfahani, & Malek, 2010) est une approche basée sur l'apprentissage pour l'ingénierie des systèmes auto-adaptatifs. Au lieu de s'appuyer sur des modèles analytiques statiques soumis à des hypothèses simplificatrices, FUSION utilise l'apprentissage automatique, à savoir l'apprentissage des arbres modèles, pour ajuster le comportement adaptatif du système aux changements imprévus. Cela permet à FUSION d'atténuer l'incertitude associée au changement dans le contexte du système logiciel au fur et à mesure qu'il apprend que la bonne adaptation doit être effectuée dans le nouvel environnement. Le résultat de l'apprentissage est un ensemble de relations entre les actions d'adaptation dans le système et les attributs de qualité d'intérêt (par exemple, temps de réponse, disponibilité). Ces règles tiennent compte de l'interaction des actions d'adaptation et, par conséquent, atténuent dans une certaine mesure l'incertitude causée par la synergie. Les attributs de qualité présentant un intérêt peuvent être mesurés et collectés à partir du système en cours d'exécution via l'instrumentation du logiciel ou des capteurs fournis par la plate-forme d'implémentation. Les actions d'adaptation correspondent aux points de variation du logiciel pouvant être exercés à l'exécution.

FUSION a deux cycles complémentaires: cycle d'apprentissage et cycle d'adaptation. Le cycle d'apprentissage relie les mesures d'attributs de qualité aux actions de l'adaptation. Le cycle d'apprentissage surveille en permanence l'environnement afin de détecter d'éventuelles erreurs dans les relations apprises. La persistance de telles erreurs, qui peuvent être dues à une dérive ou à un changement de contexte, déclenche un réapprentissage du nouveau comportement. Lorsque la qualité du logiciel diminue avec le temps et descend au-dessous d'un certain seuil, le cycle d'adaptation se déclenche et utilise les connaissances acquises pour prendre une décision d'adaptation éclairée afin d'améliorer les attributs de qualité. La qualité du logiciel est définie comme la collecte globale des attributs de la qualité individuelle. Cependant, certains attributs de qualité pouvant entrer en conflit, la notion d'utilité est utilisée pour permettre des compromis.

POISED (Esfahani, Kouroshfar, & Malek, 2011) est une approche quantitative pour aborder la complexité de la prise automatique de décisions d'adaptation dans des conditions d'incertitude. Il s'appuie sur la théorie des possibilités et les mathématiques floues pour évaluer à la fois les

conséquences positives et négatives. Le but de POISED est d'améliorer les attributs de qualité d'un système logiciel en reconfigurant ses composants pour obtenir une configuration globale optimale du système logiciel. POISED redéfinit la définition conventionnelle de la décision d'adaptation optimale à celle qui présente la meilleure gamme de comportement. À son tour, la solution sélectionnée a la plus grande probabilité de satisfaire aux objectifs de qualité du système, même si, en raison de l'incertitude, les propriétés attendues du système ne sont pas confirmées dans la pratique. Ceci est différent des approches conventionnelles, qui n'intègrent pas l'incertitude dans leur analyse. De telles approches considèrent le comportement du système comme une estimation ponctuelle, tandis que POISED considère une gamme de comportements.

POISED fournit un cadre pour rassembler et créer des incertitudes dans une représentation cohérente, qui se prête bien à la prise de décision. POISED repose sur la programmation linéaire possibiliste pour faire le compromis entre différentes alternatives de configuration. Les boutons de configuration dans POISED permettent au décideur de spécifier quel aspect de l'incertitude est le plus important: dans certains cas, une solution capable de fournir certaines garanties dans le pire des cas serait souhaitable, dans d'autres une solution présentant un risque plus élevé, mais le potentiel d'une meilleure qualité pourrait être souhaitable.

Exigences de sensibilisation : Souza (Silva Souza, Lapouchnian, Robinson, & Mylopoulos, 2011) a présenté un nouveau type d'exigence, appelée exigence de sensibilisation, pouvant faire référence à d'autres exigences et à leurs succès/échecs. Ils ont proposé un moyen d'élire et de formaliser ces exigences et de leur proposer un cadre de surveillance des exigences.

Exigences d'évolution : un autre travail de Souza (Souza, Lapouchnian, & Mylopoulos, 2012), il a proposé un nouveau type d'exigence appelé exigences d'évolution, ce genre d'exigence jouent un rôle important dans la durée de vie d'un système logiciel en ce sens qu'elles définissent les modifications possibles des exigences, ainsi que les conditions dans lesquelles ces modifications s'appliquent. Dans son travail, il s'intéresse à cette famille d'exigences, à leur modélisation et à leur opérationnalisation au moment de l'exécution.

Ahmad, Belloir et Bruel (Ahmad, Belloir, & Bruel, 2015) ont proposé une approche intégrée pour la modélisation et la vérification des exigences des systèmes auto-adaptatifs à l'aide de techniques d'ingénierie dirigée par les modèles. Pour cela, ils ont utilisé RELAX (Whittle J. a.-M., 2009). Ils ont utilisé ensuite les concepts de l'ingénierie des exigences orientée objectifs pour déterminer et modéliser les exigences des systèmes auto-adaptatifs.

MAPE-K FT : Les auteurs dans (Iglesia & Weyns, 2015) contribue à un ensemble de modèles MAPE-K formellement spécifiés qui encodent l'expertise de conception pour une famille de systèmes auto-adaptatifs. Les modèles comprennent: (1) des modèles de spécification de

comportement pour la modélisation des différents composants d'une boucle de rétroaction MAPE-K (basée sur des réseaux d'automates temporisés), et (2) des modèles de spécification de propriétés permettant de vérifier l'exactitude des comportements d'adaptation (basé sur une logique d'arborescence de calcul temporisée). Pour démontrer la réutilisabilité des modèles formels, les auteurs ont effectué quatre études de cas dans lesquelles les étudiants de dernière année en maîtrise utilisaient les modèles pour concevoir différents systèmes auto-adaptatifs.

E. Vassev et M. Hinchey (Vassev & Hinchey, 2015) ont proposé une approche pour l'ingénierie des exigences d'autonomie (ARE). Cette approche, basée sur l'ingénierie des exigences orientée objectifs (GORE) et les exigences génériques d'autonomie (GAR), a l'objectif d'éliciter et spécifier les exigences d'autonomie ainsi que la définition des objectifs alternatifs, pour les systèmes auto-adaptatifs.

Tropos4AS (Morandini, Penserini, Perini, & Marchetto, 2017) : Dans ce travail, les auteurs ont présenté un cadre pour les exigences techniques des systèmes logiciels adaptatifs. Cette approche, appelée Tropos4AS, combine des concepts orientés objectifs et des méthodes de conception à forte variabilité. Le modèle de gestion des exigences de Tropos4AS peut être directement mappé sur des prototypes logiciels avec une architecture orientée agent qui peut être exécutée pour la validation et le raffinement des exigences. Ils donnent une description complète du cadre, avec des modèles conceptuels, des directives de modélisation et des outils de support. L'applicabilité du cadre à la validation et à l'amélioration des exigences est illustrée par une étude de cas. Deux expériences contrôlées avec des sujets fournissent une évaluation empirique du langage de modélisation proposé, avec des preuves statistiques de l'efficacité de l'approche de modélisation pour la collecte des exigences des systèmes adaptatifs.

SA-RBAC : Les auteurs dans (da Silva, da Silva, Paterson, & Calinescu, 2017) ont proposé une approche pour la reconfiguration dynamique du contrôle d'accès basé sur les rôles (RBAC) des systèmes d'information exécutant des processus métier, afin de les protéger contre les menaces internes. La nouvelle approche utilise les traces d'exécution de processus métier et la vérification de modèle stochastique pour établir des intervalles de confiance pour les attributs clés mesurables du comportement des utilisateurs, et pour identifier et rétrograder de manière adaptative les utilisateurs qui utilisent abusivement leurs autorisations d'accès. Les auteurs ont mis en œuvre et évalué l'approche et son formalisme de spécification de politique pour un véritable processus métier de support informatique, en montrant leur capacité à exprimer et à appliquer un large éventail de stratégies RBAC auto-adaptables.

D'autres approches ont été proposées, comme les approches basées sur les objectifs et les aspects. Ceux fondés sur des modèles d'objectifs sont par exemple i^* (Yu, 1997), KAOS (Dardenne, 1993),

FLAGS (Baresi, Pasquale, & Spoletini, 2010). D'autre part, les approches orientées aspects tel que AspeCiS (Amroune, 2014).

3.4 Travaux sur les patrons de spécification

Les travaux qui suivent se concentrent sur les approches basées sur le concept de patron de spécification qui représente la brique de base pour le langage proposé:

Dwyer et al (Dwyer, Avrunin, & Corbett, 1999) ont proposé une approche basée sur des patrons pour la présentation, la codification et la réutilisation des spécifications de propriété pour la vérification par états finis. Ils ont effectué une enquête sur les spécifications disponibles, recueillant plus de 500 exemples de spécifications de propriétés. Ils ont constaté que la plupart sont des exemples de leurs patrons proposés.

Les modèles de spécification de propriété ont été utilisés avec succès pour combler cet écart entre les praticiens et les outils de vérification de modèle. Le système de patron proposé dans (Dwyer, Avrunin, & Corbett, 1999) ne tient pas compte des informations sur le temps. Les auteurs dans (Gruhn & Laue, 2006) ont proposé un catalogue de patrons pour les besoins en temps réel. Pour chaque patron, ils ont construit des automates d'observateurs (observateurs) qui peuvent être appliqués directement dans les outils de vérification de modèles temporisés. L'utilisateur qui utilise le système de patron n'a pas à se soucier de ces observateurs, il est possible de les construire automatiquement.

Grunske Lars (Grunske, 2008) présente, dans son papier, un système de patron de spécification de propriétés probabilistes courantes appelé ProProST. Ce système de configuration a été développé sur la base d'une étude de 152 propriétés d'exemples académiques et de 48 propriétés répondant à des critères de qualité réels, issus des systèmes avionique, de défense et automobile. En outre, une grammaire anglaise structurée pouvant guider la spécification des propriétés probabilistes est donnée. Semblable aux patrons de spécification précédents pour les propriétés traditionnelles et en temps réel, le système de modèle de spécification présenté et la grammaire anglaise structurée saisissent les connaissances des experts et aident les praticiens à appliquer correctement les techniques de vérification formelle.

Autili et al (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015) ont proposé un cadre combinant des patrons de spécification de propriétés qualitatifs, en temps réel et probabilistes. Les avantages d'un tel cadre est de permettre la suppression de la distinction entre les aspects qualitatifs et quantitatifs des événements et de fournir une structure pour découvrir systématiquement les nouveaux patrons de spécification de propriété. Dans ce travail, les auteurs présentent un catalogue unifié qui regroupe tous les modèles connus et 40 modèles récemment

identifiés ou étendus. Ils proposent également une interface en langage naturel pour associer des patrons à une logique temporelle de choix. Ils ont utilisé également, l'outil PSPWizard spécialement développé pour fonctionner avec leur catalogue de patrons unifié, afin de restituer automatiquement des instances concrètes de patrons de spécification de propriétés sous forme de formules logiques de choix.










Afin d'évaluer les travaux connexes présenté ci-dessus, nous avons identifié 4 critère d'évaluation d'approches d'ingénierie des exigences pour les SAAs, ces dernier sont:

- L'ambiguïté des spécifications résultantes de l'approche : Identifié par le fait que l'approche est équipée par une grammaire non-réursive.
- Le support natif de l'auto-adaptation : Le fait que le travail présente une approche pour l'auto-adaptation, en particulier le support explicite de l'incertitude.
- La support de vérification automatique : Si l'approche présente un support formelle pour les spécifications.
- La simplicité d'usage : Un critère très important par rapport à notre point de vue, ce dernier permet aux ingénieurs peu expérimentés de maîtriser mieux la spécification des exigences.

Le tableau ci-dessous résume les approches connexes dans l'ingénieries des exigences pour le systèmes auto-adaptatif ainsi que les patrons de spécification.

Tableau 6. Ingénieries des exigences pour le systèmes auto-adaptatif et patrons de spécification.

Approche	Réponse aux critères d'évaluation	Année
Whittle et al (RELAX) (Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2010)	<input type="checkbox"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/>	2009
Cheng et al (Cheng, Sawyer, Bencomo, & Whittle, 2009)	<input type="checkbox"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/>	2009
Baresi et al (FLAGS) (Baresi, Pasquale, & Spoletini, 2010)	<input type="checkbox"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/>	2010
Souza et al (Silva Souza, Lapouchnian, Robinson, & Mylopoulos, 2011)	<input type="checkbox"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/>	2011

Souza et al (Souza, Lapouchnian, & Mylopoulos, 2012)		2012
Ahmad, Belloir et Bruel (Ahmad, Belloir, & Bruel, 2015)		2015
Tropos4AS (Morandini, Penserini, Perini, & Marchetto, 2017)		2017
da Silva et al (da Silva, da Silva, Paterson, & Calinescu, 2017)		2017
Dwyer et al (Dwyer, Avrunin, & Corbett, 1999)		1999
Gruhn Volker & Laue Ralf (Gruhn & Laue, 2006)		2006
Grunske Lars (Grunske, 2008)		2008
Autili et al (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015)		2015
MAPE-K FT (Iglesia & Weyns, 2015)		2015

Les travaux décrits ci-dessus ont suscité un intérêt particulier pour l'incertitude qui est considérée comme un principal problème dans le développement des systèmes auto-adaptatifs. Le traitement de ce dernier dans les premières phases de développement est nécessaire. Cependant, certains problèmes dans un langage de spécification doivent être manipulés. Par exemple, la grammaire du langage doit être d'une part non ambiguë, et d'autre part le langage naturel dans la spécification est limité en raison de son ambiguïté. Notre travail traite à la fois le langage naturel et l'ambiguïté grammaticale, via le catalogue de patrons et la grammaire anglaise structurée. Le tableau ci-dessous résume certaines approches similaires.

Tableau 7 Vue générale sur les langages d'auto-adaptation

Langage	Support d'auto-adaptation	Gestion d'incertitude	Ambiguïté	Basé-patron	Cartographie
RELAX	Oui	Oui	Ambigu	Non	FBTL
FLAGS	Oui	Oui	Ambigu	Non	LTL/FTL
PSAS	Oui	Oui	Non-ambigu	Oui	FMTL

3.5 Conclusion

Dans ce chapitre, nous avons décrit le contexte de notre travail, à savoir le SAA. Ensuite, nous avons décrit la spécificité de ces systèmes et montré en quoi ces systèmes diffèrent des autres systèmes. Nous avons cité ensuite quelques cadres de SAA. Par ailleurs, nous avons présenté un ensemble de travaux dans l'ingénierie des exigences pour les SAA ainsi que les patrons de spécification. Le défi majeur en termes d'exigences consiste à définir un nouveau langage capable de saisir l'incertitude à un niveau abstrait. Une fois l'incertitude prise en compte au stade des exigences, nous devons également trouver des moyens de la gérer. Nous devons donc représenter les compromis entre la flexibilité fournie par l'incertitude et les assurances requises par la demande.

Chapitre 4

Eléments de base

Dans ce chapitre, nous donnons une description détaillée des éléments de base et de différentes techniques nécessaires pour la mise en place de notre langage PSAS (Pattern Specification for Self-Adaptive Systems). Les concepts tels que « patron de spécification » est présenté dans les deux premières sections, ainsi que les logiques mathématiques requises pour la cartographie des patrons.

4.1 Patron de spécification

Un patron de spécification est un modèle pour représenter un sous-ensemble de propriétés qui s'expriment de la même manière (Dwyer, Avrunin, & Corbett, 1999). Selon (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015), les patrons sont regroupés en trois familles: les patrons qualitatifs qui décrivent la matérialisation des événements, les patrons de temps réel qui étendent la première famille en ajoutant une contrainte temporelle et les patrons probabilistes qui étendent également les patrons qualitatifs en ajoutant la contrainte probabiliste.

4.2 Portée de patron

Afin de décrire le moment où le patron s'applique, chaque classe décrite ci-dessus doit être couplée avec une portée. En d'autres termes, tous les patrons doivent correspondre à la règle « spécification = portée, patron ». Le Tableau 8 donne les définitions des portées présentées dans (Dwyer, Avrunin, & Corbett, 1999). La Fig. 14 illustre un aperçu du fonctionnement des portées.

Tableau 8 Portée du patron

Portée	Définition
Globally	Le patron peut se maintenir en tout point de l'exécution du système.
After{P}	Le patron s'applique après la première occurrence de P.
Before{P}	Le patron s'applique jusqu'à la première occurrence de P.
Between {P} and {R}	Le patron ne peut s'appliquer qu'entre P et R, autrement dit, si le patron se produit, il doit être précédé de P et suivi de R.
After {P} until {R}	Le patron ne s'applique qu'après l'occurrence de P jusqu'à ce que R soit vrai (R n'est pas nécessaire).

Note : la différence entre « After {P} until {R} » et « Between {P} and {R} » est : dans la première portée, si R ne se produit pas, le modèle est toujours valide. Alors que dans la seconde, il n'est pas.

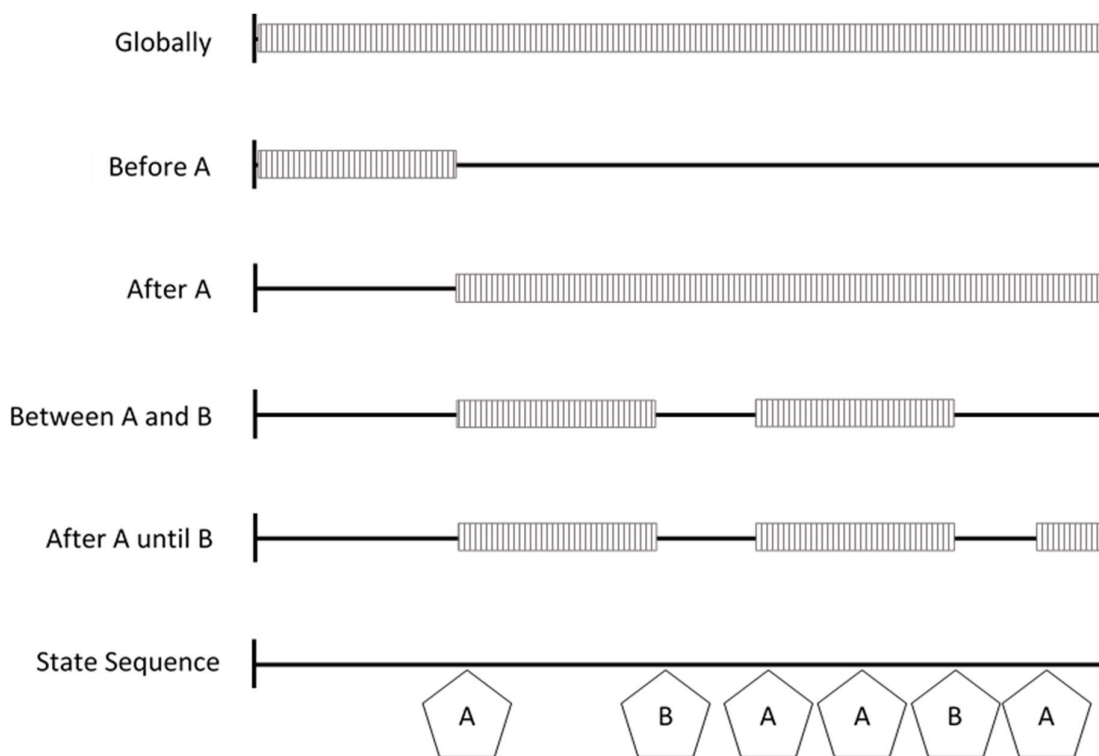


Fig. 14 Portée du patron : présente la chronologie autorisée d'un motif spécifique, délimitée par les événements A et B (Dwyer, Avrunin, & Corbett, 1999)

4.3 Logiques temporelles

4.3.1 Logique temporelle linéaire (LTL)

La logique temporelle linéaire (Zohar & Amir, 1992) est interprétée sur un modèle, qui est une séquence infinie d'états: $s:s_0, s_1, \dots$. C'est une extension de la logique ordinaire pour traiter de la nature du temps et des concepts temporels.

Étant donné $\sigma = (s_0, s_1, \dots, s_i, \dots, s_j, \dots)$ et une formule temporelle f , nous définissons

- $(\sigma, i) \models \phi$ ssi $s_i \models \phi$, ce qui signifie que f est vrai dans l'état s_i de la séquence s ;
- $(\sigma, i) \models \phi_1 \wedge \phi_2$ ssi $(\sigma, i) \models \phi_1$ and $(\sigma, i) \models \phi_2$ and;
- $(\sigma, i) \models \neg\phi$ ssi not $(\sigma, i) \models \phi$;

Les futurs opérateurs typiques utilisés dans la logique temporelle sont O (suivant), \square (toujours), \diamond (éventuellement), et U (jusqu'à).

- $(\sigma, i) \models O\phi$ ssi $(\sigma, i+1) \models \phi$;
- $(\sigma, i) \models \phi_1 U \phi_2$ ssi $\exists k \geq i$ tels que $(\sigma, k) \models \phi_2$, et pour tout j , tels que $i \leq j < k$, $(\sigma, j) \models \phi_1$;
- $(\sigma, i) \models \diamond\phi$ ssi $\exists k \geq i$ tels que $(\sigma, k) \models \phi$; and
- $(\sigma, i) \models \square\phi$ ssi $\forall k \geq i$ tels que $(\sigma, k) \models \phi$.

4.3.2 Logique temporelle métrique (MTL)

Il existe une classe substantiellement large de programmes qui fonctionnent dans des environnements distribués et en temps réel, dont l'exactitude de la spécification requiert l'expression de propriétés critiques en fonction du temps qui se rapportent aux occurrences d'événements du système. Ces exemples incluent les protocoles de communication réseau et les systèmes de contrôle en temps réel intégrés.

La logique temporelle linéaire n'accepte cependant que la spécification d'exigences temporelles qualitatives, telles qu'un événement survenant « éventuellement ». Pour permettre un raisonnement quantitatif sur les retards de temporisation dans les applications en temps réel, la logique en temps réel inclut des références temporelles explicites interprétées sur des séquences chronométrées, qui associent un temps à chaque état (Zhou & Murata, 1999).

MTL associe le temps à chaque état. La séquence temporisée :

$$\sigma = ((s_0, T_0), (s_1, T_1), \dots, (s_i, T_i), \dots, (s_j, T_j) \dots),$$

Avec $T_0, T_1, T_2, \dots, T_i, \dots, T_j, \dots$ remplissent les conditions suivantes:

1. $T_0 \leq T_1 \leq T_2 \leq \dots \leq T_i \leq \dots \leq T_j \leq \dots$
2. T_i devrait augmenter sans limite.

Soit un ensemble de propositions atomiques Q , la formule ϕ de MTL est définie de manière inductive comme suit:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid O\varphi \mid \Box\varphi \mid \Diamond\varphi \mid \varphi_1 U \varphi_2 \mid O_{\sim c}\varphi \mid \Box_{\sim c}\varphi \mid \Diamond_{\sim c}\varphi \mid \varphi_1 U_{\sim c}\varphi_2$$

où $p \in Q$, et \sim est l'un de \leq, \geq et $=$.

Les opérateurs limités dans le temps sont $O_{\sim c}\varphi \mid \Box_{\sim c}\varphi \mid \Diamond_{\sim c}\varphi \mid \varphi_1 U_{\sim c}\varphi_2$, qui sont définis comme suit:

- $(\sigma, i) \models O_{\sim c}\phi$ ssi $(\sigma, i+1) \models \phi$ $T_{i+1} \sim T_i + c$;
- $(\sigma, i) \models \varphi_1 U_{\sim c}\varphi_2$ ssi $\exists k \geq i$ tels que $(\sigma, k) \models \varphi_2$ et $T_k \sim T_i + c$, et pour tout j , tels que $i \leq j < k$, $(\sigma, j) \models \varphi_1$;
- $(\sigma, i) \models \Diamond_{\sim c}\phi$ ssi $\exists k \geq i$ tels que $(\sigma, k) \models \phi$ and $T_k \sim T_i + c$; and
- $(\sigma, i) \models \Box_{\sim c}\phi$ ssi $\forall k \geq i$ tels que $(\sigma, k) \models \phi$ and $T_k \sim T_i + c$.

4.3.3 Logique temporelle métrique floue (FMTL)

Dans la logique temporelle métrique, le temps associé aux états et les contraintes de temps sont des valeurs de temps précises. Mais dans de nombreux systèmes de temps réel, les informations temporelles sont incertaines et imprécises. Comme dans l'exemple de partage de ressources, un point temporel demandé par un processus pour une ressource est flou, et une durée d'utilisation de la ressource est également floue. Pour traiter de telles incertitudes temporelles, les auteurs dans (Zhou & Murata, 1999) ont proposé de modifier MTL et d'introduire une nouvelle classe de MTL appelée logique temporelle métrique floue FMTL. L'objectif de la définition de FMTL consiste à l'appliquer comme une restriction sur les séquences de déclenchement de transition de FTN (Fuzzy Timing Net), puisque FMTL peut spécifier non seulement les propriétés de vivacité, d'équité, d'éventualité et en temps réel dans les systèmes concurrents, mais également les exigences incertaines.

Un réseau de PETRI à temporisation floue et à logique temporelle (FTLFTN) est défini comme un couple (FTN, ϕ) , où FTN est un réseau de PETRI à synchronisation floue et ϕ est une formule temporelle métrique floue. ϕ fonctionne comme une restriction sur les séquences de tir dans un FTN. Nous nous intéressons qu'aux séquences de tir satisfaisant aux exigences temps réel incertain

spécifiées par ϕ . Ainsi, seules les séquences de déclenchement qui satisfont ϕ sont autorisées. Il sera interdit aux autres de tirer.

Soit un ensemble de propositions atomiques Q , la formule ϕ de MTL est définie de manière inductive comme suit:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid O\varphi \mid \square\varphi \mid \diamond\varphi \mid \varphi_1 U \varphi_2 \mid \sim_d \varphi \mid O_{\sim_d} \varphi \mid \square_{\sim_d} \varphi \mid \diamond_{\sim_d} \varphi \mid \varphi_1 U_{\sim_d} \varphi_2$$

où $p \in Q$, et \sim est l'un de \leq, \geq et $=$.

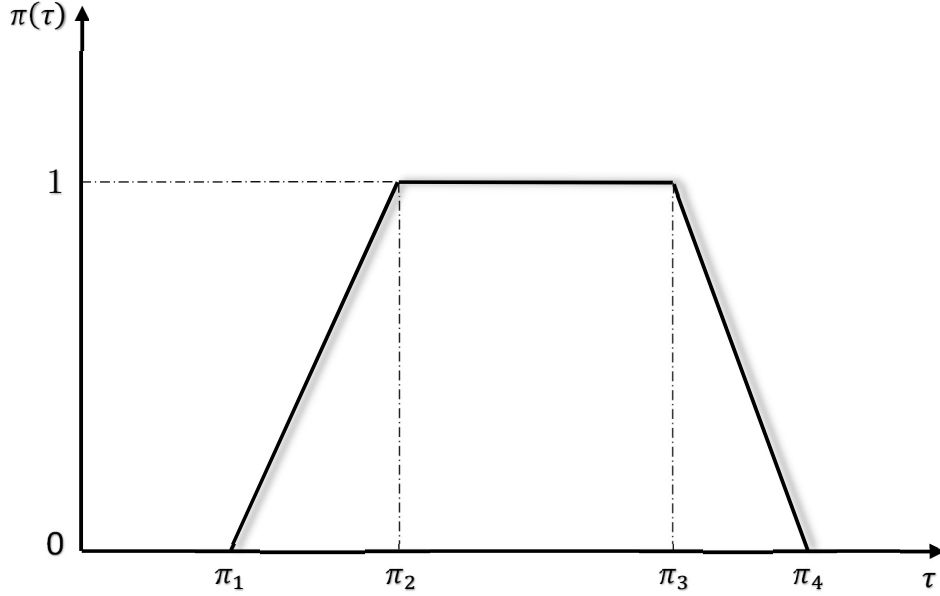


Fig. 15 Distribution de possibilité trapézoïdale

À l'exception de $\sim_d \varphi$, une formule dans FMTL ressemble à une formule dans MTL. Une différence est que le temps associé à chaque état et les contraintes de temps d peuvent être une fonction de temps flou exprimée dans une distribution de possibilité trapézoïdale (Fig. 15). Ainsi, étant donné deux distributions de possibilité π_e et π_f de deux dates floues e et f , nous avons besoin de définir les relations $e \leq f$, $e \geq f$ et $e = f$ entre les dates floues e et f , et de définir les possibilités de $e \leq f$, $e \geq f$ et $e = f$.

a) Possibilités de $e \leq f$, $e \geq f$ et $e = f$

Étant donné deux distributions possibles π_e et π_f de deux dates floues e et f , $e \leq f$ signifie que e a lieu avant f , $e \geq f$ signifie que e a lieu après f , et $e = f$ signifie que e a lieu en même temps que f . En (A) ci-dessous, nous rappelons la définition de « avant/après » donnée dans (DuBois &

Prade, 1989). Ensuite, dans (B), l'estimation de la possibilité d'évaluer l'étendue de la relation « avant/après » entre deux dates floues e et f .

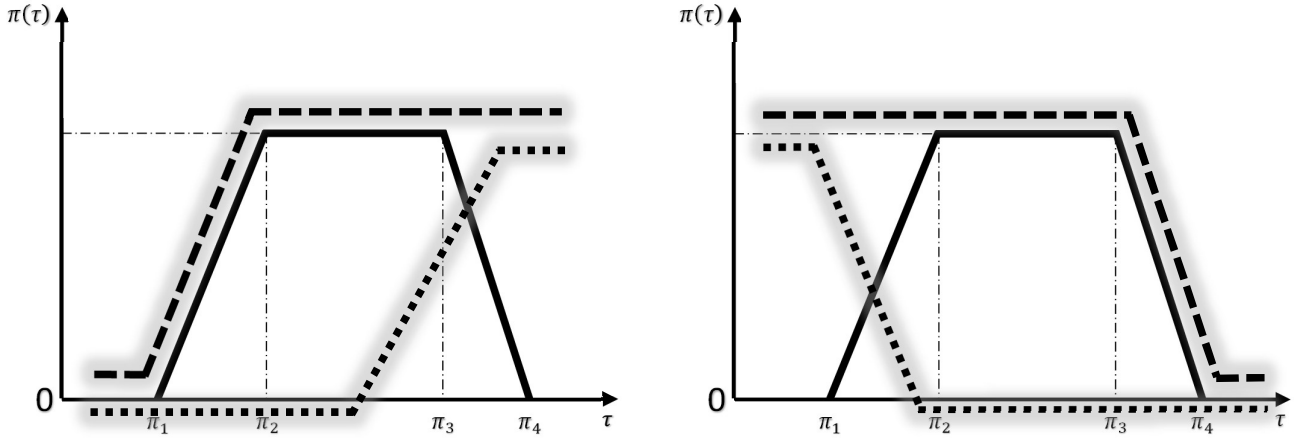


Fig. 16 Période éventuellement/nécessairement après/avant une date inconnue

(A) Une période après/avant une date. Étant donné une distribution unimodale de possibilités π_a d'une date floue a , les quatre ensembles suivants ont été définis dans (DuBois & Prade, 1989). Ils sont illustrés à la Fig. 16.

- L'ensemble flou $[A, +\infty)$ nombre de points temporels éventuellement postérieurs à la date a , définis par la fonction d'appartenance :

$$\forall \tau \in \Gamma, \text{ où } \Gamma \text{ est l'échelle de temps, } u_{[A, +\infty)}(\tau) = \sup_{s \leq \tau} \pi_a(s) .$$

- L'ensemble flou $]A, +\infty)$ des points de temps qui sont nécessairement strictement postérieurs à la date a , définie par la fonction d'appartenance:

$$\forall \tau \in \Gamma, \text{ où } \Gamma \text{ est l'échelle de temps, } u_{]A, +\infty)}(\tau) = \inf_{s \geq \tau} (1 - \pi_a(s)) .$$

- L'ensemble flou $(-\infty, A]$ de points temporels éventuellement situés avant la date a , définis par la fonction d'appartenance:

$$\forall \tau \in \Gamma, \text{ où } \Gamma \text{ est l'échelle de temps, } u_{(-\infty, A]}(\tau) = \sup_{s \geq \tau} \pi_a(s) = 1 - u_{]A, +\infty)}(\tau) .$$

- L'ensemble flou $(-\infty, A[$ de points temporels nécessairement strictement antérieurs à la date a , définis par la fonction d'appartenance:

$$\forall \tau \in \Gamma, \text{ où } \Gamma \text{ est l'échelle de temps, } u_{(-\infty, A[}(\tau) = \inf_{s \leq \tau} (1 - \pi_a(s)) = 1 - u_{[A, +\infty)}(\tau) .$$

(B) Les possibilités de $e \leq f$, $e \geq f$ et $e = f$. Étant donné deux distributions de possibilités π_e et π_f de deux dates floues e et f , la possibilité de $e \leq f$ est définie dans (DuBois & Prade, 1989) par

$$\Pi(e \leq f) = \sup_{s \leq \tau} \min\{\pi_e, \pi_f\} = \text{height}([E, F]) .$$

Et la nécessité de $e \leq f$ est de

$$N(e \leq f) = 1 - \sup_{s \leq \tau} \min\{\pi_e, \pi_f\} = \text{height}(]E, F[) .$$

où $[E, F] = [E, +\infty) \cap (-\infty, F]$, comme indiqué par le trait gras de la Fig. 17 (à gauche), et $]E, F[=]E, +\infty) \cap (-\infty, F[$, comme le montre la ligne en trait gras de la Fig. 17 (à droite). $\Pi(e \leq f)$ estime dans quelle mesure il existe au moins une paire de valeurs s et τ pour e et f compatibles

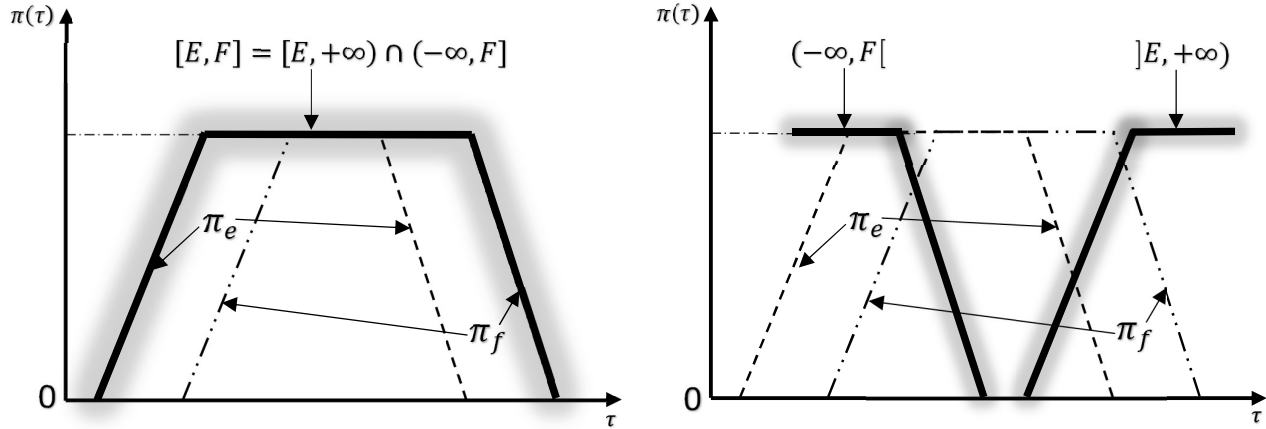


Fig. 17 à gauche : π_e, π_f et la ligne en gras montrée $[E, F]$, à droite : $]E, F[=]E, +\infty) \cap (-\infty, F[$, dans ce cas $]E, F[$ est vide.

avec π_e et π_f , respectivement, telles que $s \leq \tau$. $N(e \leq f)$ estime dans quelle mesure $]E, F[$ n'est pas vide.

La possibilité et la nécessité sont utilisées pour évaluer l'étendue du facteur $e \leq f$ dans (DuBois & Prade, 1989). Cependant, si on veut utiliser une valeur pour évaluer l'étendue de $e \leq f$, $\Pi(e \leq f)$ est trop faible, et $N(e \leq f)$ est trop strict. Comme le montre la Fig. 17 (à gauche), pour les e et f représentés, $\Pi(e \leq f) = \text{height}([E, F]) = 1$, Fig. 17 (à droite) montre que $N(e \leq f) = \text{height}(]E, F[) = 0$. La Fig. 18 (à gauche) montre que $\Pi(e \leq f) = \Pi(e \geq f) = \text{height}([E, F]) = 1$, et la Fig. 18 (à droite) montre que $N(e \leq f) = N(e \geq f) = \text{height}(]E, F[) = 0$.

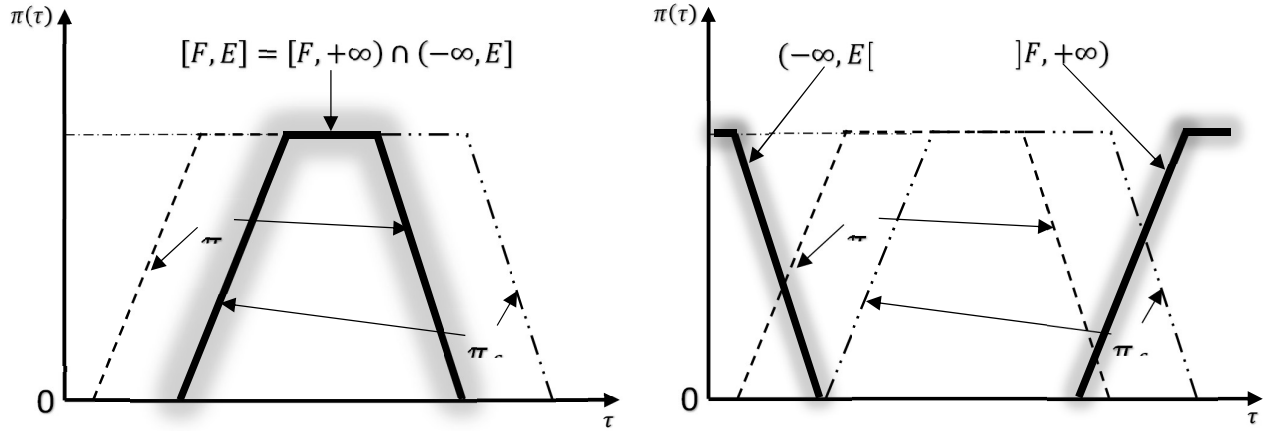


Fig. 18 à gauche : La ligne en gras montrée $[F, E] = [F, +\infty) \cap (-\infty, E]$, à droite : $]F, E[=]F, +\infty) \cap (-\infty, E[$, dans ce cas $]F, E[$ est vide.

b) Syntaxe de FMTL

Il existe trois types de propositions atomiques:

- $p_j \equiv$ place p_j a au moins un jeton avec son horodatage flou $\pi_{p_j}(\tau)$.
- $e_{t_j} \equiv$ transition t_j est activé avec son temps d'activation flou $e_{t_j}(\tau)$.
- $f_{t_j} \equiv$ transition t_j se déclenche avec son temps d'occurrence floue $o_{t_j}(\tau)$.

La formule FMTL est définie à l'aide des propriétés suivantes dans TLFTN:

$M_n \models p_j$ (la possibilité de p_j à M_n est 1) ssi $p_j \in M_n$, c'est-à-dire, $M_n(p_j) > 0$,

$M_n \models e_{t_j}$ (la possibilité de e_{t_j} à M_n est 1) ssi la transition t_j est quasi-activé à M_n (Murata, 1996)

c'est-à-dire, $*t_j \leq M_n^u$, où M_n^u est M_n avec l'attribut d'horodatage ignoré et $*t_j$ est l'ensemble d'entrée de la transition t_j . C'est-à-dire que t_j est quasi-activé s'il y a suffisamment de jetons dans chacune de ses positions d'entrée en M_n .

$M_n \models f_{t_j}$ (la possibilité de f_{t_j} à M_n est 1) ssi on atteint M_n par la transition de tir t_j .

On utilise les formules FMTL pour spécifier les exigences temporelles des séquences de déclenchement de transition. Si une séquence de déclenchement satisfait à la formule FMTL donnée, il s'agit d'une séquence de déclenchement autorisée. Sinon, il s'agit d'une séquence de déclenchement qui devrait être empêchée ou interdite.

Comme en MTL, chaque état d'une séquence de déclenchement de TLFTN est associé à un temps. Ainsi, une séquence de déclenchement est une séquence chronométrée. Une différence entre FMTL et MTL réside dans le fait que le temps associé à chaque état dans une séquence de déclenchement d'une FMTL est une fonction temporelle floue, au lieu d'un temps précis dans MTL.

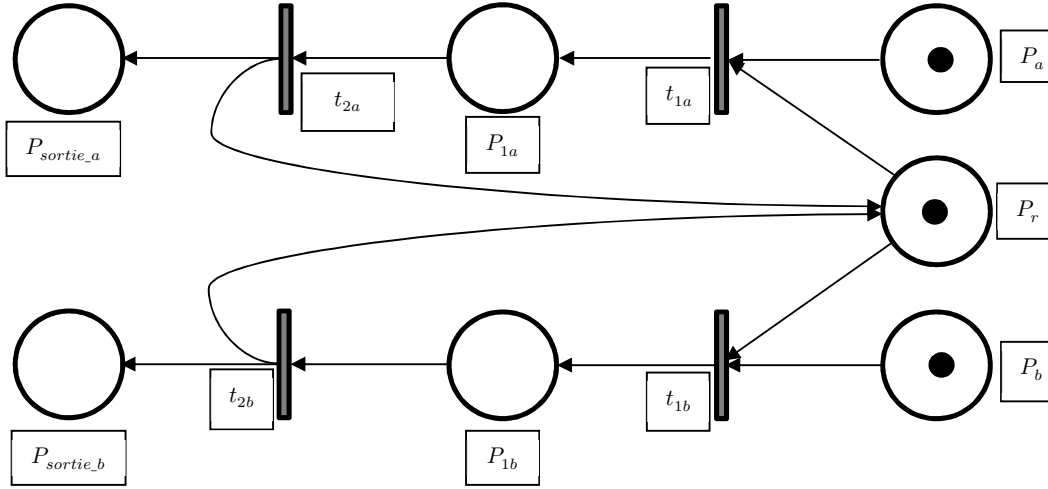


Fig. 19 Exemple d'une exclusion mutuelle dans laquelle une ressource commune P_r est partagée par deux processus.

Pour chaque état M_i d'une séquence de déclenchement de TLFTN, trois types de fonctions de temps flou sont associés:

1. $e_{t_j}(\tau)$, temps flou auquel la transition t_j est activée à M_i ;
2. $o_{t_j}(\tau)$, temps d'occurrence flou auquel l'état M_i commence par le déclenchement t_j ; et
3. $\pi_{p_k}(\tau)$, Horodatage flou auquel un jeton arrive à la position p après la transition de déclenchement t et après le retard flou de $d_{tp}(\tau)$.

Afin de conserver la propriété monotone croissante du temps, pour chaque état M_i accessible depuis l'état initial, un temps d'occurrence flou $o_{t_j}(\tau)$ est associé à M_i , auquel ce dernier commence, désigné par $Time_{M_i}(\tau)$. Et pour l'état initial M_0 , on associe $(0,0,0,0)$ à M_0 , noté $Time_{M_i}(\tau)$.

Il existe deux types de formules propositionnelles dans FMTL: l'une a une fonction de temps flou, par exemple, p_j a l'horodatage flou $\pi_{p_j}(\tau)$ a $e_{t_j}(\tau)$, et f_{t_j} a $o_{t_j}(\tau)$, qui a aussi une mesure scalaire, par exemple, la possibilité de p_j à M_n est 1 ssi $M_n(p_j) > 0$. Et l'autre n'a qu'une mesure scalaire de possibilité, telle que $\sim_d\varphi$, $0\sim_d\varphi$, $\square\sim_d\varphi$, $\diamond\sim_d\varphi$ et $\varphi_1 U\sim_d\varphi_2$.

Exemple :

Considérons la formule FMTL donnée par $\diamond_{\leq 10} P_{sortie_a}$ dans le FTN de la Fig. 19. Cette formule indique qu'il existe un état futur dans lequel un jeton se trouve dans P_{sortie_a} après la durée de l'état actuel à un jeton. L'arrivée à P_{sortie_a} est inférieure à 10 unité de temps. Calculons la possibilité de $\diamond_{\leq 1} P_{sortie_a}$ à M_0 , dans la séquence de tir $\sigma_1: M_0[t_{1a} \gg M_1[t_{2a} \gg M_2[t_{3a} \gg M_3[t_{1a} \gg M_4$. Depuis $M_2 \models P_{sortie_a}$ et l'horodatage flou auquel un jeton arrive à P_{sortie_a} est $\pi_{2a}(\tau) = (5,8,10,14)$. $\pi_{2a}(\tau)\Theta Time_{M_0}(\tau) = (5,8,10,14)\Theta(0,0,0,0) = (5,8,10,14)$, où Θ est la soustraction étendue, comme le montre la Fig. 20.

Ainsi, à M_0 , la possibilité de $\diamond_{\leq 10} P_{sortie_a}$ est donnée par $\frac{(la\ surface\ du\ trapèze\ avant\ \tau=10)}{(la\ surface\ du\ trapèze)} = \frac{3.5}{5.5} = 0.636$.

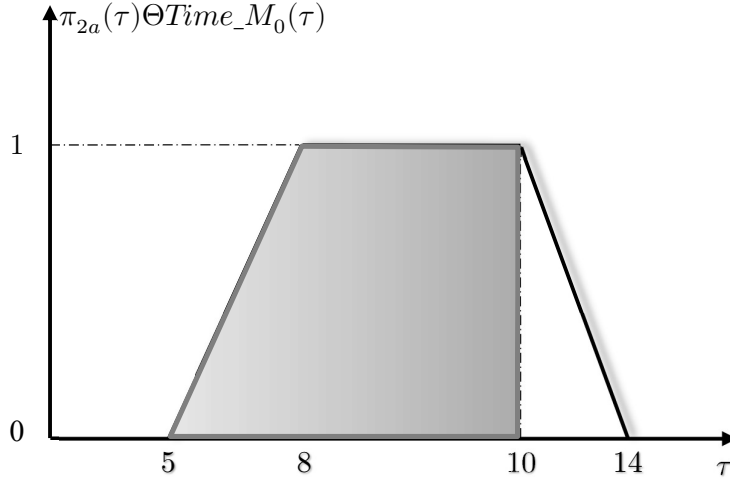


Fig. 20 La partie de (5, 8, 10, 14) avant $\tau = 10$ indiquée par la zone en dégradé

Θ est la soustraction étendue (DuBois & Prade, 1989) et est défini comme suit:

Soit deux fonctions temporelles floues $\pi_a(\tau) = (a_1, a_2, a_3, a_4)$ et $\pi_b(\tau) = (b_1, b_2, b_3, b_4)$, $D = B\Theta A$, dont la fonction d'appartenance est :

$$\forall \tau \in \Gamma, \text{ où } \Gamma \text{ est l'échelle de temps, } \pi_d(\tau) = \sup_{\tau=\tau_2-\tau_1} \min\{\pi_a(\tau_1), \pi_b(\tau_2)\}.$$

Pour les distributions de possibilité trapézoïdales, la soustraction étendue Θ est simplifiée et donnée par :

$$\pi_d(\tau) = \pi_b(\tau)\Theta\pi_a(\tau) = (b_1, b_2, b_3, b_4)\Theta(a_1, a_2, a_3, a_4) = (b_1 - a_4, b_2 - a_3, b_3 - a_2, b_4 - a_1).$$

4.3.4 Grammaire anglaise structurée

Afin de faciliter l'utilisation des patrons de spécification, des travaux dans la littérature ont proposé une grammaire anglaise structurée (illustrée dans 3.4) qui peut prendre en charge les patrons de spécification qualitatifs, temps réel, probabiliste et auto-adaptatifs. Dans la grammaire, les terminaux littéraux sont écrit en gras, les terminaux non littéraux sont indiqués entre deux accolades et capitaliser chaque mot pour les non-terminaux.

Le symbole de départ de la grammaire est la propriété et le langage $L(G)$ de la grammaire est finie, car la grammaire est non circulaire et ne se répète pas. Chaque phrase (ou chaîne) s avec $s \in L(G)$ sert de descripteur qui accompagne une formule étendue d'un patron de spécification. Par conséquent, la grammaire aide à comprendre le sens d'une propriété sans avoir à analyser la représentation de la logique temporelle. Les utilisateurs sont capables de dériver des représentations en langage naturel de propriétés qualitatives, temps réel et auto-adaptatives spécifiées en termes de logiques temporelles. Les réactions de l'industrie ont indiqué qu'une représentation anglaise structurée est moins intimidante que la notation de la logique temporelle. L'expérience a montré qu'une exposition directe aux spécifications formelles entrave souvent l'utilisation généralisée de méthodes formelles (Dwyer, Avrunin, & Corbett, 1999). La grammaire anglaise structurée est organisée en fonction de classification des patrons (Konrad & Cheng, 2005). En général, la procédure d'utilisation de grammaire pour créer une représentation en langage naturel est la suivante: Initialement, l'utilisateur choisit la portée de la propriété spécifiée (globalement, avant, après, entre ou après jusqu'à), suivi du type (qualitatif, temps réel ou auto-adaptatif). Ensuite, la catégorie de la propriété spécifiée est sélectionnée (durée, ordre périodique ou en temps réel pour les propriétés en temps réel et occurrence ou ordre flou pour les propriétés auto-adaptatives). La phrase anglaise structurée finale est construite en choisissant le modèle de spécification correspondant.

Par exemple, l'utilisateur souhaite créer une représentation en langage naturel pour une propriété de réponse liée capturée par la formule FMTL suivante:

$$(a = 0) \rightarrow \diamond_{\leq 10}(b = 1)$$

Initialement, l'utilisateur tire la phrase suivante de la grammaire (les règles de grammaire sont données entre parenthèses):

« Globalement, il est toujours vrai que si A est valide, alors B reste après au plus f unité (s) de temps. »

Cette phrase correspond au modèle FMTL à portée globale suivant du modèle de réponse lié:

$$A \rightarrow \diamond_{\leq f} B$$

Les terminaux non littéraux doivent être instanciés pour la représentation en langage naturel. A et B désignent des formules propositionnelles booléennes décrivant les propriétés des états et utilisées pour capturer les propriétés du système ainsi que pour désigner les états servant de limites aux portées. Pour les propriétés auto-adaptatives, f doit également être instancié avec les valeurs entières utilisées avec les opérateurs contraints dans une formule de logique temporelle floue. Après avoir remplacé A par $(a = 0)$, B par $(b = 1)$ et f par 10, la représentation finale en langage naturel de la propriété susmentionnée est obtenue:

« Globalement, il est toujours vrai que si $(a = 0)$ est valide, alors $(b = 1)$ reste après au plus 10 unité de temps ».

Dans cette section, j'ai présenté les concepts de base pour PSAS (la définition d'un patron de spécification ainsi que sa portée). J'ai également introduit la logique temporelle pour la cartographie des expression de PSAS. Dans ce qui suit, nous présentons le langage PSAS avec ses briques de base.

4.4 Conclusion

Dans ce chapitre, nous avons présenté les éléments de base sur lesquels notre approche est basée, en particulier les patrons de spécification qui représente une approche prometteuse dans la spécification des exigences. Nous avons également présenté certaines logiques temporelles utilisés comme cartographie pour les patrons de spécification notamment la logique FMTL qui représente le support formel de notre approche. Dans le chapitre suivant, nous allons présenter le langage proposé avec son vocabulaire, grammaire et sémantique.

Chapitre 5

PSAS : Les patrons de spécification pour l'auto-adaptation

5.1 Introduction

Dans ce chapitre, nous présentons notre langage de spécification qui permet aux ingénieurs d'exprimer les propriétés du système de manière flexible. Les patrons sont conçus pour permettre aux concepteurs de savoir que l'exigence peut être modifiée durant l'exécution lorsqu'un changement environnemental se produit. Le système doit être capable d'ignorer temporairement les exigences non critiques afin de s'assurer que les exigences essentielles puissent être satisfaites.

5.2 Catalogue de patron auto-adaptatif

Les patrons de spécification sont organisés en deux catégories : l'occurrence et l'ordre. La première catégorie présente la matérialisation de l'événement tandis que la seconde capture une séquence d'événements. Nous proposons dans ce papier un catalogue de patrons auto-adaptatifs. Ils peuvent être considérés comme des versions soulagées de ceux existants (Yahiaoui, Bendjenna, Roose, Chung, & Mohamed, 2019), par exemple : « *TheLongestPossible* » peut être considéré comme une version soulagée du patron « *Universality* » (Dwyer, Avrunin, & Corbett, 1999). La classification des patrons proposés est la même que celle présente dans (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015). Les patrons sont divisés en deux classes principales, l'occurrence auto-adaptative et l'ordre auto-adaptatif.

5.2.1 Patrons auto-adaptatifs d'occurrence

Utilisés pour affirmer de manière flexible qu'une certaine configuration de propriétés doit toujours, éventuellement avoir lieu ou ne doit pas se produire.

- *TheLongestPossible* : ce patron est proposé afin de gérer des situations où la propriété doit être vraie le plus longtemps possible.
- *TheShortestPossible* : ce patron est à l'opposé du premier, il décrit une propriété devant être vraie sur une période la plus courte possible.
- *TheEarliestPossible* : ce patron a pour objectif que la propriété se produise le plus tôt possible.
- *TheLatestPossible* : contrairement, ce patron exprime la propriété qui se produit le plus tard possible.

- *SA-MinDuration* : ce patron implique que la propriété est vraie pendant un temps minimum assoupli pour permettre à la propriété de tenir en dessous.
- *SA-MaxDuration* : ce patron indique que la propriété doit tenir sous un seuil de temps, le seuil est assoupli pour permettre à la propriété de tenir au-dessus.
- *TheClosestPossibleTo-q* : est utilisé pour exprimer la quantité (q) de manière polyvalente, où les valeurs qui l'entourent peuvent être tolérées.
- *As {many/few} as possible* : cette expression est utilisée pour exprimer le plus près possible d'une quantité idéale et indéterminée.
- *SA-recurrence* : Analogue à «TheClosest-PossibleTo-q». Ce modèle présente la fréquence de détention des propriétés, c'est-à-dire que la propriété détient le plus possible chaque fois que t (t est une durée).
- *Alternative* : utilisé pour spécifier des alternatives pour un événement/état particulier, ce modèle est utile pour la spécification de reconfiguration du système.
- *Alternative-SA-recurrence* : il s'agit d'un patron hybride utilisé pour exprimer des versions relâchées d'un état temporel.

5.2.2 Patrons auto-adaptatifs d'ordre

Ils sont utilisés pour spécifier l'ordre dans lequel certaines propriétés doivent se produire. Nous proposons une version auto-adaptative pour les deux patrons « *Until* » et « *Response* ».

- *SA-Until* : le patron « *Until* » est natif dans la plupart des logiques temporelles. Il a été étendu dans (Grunske, 2008), afin de gérer les propriétés temporisées. Nous proposons une version soulagée de ce dernier avec la cartographie liée à chaque portée, où le temps associé est flou.
- *TheEarliestPossible-Response* : ce patron indique qu'à chaque fois qu'une propriété X est vraie, elle doit être suivie par une autre propriété Y, aussitôt que possible (seule la première occurrence de la seconde propriété est considérée).
- *The{Earliest/Latest}Possible-Response (1-N | N-1)* : ce patron signifie qu'à chaque fois qu'une propriété X (stimulus) est vérifiée, elle doit être poursuivie (causée) par la chaîne de propriétés Y_i $1 \leq i \leq n$ (réponses), le plus tôt|tard possible.

Le catalogue de patrons présenté ci-dessus vise à apporter plus d'expressivité durant la phase de spécifications. Il est généré à partir de la grammaire présentée dans ce qui suit, sa sémantique est également présentée dans la section « Sémantique du langage ».

5.3 Syntaxe du langage

La formulation des spécifications est soumise à la grammaire anglaise structurée présentée dans ce qui suit. Le Tableau 9 illustre la première étape du processus de spécification. Chaque propriété est formée de deux éléments, une portée « scope » et un patron « Self-Adaptive-Pattern ». Pour la première partie, nous choisissons l'un des mots-clés suivants: « Globally », « After », « Before », « Between » et « After-until ». En ce qui concerne la partie patron, elle est générée via un non-terminal (partie remplaçable dans une grammaire écrite en « italique »). Ce dernier génère deux non-terminaux, « Self-Adaptive-Occurrence » ou « Self-Adaptive-Order », en fonction de ce que l'on cherche à exprimer (Yahiaoui, Bendjenna, Roose, Chung, & Mohamed, 2019). Par exemple, si une séquence d'événements spécifique est souhaitée, « Self-Adaptive-Order » est utilisé, après on choisit le non-terminal « TheEarliestPossible-Response », et enfin on remplace ce dernier par sa formule « If { événement } then in response {propriété} holds as early as possible ». Le Tableau 10 présente l'anglais structuré (généré à partir des modèles non-terminaux). Il contient des phrases fixes telles que « It is the case that », des parties facultatives entre crochets, telles que « [Time(A)] » et des parties correspondant à l'exigence d'origine entre « {} », ex: « It is the case that {Les systèmes de photovoltaïque terrestre (PV) font face au soleil} ». Le Tableau 11 détaille les définitions de temps et de quantité.

Tableau 9 Génération de patron.

<i>Property</i>	::=	<i>Scope, Self-Adaptive-Pattern</i>
<i>Scope</i>	::=	Globally Before {B} After {C} Between {C} and {B} After {C} until {B}
<i>Self-Adaptive-Pattern</i>	::=	<i>Self-adaptive-Occurrence / Self-adaptive-Order</i>
<i>Self-Adaptive-Occurrence</i>	::=	<i>TheLongestPossible / TheShortestPossible / TheEarliestPossible / TheLatestPossible / SA-MinDuration / SA-MaxDuration</i>
<i>Self-Adaptive-Order</i>	::=	<i>SA-Until / TheEarliestPossibleResponse</i>

Tableau 10 Anglais structuré pour les patrons auto-adaptatifs.

<i>TheLongestPossible</i>	::=	It is the case that {A} [holds] [as long] as possible [for t_A <i>TimeUnits</i>] [<i>Time (A)</i>].
<i>TheShortestPossible</i>	::=	It is the case that {A} [holds] [as short] as possible [for t_A <i>TimeUnits</i>] [<i>Time (A)</i>].
<i>TheEarliestPossible</i>	::=	It is the case that {A} [holds] [as early] as possible [<i>Time (A)</i>].
<i>TheLatestPossible</i>	::=	It is the case that {A} [holds] [as late] as possible [<i>Time (A)</i>].
<i>SA-MinDuration</i>	::=	Once {A} [becomes satisfied] it remains as possible up to t_u^A <i>TimeUnits</i> .
<i>SA-MaxDuration</i>	::=	Once {A} [becomes satisfied] it remains as possible less than t_u^A <i>TimeUnits</i> .
<i>SA-Recurrence</i>	::=	{A} [holds] repeatedly almost every t_u^A <i>TimeUnits</i> .
<i>Alternative</i>	::=	May {A ₀ } [holds] or May {A ₁ } [holds] or May {A _i }
<i>Alternative-recurrence</i>	::=	May {A ₀ } [holds] or May {A ₁ } [holds] or May {A _j }, where A _j [holds] repeatedly almost every T _j <i>TimeUnits</i> .
<i>TheClosestPossibleT o-q</i>	::=	{A} is as close as possible to q <i>QuantityUnits</i> .
<i>As many as possible</i>	::=	{A} [holds] as many as possible.
<i>As few as possible</i>	::=	{A} [holds] as few as possible.
<i>SA-Until</i>	::=	{A} [holds] without interruption almost until {D} [holds] [<i>Time(A)</i>].
<i>TheEarliestPossible-Response</i>	::=	If {A} [has occurred] then in response {D} [holds] as early as possible [<i>Time (D)</i>].
<i>TheEarliestPossible-Response (1-N)</i>	::=	If {A} [has occurred] then in response {D} [holds] as early as possible [<i>Time (D)</i>] followed by ($\{T_i\} [Time (T_i)]$) ^(1≤i≤N-1; ",") [eventually hold].

<i>TheEarliestPossible-Response (N-1)</i>	::=	If {S} followed by $(\{T_i\} [Time(T_i)])^{(1 \leq i \leq N-1; ",")}$ [have occurred] then in response {A} [holds] as early as possible [<i>Time(A)</i>]
<i>TheLatestPossible-Response</i>	::=	If {A} [has occurred] then in response {D} [holds] as late as possible [<i>Time (D)</i>].
<i>TheLatestPossible-Response (1-N)</i>	::=	If {A} [has occurred] then in response {D} [holds] as late as possible [<i>Time (D)</i>] followed by $(\{T_i\} [Time (T_i)])^{(1 \leq i \leq N-1; ",")}$ [eventually hold]
<i>TheLatestPossible-Response (N-1)</i>	::=	If {D} followed by $(\{T_i\} [Time (T_i)])^{(1 \leq i \leq N-1; ",")}$ [have occurred] then in response {A} [holds] as late as possible [<i>Time(A)</i>]

Tableau 11 Définitions de temps et de quantité.

<i>Time (A)</i>	::=	<i>UpperTimeBound (A) LowerTimeBound (A) Interval (A)</i>
<i>UpperTimeBound (A)</i>	::=	Within t_u^A <i>TimeUnits</i>
<i>LowerTimeBound (A)</i>	::=	After t_l^A <i>TimeUnits</i>
<i>Interval (A)</i>	::=	Between t_l^A and t_u^A <i>TimeUnits</i>
<i>TimeUnits</i>	::=	Any denomination of time (e.g., seconds, minutes, hours, days, or years)
<i>QuantityUnits</i>	::=	Any denomination of quantity (e.g., number of connected devices)

5.4 Sémantique du langage

La sémantique de PSAS est définie en termes de Logique Métrique Temporelle Floue (FMTL) (Zhou & Murata, 1999). FMTL peut décrire un réseau de pétri temporel avec des informations temporelles incertaines. C'est la représentation de l'incertitude dans FMTL qui la rend approprié comme formalisme pour notre langage. Par exemple, la déclaration « After{B},{A} holds as early as possible », que « A » exprime une exigence qui survient après l'apparition de l'événement « B », mais il est incertain combien de temps il faut pour que « A » se produit après l'événement « B ». La déclaration exprime simplement le désir de la période après l'occurrence de « B » soit

aussi petite que possible. Une logique avec une incertitude intégrée est donc nécessaire pour formaliser la sémantique de PSAS.

Un ensemble flou est un ensemble dont les éléments ont des degrés d'appartenance. Dans la théorie des ensembles classiques, un membre appartient à un ensemble ou non. La théorie des ensembles flous permet l'évaluation progressive de l'appartenance à des éléments dans un ensemble, qui est décrit en utilisant une fonction d'appartenance dans la plage des nombres réels $[0,1]$. En d'autres termes, un ensemble flou est une paire (A, m) où A est un ensemble et $m: A \rightarrow [0,1]$. Un nombre flou est un sous-ensemble flou de nombres réels dont la fonction d'appartenance est convexe et normalisée, c'est-à-dire $\max(m(a)) = 1$. Un nombre flou peut être triangulaire ou trapézoïdale, dans le sens où son graphique d'appartenance décrit un triangle ou un trapèze avec un Vertex montrant l'appartenance à 1. Par exemple, un nombre flou « 2 », la valeur précise du nombre est incertaine, en d'autres termes, le nombre représente environ 2. La fonction d'appartenance trapézoïdale indique que toute valeur inférieure à 1,5 ou supérieure à 2,5 n'est certainement pas considérée comme approximativement 2, que $[1.75, 2.25]$ est absolument considéré comme étant environ 2, alors que les valeurs comprises entre 1.5 et 2.5 sont environ 2 avec des degrés de vérité différents. La notion de nombre flou est facilement étendue à une durée floue. La durée $d \in \mathbb{R}^+$ est une durée floue s'il existe une incertitude sur la durée exacte. C'est-à-dire qu'il est associé à un nombre flou qui définit une longueur de temps floue.

« \square » est l'opérateur « toujours » habituel. « \mathbf{U} » désigne « jusqu'à » comme avec la logique temporelle standard. « \mathbf{O} » qui prend la valeur de vérité de sa formule après une durée. La durée $d \in \mathbb{R}^+$ peut-être une durée floue ou non. L'expression $\mathbf{O}_{=d}$ signifie « après exactement d », $\mathbf{O}_{<d}$ représente « avant que d soit passé », Et $\mathbf{O}_{>d}$ est « après que d soit passé ». Par conséquent, si d est flou, l'opérateur de délai peut être utilisé pour exprimer des relations avec un intervalle de temps incertain. Les notations abrégées habituellement utilisées sont également disponibles. En particulier, $\diamond A = \text{vraie } \mathbf{U} A$, où \diamond signifie finalement.

On est maintenant prêts à définir la sémantique des expressions PSAS en termes de FMTL. Les définitions pour ces opérateurs, y compris l'incertitude, dépendent d'une durée floue ou d'un ensemble flou. Ceux-ci ont généralement un maximum à un point particulier, puis se replient progressivement à l'infini, c'est-à-dire qu'ils ont un graphique d'appartenance trapézoïdale qui est asymptotique. Par exemple, dans le cas de « TheEarliestPossible », la fonction d'adhésion à son maximum à l'instant actuel. Cependant, le patron « TheEarliestPossible » techniquement permet de devenir vrai à tout moment après l'instant actuel. Par conséquent, la fonction d'appartenance pour la durée n'est jamais nulle mais approche de zéro progressivement à mesure que le temps augmente.

Dans PSAS, les patrons sont des versions étendues, en majeure partie, de ceux présentés dans (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015), ces patrons sont une version relâchée de ceux qui existent, la cartographie est étendue à partir de (Dwyer, Avrunin, & Corbett, 1999; Konrad & Cheng, 2005; Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015) avec des contraintes temporelles floues. Pour les patrons comme « TheLongestPossible » ou « TheEarliestPossible », il n'y a pas de cartographie existante. Nous proposons une nouvelle cartographie pour eux en FMTL. Ci-dessous, nous montrons la cartographie détaillée de notre catalogue de patrons. Dans ce qui suit, la valeur de vérité de chaque formule est une valeur de possibilité $\in [0,1]$. Le seuil de vérité peut être défini en fonction des besoins dans $[0,1]$.

t_A	=	t_A si t_A est défini sinon $+\infty$
t_l^A	=	t_l^A si t_l^A est défini sinon 0
t_u^A	=	t_u^A si t_u^A est défini sinon $+\infty$
$[[\text{time_A}]]$	=	$[t_l^A, t_u^A]$
$[[\text{gap_A}]]$	=	$t_u^A - t_l^A$
$[[\text{tl_A}]]$	=	t_l^A
$[[\text{tu_A}]]$	=	t_u^A
$[[\text{end_A}]]$	=	$[[\text{time_A}]] + t_A$

5.4.1 Famille d'occurrence auto-adaptative

a) Cartographie de « *TheLongestPossible* »

La cartographie de ce patron est basé sur celle du patron d'universalité dans (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015), la différence apparait dans la valeur de vérité de chaque formule qui appartient à $[0,1]$.

Globally	$\square_{[[\text{time_A}]]}(A \wedge \diamond_{\leq[[\text{end_A}]]}\neg A)$
Before B	$\diamond_{\geq[[\text{tl_A}]]}B \rightarrow \left((A \wedge \diamond_{\leq[[\text{end_A}]]}\neg A) \cup_{[[\text{time_A}]]} B \vee \square_{[[\text{time_A}]]}(A \wedge \diamond_{\leq[[\text{end_A}]]}\neg A) \right)$
After C	$\square \left(C \rightarrow \square_{[[\text{time_A}]]}(A \wedge \diamond_{\leq[[\text{end_A}]]}\neg A) \right)$
Between C and B	$\square \left((C \wedge \square_{\leq[[\text{tl_A}]]}\neg B) \wedge \diamond_{\geq[[\text{tl_A}]]}B \rightarrow \left((A \wedge \diamond_{\leq[[\text{end_A}]]}\neg A) \cup_{[[\text{time_A}]]} B \vee \square_{[[\text{time_A}]]}(A \wedge \diamond_{\leq[[\text{end_A}]]}\neg A) \right) \right)$

After C until B	$\Box \left((C \wedge \Box_{\leq [tl_A]} \neg B) \rightarrow \left((A \wedge \Diamond_{\leq [end_A]} \neg A) W_{[time_A]} B \right) \right)$
--------------------	---

b) Cartographie pour « *TheShortestPossible* »

La cartographie de ce patron est basé sur celle du patron d'absence dans (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015), la différence est dans la valeur de vérité de chaque formule qui appartient à [0,1].

Globally	$\Diamond_{[time_A]} (A \wedge \Diamond_{\leq [end_A]} \neg A)$
Before B	$\neg B W_{[time_A]} \left((A \wedge \Diamond_{\leq [end_A]} \neg A) \wedge \neg B \right)$
After C	$\Box \neg C \vee \Diamond \left(C \wedge \Diamond_{[time_A]} (A \wedge \Diamond_{\leq [end_A]} \neg A) \right)$
Between C and B	$\Box \left((C \wedge \Box_{\leq [tl_A]} \neg B) \wedge \Diamond_{\geq [tl_A]} B \rightarrow \left(\neg B W_{[time_A]} \left((A \wedge \Diamond_{\leq [end_A]} \neg A) \wedge \neg B \right) \right) \right)$
After C until B	$\Box \left((C \wedge \Box_{\leq [tl_A]} \neg B) \rightarrow \left(\neg B U_{[time_A]} \left((A \wedge \Diamond_{\leq [end_A]} \neg A) \wedge \neg B \right) \right) \right)$
Where :	$[end_A] = [time_A]$

c) Cartographie pour « *TheEarliestPossible* » et « *TheLatestPossible* »

La cartographie pour ces deux patrons est la même, pour le patron *TheEarliestPossible*, la valeur de vérité est égal à 1 quand $[tl_A] = 0$. Par ailleurs, le patron *TheLatestPossible* aura sa valeur de vérité égal à 0 quand $[tl_A] = 0$.

Globally	$O_{=[tl_A]} A$
Before B	$\Diamond_{\geq [tu_A]} B \rightarrow (A U_{[time_A]} B \vee O_{=[tl_A]} A)$
After C	$\Box (C \rightarrow O_{=[tl_A]} A)$
Between C and B	$\Box \left((C \wedge \Box_{\leq [tu_A]} \neg B) \wedge \Diamond_{\geq [tu_A]} B \rightarrow (A U_{[time(A)]} B \vee O_{=[tl_A]} A) \right)$
After C until B	$\Box \left((C \wedge \Box_{\leq [tu_A]} \neg B) \rightarrow O_{=[tl_A]} (A W_{[time_A]} B) \right)$

d) Cartographie pour « SA-MinDuration »

La cartographie de ce patron est basé sur celle du patron min-duration dans (Konrad & Cheng, 2005), la validité du patron est toujours entre 0 et 1.

Globally	$\Box (A \vee (\neg A W_{\leq [tu_A]} A))$
Before B	$\Diamond B \rightarrow \left(\left(A \vee (\neg A U (\Box_{\leq [tu_A]} (A \wedge \neg B) \vee B)) \right) \right) U B$
After C	$\Box (C \rightarrow \Box (A \vee (\neg A W_{\leq [tu_A]} A)))$
Between C and B	$\Box \left(\begin{array}{c} (C \wedge \neg B \wedge \Diamond B) \\ \rightarrow \left(\left(A \vee (\neg A U (\Box_{\leq [tu_A]} (A \wedge \neg B) \vee B)) \right) \right) U B \end{array} \right)$
After C until B	$\Box \left((C \wedge \neg B) \rightarrow \left(\left(A \vee (\neg A W (\Box_{\leq [tu_A]} (A \wedge \neg B) \vee B)) \right) \right) W B \right)$

e) Cartographie pour « SA-MaxDuration »

La cartographie de ce patron est basé sur celle du patron max-duration dans (Konrad & Cheng, 2005), la validité du patron peut avoir une valeur entre 0 et 1.

Globally	$\Box (A \vee (\neg A W (A \wedge \Box_{\leq [tu_A]} \neg A)))$
Before B	$\Diamond B \rightarrow \left(\left(A \vee (\neg A U (A \wedge \Diamond_{\leq [tu_A]} (\neg A \wedge B) \vee B)) \right) \right) U B$
After C	$\Box (C \rightarrow \Box (A \vee (\neg A W (A \wedge \Box_{\leq [tu_A]} \neg A))))$
Between C and B	$\Box \left(\begin{array}{c} (C \wedge \neg B \wedge \Diamond B) \\ \rightarrow \left(\left(A \vee (\neg A U (P \wedge \Diamond_{\leq [tu_A]} (\neg A \wedge B) \vee B)) \right) \right) U B \end{array} \right)$
After C until B	$\Box \left(\begin{array}{c} (C \wedge \neg B) \\ \rightarrow \left(\left(A \vee (\neg A W (P \wedge \Diamond_{\leq [tu_A]} (\neg A \wedge B) \vee B)) \right) \right) W B \end{array} \right)$

f) Cartographie pour « *TheClosestPossibleTo-q* » et « *as (many/few) as possible* »

Globally	$\diamond A$
Before B	$\neg B \text{ W } (A \wedge \neg B)$
After C	$\square \neg C \vee \diamond (C \wedge \diamond A)$
Between C and B	$\square ((C \wedge \neg B \wedge \diamond B) \rightarrow (\neg B \text{ W } (A \wedge \neg B)))$
After C until B	$\square ((C \wedge \neg B) \rightarrow (\neg B \text{ U } (A \wedge \neg B)))$
Où:	
\mathbb{Q} est un ensemble flou, m est sa fonction d'appartenance, et Δ est une fonction pour le quantifiable dans une propriété donnée A .	
Pour : <i>TheClosestPossibleTo-q</i>	
$A = (\Delta(A) - q) \in \mathbb{Q}$, $m()$ a sa valeur max à 0 ($m(0) = 1$) et décroît continuellement autour de 0.	
Pour : <i>As few as possible</i>	
$A = \Delta(A) \in \mathbb{Q}$, $m()$ a sa valeur max à 0 ($m(0) = 1$) et décroît continuellement au-dessus de 0.	
Pour : <i>As many as possible</i>	
$A = \Delta(A) \in \mathbb{Q}$, $m()$ a sa valeur min à 0 ($m(0) = 0$) et augmente continuellement jusqu'à ∞ .	
Pour : <i>SA-MinQuantity</i>	
$A = \Delta(A) \in \mathbb{Q}$, $m()$ a sa valeur max à A ($m(A) = 1$) et décroît continuellement au-dessus A .	
Pour : <i>SA-MaxQuantity</i>	
$A = \Delta(A) \in \mathbb{Q}$, $m()$ a sa valeur max à A ($m(A) = 1$) et décroît continuellement <u>au-dessous</u> A .	

g) Cartographie pour « *SA-Recurrence* »

La cartographie de ce patron étend celle du patron « *Recurrence* » (Konrad & Cheng, 2005; Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015), avec $\llbracket tu_A \rrbracket$ durée floue.

Globally	$\square (\diamond_{\leq \llbracket tu_A \rrbracket} A)$
Before B	$\diamond B \rightarrow \left(\left(\diamond_{\leq \llbracket tu_A \rrbracket} (A \vee B) \right) \text{ U } B \right)$
After C	$\square \left(C \rightarrow \square (\diamond_{\leq \llbracket tu_A \rrbracket} A) \right)$
Between C and B	$\square \left((C \wedge \neg B \wedge \diamond B) \rightarrow \left(\diamond B \rightarrow \left(\left(\diamond_{\leq \llbracket tu_A \rrbracket} (A \vee B) \right) \text{ U } B \right) \right) \right)$
After C until B	$\square \left((C \wedge \neg B \wedge \diamond B) \rightarrow \left(\left(\diamond_{\leq \llbracket tu_A \rrbracket} (A \vee B) \right) \text{ W } B \right) \right)$

h) Cartographie pour « Alternative » et « Alternative-SA-recurrence »

$\forall i, i \in \{1, 2, \dots, n\}$, tu_{A_i} est une durée foule.

Globally	$\bigvee_{i=1}^n \square_{\llbracket time_{A_i} \rrbracket} A_i$
Before B	$\bigvee_{i=1}^n \left(\diamond_{\geq \llbracket tl_{A_i} \rrbracket} B \rightarrow (A_i U_{\llbracket time_{A_i} \rrbracket} B \vee \square_{\llbracket time_{A_i} \rrbracket} A_i) \right)$
After C	$\bigvee_{i=1}^n \square (C \rightarrow \square_{\llbracket time_{A_i} \rrbracket} A_i)$
Between C and B	$\bigvee_{i=1}^n \left(\square \left((C \wedge \square_{\leq \llbracket tl_{A_i} \rrbracket} \neg B) \wedge \diamond_{\geq \llbracket tl_{A_i} \rrbracket} B \rightarrow (A_i U_{\llbracket time_{A_i} \rrbracket} B \vee \square_{\llbracket time_{A_i} \rrbracket} A_i) \right) \right)$
After C until B	$\bigvee_{i=1}^n \left(\square (C \wedge \square_{\leq \llbracket tl_{A_i} \rrbracket} \neg B) \rightarrow A_i W_{\llbracket time_{A_i} \rrbracket} B \right)$
Pour : Alternative-SA-recurrence	$A_1 = A_2 = A_3 = \dots = A_n, \text{ et } \forall i, i \in \{1, 2, \dots, n\}, \quad tu_{A_i} \leq tl_{A_{i+1}}$

5.4.2 Famille d'ordre auto-adaptatif

a) La cartographie pour « SA-until »

La cartographie de ce patron étend celle du patron Until (Konrad & Cheng, 2005) avec valeur de vérité dans $[0, 1]$.

Globally	$A U_{\llbracket time_A \rrbracket} D$
Before B	$\diamond_{\geq \llbracket tl_A \rrbracket} B \rightarrow \left((A \wedge \neg B) U_{\llbracket time_A \rrbracket} (D \vee B) \right)$
After C	$\square \left(C \rightarrow (A U_{\llbracket time_A \rrbracket} D) \right)$
Between C and B	$\square \left((C \wedge \neg B \wedge \diamond_{\geq \llbracket tl_D \rrbracket} B) \rightarrow \left((A \wedge \neg B) U_{\llbracket time_A \rrbracket} (D \vee B) \right) \right)$
After C until B	$\square \left((C \wedge \square_{\leq \llbracket tl_D \rrbracket} \neg B) \rightarrow \left((A U_{\llbracket time_A \rrbracket} D) W B \right) \right)$

Pour ce qui suit, la cartographie des patrons de réponse : « *TheEarliestPossibleResponse* », « *TheEarliestPossibleResponse-1-N* » et « *TheEarliestPossibleResponse-N-1* », est basé sur celles de *Response*, *Response-1-N* et *Response-N-1*, présenté dans (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015) avec valeur de vérité dans $[0, 1]$.

b) TheEarliestPossibleResponse

Globally	$\Box(A \rightarrow \Diamond_{\llbracket time_D \rrbracket} D)$
Before B	$\Diamond_{\geq \llbracket tl_D \rrbracket} B \rightarrow \left(A \rightarrow \left(\neg B \text{ U}_{\llbracket time_D \rrbracket} (D \wedge \neg B) \right) \right) \text{ U } B$
After C	$\Box \left(C \rightarrow \Box(A \rightarrow \Diamond_{\llbracket time_D \rrbracket} D) \right)$
Between C and B	$\Box \left(\left(C \wedge \Box_{\leq \llbracket tl_D \rrbracket} \neg B \wedge \Diamond_{\geq \llbracket tl_D \rrbracket} B \right) \rightarrow \left(\left(A \rightarrow \left(\neg B \text{ U}_{\llbracket time_D \rrbracket} (D \wedge \neg B) \right) \right) \text{ U } B \right) \right)$
After C until B	$\Box \left(\left(C \wedge \Box_{\leq \llbracket tl_D \rrbracket} \neg B \right) \rightarrow \left(\left(A \rightarrow \left(\neg B \text{ U}_{\llbracket time_D \rrbracket} (D \wedge \neg B) \right) \right) \text{ W } B \right) \right)$

c) TheEarliestPossibleResponse-1-N

Globally	$\Box \left(A \rightarrow \Diamond_{\llbracket time_D \rrbracket} (D \llbracket Ch(1) \rrbracket) \right)$
Before B	$\Diamond B \rightarrow \left(A \rightarrow \left(\neg B \text{ U}_{\leq \llbracket tl_D \rrbracket} (D \llbracket Ch(1) \rrbracket) \wedge \neg B \right) \right) \text{ U } B$
After C	$\Box \left(C \rightarrow \Box \left(A \rightarrow \Diamond_{\leq \llbracket tl_D \rrbracket} (D \llbracket Ch(1) \rrbracket) \right) \right)$
Between C and B	$\Box \left(\left(C \wedge \Diamond B \right) \rightarrow \left(\left(A \rightarrow \left(\neg B \text{ U}_{\leq \llbracket tl_D \rrbracket} ((D \llbracket Ch(1) \rrbracket) \wedge \neg B) \right) \right) \text{ U } B \right) \right)$
After C until B	$\Box \left(C \rightarrow \left(\left(A \rightarrow \left(\neg B \text{ U}_{\leq \llbracket tl_D \rrbracket} ((D \llbracket Ch(1) \rrbracket) \wedge \neg B) \right) \right) \text{ W } B \right) \right)$
With :	$\llbracket Ch(i) \rrbracket = \bigwedge 0 \left(\Diamond_{\llbracket tu_R \rrbracket} (R \llbracket Ch(i+1) \rrbracket) \right), R = T_i \text{ and } i \leq N$

d) TheEarliestPossibleResponse-N-1

Globally	$\Box \left((D \llbracket Ch(1) \rrbracket) \rightarrow \Diamond_{\leq \llbracket tu_A \rrbracket} A \right)$
Before B	$\Diamond B \rightarrow \left(\left((D \llbracket Ch_B(1) \rrbracket) \rightarrow \Diamond_{\leq \llbracket tu_A \rrbracket} A \right) \text{ U } B \right)$
After C	$\Box \left(C \rightarrow \Box \left((D \llbracket Ch(1) \rrbracket) \rightarrow \Diamond_{\leq \llbracket tu_A \rrbracket} A \right) \right)$

Between C and B	$\square \left((C \wedge \neg B \wedge \diamond B) \rightarrow \left((D[\text{Ch}_B(1)]) \rightarrow \diamond_{\leq [tu_A]} A \right) \cup B \right)$
After C until B	$\square \left((C \wedge \neg B) \rightarrow \left((D[\text{Ch}_B(1)]) \rightarrow \diamond_{\leq [tu_A]} A \right) \cup B \right)$
With :	$\llbracket \text{Ch}(i) \rrbracket = \wedge O \left(\diamond_{\leq [tu_R]} (R[\llbracket \text{Ch}(i+1) \rrbracket]) \right), R = T_i \text{ and } i \leq N$ $\llbracket \text{Ch}_B(i) \rrbracket = \wedge O \left(\neg B \cup_{\leq [tl_R]} (R[\llbracket \text{Ch}_B(i+1) \rrbracket]) \right), R = T_i \text{ and } i \leq N$

5.5 La vérification automatique

Afin d'expliquer comment un patron de spécification peut être vérifié, nous donnons l'exemple suivant: Donner la propriété P = « **Globally, it is the case that** {Le réseau de communication est actif} **holds as long as possible** », avec un seuil de vérité de 0,99. On vérifie la validité de la propriété P sur la partie du système illustrée dans la Fig. 19, qui est un exemple d'exclusion, où une ressource commune F est partagée entre deux propriétés X et Y. Il existe deux séquences de déclenchement possibles pour atteindre le but que les deux endroits $\neg A$ et E obtiennent un jeton.

$$s_1: M_0[t_B \gg M_1[t_C \gg M_2[t_D \gg M_3[t_E \gg M_4$$

$$s_2: M_0[t_D \gg M_5[t_E \gg M_6[t_A \gg M_7[t_B \gg M_8$$

$$\pi_{0A}(\tau) = (1,3,3,5), \pi_{0C}(\tau) = (3,5,5,7), \pi_{0F}(\tau) = (0,0,0,0)$$

M_x : état « x » du réseau de Petri flou.

T_x : transition de réseau de Petri.

$\pi_{0X}(\tau)$: fonction de synchronisation initiale.

Supposons que la durée de fonctionnement du système soit de 10 jours, de sorte que P devient

$$\square_{[0,10]}(\{\text{Le réseau de communication est actif}\} \\ \wedge \diamond_{[0,10]} \neg \{\text{Le réseau de communication est actif}\})$$

Nous vérifions si P est satisfait ou non sur les deux séquences possibles (s_1, s_2). En appliquant l'algorithme de vérification de modèle (Zhou & Murata, 1999), nous trouvons que s_1 donne 63,6% de degré de satisfaction et 22,2% pour s_2 . En d'autres termes, la place $\neg A$ peut être atteinte avec une valeur de possibilité de 0,636 ou 0,222, ce qui n'est pas accepté en raison du seuil 0.01 ($1 - 0.99 = 0.01$). Donc, P est satisfaite sur ce modèle.

5.6 Conclusion

Dans ce chapitre, nous avons présenté notre catalogue de patrons, pour traiter l'expression de l'auto-adaptation dans la phase de spécification des exigences. Nous avons également spécifié la grammaire de PSAS basée sur l'anglais structuré, de la sélection jusqu'à la formulation du patron. Ensuite, nous avons présenté la sémantique du langage en termes de logique floue, et enfin nous avons montré comment un patron peut être vérifié sur un modèle de PETRI. Cependant, l'efficacité de langage doit être testée à travers des instances réelles et des cas d'études. Dans le chapitre suivant, nous allons mettre en application le langage PSAS, en utilisant notre outil de support, à travers différents cas d'études.

Chapitre 6

PSAS : Tests et mise en œuvre

6.1 Introduction

L'objectif dans ce chapitre est de montrer que les patrons de spécification présentés traitent des problèmes actuels de spécifications auto-adaptatives. Ces patrons de spécification sont principalement conçus pour l'expression de l'incertitude du temps.

6.2 PSAS-tool

Cette section est dédiée à l'outil de support ainsi qu'un ensemble d'exemples de différents systèmes dont les spécifications doivent être flexibles pour assurer l'auto-adaptation.

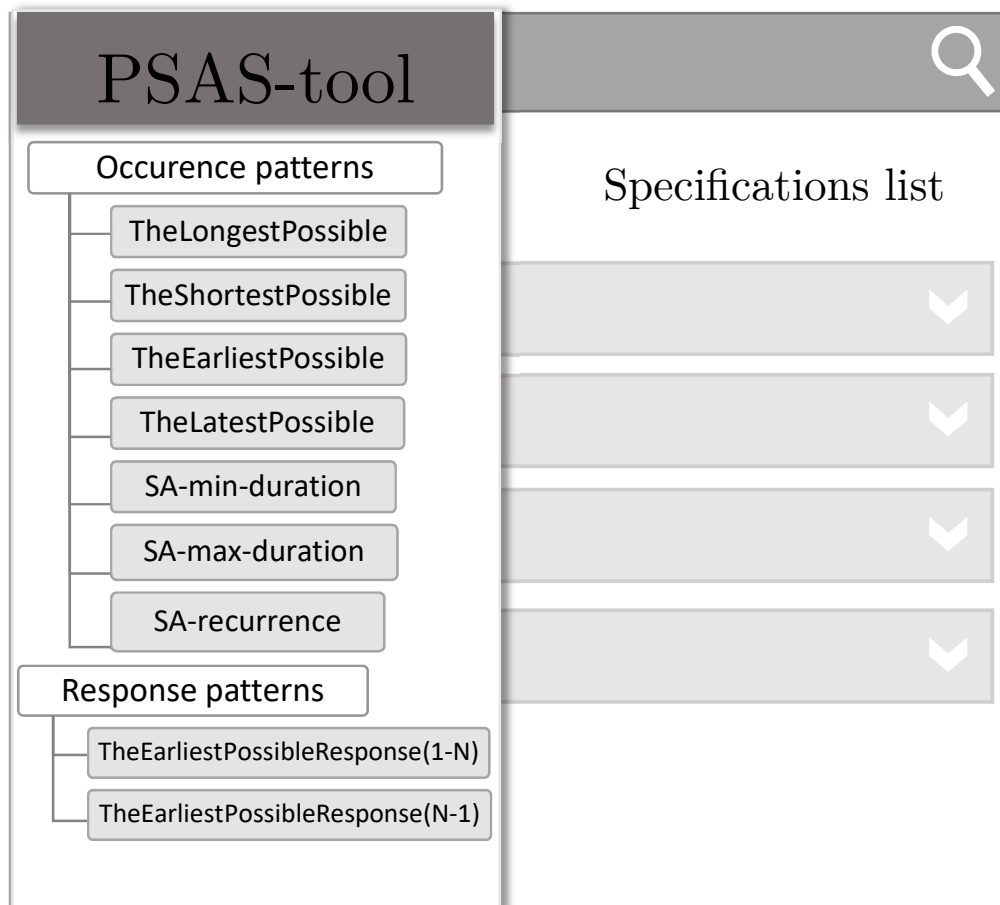


Fig. 21 Vue d'ensemble de l'outil PSAS-tool

6.2.1 Présentation de PSAS-tool

Pour les concepteurs d'exigences, non familiers avec les logiques temporelles, la spécification des propriétés peut être une tâche difficile (Dwyer, Avrunin, & Corbett, 1999). Nous avons donc conçu un outil de support afin de faciliter le travail du concepteur. L'objectif principal de l'outil PSAS-tool (Fig. 21) est d'automatiser la traduction des expressions de grammaire anglaise

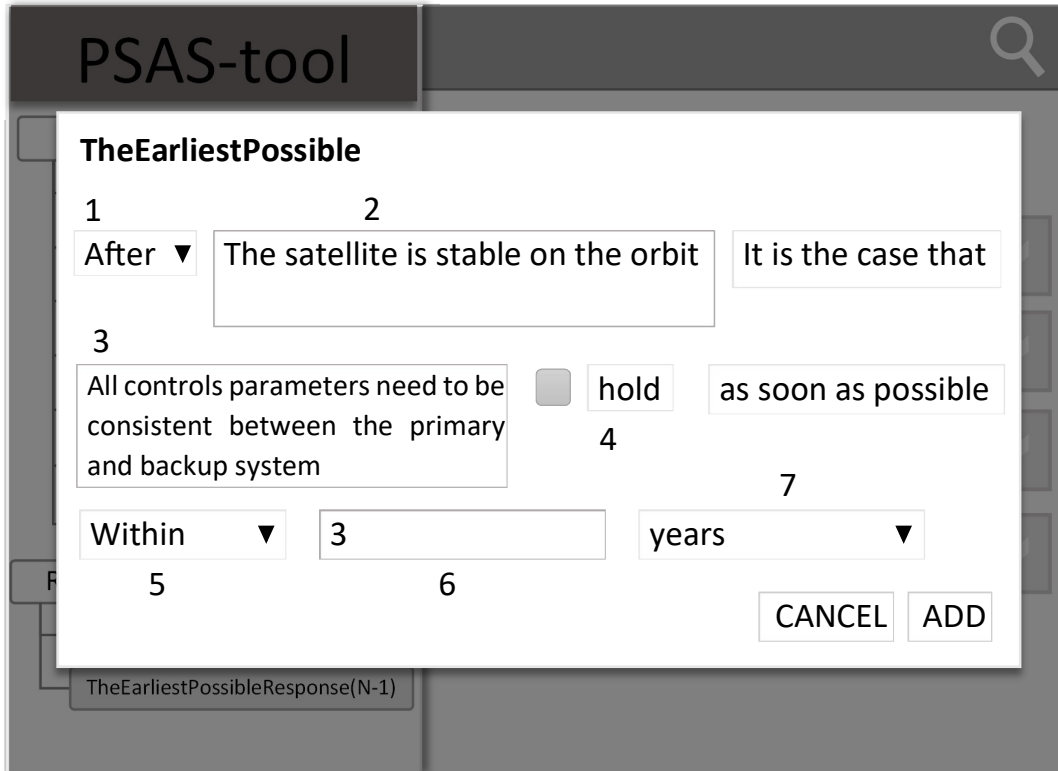


Fig. 22 La sélection de patron et la création de propriété

structurée en formules logiques temporelles métriques floues (FMTL). Le processus de « PSAS-tool » est comme suit : lors de la sélection du patron dans le menu, une fenêtre modale apparaît (Fig. 22), puis on remplit les zones de texte avec l'exigence originale. Après avoir appuyé sur le bouton "ADD", la spécification est insérée dans le document principal et la traduction en FMTL est générée automatiquement (Fig. 23). Enfin, la liste des spécifications peut être exportée au format pdf.

L'exemple dans la figure (Fig. 22) montre le patron « TheEarliestPossible » :

Liste des extensions. Elles incluent des événements comme « After{C} », dans ce cas PSAS-tool ajoute automatiquement une zone de texte pour insérer l'événement.

1. Choix de portée.
2. Zone de texte pour l'événement.
3. Zone de texte qui décrit l'exigence originale.
4. Checkbox optionnelle "holds", pour exprimer clairement l'exigence.
5. Définition d'intervalle de temps (within, at least, ...).
6. Entrée de la durée liée à la spécification.
7. Unité de temps (seconde, minute, heure ...)

Property no: 1

Structured English: ^

After (the satellite is stable on the orbit) It is the case that (all control parameters need to be consistent between the primary and backup system) as early as possible within 3 years

FMTL mapping: ^

$(C \rightarrow \bigcirc_{[[time_A]]} A)$

C = {the satellite is stable on the orbit}

A = {all control parameters need to be consistent between the primary and backup system}

time_A = [0,94670778[

Fig. 23 Spécification en terme d'anglais structuré et FMTL

La conception de PSAS-tool permet aux concepteurs des exigences de spécifier les propriétés du système de manière flexible, sans utiliser des symboles logiques. PSAS-tool est une application orientée web, ce choix implique divers avantages, d'abord, les applications web permettent d'éviter le déploiement sur chaque machine et facilite les mises à jour, deuxièmement, l'accessibilité de n'importe quel endroit avec accès internet, troisièmement, les applications Web sont indépendantes de la plate-forme. L'outil PSAS-tool fournit également l'avantage de l'adaptabilité mobile grâce à sa conception responsive.

Les spécifications présentées dans la sous-section suivante sont générées par l'application de notre approche sur un ensemble d'instances du monde réel.

6.2.2 Exemples de spécifications PSAS

Toutes les spécifications doivent être conformes à la règle : « Portée, patron » (section 0). La portée est l'intervalle de temps, dans lequel le patron peut être valide. La deuxième partie représente le corps du patron. Dans le premier exemple ci-dessous, la portée est « Globally » ce qui signifie que le patron est valable pour tout le temps d'exécution du système. Le modèle ici est

« TheLongestPossible », il sera par la suite remplacé par « It is the case that {P} holds as long as possible », les parties en gras restent inchangées, tandis que P sera remplacé par l'exigence initiale.

a. TheLongestPossible

= Portée, Patron

= Globally, TheLongestPossible.

= Globally, It is the case that {P} holds as long as possible.

= **Globally, It is the case that {Terrestrial Photovoltaics (PV) systems face the sun} holds as long as possible.**

b. TheEarliestPossible

Patron: After{C}, It is the case that {A} holds as early as possible [*Time(A)*].

Exemple: After {the satellite is stable on the orbit}, It is the case that{all control parameters need to be consistent between the primary and backup system for the 3 year mission time} holds as early as possible. (Source: satellite control system).

c. SA-MaxDuration

Patron: Globally, Once {A} [becomes satisfied] it remains as possible less than t_u^A *TimeUnits*.

Exemple: Globally, Once {insulin pump starts injection} becomes satisfied it remains as possible less than d/ d= Duration of insulin action (DIA). (The Fault-Tolerant Insulin Pump Therapy (Capozucca, Guelfi, & Pelliccione, 2006)).

Les exemples en dessus sont représentés par sous-ensemble de patrons. Dans la section suivante, nous présentons un cas d'étude concret pour illustrer comment notre langage ajoute de la souplesse aux exigences.

6.3 Evaluation empirique

Nous présentons ici trois cas d'étude, le premier est sur le protocole de communication "Message Queue Telemetry Transport (MQTT)", le deuxième est un réfrigérateur intelligent d'une maison de retraite assistée (Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2010), et le dernier est un système de répartition d'ambulance (Silva Souza, Lapouchnian, Robinson, & Mylopoulos, 2011).

6.3.1 Scénario du protocole de communication MQTT

Nous présentons ici un cas d'étude sur le protocole de communication "Message Queue Telemetry Transport (MQTT)" où les exigences nécessitent la spécification du temps. Le choix de MQTT est également approprié par le fait que MQTT est normalisé et adapté au réseau de capteurs, qui est l'un des domaines d'application actifs des techniques d'auto-adaptation (Macias-Escriva, Haber, del Toro, & Hernandez, 2013). Nous présentons les principales opérations incluant le processus de connexion, de publication, de souscription et de désabonnement. MQTT est un protocole de transport de messagerie de publication/abonnement client-serveur. Il est idéal pour

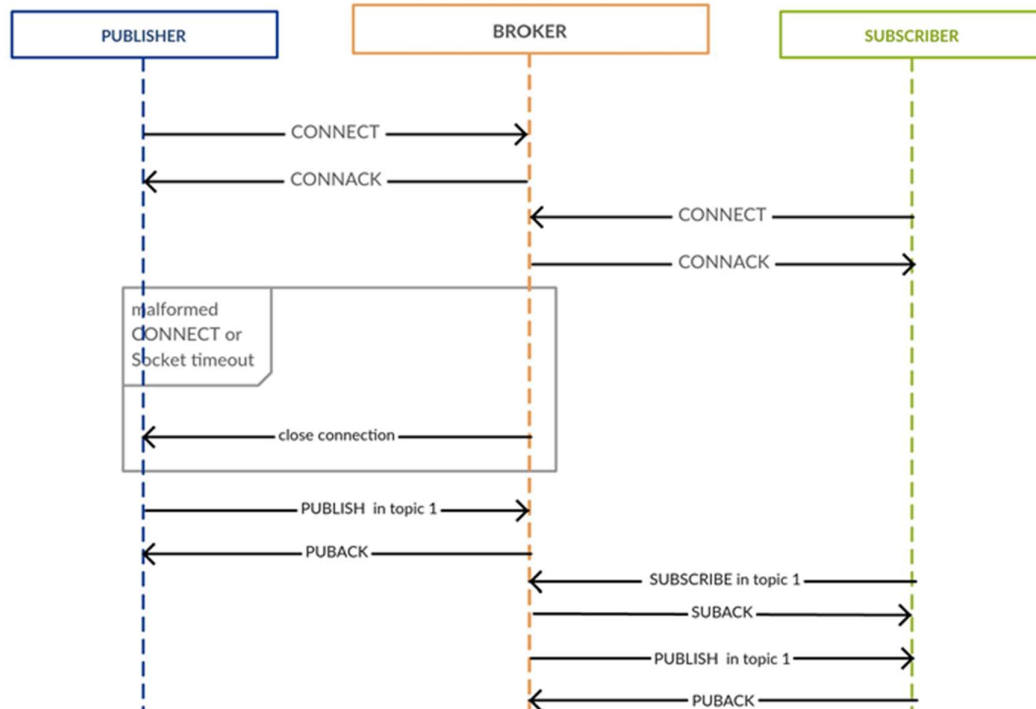


Fig. 24 Aperçu sur le protocole MQTT

une utilisation dans des environnements avec contraintes comme l'Internet of Things (IoT) et le Machine to Machine (M2M). La Fig. 24 présente une vue d'ensemble du protocole MQTT dans un diagramme de séquence simplifié.

Dans ce qui suit, nous présentons un ensemble de spécifications MQTT en utilisant les patrons de (Autili, Grunske, Lumpe, Pelliccione, & Tang, 2015) pour les exigences invariantes et notre catalogue proposé pour les exigences d'auto-adaptation.

1. Exigences invariantes
 - Globally, it is never the case that {a client is connected to another directly} holds.

- Globally, it is always the case that {all communications get through the broker} holds.
2. Processus de connexion (exigences non-invariantes)
- TheEarliestPossible-Response: Globally, if {the client sends a CONNECT message to the broker} then in response {the broker sends a CONNACK and a status code to the client} as early as possible.
 - TheLongestPossible: Before{a client sends a disconnect command or loses connection}, It is the case that {the broker keep the established connection open} holds as long as possible.
 - SA-max-duration: Before{the broker close the connection}, Once {the broker waits CONNECT message from the client} it remains as possible less than MAXTIME. / MAXTIME = a reasonable amount of time defined by the broker.
 - TheEarliestPossible-Response: After{a client is connected}, if{a client set CleanSession to true} then in response {the broker sends a CONNACK with session flag is false to the client} as early as possible.
 - After{client is connected}, if{client has set CleanSession to false and stored client session information exist} then in response {the broker sends to the client a CONNACK with session flag is true} as early as possible.
 - After{client is connected}, if{client has set CleanSession to false and stored client session information does not exist} then in response {the broker sends to client a CONNACK with session flag is false} as early as possible.
3. Publier
- After{client is connected}, if{a client sends a publish to the broker} then in response {the broker will read the publish} holds as early as possible followed by({the broker acknowledge the publish if needed (according to the QoS Level)}){the broker process the publish}).
4. Abonnement/Désabonnement
- After{client is connected}, if{a client sends a SUBSCRIBE/ UNSUBSCRIBE message to the MQTT broker} then in response {the broker sends a SUBACK/UNSUBACK message to client} holds as early as possible.

Les patrons proposés dans les exemples ci-dessus sont des versions souples de ceux existants pour les systèmes non adaptatifs. PSAS à l'avantage par deux aspects principaux. Tout d'abord, les

spécifications générées sont simples grâce à sa grammaire non récursive. Deuxièmement, notre langage offre un moyen de gérer l'intervalle d'exécution des spécifications via le concept de portée.

6.3.2 Autres scénarios : AAL et ADS

Afin de renforcer ma proposition, nous montrons comment le langage PSAS traite des problèmes actuels des spécifications industrielles. Deux autres cas d'études sont présentés. Le premier est le réfrigérateur intelligent d'une maison de retraite assistée sous forme de comparatif avec l'étude dans (Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2010) et le second est un ADS (Ambulance Dispatch System) (Silva Souza, Lapouchnian, Robinson, & Mylopoulos, 2011). Dans ce qui suit, nous présentons un ensemble de spécifications d'auto-adaptation utilisant le catalogue proposé pour les besoins d'auto-adaptation.

Tableau 12 Spécifications du réfrigérateur intelligent (RELAX vs PSAS).

	RELAX	PSAS
R1	The fridge SHALL detect and communicate information with AS MANY AS POSSIBLE food packages.	Globally, it is the case that {The fridge detects and communicate information with food package} as many as possible.
R2	The fridge SHALL suggest a diet plan with total calories AS CLOSE AS POSSIBLE TO the daily ideal calories.	Globally, it is the case that {The fridge suggests a diet plan with total calories} as close as possible to $q / q =$ the daily ideal calories.
R3	The system SHALL consume AS FEW units of energy AS POSSIBLE during normal operation.	Globally, it is the case that {The fridge detect and communicate information with food package} as few as possible.

Tableau 13 Spécifications ADS.

	Description d'exigence	Spécification PSAS
R4	Input emergency information should never fail.	Globally, it is the case that {Input emergency information} holds as long as possible.
R5	Communications networks working should have 99% success rate.	Globally, it is the case that {Communications networks working should have 99% success rate } holds as long as possible.
R6	Search call database should have a 95% success rate over 1-week periods.	Globally, {Search call database should have a 95% success rate} holds repeatedly almost every <i>1 week</i> .
R7	Dispatch ambulance should fail at most once a week.	Globally, {Dispatch ambulance} holds repeatedly almost every <i>1 week</i> .
R8	Ambulance arrives in 10 minutes; should succeed 60% of the time.	Globally, it is the case that {Ambulance arrives should succeed 60% of the time} holds as possible within <i>10 minutes</i> .
R9	Ambulance arrives in 15 minutes; should succeed 80%, measured daily.	Globally, it is the case that {Ambulance arrives should succeed 80%} holds as possible within <i>15 minutes</i> .
		Globally, {Ambulance arrives should succeed 80%} holds repeatedly almost every <i>1 day</i> .

R10	Update automatically should succeed 100 times more than the task Update manually.	Globally, it is the case that {Automatic update should succeed 100 times more than manual update} holds as long as possible.
R11	The success rate of No unnecessary extra ambulances for a month should not decrease, compared to the previous month, two times consecutively.	Globally, May {The success rate of No unnecessary extra ambulances should not decrease, compared to the previous month} or May {The success rate of No unnecessary extra ambulances decrease one time compared to the previous month } or May {The success rate of No unnecessary extra ambulances decrease two times compared to the previous month}, repeatedly almost every <i>1 month</i> .
R12	Update arrival at site should be successfully executed within 10 minutes of the successful execution of Inform driver, for the same emergency call.	After {successful execution of Inform driver}, it is the case that {update arrival at site} holds as early as possible within <i>10 minutes</i> .
R13	Mark as unique or duplicate should be decided within 5 minutes.	Globally, it is the case that {Mark as unique or duplicate should be decided} as early as possible within <i>5 minutes</i> .
R14	(Search call database should have a 95% success rate over 1-week periods); should have 75% success rate over 1-month periods.	Globally, {Search call database should succeed} holds repeatedly almost every <i>1 month</i> .

6.4 Discussion

PSAS est un citoyen de première classe dans le monde auto-adaptatif, c'est un peu comme JAVA pour l'orienté-objet. Tous les modèles sont assouplis à l'aide d'opérateurs tels que «as possible» (Fig. 25). Cela signifie que la propriété peut être assouplie si elle ne peut pas être satisfaite dans certains états du système au moment de l'exécution. D'autre part, chaque patron est représenté par une formule générée à l'aide de la logique temporelle métrique floue qui peut être vérifiée automatiquement sur un modèle de réseau de Petri temporel flou. La courbe d'apprentissage est vraiment facile quand il s'agit de PSAS, les approches de patrons sont éprouvées (500 exigences exprimées en utilisant seulement 20 modèles (Dwyer, Avrunin, & Corbett, 1999)), dans nos études de cas, 14 exigences exprimées en utilisant seulement 7 patrons. Un autre avantage intéressant, avec l'utilisation de PSAS, est l'amélioration de la lisibilité de la spécification de document en raison du nombre réduit de patrons utilisés.

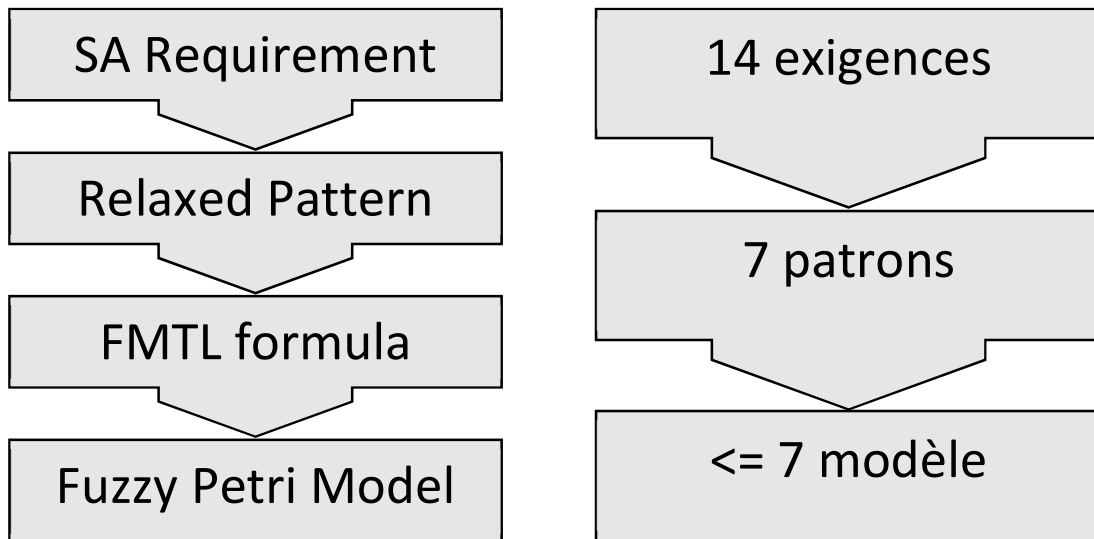


Fig. 25 Traitement des exigences avec PSAS.

Les spécifications, dans les scénarios précédents, illustrent la facilité d'utilisation du système PSAS, qui constitue l'un de ses meilleurs avantages. Nous avons constaté qu'il est plus facile à manipuler en raison de la structure bien définie du patron. Le Tableau 12 présente les spécifications exprimées avec RELAX (Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2010) et PSAS; les approches sont similaires en termes de simplicité. Cependant, PSAS présente l'avantage d'une grammaire non récursive qui facilite le langage pour les ingénieurs moins expérimentés. Comme mentionné dans la section 5.2, chaque patron est associé à l'un des cinq portées afin que les dix-sept motifs puissent constituer jusqu'à 85 propriétés différentes. La Fig. 26 illustre le

nombre d'opérateurs disponibles dans PSAS ainsi que trois ouvrages de référence, par rapport au nombre d'études de cas qui leur ont été appliquées. Le graphique montre que PSAS est la langage le plus riche en termes de vocabulaire (17 formule), 13 pour RELAX (Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2010), 3 pour MAPE-K-FT (Iglesia & Weyns, 2015) et uniquement l'opérateur « **le plus possible** » pour FLAGS (Baresi, Pasquale, & Spoletini, 2010). En ce qui concerne l'applicabilité du langage, MAPE-K-FT prend le trône avec quatre études de cas, trois pour PSAS et un pour les autres. Néanmoins, PSAS est un langage générique qui peut être appliqué dans n'importe quel contexte d'auto-adaptation. Dans le Tableau 13, les exigences doivent être assouplies, mais cet assouplissement est limité à un certain seuil. Par exemple, «Une ambulance qui arrive dans 10 minutes devrait réussir 60% du temps», 60% du succès définissant le degré de réussite minimum toléré, et le patron doit avoir une valeur réelle de 1 dans 60% du temps. Les 40% restants ont une valeur réelle inférieure à 1. Ainsi, PSAS gère non seulement le degré de réussite minimal, mais également le reste qui peut donner une valeur réelle inférieure à 1, ce qui peut être utile au niveau de raisonnement au moment de l'exécution.

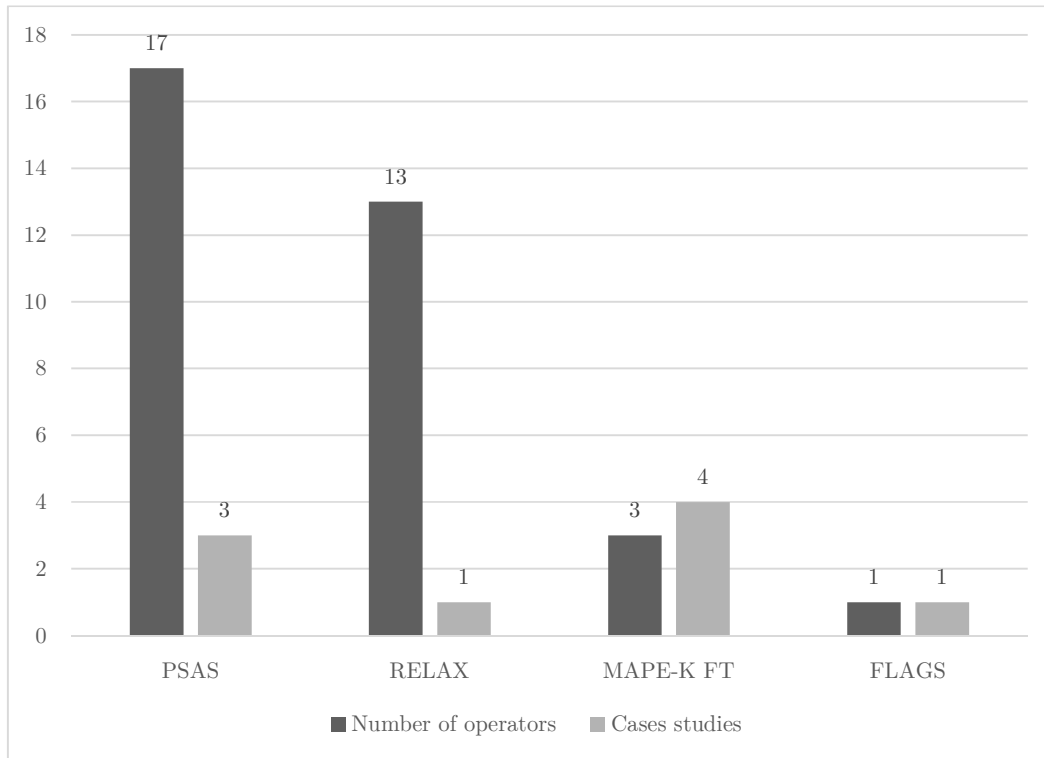


Fig. 26 Nombre d'opérateurs et le nombre de cas d'étude (PSAS et autres)

L'approche dans cette thèse est conçue pour être utilisée lors de la phase de conception. Ainsi, en permettant aux ingénieurs de spécifier l'auto-adaptation au niveau des exigences, guide le raisonnement des concepteurs sur l'auto-adaptation dans la phase de spécification. Les PSAS

peuvent également être utiles au moment de l'exécution. S'agissant de la boucle de rétroaction MAPE (Monitoring Analyze Planning Execution) (Kephart & Chess, 2003), la valeur de vérité de la spécification peut être très utile pendant l'exécution (phases de surveillance et de planification). Cependant, l'approche des chercheurs ne donne pas une solution d'exécution explicite. Grâce à la logique floue, les modèles proposés dans les exemples présentés ci-dessus sont plus souples que ceux existants pour les systèmes non adaptatifs. D'autre part, les langages existants pour les systèmes auto-adaptatifs tels que RELAX offraient la flexibilité requise, mais le système PSAS offre un avantage par deux aspects principaux. Premièrement, les spécifications générées sont simples dans PSAS en raison de sa grammaire non-réursive. Deuxièmement, le langage des chercheurs permet de gérer l'intervalle d'exécution des spécifications via le concept de « portée ».

6.5 Conclusion

Dans ce chapitre, nous avons présenté l'outil de support PSAS-tool ainsi que la méthode de spécification d'une propriété en utilisant PSAS. Nous avons également renforcé l'approche à travers des instances diverses du monde réel, ensuite l'application de PSAS sur des cas d'études industriels, et enfin, nous avons mis en place une comparaison entre PSAS et certains travaux connexes afin de montrer l'efficacité de notre langage en matière de richesse de vocabulaire, de facilité d'usage et d'applicabilité.

Conclusions générales et travaux futurs

Le contexte de ce travail de recherche se situe dans le domaine de l'ingénierie des exigences pour les systèmes auto-adaptatifs. Ce travail se situe aux toutes premières étapes du cycle de vie du développement logiciel, c'est-à-dire à la phase de l'ingénierie des exigences. La contribution globale de cette thèse est de proposer une approche intégrée pour la spécification des exigences de SAA en utilisant les techniques de patrons de spécification.

Notre approche proposée prend en compte les exigences puis, en appliquant notre processus et outil, nous intégrons la notion d'adaptabilité aux exigences que nous spécifions. Une fois le système conçu, la vérification des propriétés de SAA peut être faite automatiquement grâce à la cartographie dans FMTL.

Notre Contribution

Cette thèse vise à présenter un nouveau langage de spécification pour les systèmes auto-adaptatifs « PSAS ». L'objectif principal de ce langage est de faire face aux changements d'exécution, y compris l'incertitude, afin de spécifier le comportement d'un système auto-adaptatif pour répondre aux changements inattendus, qui se produisent dans l'environnement d'exécution. L'absence d'informations suffisantes sur le comportement prévu de l'application peut causer une incertitude comportementale pendant la phase de développement, de sorte qu'il nécessite encore une adaptation au moment de l'exécution.

Nous traitons les exigences liées à l'auto-adaptation dès le début, c'est-à-dire au même niveau d'abstraction que les exigences non-invariantes. Les méthodes d'ingénierie des exigences pour SAA doivent prendre en compte l'incertitude inhérente à ces systèmes. Cela se traduit par la conception de nouveaux langages de spécification. Gardant à l'esprit cette incertitude, nous fournissons une approche intégrée pour traiter la spécification des exigences et la vérification des propriétés de SAA. Pour cela, nous avons présenté notre approche proposée qui vise à résoudre de différents problèmes que nous avons identifiés concernant la spécification des exigences et la vérification des propriétés. Nous sommes intéressés par l'utilisation de techniques de patron pour la spécification des propriétés et des propriétés de SAA.

PSAS fournit un nouveau vocabulaire et une nouvelle terminologie pour l'ingénierie des exigences pour les SASs. Il fournit la notion de facteurs d'incertitude associés aux exigences. Le processus PSAS prend les entrées en tant que des exigences fonctionnelles ou non-fonctionnelles et permet de distinguer les exigences adaptables et celles qui sont invariantes. Pour tirer parti des exigences assouplies et invariantes, nous avons développé un outil appelé PSAS-tool, utilisé pour

automatiser la formalisation des exigences SAA en tenant compte d'incertitude associée à chaque exigence assouplie.

Dans notre approche proposée, nous avons introduit un catalogue de patrons pour les exigences non-invariantes. Le langage est basé sur deux piliers majeurs : la logique floue et les patrons de spécification. Le premier a pour objectif de supporter à la fois les incertitudes temporelles et ordinales grâce à la flexibilité de la logique floue alors que le second est destiné à traiter les ambiguïtés du langage naturel à travers l'anglais structuré. Nous avons fourni une corrélation des concepts PSAS dans des formules FMTL. Les exigences créées avec PSAS sont ensuite transformées en formules FMTL. Ce mappage est réalisé à l'aide de l'outil PSAS-tool.

Afin de valider notre approche proposée, nous avons présenté un ensemble de cas d'études pour montrer l'efficacité du langage PSAS. Un cas d'étude industriel à partir d'un protocole dédié au domaine des réseaux de capteurs, l'un des secteurs les plus actifs dans l'auto-adaptation. Le deuxième cas d'étude est réfrigérateur intelligent et le troisième est un système de distribution d'ambulance.

Perspectives

Pour améliorer l'efficacité de ce travail, il faut des méthodes et des outils sophistiqués afin d'assister les spécifications générées par PSAS dans les phases avancées du GL, en particulier la phase de vérification. Dans cette thèse, les patrons proposés, sont mappés sur la logique temporelle FMTL, alors qu'il est possible de mettre en place d'autre mappage dans d'autre logique (supportant l'incertitude), afin d'être compatible avec des outils de conception ou de vérification qui existent déjà afin de faciliter la migration d'autre approche de spécification vers PSAS. L'objectif de cela est de combler l'écart entre PSAS et les outils et techniques utilisées dans l'industrie des logiciels. Dans ce qui suit, nous mentionnons certaines des perspectives que nous avons identifiées concernant la modélisation des exigences et la vérification des propriétés de SAA.

Perspective pour la modélisation des exigences des systèmes auto-adaptatifs

Pour le moment, notre outil PSAS-tool est capable de mapper les concepts PSAS sur les concepts FMTL, mais pas dans d'autres logiques temporelles. Nous avons identifié la mise en correspondance entre les concepts de PSAS et d'autres logiques, en tant que perspective afin que les personnes familiarisées avec des approches, avec une cartographie autre que FMTL, puissent mapper leurs concepts en concepts PSAS auxquels ils ne sont pas familiers, afin de fournir une base de connaissances supplémentaire concernant la spécification des exigences de SAA.

Cela aidera à identifier les facteurs de l'auto-adaptation associés à chaque exigence. Les exigences assouplies résultantes peuvent être soumises à divers outils et processus, et peuvent être utilisées avec diverses autres méthodes pour répondre aux questions relatives à l'auto-adaptation, telles que les conditions à surveiller, les procédures de prise de décision et les stratégies d'adaptation possibles à suivre.

Perspectives pour la vérification des propriétés des systèmes auto-adaptatifs

On s'attend souvent à ce que les systèmes logiciels critiques essentiels à la mission s'adaptent en toute sécurité aux modifications de leur environnement d'exécution. Étant donné des rôles critiques que jouent SAA, il est souvent peu pratique de les déconnecter pour adapter leurs fonctionnalités. Par conséquent, ces systèmes doivent, lorsque cela est possible, adapter leur comportement au moment de l'exécution avec une intervention humaine minimale, voire aucune.

Nous n'avons pas traité ces aspects, car nous en étions au tout début du cycle de développement de SAA. Pour être précis, notre travail réside dans le cadre de l'auto-adaptation mais nous ne traitons pas le développement de mécanismes d'auto-adaptation. Nous aidons les développeurs SAA en fournissant un mécanisme d'identification des fonctionnalités adaptatives associées aux exigences de ces systèmes. L'utilisation des approches MDE pour assister notre approche proposée dans les étapes avancées est une perspective que nous avons identifiée.

Nous aimerions automatiser le processus de modélisation des patrons en le générant automatiquement à partir d'exigences assouplies et invariantes. Nous pouvons ensuite fournir la conception de système résultante de l'application de l'approche que nous proposons à utiliser avec ces micro-modèles générés automatiquement, afin de vérifier les propriétés par rapport à la conception de système.

Perspective pour les études empiriques

Comme preuve de concept, nous avons appliqué notre approche proposée à trois cas d'études dans le domaine des SASs. Les exigences des trois cas d'études ont été spécifiées et les propriétés de l'une des études de cas ont été comparées à une approche existante. Une situation idéale sera de l'appliquer à un système réel, ce qui nous aidera à évaluer la validité de l'approche proposée par rapport à certains critères d'évaluation prédéfinis. Pour cela, la première étape serait de définir les critères d'évaluation. Ensuite, nous devons fournir suffisamment d'expérience pour mesurer les critères définis, qui fourniront des données statistiques sur son applicabilité d'une part et les améliorations pouvant être apportées d'autre part.

Les critères d'évaluation peuvent être définis, par exemple en termes de facilité d'utilisation de l'approche proposée. Nous pouvons ensuite modéliser et vérifier les exigences du système réel SAA en utilisant notre approche et les comparer aux méthodes classiques de spécification et de vérification des exigences. Cela fournira une véritable évaluation de notre approche proposée.

Production scientifique

- **Revue internationale avec comité de lecture**
 - Yahiaoui, A., Bendjenna, H., Roose, P., Chung, L., & Mohamed, A. (2019). Temporal Pattern Specifications for Self-Adaptive Requirements. *Recent Patents on Computer Science*, 12(1), 58.
- **Conférences internationales avec comité de lecture**
 - Ayoub Yahiaoui*, Hakim Bendjenna, Philippe Roose, Lawrence Chung and Mohammed Amroune. “Requirement engineering for self-adaptive system : A new specification language”. *The International Conference on Pattern Analysis and Intelligent Systems (PAIS’2015)*. University of Larbi Tebessi, Algeria. October 26-27, 2015.
 - Ayoub Yahiaoui*, Hakim Bendjenna, Philippe Roose. “Patrons temporels pour spécifier les systèmes auto-adaptatifs”. *Actes du XXXVème Congrès INFORSID*, Toulouse, France, May 30 - June 2, 2017. 213—228 2017.
- **Stage de recherche**
 - Stage de perfectionnement de niveau à l'étranger PNE 2016-2017.

Bibliographie

- Ahmad, M., Belloir, N., & Bruel, J.-M. (2015). Modeling and Verification of Functional and Non-Functional Requirements of Ambient Self-Adaptive Systems. *Journal of Systems and Software*, 107, 50-70.
- Amroune, M. (2014). *Vers une approche orientée aspect d'ingénierie des besoins dans les organisations multi-entreprises*. Toulouse, France: Université Toulouse 2 Jean Jaurès.
- ARM: Application Response Measurement. (2011).
- Asadollahi, R. a. (2009). StarMX: A framework for developing self-managing Java-based systems. Dans *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on* (pp. 58--67). IEEE.
- Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., & Tang, A. (2015). Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. *IEEE Transactions on Software Engineering*, 41(7), 620-638.
- Baresi, L., Pasquale, L., & Spoletini, P. (2010). Fuzzy goals for requirements-driven adaptation. *2010 18th IEEE International Requirements Engineering Conference* (pp. 125-134). Sydney, NSW, Australia: IEEE.
- Bendjenna, H. (2010). *Ingenierie des exigences pour les processus inter-organisationnels*. Constantine, Algérie: Université Mentouri de Constantine, Université de Toulouse.
- Brake, N. a. (2008). Automating discovery of software tuning parameters. Dans *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems* (pp. 65--72). ACM.
- Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1), 14--23.
- Bruhn, J. a. (2008). Comprehensive support for management of Enterprise Applications. Dans *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on* (pp. 755--762). IEEE.
- Brun, Y. a. (2009). Engineering self-adaptive systems through feedback loops. Dans *Software engineering for self-adaptive systems* (pp. 48--70). Springer.

- Capozucca, A., Guelfi, N., & Pelliccione, P. (2006). The fault-tolerant insulin pump therapy. *Rigorous Development of Complex Fault-Tolerant Systems* (pp. 59-79). Berlin, Heidelberg: Springer.
- Cazzola, W. a. (2004). Software evolution through dynamic adaptation of its oo design. Dans *Objects, Agents, and Features* (pp. 67--80). Springer.
- Cheng, B. H., Sawyer, P., Bencomo, N., & Whittle, J. (2009). A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. *International Conference on Model Driven Engineering Languages and Systems* (pp. 468-483). Berlin, Heidelberg: Springer.
- Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., . . . Malek, S. (2009). Software Engineering for Self-Adaptive Systems: A Research Roadmap. *SOFTWARE ENGINEERING FOR SELF-ADAPTIVE SYSTEMS*, 5525, 1 - 26.
- Cheng, S.-W., & Garlan, D. (2007). Handling uncertainty in autonomic systems. *International Workshop on Living with Uncertainties (IWLU'07)*. Atlanta, Georgia, USA: Citeseer.
- Cooray, D. a. (2010). RESISTing reliability degradation through proactive reconfiguration. *Proceedings of the IEEE/ACM international conference on Automated software engineering* (pp. 83--92). Antwerp, Belgium: ACM.
- da Silva, C. E., da Silva, J. D., Paterson, C., & Calinescu, R. (2017). Self-adaptive role-based access control for business processes. *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (pp. 193--203). Buenos Aires, Argentina: IEEE Press.
- Dahm, M. a. (2002). Byte code engineering library.
- Dardenne, A. a. (1993). Goal-directed requirements acquisition. *Science of computer programming*, 20(1- 2), 3--50.
- De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., & Weyns, D. (2013). Software engineering for self-adaptive systems: A second research roadmap. *Software Engineering for Self-Adaptive Systems II*. 5, pp. 1-32. Berlin, Heidelberg: Springer.
- De Lemos, R., Giese, H., Müller, H., Shaw, M., Andersson, J., Baresi, L., . . . others. (2009). Software engineering for self-adaptive systems. Dans *Dagstuhl Seminar* (Vol. 10431). Springer.
- Dick, J., Hull, E., & Jackson, K. (2017). *Requirements engineering*. Springer.

- Dowling, J. (2004). The Decentralised Coordination of Self-Adaptive Components. University of Dublin, Trinity College. Department of Computer Science.
- DuBois, D., & Prade, H. (1989). Processing fuzzy temporal knowledge. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(4), 729 - 744.
- Dwyer, M. B., Avrunin, G. S., & Corbett, J. C. (1999). Patterns in property specifications for finite-state verification. *Proceedings of the 21st international conference on Software engineering* (pp. 411-420). Los Angeles: ACM.
- Easterbrook, S. (2001). *Software Lifecycles*. University of Toronto Department of Computer Science.
- Eclipse, F. (2008). *Aspectj: Crosscutting objects for better modularity*. Retrieved 10-07-08, 2008, from <http://www.eclipse.org/aspectj>.
- Elizabeth, H., Ken, J., & Jeremy, D. (2004). *Requirements Engineering*. Springer.
- Elkhodary, A., Esfahani, N., & Malek, S. (2010). FUSION: a framework for engineering self-tuning self-adaptive software systems. *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering* (pp. 7--16). Santa Fe, New Mexico, USA: ACM.
- Esfahani, N., Kourosfar, E., & Malek, S. (2011). Taming uncertainty in self-adaptive software. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 234--244). Szeged, Hungary: ACM.
- Fleurey, F. a. (2009). A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. Dans *International Conference on Model Driven Engineering Languages and Systems* (pp. 606--621). Springer.
- Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self adaptation with reusable infrastructure. *Computer Science Department*, 668.
- Gotel, O. C., & Finkelstein, C. (1994). An analysis of the requirements traceability problem. *the First International Conference on Requirements Engineering* (pp. 94--101). IEEE.
- Grace, P., Lagaisse, B., Truyen, E., & Joosen, W. (2008). A reflective framework for fine-grained adaptation of aspect-oriented compositions. Dans *International Conference on Software Composition* (pp. 215--230). Springer.

- Graham, S. L., Kessler, P. B., & Mckusick, M. K. (1982). Gprof: A call graph execution profiler. Dans *ACM Sigplan Notices* (Vol. 17, pp. 120--126). ACM.
- Greenwood, P. a. (2004). Using dynamic aspect-oriented programming to implement an autonomic system. Dans *Proceedings of Dynamic Aspects Workshop (DAW)(held with AOSD 2004)*.
- Group, T. S. (2009). Chaos report 2009. *Technical report*.
- Gruhn, V., & Laue, R. (2006). Patterns for timed property specifications. *Electronic Notes in Theoretical Computer Science*, 153(2), 117-133.
- Grunske, L. (2008). Specification patterns for probabilistic quality properties. *2008 ACM/IEEE 30th International Conference on Software Engineering* (pp. 31-40). Leipzig, Germany: IEEE.
- IBM. (2005). Autonomic computing toolkit: Developer's guide.
- IBM. (2006). An architectural blueprint for autonomic computing. *IBM White Paper*, 31, 1--6.
- IEEE. (1990). Ieee standard glossary of software engineering terminology.
- Iglesia, D. G., & Weyns, D. (2015). MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(3), 15.
- Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1), 41--50.
- Khan, K. (2006). *JBOSSAOP: Framework for Organizing Cross Cutting Concerns*.
- Kiczales, G., Des Rivieres, J., & Bobrow, D. G. (1991). *The art of the metaobject protocol*. MIT press.
- Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., & Irwin, J. (1997). Aspect-oriented programming. Dans *European conference on object-oriented programming* (pp. 220--242). Springer.
- Konrad, S., & Cheng, B. H. (2005). Real-time specification patterns. *Proceedings of the 27th international conference on Software engineering* (pp. 372-381). St. Louis, MO, USA: ACM.

- Kuleshov, E. (2007). Using the ASM framework to implement common Java bytecode transformation patterns. *Aspect-Oriented Software Development*.
- Laddaga, R. (2000). Active software. Dans *International Workshop on Self-Adaptive Software* (pp. 11--26). Springer.
- Lee, G. L. (2007). Dynamic binary instrumentation and data aggregation on large scale systems. *International Journal of Parallel Programming*, 35(3), 207--232.
- Lieberherr, K. J. (1994). Customizing adaptive software to object-oriented software using grammars. *International Journal of Foundations of Computer Science*, 5(2), 179--208.
- Littman, M. L. (2004). Reinforcement learning for autonomic network repair. IEEE.
- Macias-Escriba, F. D., Haber, R., del Toro, R., & Hernandez, V. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18), 7267-7279.
- Maes, P. (1987). Concepts and experiments in computational reflection. Dans *ACM Sigplan Notices* (Vol. 22, pp. 147--155). ACM.
- Maes, P. (1990). Situated agents can have goals. *Robotics and autonomous systems*, 6(1-2), 49--70.
- Moon, S.-i., Lee, K. H., & Lee, D. (2004). Fuzzy branching temporal logic. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2), 1045-1055.
- Morandini, M., Penserini, L., Perini, A., & Marchetto, A. (2017). Engineering requirements for adaptive systems. *Requirements Engineering*, 22(1), 77-103.
- Morin, B. a.-M. (2009). Models@ run. time to support dynamic adaptation. *Computer*, 42(10).
- Morin, B. a.-M. (2009). Taming dynamically adaptive systems using models and aspects. Dans *Proceedings of the 31st International Conference on Software Engineering* (pp. 122--132). IEEE Computer Society.
- Murata, T. (1996). Temporal uncertainty and fuzzy-timing high-level Petri nets. *17th International Conference on Application and Theory of Petri Nets*. Osaka, Japan.
- Naur, P., & Randell, B. (1968). Software Engineering: Report on a conference sponsored by the NATO SCIENCE COMMITTEE, Garmisch, Germany, 7th to 11th October 1968.
- Ogata, K., & Yang, Y. (2002). *Modern control engineering* (Vol. 4). India: Prentice hall.

- Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimhigner, D., Johnson, G., Medvidovic, N., . . . Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, 14(3), 54--62.
- Oreizy, P., Medvidovic, N., & Taylor, R. N. (1998). Architecture-based runtime software evolution. Dans *Proceedings of the 20th international conference on Software engineering* (pp. 177--186). IEEE Computer Society.
- Pawlak, R., Seinturier, L., Duchien, L., Florin, G., Legond-Aubry, F., & Martelli, L. (2004). JAC: an aspect-based distributed dynamic framework. *Software: Practice and Experience*, 34(12), 1119--1148.
- Pohl, K. (2010). *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated.
- Popovici, A., Gross, T., & Alonso, G. (2002). Dynamic weaving for aspect-oriented programming. Dans *Proceedings of the 1st international conference on Aspect-oriented software development* (pp. 141--147). ACM.
- Rosenblatt, J. K. (1997). DAMN: A distributed architecture for mobile navigation. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3), 339--360.
- Ryan, K. (1993). The role of natural language in requirements engineering. *the IEEE International Symposium on Requirements Engineering* (pp. 240--242). IEEE.
- Sadjadi, S. M., McKinley, P. K., Cheng, B. H., & Stirewalt, R. K. (2004). TRAP/J: Transparent generation of adaptable Java programs. Dans *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (pp. 1243--1261). Springer.
- Salehie, M. a. (2009). Change support in adaptive software: A case study for fine-grained adaptation. Dans *2009 Sixth IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems* (pp. 35--44). IEEE.
- Salehie, M., & Tahvildari, L. (2007). A weighted voting mechanism for action selection problem in self-adaptive software.
- Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2), 14.
- Shaw, M., & Garlan, D. (1996). *Software architecture* (Vol. 101). Prentice Hall Englewood Cliffs.

- Silva Souza, V. E., Lapouchnian, A., Robinson, W. N., & Mylopoulos, J. (2011). Awareness requirements for adaptive systems. *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems* (pp. 60-69). Waikiki, Honolulu, HI, USA: ACM.
- Sommerville, I. a. (2010). Software testing. *Software Engineering, 9th edn. Addison-Wesley*.
- Souza, V. E., Lapouchnian, A., & Mylopoulos, J. (2012). (Requirement) evolution requirements for adaptive systems. *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (pp. 155-164). Zurich, Switzerland: IEEE.
- Srivastava, A. a. (1994). *ATOM: A system for building customized program analysis tools* (Vol. 29). ACM.
- Sun Microsystems, I. (2006). Java Management Extensions (JMX) Specification. *Sun Microsystems, Inc.*
- Sun Microsystems, I. (2006). Java Virtual Machine Tool Interface (JVMTI).
- Sutton, R. S. (1998). *Reinforcement learning: An introduction*. MIT press.
- Tesauro, G. (2007). Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing, 11*(1).
- Vassev, E., & Hinchey, M. (2015). Engineering Requirements for Autonomy Features. *Software Engineering for Collective Autonomic Systems* (pp. 379-403). Cham: Springer.
- Vogel, T. a. (2010). Adaptation and abstract runtime models. Dans *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* (pp. 39--48). ACM.
- Weyns, D. a. (2010). FORMS: a formal reference model for self-adaptation. Dans *Proceedings of the 7th international conference on Autonomic computing* (pp. 205--214). ACM.
- Whittle, J. a.-M. (2009). Relax: Incorporating uncertainty into the specification of self-adaptive systems. Dans *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International* (pp. 79--88). IEEE.
- Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H., & Bruel, J.-M. (2010). RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering, 177-196*.

Yahiaoui, A., Bendjenna, H., Roose, P., Chung, L., & Mohamed, A. (2019). Temporal Pattern Specifications for Self-Adaptive Requirements. *Recent Patents on Computer Science*, 12(1), 58.

Yu, E. S. (1997). Towards modelling and reasoning support for early-phase requirements engineering. *Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering* (pp. 226-235). Annapolis, MD, USA, USA: IEEE.

Zhou, Y., & Murata, T. (1999). Petri net model with fuzzy timing and fuzzy-metric temporal logic. *International Journal of Intelligent Systems*, 14(8), 719-745.

Zohar, M., & Amir, P. (1992). *The Temporal Logic of Reactive and Concurrent Systems*. New York: Springer.

Acronymes

SAA	Systeme Auto-Adaptatif
SAS	Self adaptive system
IE	Ingénierie des Exigences
PS	Patrons de Spécification
FMTL	Logique Métrique Temporelle Floue
PSAS	Patron de Spécification pour Les Systemes Auto-Adaptatif
PSAS-tool	PSAS tool
GORE	Goal Oriented Requirements Engineering
GL	Génie Logiciel
UML	Unified Modeling Language
EUC	Exigence Utilisateur Clé
IPC	Indicateur de Performance Clé
APRAD	Agence des Projets de Recherche Avancée de la Défense
MAPE-loop	Monitoring Analyse Planning Execution loop
AJG	Adaptateur de Journal Générique
ATJ	Analyseur de Trace de Journal
ARM	Application Response Measurement
SDK	Standard Development Kit
JVMTI	Interface Java Virtual Machine Tools
JVM	Java Virtual Machine
JMX	Java Management Extensions
JDK	Kit de Développement Java
MBean	Managed Bean
POA	Programmation Orientée Aspect
API	Application Programming Interface
ASP	Problème De Sélection D'action
MAPE-K	MAPE Knowledge
LTL	Logique Temporelle Linéaire
MTL	Logique Temporelle Métrique
TLFTN	Temporal-Logic Fuzzy-Timing Petri Net
MQTT	Message Queue Telemetry Transport

IoT	Internet Of Things
M2M	Machine To Machine
AAL	Assisted Ambient Living
ADS	Ambulance Dispatching System