Patrons temporels pour spécifier les systèmes auto-adaptatifs

Yahiaoui Ayoub, Hakim Benjenna, Philippe Roose

Patrons temporels pour spécifier les systèmes auto-adaptatifs

YAHIAOUI Ayoub[1], BENJENNA Hakim[1], ROOSE Philippe[2]

1. Laboratoire LAMIS, Université Larbi Tébéssi,
Route de Constantine, 12002 Tébessa, Algérie
ayoub.yahiaoui@univ-tebessa.dz , hakim.bendjenna@univ-tebessa.dz

2. Laboratoire LIUPPA, Univ Pau & Pays Adour,
EA 3000, 64600 Anglet, France
Philippe.Roose@iutbayonne.univ-pau.fr

RESUME : La demande est croissante de systèmes qui nécessitent une adaptation. Ces systèmes doivent avoir la capacité d'adapter leur comportement de façon autonome durant l'exécution en fonction de l'évolution de leur environnement. Parmi les applications nécessitant une capacité d'auto-adaptation : les systèmes automobiles, de télécommunication, de surveillance et les systèmes de maison intelligente. Cependant, malgré son importance, l'auto-adaptation est souvent construite de manière ad-hoc. Dans cet article, nous présentons « Pattern-based Specification for Self-Adaptive Systems (PSAS) », un langage de spécification pour l'auto-adaptation avec un outil support pour faciliter le processus de spécification. La sémantique est présentée en termes de logique floue. Ainsi, un traitement minutieux des exigences, afin de formuler avec précision des exigences relatives aux systèmes auto-adaptatifs, facilite la conception de systèmes flexibles et adaptatifs de manière systématique. Pour montrer l'applicabilité et l'efficacité de notre langage, nous l'appliquons sur un protocole de communication dans les réseaux de capteurs.

ABSTRACT. There is a growing demand for systems that require adaptation. These systems must have the ability to adapt their behavior independently during execution according to the evolution of their environment. Among applications requiring a self-adaptive capacity: automotive systems, telecommunication systems, monitoring systems and intelligent home systems. However, despite its importance, self-adaptation is often constructed in ad-hoc manner. In this paper, we present "Pattern-based Specification for Self-Adaptive Systems (PSAS)", a specification language for self-adaptation with a support tool to facilitate the specification process. Semantics is presented in terms of fuzzy logic. Thus, careful processing of requirements, in order to accurately formulate requirements for self-adaptive systems, facilitates the design of flexible and adaptive systems in a systematic manner. To demonstrate the applicability and effectiveness of our language, we apply it to a protocol of sensors network.

MOTS-CLES Ingénierie des exigences, Systèmes auto-adaptatifs, Patron de spécification, Logique Temporelle Métrique Floue.

KEYWORDS Requirement engineering, Self-adaptive systems, Specification patterns, Fuzzy metric temporal logic.

1. Introduction

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 comme par exemple dans un système de bureau intelligent où les appareils doivent être synchronisés (exigence), mais le nombre des appareils ne peut pas être connu durant la conception. A la lumière de cela, il devient vital pour ces systèmes de s'adapter automatiquement aux changements qui se produisent dans l'environnement comme le nombre de périphériques dans l'exemple précédent. 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 (Iftikhar et Weyns, 2012). 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 (Whittle et al., 2009). Pour clarifier le concept de SAA, nous rappelons quelques définitions :

- **Définition 1**. Un système auto-adaptatif évalue son propre comportement et modifie sa propre performance lorsque l'évaluation indique que n'est pas accompli ce que le logiciel est censé faire, ou lorsque de meilleures fonctionnalités ou performances sont possibles (Salehie et Tahvildari, 2005).
- **Définition 2**. Un système auto-adaptatif est un système en boucle fermée qui peut se modifier lui-même dû aux changements continus du système, ses exigences et les tendances existantes de développement et de déploiement des systèmes complexes, réduisant les interactions humaines. La conception de systèmes auto-adaptatifs dépend des besoins de l'utilisateur et des propriétés et caractéristiques environnementales du système. Les logiciels auto-adaptatifs nécessitent une grande fiabilité, robustesse, adaptabilité et disponibilité (Weyns et al., 2012).

Les SAAs ont besoin de spécifications flexibles tout en étant formelles, d'où la nécessité d'une logique souple. Nous choisissons la logique floue (Zadeh, 1965) afin de gérer des situations où l'environnement contient des incertitudes.

Dans de nombreux systèmes réels avec incertitude, l'application de la logique floue, a donné des résultats très fructueux. Elle est le résultat de plus de 3 décennies d'études dans le domaine de la spécification et de la vérification et offre de nombreux avantages aux praticiens. Elle reste néanmoins difficile (Dwyer et al., 1999) en raison de la distance importante entre les formalismes employés par les outils de vérification

des modèles et le langage naturel des exigences (Abid et al., 2011). Cette limitation implique l'expression de propriétés à un niveau d'abstraction élevé en utilisant un langage basé sur l'anglais structuré (Konrad et Cheng, 2005), qui est naturel et limité par le formalisme.

Cet article présente un nouveau langage de spécifications basé sur des patrons pour les SAAs afin de faciliter l'expression des contraintes temporelles d'auto-adaptation.

La suite de l'article est organisée comme suit : dans la section suivante, nous définissons les concepts nécessaires pour notre catalogue de patrons. La section 3 se concentre sur nos patrons et leurs sémantiques, nous présentons dans cette section la grammaire d'anglais structuré pour notre langage. Dans la section 4, nous présentons notre outil de support ainsi que nous justifions l'efficacité de notre langage via des instances du monde réel, nous renforçons également notre proposition par une évaluation empirique dans la section 5. Avant de conclure, nous décrivons les travaux connexes effectués dans ce domaine dans la section 6.

2. Contexte

2.1. Patrons de spécification

Un patron de spécification est un modèle pour représenter un sous ensemble de propriétés qui s'expriment de la même manière (Dwyer et al., 1999). Selon (Autili et al., 2015), les patrons sont regroupés en trois familles : les patrons qualitatifs qui décrivent la matérialisation des événements, les patrons 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.

2.2. Portée

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 1 donne les définitions des portées présentées dans (Dwyer et al., 1999). La figure 1 illustre un aperçu du fonctionnement des portées.

Tableau 1 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.

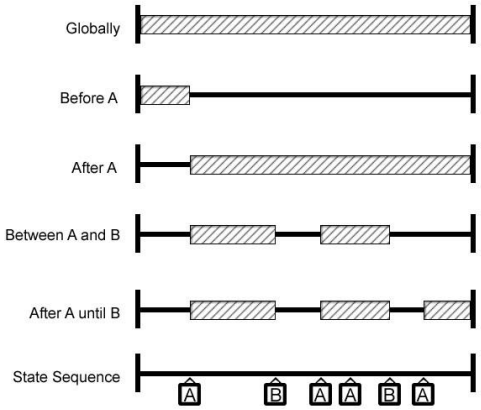

Figure 1 Portée du motif : présente la chronologie autorisée d'un motif spécifique, délimitée par les événements A et B (Dwyer et al., 1999)

3. Présentation du langage

Dans cette section, nous présentons notre langage qui permet aux ingénieurs d'exprimer les propriétés du système de manière flexible. Nous avons conçu des patrons afin de permettre aux concepteurs de savoir que l'exigence peut être changée 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. Notre langage supporte la spécification de sources multiples d'incertitude (Whittle et al., 2009 ; Esfahani et Malek, 2013), de manière déclarative flexible (as early as possible) ou d'énumérer les alternatives possibles (may R1 or may R2…) ce qui permet au développeur d'introduire l'auto-adaptation dans le système, par exemple : « le message d'acquittement doit parvenir dans 10 millisecondes », cette exigence est invalide si le message parvient dans 12 millisecondes. Par contre si on utilise le patron « TheEarliestPossible » (présenté ci-dessous) l'exigence est toujours valide mais avec un degré de vérité inferieur à 1.

Notre langage présente plusieurs atouts. Le premier est la possibilité que les exigences soient soulagées par la souplesse de l'expression du patron (as possible), qui se traduit après par la logique floue, afin d'assurer le fonctionnement régulier du système aussi longtemps que possible. Le second atout qu'il repose sur la grammaire anglaise structurée présentée dans (Konrad et Cheng, 2005 ; Autili et al., 2015), ce qui fait de lui un langage sans ambiguïté.

3.1. Catalogue de patron auto-adaptatif

Les patrons (abordés dans la section 2) 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. Pour des raisons de place, nous ne proposons ici qu'un sous-ensemble de patrons de spécification auto-adaptatifs. Ils peuvent être considérés comme des versions soulagées de ceux existants, par exemple : "TheLongestPossible" peut être considéré comme une version soulagée du patron "Universality" (Dwyer et al., 1999). La classification des patrons proposés est la même que celle présente dans (Autili et al., 2015). Les patrons sont divisés en deux classes principales, l'occurrence auto-adaptative et l'ordre auto-adaptatif.

3.1.1. 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 : nous proposons ce patron 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.

3.1.2. Patrons auto-adaptatifs d'ordre

Ils sont utilisés pour spécifier l'ordre dans lequel certaines propriétés doivent se produire. Dans cet article, 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).

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 sous-section 3.3.

3.2. Syntaxe du langage

La formulation des spécifications est soumise à la grammaire anglaise structurée présentée ci-dessous. Dans la première étape du processus de spécification, chaque propriété est représenté par le couple "portée, patron", ensuite on choisit la portée (présenté dans la section 2). En ce qui concerne la partie du patron (pattern), elle est générée par le non-terminal "Self-adaptative-Pattern". Ce dernier génère deux non-terminaux, Self-adaptive-Occurrence ou Self-adaptive-Order, ensuite le modèle non-terminal du patron, et enfin l'anglais structuré, le tableau 2 montre la grammaire d'anglais structurée utilisée dans ce papier.

Tableau 2 Grammaire structurée du langage A,B,C ∈{Exigences}, t_l^A ,t_u^A ∈R^+, t_l^A,t_u^A sont les bornes d'occurrence inférieure et supérieure, respectivement.

Property	::=	Scope, Self-adaptive-Pattern
Scope	::=	**Globally** | **Before** *{B}* | **After** *{C}* | **Between** *{C}* **and** *{B}* | **After** *{C}* **until** *{B}*
Self-adaptive-Pattern	::=	Self-adaptive-Occurrence | Self-adaptive-Order
Self-adaptive-Occurrence	::=	TheLongestPossible | TheShortestPossible | TheEarliestPossible | TheLatestPossible | SA-MinDuration | SA-MaxDuration
Self-adaptive-Order	::=	SA-Until | TheEarliestPossibleResponse
TheLongestPossible	::=	**It is the case that** *{***A***}* **[holds] as long as possible** [Time (A)].
TheShortestPossible	::=	**It is the case that** *{***A***}* **[holds] as short as possible** [Time (A)].
TheEarliestPossible	::=	**It is the case that** *{***A***}* **[holds] as early as possible** [Time (A)].
TheLatestPossible	::=	**It is the case that** *{***A***}* **[holds] as late as possible** [Time (A)].
SA-MinDuration	::=	**Once** *{***A***}* **[becomes satisfied] it remains as possible up to** t_u^A TimeUnits.
SA-MaxDuration	::=	**Once** *{***A***}* **[becomes satisfied] it remains as possible less than** t_u^A TimeUnits.
SA-Until	::=	**{A} [holds] without interruption until {D} [holds]** [Time(A)].

TheEarliestPossibleResponse	::=	**if {A} [has occurred] then in response {D} [holds] as early as possible** [Time(D)].
Time(A)	::=	UpperTimeBound(A) | LowerTimeBound(A) | Interval(A)
UpperTimeBound(A)	::=	**within** t_u^A TimeUnits
LowerTimeBound(A)	::=	**after** t_l^A TimeUnits
Interval(A)	::=	**between** t_l^A **and** t_u^A TimeUnits
TimeUnits	::=	**any denomination of time (e.g., seconds, minutes, hours, days, or years)**
QuantityUnits	::=	**any denomination of quantity (e.g., number of connected devices)**

3.3. Sémantique du langage

La sémantique de PSAS est définie en termes de Logique Métrique Temporelle Floue (LMTF) (Zhou, 1999). LMTF peut décrire un réseau de pétri temporel avec des informations temporelles incertaines. C'est la représentation de l'incertitude dans LMTF 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→[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 $d \in R^+$ est une durée floue s'il existe une incertitude floue sur la durée exacte de la durée. C'est-à-dire qu'il est associé à un nombre flou qui définit une longueur de temps floue.

□ est l'opérateur "toujours" habituel. **U** désigne "jusqu'à" comme avec la logique temporelle standard. **O**, qui prend la valeur de vérité de sa formule après une durée. La durée $d \in R^+$ 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, ◊ A = vraie **U** A, où ◊ signifie finalement.

Nous sommes maintenant prêts à définir la sémantique des expressions PSAS en termes de LTMF. Les définitions pour ces opérateurs, y compris l'incertitude, dépendent d'une durée flou 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.

Nos patrons sont des versions étendues, en majeure partie, de ceux présentés dans (Autili et al., 2015), ces patrons sont une version relâchée de ceux qui existent, la cartographie est étendue à partir de (Dwyer et al., 1999 ; Konrad et Cheng, 2005 ; Autili et al., 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 du patron d'occurrence **TheLongestPossible**. La cartographie des autres patrons se trouve sur le site http://www.psas-tool.com/support-tool/sa-patterns.

Globally	$\Box_{[\![time\_A]\!]}\left(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A\right)$
Before B	$\Diamond_{\geq [\![tl\_A]\!]} B \rightarrow \left(\left(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A\right) \mathrm{U}_{[\![time\_A]\!]}\ B \vee \Box_{[\![time\_A]\!]}\left(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A\right)\right)$
After C	$\Box\left(C \rightarrow \Box_{[\![time\_A]\!]}\left(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A\right)\right)$
Between C and B	$\Box\ \Big(\left(C \wedge \Box_{\leq [\![tl\_A]\!]} \neg B\right) \wedge \Diamond_{\geq [\![tl\_A]\!]} B \rightarrow \left(\left(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A\right) \mathrm{U}_{[\![time\_A]\!]}\ B \vee \Box_{[\![time\_A]\!]}\left(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A\right)\right)\Big)$
After C until B	$\Box\left(\left(C \wedge \Box_{\leq [\![tl\_A]\!]} \neg B\right) \rightarrow \left(\left(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A\right) \mathrm{W}_{[\![time\_A]\!]}\ B\right)\right)$
Where : $[\![time\_A]\!] = [t_l^A, t_u^A]$, $[\![end\_A]\!] = \infty$, $[\![tl\_A]\!] = t_l^A$	

Dans cette section, 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 notre grammaire basée sur l'anglais structuré, de la sélection jusqu'au détail du patron. Ensuite, la sémantique du langage a été présentée en termes de logique floue. Cependant, l'efficacité de notre langage doit être testée à travers des instances réelles et des cas d'études.

4. PSAS-tool

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

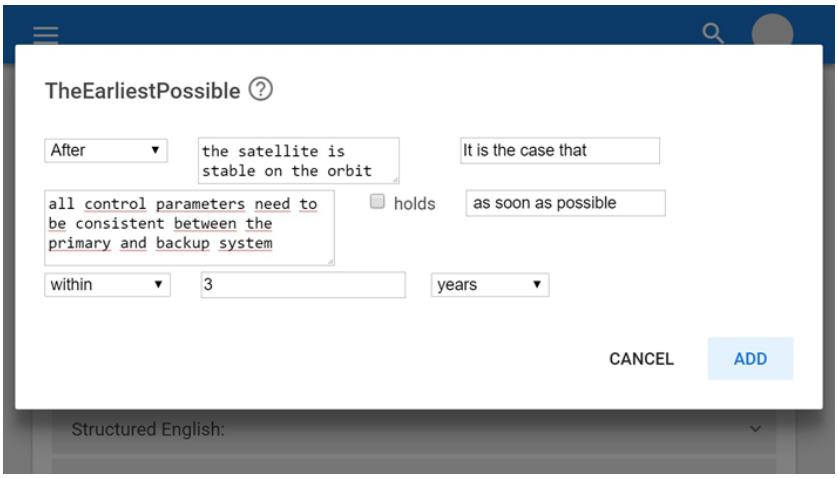

Figure 2 Sélection du patron et création de spécification

4.1. 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 et al., 1999). Nous avons donc conçu un outil de support afin de faciliter le travail du concepteur. L'objectif principal de l'outil PSAS-tool est d'automatiser la traduction des expressions de grammaire anglaise 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 (figure 2), puis le remplissage des 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 généré automatiquement (figure 3). Enfin, la liste des spécifications peut être exportée au format pdf.

L'exemple dans la figure 2 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.

- Zone de texte pour l'événement.

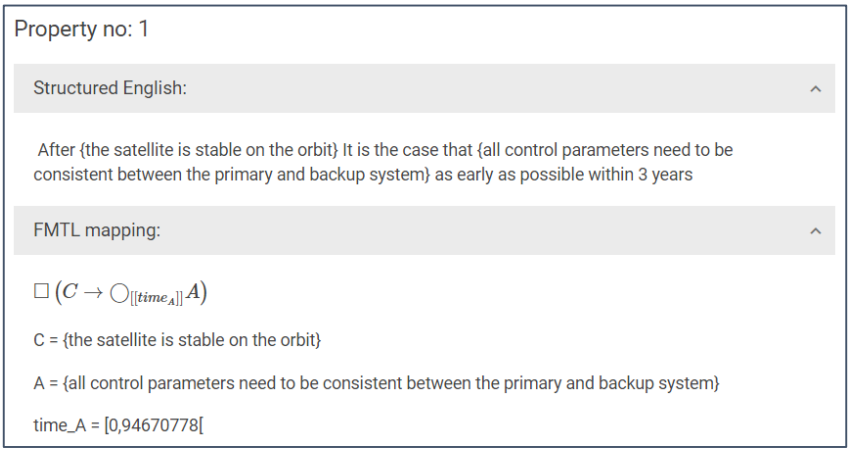

Figure 3 Spécification en anglais structuré et en formule FMTL

- Zone de texte qui décrit l'exigence originale.
- Checkbox optionnelle "holds", pour exprimer clairement l'exigence.
- Définition d'intervalle de temps (within, at least).
- Entrée de la durée liée à la spécification.
- Unité de temps (second, minute, hour ...)

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 orienté web, ce choix implique divers avantages, d'abord, les applications web permet 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.

4.2 Exemples de patrons auto-adaptatifs

Toutes les spécifications doivent être conformes à la règle : "Portée, patron" (section 3). 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.

TheLongestPossible

Spécification = Portée, Patron

= Globally, TheLongestPossible.

= Globally, It is the case that {P} holds as long as possible [Time(P)]. */Time(P) optionnel, s'il n'est pas spécifié donc Time(P) = [0,∞[.*

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

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).

SA-MaxDuration

Patron: Globally, Once {A} [becomes satisfied] it remains as possible less than tuATimeUnits.

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, 2006))

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

5 Evaluation empirique

L'objectif est ici de montrer que les patrons de spécification présentés traitent des problèmes actuels de spécifications auto-adaptatives. Nos patrons de spécification sont conçus principalement pour l'expression du temps. Nous présentons ici un cas d'étude sur le protocole de communication "Message Queue Telemetry Transport (MQTT)" où le temps intervient. 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, 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 une

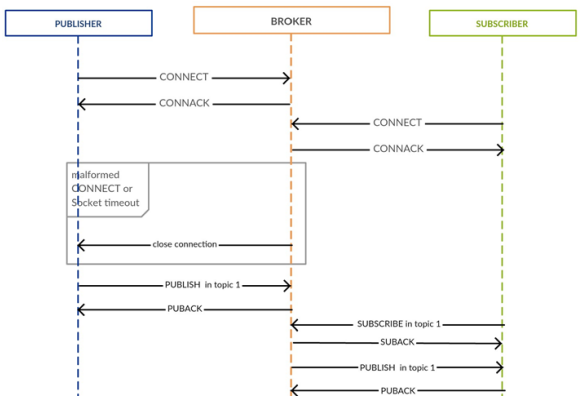

Figure 4 Aperçu sur le protocole MQTT

utilisation dans des environnements avec contraintes comme l'Internet of Things (IoT) et le Machine to Machine (M2M). La figure 4 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 et al., 2015) pour les exigences invariantes et notre catalogue proposé pour les exigences d'auto-adaptation.

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.

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.

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}).

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 plus souples que les patrons 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 Travaux connexes

C. Krupitzer et al. (2015) ont parlé des systèmes auto-adaptatifs et des défis dans ce domaine. Faire face à l'incertitude est l'un des défis importants pour le domaine de l'ingénierie des exigences. Il peut être matérialisé dans un nouveau langage pour spécifier les exigences (Dey, 2001). Un travail similaire à celui proposé dans cet article est RELAX (Whittle et al., 2009). Les auteurs visent, par leur langage, à gérer l'incertitude d'une manière déclarative à l'aide d'opérateurs temporels, ordinaux et modaux. Ils choisissent pour leur cartographie, la logique temporelle de branchement floue (Moon et al., 2004), alors que notre langage est basé sur des patrons, avec une grammaire anglaise structurée (Autili et al., 2015) pour traiter l'ambiguïté dans les spécifications. De plus, notre langage permet l'expression de tous les opérateurs RELAX et plus. De manière similaire à RELAX, notre langage vise également à soutenir des adaptations non prévues. Dans un article ultérieur (Cheng et al., 2009), RELAX a été utilisé avec la modélisation d'objectifs pour spécifier l'incertitude dans d'autres sources. Ils construisent d'abord le modèle de but, puis l'utilisent de manière ascendante pour rechercher les sources d'incertitude qui sont les éléments du domaine/environnement et peuvent compromettre la satisfaction des objectifs.

James F. Allen (Allen, 1983) décrit un système de raisonnement sur les intervalles temporels expressif. Cette approche est utile dans les domaines où l'information temporelle est imprécise et relative, cependant cette dernière ne donne pas d'information précise concernant le degré de vérité des expressions. Baresi et al. (2010) traitent l'incertitude des objectifs via FLAGS. Analogue à RELAX, ils ont pour but d'atteindre l'objectif principal des systèmes adaptatifs au niveau des exigences : atténuation de l'incertitude attachée aux besoins de l'environnement en intégrant l'adaptabilité dans le système dès l'élicitation des exigences. Les exigences

spéciales à FLAGS sont appelées objectifs adaptatifs. Ils permettent les stratégies de prévention qui doivent être réalisées si certains objectifs ne sont pas satisfaits comme prévu. FLAGS traite également une autre source d'incertitude. L'incertitude dans les objectifs eux-mêmes. FLAGS est basé sur des objectifs flous pour lesquels les propriétés ne sont pas complètement connues. La spécification complète n'est pas disponible et de petites violations temporaires sont tolérées. Ainsi, FLAGS se terminent par deux séries d'objectifs : nettes et floues. Il formalise les buts en utilisant la logique temporelle floue pour les langages flous et la logique temporelle linéaire pour les autres. FLAGS s'appuie également sur une grammaire ambigüe, ce qui le rend difficile à gérer par des ingénieurs moins expérimentés (Autili et al., 2015).

Tableau 3 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

Deux nouvelles catégories d'exigence ont été proposées. Les exigences de sensibilisation (Souza et al., 2011) servent à surveiller les exigences du système et les exigences d'évolution dédiées à la représentation des plans d'adaptation afin de faire face aux changements dans les modèles d'exigences (Souza et al., 2012). A. Manzoor et al. (2015) emploient à la fois des approches déclaratives et basées sur des buts. Ils ont utilisé RELAX (Whittle et al., 2009) pour spécifier les exigences non-fonctionnelles et les notions GORE pour susciter et modéliser les exigences des systèmes auto-adaptatifs. E. Vassev et M. 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. D'autres approches ont été proposées, comme les approches basées sur les objectifs et les agents. Ceux fondés sur des modèles d'objectifs sont par exemple i* (Yu, 1997), KAOS (Dardenne et al., 1993), FLAGS (Baresi et al., 2010). Tropos4AS (Morandini et al., 2017) est une approche récente basée sur les agents, qui repose sur Tropos (Bresciani et al., 2004).

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 ambigüe, 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 3 résume certaines approches similaires.

7 Conclusion

Cet article vise à présenter un nouveau langage de spécification pour les systèmes auto-adaptatifs. L'objectif principal de notre 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 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 comme objectif de traiter 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 donné un ensemble d'exemples pour apporter un sens à notre proposition. Nous avons également présenté 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.

8 Références

Abid N. et al. (2011). A Real-Time Specification Patterns Language, https://hal.archives-ouvertes.fr/hal-00593965

Autili M. et al. (2015). Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. IEEE Transactions on Software Engineering, vol. 41, n° 7, p. 620-638.

Banks A., Gupta R. (2014). Version 3.1.1. 29 October 2014. OASIS Standard. s.f.

Baresi L. et al. (2010). Fuzzy goals for requirements-driven adaptation. Requirements Engineering Conference (RE), 18th IEEE International 2010.

Bellini P. et al. (2009). Expressing and organizing real-time specification patterns via temporal logics. Journal of Systems and Software, vol. 82, n° 2, p. 183-196.

Bresciani P. et al. (2004). Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems, vol. 8, n° 3, p. 203-236.

Capozucca A. (2006). The fault-tolerant insulin pump therapy. Rigorous Development of Complex Fault-Tolerant Systems, Springer Berlin Heidelberg, p. 59-79.

Cheng B. HC. et al. (2009). A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. Model Driven Engineering Languages and Systems, Springer, p. 468-483.

Dardenne A. et al. (1993). Goal-directed requirements acquisition. Science of computer programming, vol. 20, n° 1-2, p. 3-50.

Dey A K. (2001). Understanding and using context. Personal and ubiquitous computing 2001, vol. 5, n° 1, p. 4-7.

Dwyer M. et al. (1999). Patterns in property specifications for finite-state verification. Software Engineering, Proceedings of the 1999 International Conference, IEEE.

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

Gruhn V., Laue R. (2006). Patterns for timed property specifications. Electronic Notes in Theoretical Computer Science, vol. 153, n° 2, p. 117-133.

Grunske L. (2008). Specification patterns for probabilistic quality properties. in Proceedings of the 30th International Conference on Software Engineering, IEEE, p. 31-40.

Konrad S., Cheng B. HC. (2005). Real-time specification patterns. Proceedings of the 27th international conference on Software engineering, IEEE, p. 372-381.

Koymans R. (1990). Specifying real-time properties with metric temporal logic. Real-time systems, vol. 2, n° 4, p. 255-299.

Krupitzer C. et al. (2015). A survey on engineering approaches for self-adaptive systems. Pervasive and Mobile Computing, vol. 17, p. 184-206.

M. Usman Iftikhar and Danny Weyns, A Case Study on Formal Verification of Self-Adaptive Behaviors in a Decentralized System. Proceedings 11th International Workshop on Foundations of Coordination Languages and Self Adaptation, FOCLASA2012, Newcastle, U.K., September 8, 2012, pages 45-62.

Manzoor A. et al. (2015). Modeling and Verification of Functional and Non-Functional Requirements of Ambient Self-Adaptive Systems. Journal of Systems and Software, vol. 107, p. 50-70.

Macias-Escriva F D. et al. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. Expert Systems with Applications, vol. 40, n° 18, p. 7267-7279.

Moon S. et al. (2004). Fuzzy branching temporal logic. Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 34, n° 2, p. 1045-1055.

Morandini M et al. (2017). Engineering requirements for adaptive systems. Requirements Engineering, vol. 22, n° 1, p. 77-103.

Rajeev A. (1991). Techniques for automatic verification of real-time systems. Stanford University.

Salehie M., Tahvildari L. (2005). Autonomic computing: emerging trends and open problems. ACM SIGSOFT Software Engineering, vol. 30, p. 1-7.

Souza V E S. et al. (2011). Awareness requirements for adaptive systems. Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems, ACM, p. 60-69.

Souza V E S. et al. (2012). (Requirement) evolution requirements for adaptive systems. Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, IEEE, p. 155-164.

Vassev E., Hinchey M. (2015). Engineering Requirements for Autonomy Features. Software Engineering for Collective Autonomic Systems, Springer, p. 379-403.

Weyns D., Iftikhar M. U., Malek S., Andersson J. (2012). Claims and supporting evidence for self-adaptive systems-A literature study, SEAMS'12, Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, p.89-98.

Whittle, J., Sawyer P., Bencomo N., Cheng B. HC., Bruel J. M. (2009). Relax: Incorporating uncertainty into the specification of self-adaptive systems. In Requirements Engineering Conference, 2009. RE'09. 17th IEEE International. IEEE, p. 79-88.

Yu Eric SK. (1997). Towards modelling and reasoning support for early-phase requirements engineering. Requirements Engineering, Proceedings of the Third IEEE International Symposium, IEEE, p. 226-235.

Zadeh L A. (1965). Fuzzy sets. Information and control, vol. 8, n° 3, p. 338-353.

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