Temporal Pattern Specifications for Self-Adaptive Requirements

Ayoub Yahiaoui, Hakim Bendjenna, Philippe Roose, Lawrence Chung, Mohamed Amroune

Send Orders for Reprints to reprints@benthamscience.ae

Recent Patents on Computer Science **2018,** *11,* **1-11**

RESEARCH ARTICLE

Temporal Pattern Specifications for Self-Adaptive Requirements

Ayoub Yahiaoui[a,*], Hakim Bendjenna[a], Philippe Roose[b], Lawrence Chung[c] and Mohamed Amroune[a]

[a]*Department of Computer Science, Faculty of Science, University Larbi Tebessi, Tebessa, Algeria;* [b]*IUT de Bayonne/LIUPPA-T2I, France,* [c]*University of Texas at Dallas, US*

ARTICLE HISTORY

Received: July 16, 2018
Revised: September 25, 2018
Accepted: October 11, 2018

DOI:
10.2174/2213275911666181019115744

Abstract: ***Background*****:** Systems whose requirements change at a rate that necessitates adaptation without human intervention are called self-adaptive systems, and they have the ability to adjust their behavior autonomously at run-time in response to their environment's evolution. Samples of applications that require self-adaptation include Smart home systems and environmental monitoring. However, self-adaptivity is often constructed in an ad-hoc manner.

Method**:** In this paper, the authors present a pattern-based specification language for self-adaptive systems. Its semantics are presented in terms of fuzzy logic. Thus, enabling a meticulous processing of requirements, in order to permit the formulation of self-adaptive requirements accurately, thereby facilitates the design of systems that are flexible and responsive to adaptation in a systematic manner.

Results**:** To show the applicability and effectiveness of our language, the authors apply it to two case studies. One case study reviews the Smart fridge in ambient assisted living and the second case study is focused on an ambulance dispatching system using a developed support tool.

Keywords: Adaptive requirements, specification patterns, formal specification, fuzzy logic, metric temporal logic, self-adaptive system, fuzzy metric temporal logic.

1. INTRODUCTION

Technology applications have become increasingly large and even more heterogeneous and complex. In some particular tasks where environmental data cannot be defined during the design phase, a system must adapt itself dynamically. For example, in a Smart Office system, devices must have the same data all of the time, so the duration of the synchronization process is not well known, due to the changeable number of devices. Therefore, it becomes crucial for such systems to self-adapt dynamically in order to change according to what is occurring in the environment, such as synchronization time in the previous example. These systems are called SAS (self-adaptive system). However, SASs need flexible specifications, whereas specifications must be formal, hence, the necessity to apply a flexible logic. The authors chose fuzzy logic [1] in order to handle situations where the environment contains uncertainties.

The application of fuzzy logic, in many real-world systems with uncertainty, provided very fruitful outcomes. Results of over three decades of studies in the domain of formal specification and verification offer many advantages for practitioners, but they have not been well adopted [2] due to the wide distance between the low-level of formalisms employed by the tools of model checking and the natural language used in requirements specifications [3]. This limitation involves the expression of properties at a high abstraction level using the structured English language.

Moreover, the smooth functioning of an SAS can be compromised by two key sources [4]. The first source is behavioral. It invokes situations where the requirements need to be changed. For example, the requirements of a telecommunication network protocol that needs to change its configuration in order to recover its convergence in case of router a failure. The second source is environmental changes that occur, such as sensor failures and unexpected input (human interaction). It is complex to predict all requirement changes at design time and it may not be possible to enumerate all possible alternatives.

Thus, this paper presents a new pattern-based requirements language for SASs. The authors introduce some concepts to specify self-adaptation in requirements. The self-adaptive patterns used are relaxed versions of existing patterns. In other words, they can be seen as a self-adaptive extension of previous works on specification patterns [2, 5, 6]. A specification language must be, first, natural to allow the engineer to freely and easily express requirements, and second, it must be based on a formalism that enables automatic verification. For this purpose, the researchers' proposed PSAS (Pattern Specification for SAS), a language that relies on SE (Structured English) with a mapping in FMTL (Fuzzy Metric Temporal Logic). [2]. Also, in order to facilitate the use of the language, researchers developed a PSAS-tool, an application that allows for the automation of struc-

*Address correspondence to this author at the Department of Computer Science, Faculty of Science, University Larbi Tebessi, Tebessa, Algeria; Tel: +213552419087; E-mail: yahiaoui.ayoub.2015@gmail.com

2213-2759/18 $58.00+.00

tured English grammar expressions to be translated into fuzzy metric temporal logic formulas.

The remainder of the paper is organized in the following manner: the authors describe related works conducted in this domain. In the third section, the authors define notations for their patterns semantics, focus on the proposed patterns, their syntax, and semantics, and then justify the efficiency of the language via real-world instances. The authors also strengthen their proposal by providing an empirical evaluation.

2. RELATED WORKS

Authors in some studies [7, 8] talked about state-of-the-art self-adaptive systems and challenges in this area. Dealing with uncertainty is one of the important challenges nominated for requirements in the engineering domain; it can be materialized in a new language for specifying requirements [9]. A similar work is RELAX [4], Whittle *et al.* aimed, through their language, to handle uncertainty in a declarative fashion using temporal, ordinal, and modal operators. They chose fuzzy branching temporal logic for their mapping [10]. Now, the authors propose a pattern-based language with structured English grammar [6] to deal with ambiguity in the requirements specification. Furthermore, this language allows expression of all RELAX operators. Similar to RELAX, the authors' language also aims to support unexpected adaptations. In an ulterior paper [11], RELAX was used with goal modeling to specify the uncertainty in other sources. They first build the goal trellis and then use it in a bottom-up manner to search for sources of uncertainty, which are the elements of domain/environment and can jeopardize goals satisfaction.

Baresi, Pasquale, and Spoletini deal with the uncertainty of the objectives via FLAGS (Fuzzy Live Adaptive Goals for Self-adaptive systems) [12]. Analogous to RELAX, the objective of FLAGS is to achieve the main goal of adaptive systems at the requirements level: mitigation of the uncertainty attached to environment needs by incorporating adaptability in the software system as early as requirement elicitation. FLAGS special requirements are called adaptive goals. They allow the prevention strategies that must be performed if some goals are not satisfied as expected. FLAGS also deals with another source of uncertainty, which is the uncertainty in the goals themselves. FLAGS is based on fuzzy goals for which properties are not completely known. The complete specification is not available, and small temporary violations are tolerated. Thus, FLAGS finishes by two sets of goals: crisp and fuzzy goals. It formalizes goals using fuzzy temporal language for fuzzy ones and Linear Temporal Logic for the others. FLAGS also relies on an ambiguous grammar, which makes it hard to handle by less experienced engineers [6].

Souza [13] presented two new requirements categories; awareness requirements for monitoring a system and evolution requirements dedicated to representation of adaptation plans in order to address changes in requirements models [14]. Ahmad, Belloir, and Bruel [15], employed both declarative and goal-based approaches.

They used RELAX [4] to specify non-functional requirements and the GORE (Goal-Oriented Requirements Engineering) notions to elicit and model the requirements of self-adaptive systems. Hinchey and Vassev [16] proposed an approach for ARE (Autonomy Requirements Engineering). This approach, based on GORE and GAR (Generic Autonomy Requirements), has the objective for eliciting, specific autonomy requirements, and defines assistive and possible alternative objectives for self-adaptive systems. Other approaches were proposed, such as goal-based like FLAGS [12] and Tropos4AS (Tropos for Adaptive System) [17], a recent agent-based approach.

Recently, Didac Gil De La Iglesia and Danny Weyns [18] proposed a set of formal MAPE-K (Monitoring Analyse Planning Execution and Knowledge) templates for a specific family of self-adaptive systems. They comprise behavior and property specification templates based on networks of timed automata and timed computation tree logic that support automated verification of the correctness. This approach has some similarities with the one presented in this paper, they used timed temporal logic for formalism. Carlos Eduardo da Silva *et al.* [19], also presented an approach for dynamic reconfiguration of role-based access control for business processes, they relied on probabilistic computation tree logic for specifying properties. However, this approach doesn't give a clear support for uncertainty.

The works discussed above held a special interest in uncertainty problems. Dealing with uncertainty in the early phases of development is necessary. However, some problems in the specification language must be handled. For example, language grammar must be unambiguous. Therefore, using natural language in specifications is limited due to its ambiguity. The authors' work processes, both natural language and grammar ambiguity via a patterns catalog and structured English grammar. Table **1** summarizes similar approaches.

Table 1. Self-adaptation languages overview.

Language	Uncertainty Handling	Structured Grammar	Pattern-Based	Mapping
RELAX	Yes	No	No	FBTL[1]
FLAGS	Yes	No	No	FTL[2]
MAPE-K FT	Yes	No	Yes	TCTL[3]

[1]Fuzzy Branching Temporal Logic
[2]Fuzzy Temporal Language
[3]Timed computational tree logic

3. BACKGROUND

Before beginning the presentation of the researchers' language, some technical definitions are necessary.

3.1. Self-Adaptive Systems

A self-adaptive system evaluates its own behavior and changes its own performance when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible [20].

3.2. Specification Pattern

A specification pattern is a common representation of a set of commonly occurring requirements on the allowed event/state succession in a system model (scopes). It furnishes expressions of a system's behavior in common formalisms and describes the basic structure of some sides of a system's behavior. For each pattern, a scope must be coupled with it; the scope represents a part of the system execution over which the pattern must hold. There are five basic kinds of scopes: *globally, before, after, between* and *after-until*. The scope is defined by indicating a start and an ending point for the pattern. Table **2** provides definitions of scopes presented in [2]. Fig. (**1**), from [2], offers an overview of how scopes work.

Table 2. Patterns scopes.

Scope	Pattern can hold
Globally	In any point of system run-time.
After {A}	Only after the occurrence of A.
Before {A}	Only before the occurrence of A.
Between {A} and {B}	Only between A and B.
After {A} until {B}	Only after the occurrence of A until B is true (B is not required)

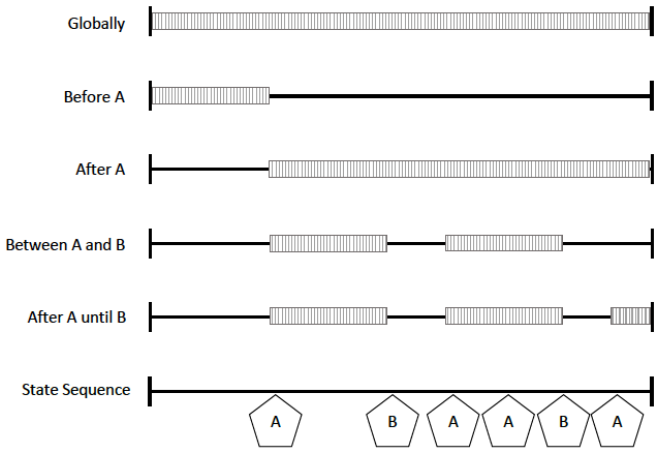

Fig. (1). Pattern scopes: present allowed timeline of a specific pattern, delimited by events A and B [2](Dwyer, Avrunin, & Corbett, 1999).

Patterns are grouped into three families: Qualitative patterns presented in a study (Dwyer, Avrunin, & Corbett, 1999), describe the materialization of events; real-time patterns [5] extend the first family by adding time constraint and probabilistic patterns [21] that also extend qualitative patterns by adding the probabilistic constraint. The authors also proposed mapping to temporal logics, including CTL (Computation Tree Logic), MTL (Metric Temporal Logic), and PCTL (Probabilistic Computation Tree Logic), as they represent formal expressions of patterns.

Works presented above focus on qualitative, real-time, and probabilistic properties. They do not give any support for self-adaptation. Table **3** presents a brief comparative description of existing patterns. Patterns in this work rely mainly on those recently presented by Autili *et al.* [6]. They can be seen as self-adaptive extensions. A specification pattern must be mapped in a temporal logic formula so that it brings formal meaning to the pattern. Thus, on one hand, a pattern plays the role of an interface for a temporal logic formula. On the other hand, the patterns catalog relies on [6] that provides mapping in MTL. That leads the authors to propose mapping in terms of FMTL (Fuzzy Metric Temporal Logic). Because of limits imposed by the formalism used in this paper, only related patterns are presented.

Table 3. Overview on patterns catalogs.

Catalog	Qualitative	Real-time	Probabilistic
[2]	Yes	No	No
[5]	Yes	Yes	No
[21]	Yes	No	Yes

3.3. Fuzzy Metric Temporal Logic (FMTL)

Yi Zhou and Tadao Murata [22] have proposed FMTL; it allows the expression of fuzzy-timing in real-time applications. It includes the references of explicit time that can be translated into time sequences.

FMTL grammar: Given a set of atomic propositions AP. An FMTL formula is as follows:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid O\varphi \mid \Box\varphi \mid \Diamond\varphi \mid \varphi_1 U \varphi_2 \mid$$
$$O_{\sim d}\varphi \mid \Box_{\sim d}\varphi \mid \Diamond_{\sim d}\varphi \mid \varphi_1 U_{\sim d}\varphi_2 \quad (1)$$

Where p ∈ AP, and $\sim \in \{\leq, =, \geq\}$ and d is a fuzzy duration.

Using FMTL provides interesting results concerning the precision of time expression. It allows for specifying the time in such a way that it does not take a specific value.

4. LANGUAGE OVERVIEW

This section presents the pattern-based language that allows analysts to express system properties in a versatile manner. The patterns conception was created to enable analysts to know what requirement must be changed during runtime when an environmental change occurs. The system might be able to ignore noncritical requirements, temporarily, in order to ensure that vital requirements can be satisfied. The researcher's pattern-based language supports the specification of multiple sources of uncertainty in a declarative way rather than listing possible alternatives [23], whereas, possible alternatives can always be expressed via an alternative pattern.

The researchers' language presents several assets. The first one is the possibility of requirements becoming relaxed due to pattern flexibility or via alternatives, in multiple forms according to the degree of relaxation (alternative and recurrence-alternative patterns) in order to ensure that regular

functioning of the system holds as long as it can. The second asset is the fact that the language relies on structured English grammar [5] to provide an unambiguous language.

4.1. Self-Adaptive Patterns Catalog

Qualitative and real-time patterns are organized into two categories: occurrence and order. The first one presents event materialization while the second captures a sequence of events. The authors propose a complete set of self-adaptive patterns. For example, "TheLongestPossible" pattern can be considered as a relaxation of the "Universality" pattern [6]. The classification of proposed patterns is similar to that proposed in another study [5].

4.1.1. Self-Adaptive-Occurrence Patterns

These patterns are employed to state that some configuration of properties must always take place eventually, or they do not occur in a versatile way.

- *The{Longest|Shortest}Possible*: In order to handle situations where property must hold as long (short) as possible.
- *The{Earliest|Latest}Possible:* This pattern has as an objective to make property happen as early (late) as possible.
- *SA-{Min|Max}Duration:* This pattern implies property to hold for a min (max) time, but this threshold is relaxed to allow the property to hold around it.
- *TheClosestPossibleTo-q: It i*-s used to express quantity (q) in a versatile manner, where values around it can be tolerated.
- *As {many| few} as possible:* This phrase is used to express, as close as possible, an undetermined and ideal quantity.
- *SA-Recurrence: It is* analogous to "The Closest-PossibleTo-q." This pattern presents the frequency of property holding, *i.e.* property holds as possible every t (t is a duration).
- *Alternative: It is* used to specify alternatives for a particular event/state, this pattern is useful for system reconfiguration specification.
- *Alternative-SA-recurrence:* This is a hybrid pattern used to express relaxed versions of a temporal state.

4.1.2. Self-Adaptive-Order Patterns

These patterns are used to specify the order in which some properties must occur. This kind of pattern includes two families: recurrence and response patterns. In this paper, the authors propose a self-adaptive version for "Until" and "Response" family patterns.

- *SA-Until:* "Until" pattern is native in most of the temporal logics; it was extended in a study [24], in order to handle timed properties. The authors propose a relaxed version of the latter, where the time bound is fuzzy.
- *The{Earliest|Latest}Possible-Response (1-N | N-1):* This pattern means that every time, a property X (stimulus) holds, it must be pursued (caused) by the chain of property Yi $1 \leq i \leq n$ (responses), as early (late) as possible.

The patterns catalog presented above aims to bring more expressiveness during the phase of specification. It is generated from the grammar presented in the earlier text along; its semantics are also presented.

4.2. Language Syntax

Pattern generations follow the structured English grammar presented below:

Table **4** illustrates the first step in the process of specification. Each property is formed of two pieces, a scope and a pattern, for the first part, we choose one of following keywords: "Globally", "After", "Before", "Between", or "After-Until". Concerning pattern part, it is generated via a Self-Adaptive-Pattern non-terminal (replaceable part in a grammar). The latter generates two non-terminals, Self-Adaptive-Occurrence or Self-Adaptive-Order, depending on what one is seeking to express. For example, if a specific sequence of events is desired, the Self-Adaptive-Order is used. Whatever the choice, it generates pattern non-terminal, the last step before structured English. Table **5** presents structured English (generated from pattern non-terminal). It contains fixed sentences such as, "it is the case that," optional parts between brackets like "[Time(A)]" and original requirement parts between "{ }", ex: {Terrestrial Photovoltaics (PV) systems face the sun}. Table **6** details time and quantity definitions.

4.3. Language Semantics

The authors present a mapping for their patterns in temporal logic in order to bring meaning to the language. The fuzzy metric temporal logic was selected to provide formalism for the language. Fuzzy logic is well adopted to handle uncertainty due to its flexibility. Table **7** presents *TheEarliestPossible* pattern translation in FMTL. The complete mapping of the catalog is presented in Appendix 1. In this section, the patterns catalog is presented to deal with self-adaptation expression in the requirements specifications. A grammar, based on structured English is also specified, from a pattern type selection to detail the core of the pattern. Then, language semantics are presented in terms of fuzzy logic. However, the efficiency of language must be tested via real-world instances and empirical case studies.

In order to explain how a pattern can be verified, the authors provide the example: Giving the property P = "**Globally**, **it is the case that** {Communications networks working } **holds as long as possible**", with truth threshold of 0.99. we check the validity of the property P over the part of the system shown in Fig. (**2**), which is an exclusion sample, where a common resource F is shared between two properties X and Y. There are the following two possible firing sequences to achieve the goal that both places $\neg A$ and E get a token.

$s_1 : M_0 [T_B> M_1 [T_C> M_2 [T_D> M_3 [T_E> M_4$

$s_2 : M_0 [T_D> M_5 [T_E> M_6 [T_A> M_7 [T_B> M_8$

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

M_x : stat of fuzzy petri net.

T_x : petri net transition.

Table 4. Pattern generation.

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

Table 5. Structured English for self-adaptive patterns.

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^ATimeUnits.
SA-MaxDuration	::=	Once *{*A*}* [becomes satisfied] it remains as possible less than t_u^ATimeUnits.
SA-Recurrence	::=	{A} [holds] repeatedly almost every t_u^ATimeUnits.
Alternative	::=	May $\{A_0\}$ [holds] or May $\{A_1\}$ [holds] or ..... May $\{A_i\}$
Alternative-recurrence	::=	May $\{A_0\}$ [holds] or May $\{A_1\}$ [holds] or ..... May $\{A_j\}$, where A_j [holds] repeatedly almost every T_jTimeUnits.
TheClosestPossibleTo-q	::=	{A} is as close as possible to q QuantityUnits.
As many as possible	::=	{A} [holds] as many as possible.
As few as possible	::=	{A} [holds] as few as possible.
SA-Until	::=	{A} [holds] without interruption almost until {D} [holds] [Time(A)] / A holds even D doesn't.
TheEarliestPossible-Response (1-N)	::=	If {A} [has occurred] then in response {D} [holds] as early as possible [Time (D)] followed by $(\{T_i\}$ [Time $(T_i)]$ $)^{(1\le i\le N-1;",")}$[eventually hold].
TheEarliestPossible-Response (N-1)	::=	If {S} followed by $(\{T_i\}$ [Time$(T_i)]$ $)^{(1\le i\le N-1;",")}$[have occurred] then in response {A} [holds] as early as possible [Time(A)]
TheLatestPossible-Response (1-N)	::=	If {A} [has occurred] then in response {D} [holds] as late as possible [Time (D)] followed by $(\{T_i\}$ [Time $(T_i)]$ $)^{(1\le i\le N-1;",")}$ [eventually hold]
TheLatestPossible-Response (N-1)	::=	If {D} followed by $(\{T_i\}$ [Time $(T_i)]$ $)^{(1\le i\le N-1;",")}$ [have occurred] then in response {A} [holds] as late as possible [Time(A)]

Table 6. Time and quantity definitions.

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

Table 7. Mapping of "TheEarliestPossible" in FMTL.

The Earliest Possible	
Globally	$O_{\geq[\![tl\_A]\!]}A$
Before B	$\Diamond_{\geq[\![tu\_A]\!]}B \rightarrow \left(A\mathrm{U}_{[\![time\_A]\!]}B \vee O_{\geq[\![tl\_A]\!]}A\right)$
Between C and B	$\Box\left(\left(C \wedge \Box_{\leq[\![tu\_A]\!]}\neg B\right) \wedge \Diamond_{\geq[\![tu\_A]\!]}B \rightarrow \left(A\mathrm{U}_{[\![time(A)]\!]}B \vee O_{\geq[\![tl\_A]\!]}A\right)\right)$
After C until B	$\Box\left(\left(C \wedge \Box_{\leq[\![tu\_A]\!]}\neg B\right) \rightarrow O_{\geq[\![tl\_A]\!]}\left(A\mathrm{W}_{[\![time\_A]\!]}B\right)\right)$

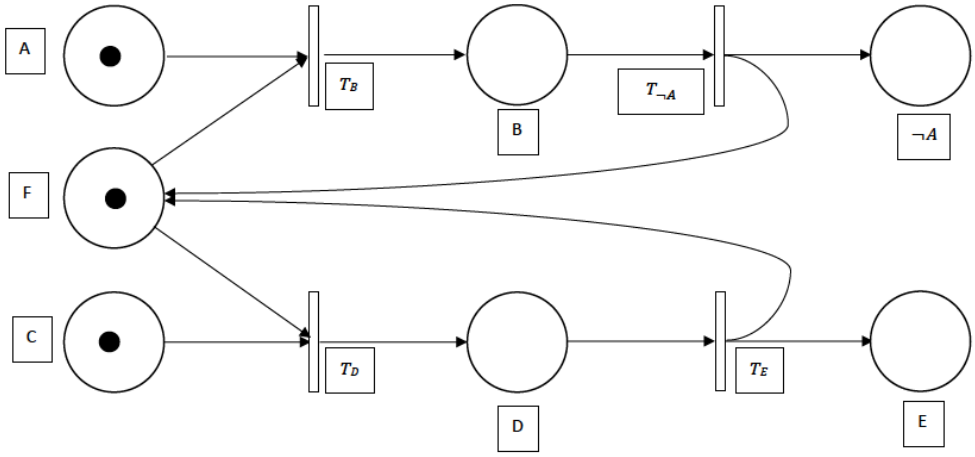

Fig. (2). Exclusion sample model.

$\pi_{0X}(\tau)$: initial trapezoidal fuzzy timing function.

Suppose that the duration of running system is 10 days so P becomes

$\Box_{[0,10]}\left(\{Communications\ networks\ working\} \wedge \Diamond_{[0,10]}\neg\{Communications\ networks\ working\}\right)$.

We check whether P is satisfied or not on the two possible sequences (s_1, s_2). By applying Model checking algorithm from [6], we find that s_1 gives 63,6 % degree of satisfaction and 22,2 % for s_2. In other words, $\neg A$ place can be reached by 0.636 or 0.222 possibility value, which is not accepted because of threshold 0.01 (1 – 0.99 = 0.01). So P is satisfied on this model.

5. VALIDATION

In this section, researchers present a validation for their proposal by providing some examples from real-world applications. Also presented are two sets of specifications from the Smart Living and ambulance dispatching systems. Before that, the authors introduce a developed support tool. The choice of case studies is supported by the need for a high level of flexibility in such systems. Therefore, specifications must be highly versatile.

5.1. The PSAS-Tool

The authors designed a new support tool in order to facilitate engineering work, which is similar to the PSPFramework presented in a study [6]. The main objective of the PSAS-tool is to automate the structured English grammar expressions translation to fuzzy metric temporal logic formulas. The PSAS-tool's main process is as follows: First, selection of a pattern in the left side bar menu (Fig. **3**) is made and a modal window appears by selecting a pattern (Fig. **4**). Second, text areas are filled with original requirement con-

tent. After clicking the "ADD" button, the specification is inserted into the main document and the mapping in FMTL is automatically generated. Finally, a specifications list can be exported into multiple formats (JSON, xml).

The example in Fig. (**4**) shows the "TheEarliest-Possible" pattern. First, scope (1) is selected, the PSAS-tool adds automatically a Textarea to enter the event (2). Next, textarea that depicts the original requirement is fulfilled (3). Next, it is checked optionally "holds" to express clearly requirement (4). Next, time interval definition is selected (within, at least) (5), finally, fulfilling pattern-holding duration and select time unit (6, 7).

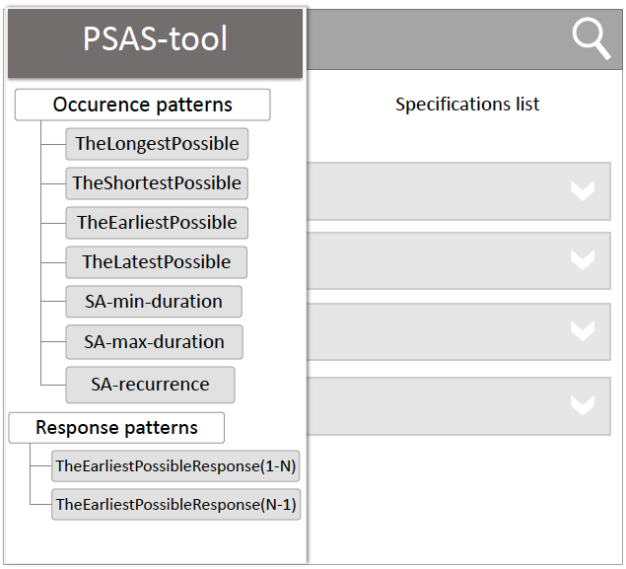

Fig. (3). PSAS-tool overview.

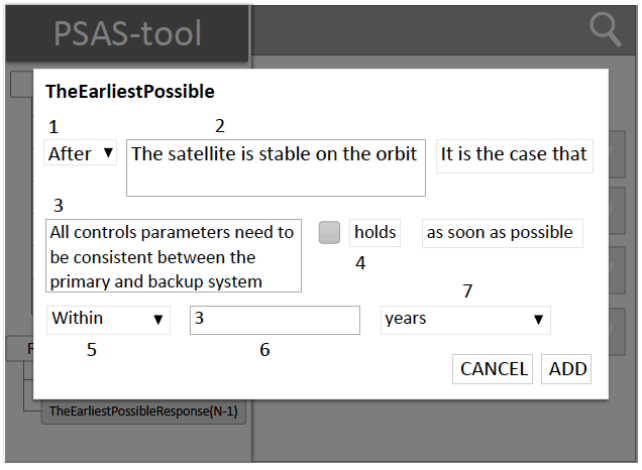

Fig. (4). Pattern selection and property creation.

The PSAS-tool conception allows requirement engineers to specify system properties in a versatile way with no need to use logic symbols (generated automatically). The PSAS-tool is a web application; this choice implies various benefits. First, web applications are centralized, therefore, they avoid deploying on each client, and updates are easier. Secondly, they enable accessibility from any place with Internet access. Third, web applications are platform independent. The PSAS-tool also offers the advantage of mobile adaptability via its responsive design, by applying PSAS using the PSAS-tool on a set of requirements taken from various applications from the real world.

5.2. Self-Adaptive Pattern Samples

All specifications must conform to the rule: "Scope, pattern." The scope is the interval of time in which a pattern can be handled. The second part depicts the core of the pattern. In the first example below, the scope is "Globally," which means the pattern is valid for the whole system run-time. The pattern here is "TheLongestPossible," following the researchers' grammar. It will be replaced by "It is the case that {A} holds as long as possible;" parts in bold remain unchangeable, while "A" will be replaced by the original requirement.

Globally, it is the case that {Terrestrial Photovoltaics (PV) systems face the sun} holds as long as possible. (It is hard to determine the sun-facing time because there is no fixed duration, even during the same day or season.).

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 three-year mission time} holds as early as possible (Source: satellite control system).

Globally, {barley hydroponics system temperature} is as close as possible to 18° (barley fodder hydroponics system).

After {Wi-Fi activation}, {smartphone detects device} holds as many as possible (Smartphone Operating System).

Due to the lack of space, only a subset of patterns was instantiated. In the following section, a concrete case study is provided to illustrate how the authors chosen language adds flexibility to requirements.

5.3. Empirical Evaluation

In order to demonstrate whether the discovered specification patterns deal with current problems of industrial specifications, two case studies are presented. The first one is the Smart Fridge from an assisted living home [4]; the second is an ADS (Ambulance Dispatch System) [13]. In the following, the authors present a set of self-adaptation specifications using our proposed catalog for self-adaptive requirements.

6. DISCUSSION

PSAS is a first-class adaptive language, all patterns are relaxed using operators like "as possible" (Fig. **5**), this means that property can be relaxed if it cannot be satisfied in a certain state of the system at runtime. On the other hand, each pattern is represented with a formula generated using fuzzy metric temporal logic grammar which can be verified automatically on a fuzzy timed Petri net model. Learning curve is really easy when it comes to PSAS, patterns approaches are proven ones (500 requirements expressed using only 20 patterns [2]), in our cases studies 14 requirements expressed using only 7 patterns. Another interesting advantage, with the use of patterns, is the improved readability of document specification because of reduced number of used patterns.

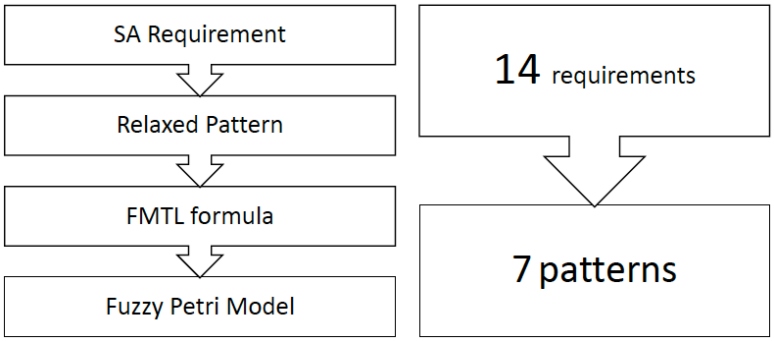

Fig. (5). Requirement processing with PSAS.

Table 8. Smart fridge specifications (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 detects and communicates information with food package} as few as possible.

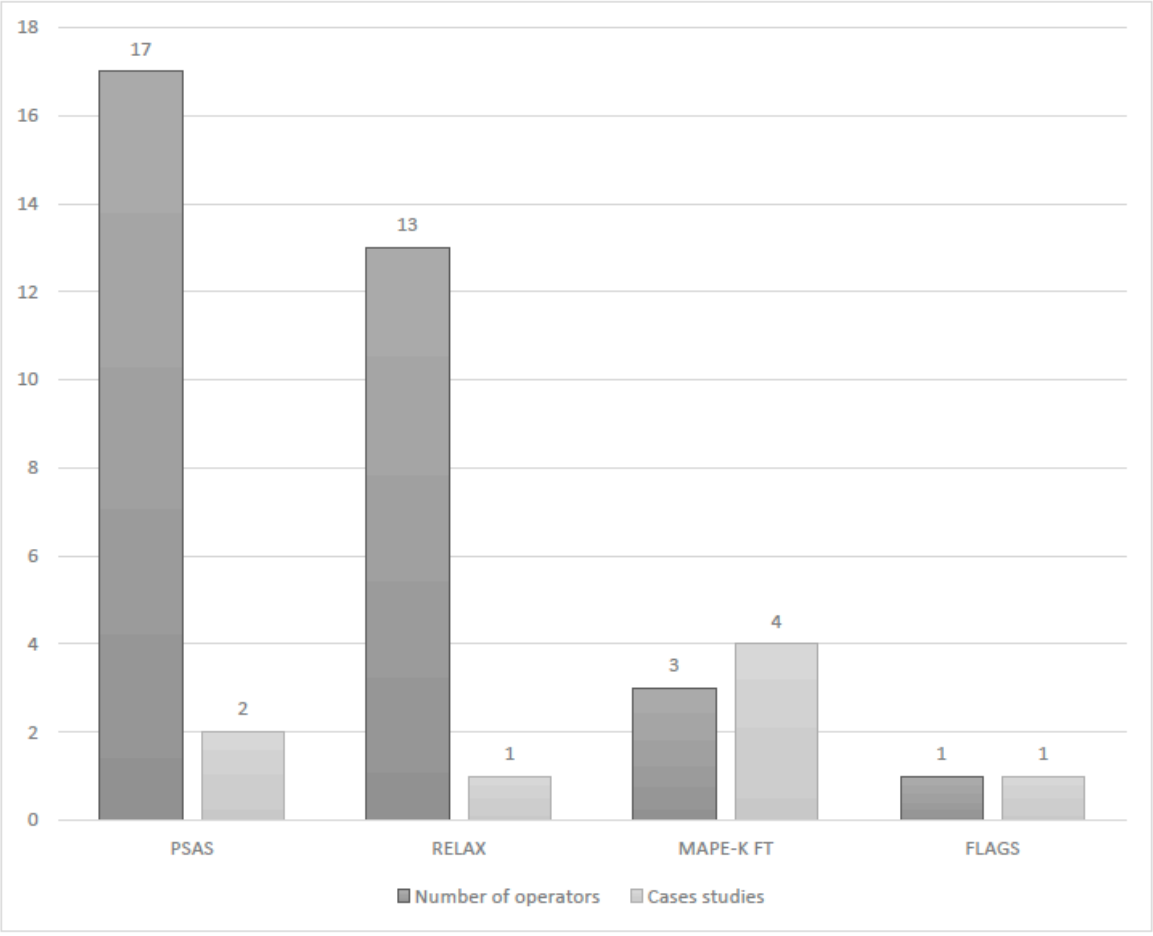

Fig. (6). Number of operators vs cases studies (PSAS and others).

Table 9. ADS specifications.

	Requirement Description	**PSAS Pattern**
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 *1 week*.
R7	Dispatch ambulance should fail at most once a week.	Globally, {Dispatch ambulance} holds repeatedly almost every *1 week*.
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 *10 minutes*.
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 *15 minutes*.
		Globally, {Ambulance arrives should succeed 80%} holds repeatedly almost every *1 day*.
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 decreases one time compared to the previous month } or May {The success rate of No unnecessary extra ambulances decreases two times compared to the previous month}, repeatedly almost every *1 month*.
R12	Update arrival at the 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 *10 minutes*.
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 *5 minutes*.
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 *1 month*.

The specifications above illustrate the ease of use of PSAS, which is one of its best advantages. The authors have found that it is easier to handle because of the well-defined structure of pattern. Table **8** presents specifications expressed using RELAX [4] and PSAS; the approaches are similar in terms of simplicity. However, PSAS has the advantage of non-recursive grammar that makes the language easier for less experienced engineers. As mentioned in section 3, each pattern is coupled with one of the five scopes so the seventeen patterns can made up to eighty five different properties. Fig. (**6**) illustrates the number of operators available in PSAS and three reference works, versus the number of case studies applied to them. The chart shows that PSAS is the richest language in terms of vocabulary (17 patterns), 13 for RELAX [4], 3 for MAPE-K-FT [18], and only the operator "as possible" for FLAGS [12]. In regard to applicability of language, MAPE-K-FT takes the throne with four cases studies, two for PSAS and one for the others. Nevertheless, PSAS is a general propose language that can be applied in any self-adaptation context.

In Table **9**, requirements must be relaxed, but this relaxation is limited at a certain threshold. For example, "Ambulance arrives in 10 minutes should succeed 60% of the time," 60% of success defines the minimum tolerated success degree, and the pattern must have a true value of one in 60% of the time. The remaining 40% has a true value of less than one. So PSAS not only handles the minimum success degree, but also the rest that can result in a true value of less than one, which can be useful in the reasoning level at run-time.

The researchers' approach is designed to be used during the design phase, so by permitting the engineers to specify self-adaptation at the requirements level, it guides designers' reasoning about self-adaptation in the specification phase. PSAS can also be useful at run-time. Speaking about the feedback loop MAPE (Monitoring Analyse Planning Execution) [25], the truth value of specification can be very helpful during run-time (monitoring and planning phases). However, the researchers' approach does not give an explicit run-time solution. Thanks to fuzzy logic, proposed patterns in the examples presented above are more flexible than the existing

ones for non-adaptive systems. On the other hand, existing languages for self-adaptive systems like RELAX provided the required flexibility, but PSAS offers an advantage by two major things. First, specifications generated are simple in PSAS due to its non-recursive grammar. Second, the researchers' language offers a way to handle specification execution interval via scopes.

CONCLUSION

This paper aimed to present a new specification language for self-adaptive systems. The main objective of this language is to deal with run-time changes including uncertainty, for indicating the behavior of a self-adaptive system. In response to the unexpected changes that occur in the execution environment, system behavior needs to be changed. The lack of sufficient information about the applications intended behavior could occasionally cause behavioral uncertainty during the development stage, so it still requires run-time adaptation. In this work, we introduced a pattern catalog to the non-invariant requirements.

This language is based on two major pillars, fuzzy logic, and specification patterns; the former has as objective to handle both temporal and ordinal uncertainties due to fuzzy logic flexibility. The latter is for handling natural language ambiguities through the Structured English language. The authors gave a set of examples to bring meaning to our proposal. The authors also presented two case studies from ambient assisted living and ambulance dispatch system domains those are active areas in self-adaptation.

CONSENT FOR PUBLICATION

Not applicable.

CONFLICT OF INTEREST

The authors declare no conflict of interest, financial or otherwise.

ACKNOWLEDGEMENTS

Declared none.

APPENDIX 1

PSAS Semantics in Terms of FMTL

In the following, the truth value of each formula is a possibility value ∈ [0,1]. The threshold of truth can be defined according to need in [0,1].

$[\![time\_A]\!]$	$[t_l^A, t_u^A]$
$[\![gap\_A]\!]$	$t_u^A - t_l^A$
[[tl_A]]	t_l^A
[[tu_A]]	t_u^A
$[\![end\_A]\!]$	$[\![time\_A]\!] + lifeTime(A)$

Self-Adaptive-Occurrence Family

The Longest Possible (Universality pattern mapping (Autili *et al.*, 2015) with truth value in [0,1])	
Globally	$\Box_{[\![time\_A]\!]}(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A)$
Before B	$\Diamond_{\geq [\![tl\_A]\!]} B \rightarrow \left((A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A) \mathrm{U}_{[\![time\_A]\!]} B \vee \Box_{[\![time\_A]\!]}(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A)\right)$
After C	$\Box\left(C \rightarrow \Box_{[\![time\_A]\!]}(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A)\right)$
Between C and B	$\Box\left(\left(C \wedge \Box_{\leq [\![tl\_A]\!]} \neg B\right) \wedge \Diamond_{\geq [\![tl\_A]\!]} B \rightarrow \left((A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A) \mathrm{U}_{[\![time\_A]\!]} B \vee \Box_{[\![time\_A]\!]}(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A)\right)\right)$
After C until B	$\Box\left(\left(C \wedge \Box_{\leq [\![t\Box\_A]\!]} \neg B\right) \rightarrow \left((A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A) \mathrm{W}_{[\![time\_A]\!]} B\right)\right)$
Where : $[\![end\_A]\!] = \infty$	
The Shortest Possible (Absence pattern mapping (Autili *et al.*, 2015) with truth value in [0,1])	
Globally	$\Diamond_{[\![time\_A]\!]}(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A)$
Before B	$\neg B \ \mathrm{W}_{[\![time\_A]\!]} \left((A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A) \wedge \neg B\right)$
After C	$\Box \neg \mathrm{C} \vee \Diamond \left(\mathrm{C} \wedge \Diamond_{[\![time\_A]\!]}(A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A)\right)$
Between C and B	$\Box\left(\left(C \wedge \Box_{\leq [\![tl\_A]\!]} \neg B\right) \wedge \Diamond_{\geq [\![tl\_A]\!]} B \rightarrow \left(\neg B \ \mathrm{W}_{[\![time\_A]\!]} \left((A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A) \wedge \neg B\right)\right)\right)$
After C until B	$\Box\left(\left(C \wedge \Box_{\leq [\![tl\_A]\!]} \neg B\right) \rightarrow \left(\neg B \mathrm{U}_{[\![time\_A]\!]} \left((A \wedge \Diamond_{\leq [\![end\_A]\!]} \neg A) \wedge \neg B\right)\right)\right)$
Where: $[\![end\_A]\!] = 0$	

The complete mapping is available on: https://zerone-01.com/psas/mapping.pdf

REFERENCES

[1] L. A. Zadeh, "Fuzzy sets," *Information and control,* vol. 8, no. 3, 1965.

[2] M. B. Dwyer, G. S. Avrunin and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *Proceedings of the 21st international conference on Software engineering*, Los Angeles, 1999.

[3] N. Abid, S. Dal Zilio and D. Le Botlan, "A Real-Time Specification Patterns Language," 2011.

[4] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng and J.-M. Bruel, "RELAX: a language to address uncertainty in self-adaptive systems requirement," *Requirements Engineering,* pp. 177-196, 2010.

[5] S. Konrad and B. H. Cheng, "Real-time specification patterns," in *Proceedings of the 27th international conference on Software engineering*, St. Louis, MO, USA, 2005.

[6] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione and A. Tang, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar," *IEEE Transactions on Software Engineering,* vol. 41, no. 7, pp. 620-638, 2015.

[7] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing,* vol. 17, pp. 184-206, 2015.

[8] D. Weyns, "Software engineering of self-adaptive systems: an organised tour and future challenges," *Chapter in Handbook of Software Engineering,* 2017.

[9] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu and D. Weyns, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, Berlin, Heidelberg, 2013.

[10] S.-i. Moon, K. H. Lee and D. Lee, "Fuzzy branching temporal logic," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 34, no. 2, pp. 1045-1055, 2004.

[11] B. H. Cheng, P. Sawyer, N. Bencomo and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *International Conference on Model Driven Engineering Languages and Systems*, Berlin, Heidelberg, 2009.

[12] L. Baresi, L. Pasquale and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *2010 18th IEEE International Requirements Engineering Conference*, Sydney, NSW, Australia, 2010.

[13] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson and J. Mylopoulos, "Awareness requirements for adaptive systems," in *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems*, Waikiki, Honolulu, HI, USA, 2011.

[14] V. E. S. Souza, A. Lapouchnian and J. Mylopoulos, "(Requirement) evolution requirements for adaptive systems," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Zurich, Switzerland, 2012.

[15] M. Ahmad, N. Belloir and J.-M. Bruel, "Modeling and Verification of Functional and Non-Functional Requirements of Ambient Self-Adaptive Systems," *Journal of Systems and Software,* vol. 107, pp. 50-70, 2015.

[16] E. Vassev and M. Hinchey, "Engineering Requirements for Autonomy Features," in *Software Engineering for Collective Autonomic Systems*, Cham, 2015.

[17] M. Morandini, L. Penserini, A. Perini and A. Marchetto, "Engineering requirements for adaptive systems," *Requirements Engineering,* vol. 22, no. 1, pp. 77-103, 2017.

[18] D. G. D. L. Iglesia and D. Weyns, "MAPE-K formal templates to rigorously design behaviors for self-adaptive systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS),* vol. 10, no. 3, p. 15, 2015.

[19] C. E. da Silva, J. D. S. da Silva, C. Paterson and R. Calinescu, "Self-adaptive role-based access control for business processes," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Buenos Aires, Argentina, 2017.

[20] M. Salehie and L. Tahvildari, "Autonomic computing: emerging trends and open problems," in *DEAS '05 Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, St. Louis, Missouri, USA, 2005.

[21] L. Grunske, "Specification patterns for probabilistic quality properties," in *2008 ACM/IEEE 30th International Conference on Software Engineering*, Leipzig, Germany, 2008.

[22] Y. Zhou and T. Murata, "Petri net model with fuzzy timing and fuzzy-metric temporal logic," *International Journal of Intelligent Systems,* vol. 14, no. 8, pp. 719-745, 1999.

[23] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II*, Berlin, Heidelberg, 2013.

[24] V. Gruhn and R. Laue, "Patterns for timed property specifications," *Electronic Notes in Theoretical Computer Science,* vol. 153, no. 2, pp. 117-133, 2006.

[25] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer,* vol. 36, no. 1, pp. 41-50, 2003.

[26] M. U. Iftikhar and D. Weyns, "A case study on formal verification of self-adaptive behaviors in a decentralized system," *arXiv preprint arXiv:1208.4635,* 2012.

[27] D. Weyns, M. U. Iftikhar, S. Malek and J. Andersson, "Claims and supporting evidence for self-adaptive systems: A literature study," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Zurich, Switzerland, 2012.

[28] F. D. Macias-Escriva, R. Haber, R. del Toro and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications,* vol. 40, no. 18, pp. 7267-7279, 2013.

[29] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems,* vol. 2, no. 4, pp. 255-299, 1990.

[30] A. Capozucca, N. Guelfi and P. Pelliccione, "The fault-tolerant insulin pump therapy," in *Rigorous Development of Complex Fault-Tolerant Systems*, Berlin, Heidelberg, 2006.

[31] J. Coleman and C. Jones, "Examples of how to determine the specifications of control systems," *Technical Report Series-University of Newcastle Upon Tyne Computing Science,* vol. 915, p. 65, 2005.

[32] E. S. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering*, Annapolis, MD, USA, USA, 1997.

[33] A. Dardenne, A. Van Lamsweerde and S. Fickas, "Goal-directed requirements acquisition," *Science of computer programming,* vol. 20, no. 1-2, pp. 3-50, 1993.

[34] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems,* vol. 8, no. 3, pp. 203-236, 2004.

[35] I. Suzuki and H. Lu, "Temporal Petri nets and their application to modeling and analysis of a handshake daisy chain arbiter," *IEEE Transactions on Computers,* vol. 38, no. 5, pp. 696-704, 1989.