Martin, T., & Anggraini, R. (2019). A Graded Approach to Requirement Satisfaction for Evolving Systems. In *2019 IEEE International Conference on Fuzzy Systems* (IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)). IEEE Computer Society. https://doi.org/10.1109/FUZZ-IEEE.2019.8858795

A Graded Approach to Requirement Satisfaction for Evolving Systems

Trevor P Martin
Intelligent Systems Lab,
University of Bristol
Bristol, United Kingdom
trevor.martin@bristol.ac.uk

Ratih N.E. Anggraini
Intelligent Systems Lab,
University of Bristol
Bristol, United Kingdom
ra16032@bristol.ac.uk

***Abstract*—There has been a strong trend towards autonomous and semi-autonomous systems in recent years. Evolving and adaptive systems embody the notion of autonomy, by changing their behavior (and possibly their structure) in response to changes in their environment. A consequence is that a designer may not be able to fully define the functional behavior of a system. Hence, formal verification and testing may not be possible. As a result, the self-adapting aspect of an evolving system is often implemented in an informal, *ad hoc*, manner and there is potential for causing significant harm if a system malfunctions in some way. A safety case requires more than an assertion that a system will work because it has not failed in testing. A more rigorous approach is essential, in which we can formally show that an evolving system meets its requirements and specifications. This paper outlines initial work in combining the *X-mu* approach (to model fuzzy uncertainty) with flexible requirements for an evolving system specified in RELAX, a formal framework to capture the uncertainty in evolving system requirements. A simple case study is used to illustrate some of the principles.**

Keywords—X-mu, Fuzzy, Graded, RELAX, Evolving Systems, Self-Adaptation, Requirements, Verification.

I. INTRODUCTION

There has been a strong trend towards autonomous and semi-autonomous systems in recent years. Such systems vary enormously in scope and complexity, and include

- vehicles (for example, cars, trains, lorries, drones),
- sub-systems of vehicles (navigation aids, cruise control, braking systems),
- robot household devices (e.g. vacuum cleaners, lawn mowers, assistive devices),
- IoT devices and groups of devices (such as a environment control in a building),
- network control and routing
- pure software systems (such as conversational agents).

Although ranging widely in application, these have aspects in common, and (in particular) can all be considered as evolving, or self-adaptive, systems, where the system is able to reconfigure and alter its behaviour in response to changes in the environment. In this context, the environment refers to the setting in which the system operates, including all entities that can interact with the system. We follow standard definitions (e.g.[1]) in distinguishing *adaptive* systems (where internal parameters are tuned to optimise some measure of performance) from *evolving* systems where there may be changes in internal structure as well as in parameters - for example, a rule-based system that generates new rules in response to previously unseen data patterns.

The possibility of adaptation means that a designer may not be able to fully define the functional behaviour of a system, with the consequence that formal verification and testing may not be possible [2]. As a result, self-adaptation is often implemented in an informal, *ad hoc*, manner and tested by running the system under a variety of inputs and operating environments [3]. Clearly in many of the examples listed above there is the potential for causing significant harm if a system malfunctions in some way, and a formal safety case requires more than an assertion that a system will work because it has not failed in testing [4]. A more rigorous approach is essential, in which we can formally show that an evolving system meets its requirements and specifications. Specifying and implementing semi-autonomous and autonomous systems is difficult precisely because the system can adapt in response to changes in its environment, possibly in ways that were not anticipated at design-time. This is particularly true when a system is interacting with humans or other autonomous agents, which can behave in inconsistent and unpredictable ways.

RELAX [5] has been proposed as a formal framework in which the uncertainty in adaptive system requirements can be captured and converted into specifications, which can then be used in formal proof or model-checking to verify that software meets its specifications. RELAX is a structured form of natural language which describes the properties and behaviour of a system. It is intended to create a system specification and consequently is unable to assess requirement satisfaction.

Verification is a way to prove or to disprove requirement satisfaction in a system. The classic way of describing requirement satisfaction is using Boolean values, i.e. a yes or no answer. Using this approach, we only know whether a requirement is satisfied or not - we are unable to know that a requirement was almost always satisfied, or that it was almost satisfied in all cases. In the work described here, we have used

This work was partly supported by the Indonesia Endowment Fund for Education (LPDP), Ministry of Finance, Indonesia.

TABLE 1 RELAX OPERATORS (TAKEN FROM [1])

RELAX operator	Description
Modal operators	
SHALL	A requirement must hold
MAY...OR	A requirement specifies one or more alternatives
Temporal operators	
EVENTUALLY	A requirement must hold eventually
UNTIL	A requirement must hold until a future position
BEFORE, AFTER	A requirement must hold before or after a particular event
IN	A requirement must hold during a particular time interval
AS EARLY, LATE AS POSSIBLE	A requirement specifies something that should hold as soon as possible or shouldbe delayed as long as possible
AS CLOSE AS POSSIBLE TO [frequency]	A requirement specifies something that happens repeatedly but the frequency may be relaxed
Ordinal operators	
AS CLOSE ASPOSSIBLE TO [quantity]	A requirement specifies a countable quantity but the exact count may be relaxed
AS MANY, FEWAS POSSIBLE	A requirement specifies a countable quantity but the exact count may be relaxed
Uncertainty factors	
ENV	Defines a set ofproperties that define the system's environment
MON	Defines a set of properties that can be monitored by the system
REL	Defines the relationship between the **ENV** and **MON** properties
DEP	Identifies the dependencies between the (relaxed and invariant) requirements

UPPAAL [1] to model and simulate a specified system. UPPAAL is intended for use with crisp specifications and input values. It cannot handle fuzzy uncertainty in parameter values (this is also true for similar verification tools) and hence we cannot work with the results of our RELAXed system model to determine the degree of requirement relaxation needed to produce a satisfactory system. In order to model the uncertainty, we use a graded approach to the satisfaction of requirements, based on the X-μ representation outlined in [6]. This aims to model the flexible definitions used in human language by allowing partial satisfaction of a predicate (in the same way as a fuzzy set); however, it differs from standard fuzzy set theory in focussing on the crisp sets of values that satisfy a predicate at different memberships, and in treating membership purely as an ordering (for example, the set *{1,2,3}* satisfies the predicate "small dice values" better than the set *{1,2,3,4}*). The precise membership values are unimportant, as long as the ordering is respected.

In cases where sets are guaranteed to be nested with increasing membership, the X-μ representation is essentially the same as an α-cut approach; however, as shown in [6], the X-μ representation is more powerful in that it can represent a single value that varies with membership - for example, the cardinality of a fuzzy set or the mid point of a fuzzy interval.

[1] Uppaal - an integrated tool environment for modeling, simulation and verification of real-time systems http://www.uppaal.org/

The focus on crisp sets or values at different memberships simplifies re-use of existing software packages based on crisp input values and crisp processing - we can derive an output membership function by evaluating the software a number of times on crisp inputs determined by the X-μ analysis.

This paper outlines initial work in combining the X-μ approach with flexible requirements for an evolving system specified in RELAX (see also [7]). We briefly cover background material and show how the fuzzy requirements can be efficiently integrated with a crisp verification package. A simple application, adapted from [8, 9]) is included for illustration.

II. SYSTEM COMPONENTS

A. RELAX

RELAX is a requirements engineering language for adaptive systems, able to capture the uncertainty in evolving system requirements to enable a subsequent verification process. A full description of the RELAX language is given in [5]. The requirement statements indicate the behaviour of a system using modal verbs such as *SHALL* or *WILL* and standard operators from temporal logic such as *BEFORE, AFTER*, *UNTIL* and *EVENTUALLY*; dependencies between requirements can be declared, as well as properties of the environment and measurements that can be taken to determine such properties (see Table 1, reproduced from [5]). In cases where not all requirements can be satisfied, the designer may

indicate that some requirements can be relaxed - typically it is possible to identify critical and non-critical requirements, where the former must always be satisfied but the latter may be relaxed under certain conditions. For example, [8] refers to this division as *functional* (describing essential services offered by a system) and *non-functional* (typically imposing quality constraints on the services). Consider a robot delivery service which should never collide with an obstacle or a moving entity, and should reach its destination within a specified time. In some cases, a small delay in delivery time would be acceptable but a collision (even a small collision) would never be acceptable. Frequently, the complete set of un-relaxed requirements may be unsatisfiable because of interactions between requirements - for example, the need to conserve battery power in a delivery robot is in direct opposition to the need to move more quickly in order to reduce delivery time. If the interaction between requirements is known (by means of the *DEP* operator in RELAX) and both requirements can be relaxed, we can determine the optimum degree of relaxation which enables the requirements to be satisfied without over-relaxing either of them.

A by-product of our approach is to enable designers to explore relaxation trade- offs - for example,

given a level of relaxation, what guarantees can be provided on achievable performance?

and

what is the minimum relaxation needed to achieve a given level of performance?

The RELAX system proposes the use of a standard fuzzy approach to represent the uncertainty but does not consider subsequent verification steps against the fuzzy requirements.

B. The X-μ Representation of Fuzzy Quantities

In order to use a verification system (or any crisp software package) with fuzzy quantities, we must modify the internal representation and processing steps in the software. This level of modification represents a considerable effort and may be impossible in some cases. Here we propose an alternative representation of the fuzzy uncertainty. Zadeh's original formulation of fuzzy sets [10] was inspired at least partly by the goal of modelling the flexible definitions used in natural language so that mathematical system descriptions could be made more understandable. This is based on the observation that humans are adept at using loose definitions which admit elements to a greater or lesser degree, rather than an absolute yes/no test. A standard example is the set of tall people in a specified population; some are definitely tall, others definitely not tall, and others are tall to some intermediate degree. Such gradation is clear in the case of categories related to an underlying numerical attribute, such as height. It is also true for more complex concepts such as "socially responsible corporation", "high value, long-term customer", etc. In these cases, we can rank the membership of different objects in the set representing the concept extension - in other words, the extension can be modeled as a fuzzy set. The interval [0, 1] is a convenient range for the membership function. It has the advantage of mapping naturally to a scale with definite membership (1) and non-membership (0), with intermediate values making it easy to say that one element has a higher membership than another.

The *X*-μ approach switches attention from the membership of individual elements to the elements that satisfy the predicate at a specified level (usually, but not exclusively, we are concerned with sets of elements). This separates the imprecision in a value (typically modeled as an interval or set of possible values) from the fuzziness, which is modeled by membership. For example (see Fig 1), a fuzzy set of integers has a cardinality which varies with membership. There is no imprecision - the cardinality is a single value at any specified membership.

There is an underlying assumption that membership scales are *commensurable*, i.e. a statement that object *O* has property *P* to degree α has a consistent meaning regardless of the set. For example, we should be able to say that person *A* belongs more strongly to the set of *tall* people than *B* belongs to the set of *rich* people, or that *A* belongs equally strongly to both sets, etc. We do not investigate ways of eliciting such judgments, but note there is considerable expertise in finding implicit beliefs using stated preference surveys and related techniques.

We use the operator *X* to denote the transformation to inverse membership function.

Given a membership function

$$\mu(x) = \alpha$$

we find the function (or relation) satisfying

$$X_{\mu}(\alpha) = x$$

In many cases, an analytic membership function leads to an analytic inverse; in the case of no analytic solution, a sampled version can be obtained to a required degree of accuracy. For example, the fuzzy set *small* shown in Fig 1 would be represented as:

$$X^{S}_{smallDiscrete}(\alpha) = \begin{cases} \{1, 2, 3, 4\} & 0 < \alpha \le 0.25 \\ \{1, 2, 3\} & 0.25 < \alpha \le 0.5 \\ etc. & \end{cases}$$

For discrete sets, this representation leads to a natural sequence of relaxation levels.

III. Use of X-μ Quantities to Fuzzify Crisp Software

When generalising a crisp approach (i.e. software, a mathematical method, etc.) to the fuzzy case, the normal strategy is to identify quantities (inputs) that are subject to fuzzy uncertainty and then extend the underlying method to cope with a fuzzy representation of these quantities. This is problematic when dealing with software implementations, particularly those provided as libraries without access to source code. Even in the case where source code is available, it can be a considerable task to re-write a system to handle the

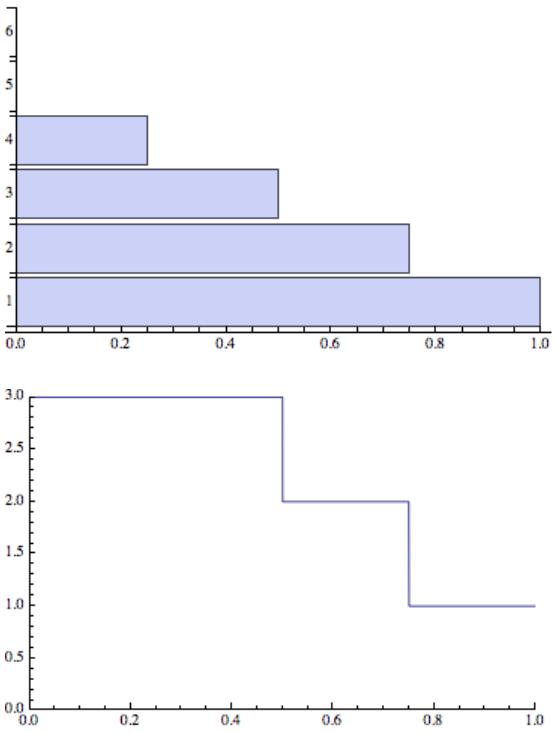

Fig. 1. *(top) X-μ* representation of the (inverse) membership function for *small* defined on the universe {1,2,3,4,5,6}. Membership is on the *x*-axis, and this is a set-valued function. For example, at membership 0.8, the set *small* is {1}, while at membership 0.4 it is {1,2,3}

(bottom) X-μ representation of a single-valued fuzzy quantity, representing the number of elements less than or equal to 3 in the fuzzy set *small*. At membership 0.8, there is one element, while at membership 0.4 there are three. This is a pure fuzzy quantity as opposed to the fuzzy intervals which are often used to model numerical fuzzy values

more general representation of values required for the fuzzy case.

Adopting the *X-μ* approach enables us to avoid these problems, at the expense of multiple software executions, in order to obtain outputs at different membership levels. In most cases, this is not a significant problem - model checkers such as UPPAAL are designed to reduce state-space explosion arising from the combination of possible input (observable) values. However, it is clear that any help in reducing the dimensionality is useful, and we utilise two approaches in this regard. The information contained in the *DEP* statements of the *RELAX* framework represents the dependencies between requirements and the influence of the monitored (observable) values on each other and on the requirements. This can be used to reduce the number of *RELAX*-ed values that must be considered (see next section). Secondly, in cases where there is no interaction (dependency) between requirement constraints, it is possible to decompose a calculation into constituents which can be treated as separate calculations; the results of those calculations can then be combined to give the overall result. The ideas are related to calculations involving fuzzy quantities [6, 11, 12] and are illustrated in the example outlined in the next section.

Of course, it is always possible to rewrite the *RELAX*-ed specifications as more complicated crisp specifications, in the same way as it is possible to reformulate any fuzzy system as a more complex crisp system which explicitly includes parameters representing the uncertainty; the advantage of using the *RELAX*-able terms in the original requirements is their closeness to natural language which enables easier expression of the system constraints and (arguably) a more understandable set of requirements.

IV. Case Study for Flexible Requirements

We adapt the smart vacuum system (SVS) example (described in [3] and reformulated in [9]) focusing on a small sub-problem to illustrate our approach. Consider a room that is divided into unit squares, each of which is dirty, and a set of autonomous robot vacuum cleaners tasked to clean the room. We ignore issues such as route planning and co-ordination between the cleaning robots. Whilst our simplified example is not an adaptive or evolving system, it is easy to envisage cases that would require greater flexibility, such as route planning in the presence of obstructions, dynamic changes in number/ position of obstructions, changing amounts of dirt including re-introduction of dirt into clean areas, etc.

We take a simple case in which the problem is specified as

- the room dimension is 20×25 i.e. 500 unit squares,
- each unit square requires 1 unit of battery power to clean,
- each cleaner has 100 units of power initially

Our crisp requirements are

- each cleaner should minimise its use of power and keep at least 20 units in reserve
- use three of the five available cleaners
- aim to clean 100% of the room.

In this case, all requirements are eligible for potential relaxation - we will accept lower limits on the amount of power to be held in reserve (down to 0), we will allow use of

TABLE II SAMPLE OF SVS REQUIREMENTS

S_A	SVS SHALL achieve AS MANY clean unit squares AS POSSIBLE ENV: room space MON: dirt and motion sensors REL: sensors indicate the room space that has been cleaned DEP: S_A is negatively impacted by S_B , S_C
S_B	Each unit SHALL have battery use AS FEW AS POSSIBLE ENV: remaining battery MON: amount of dirt REL: amount of dirt will determine how much battery power is needed DEP: S_B negatively impacts S_A
S_C	Number of units SHALL be AS FEW AS POSSIBLE DEP: S_C negatively impacts S_A

TABLE III RELAXED REQUIREMENTS

ID	RELAX Requirement	Original threshold	RELAXed change
R1	cleanAreaUnits AS MANY AS POSSIBLE	500	±100
R2	numberOfCleaners AS CLOSE AS POSSIBLE TO	3	±2
R3	batteryUsed AS FEW AS POSSIBLE	up to 80	+20

more cleaners if necessary (up to 5), and we will accept a reduced threshold on the proportion of clean space (down to a minimum of 80%). A subset of the RELAX requirements is shown in Table II.

The un*RELAX*ed problem requires us to consider

$$N_C = 3$$
$$N_S = 500$$
$$B_i = \{20, 21, \ldots, 100\} \quad i \in \{1, \ldots, N_C\}$$

We built a simulation in UPPAAL and verified it against the requirements, confirming non-satisfaction. In this simple example, we can use algebraic reasoning to show that the requirements are not satisfiable - since each square requires one unit of power, the maximum number of squares cleaned with 3 cleaners and 80 units of power is 3 × 80 = 240. We must therefore relax our initial requirements to some degree.

Relaxation proceeds by considering the range of values for the number of cleaners, number of unit squares cleaned and remaining battery power for each cleaner. The relaxed ranges are shown in Fig 2. Due to the dependency declarations in Table III, we can see that only the upper limit on the number of cleaners should be considered and only the upper limit on battery use is relevant (since S_A requires us to maximise the area cleaned, and this is negatively impacted by minimising battery use and by number of cleaners). Hence we need to consider the following input values

$\frac{2}{3} < \alpha \le 1$	$\frac{1}{3} < \alpha \le \frac{2}{3}$	$0 < \alpha \le \frac{1}{3}$
$N_C \in \{3\}$	$N_C \in \{3,4\}$	$N_C \in \{3,4,5\}$
$N_S \in \{467, \ldots, 500\}$	$N_S \in \{433, \ldots, 500\}$	$N_S \in \{401, \ldots, 500\}$
$B_i \in \{14, \ldots, 20\}$	$B_i \in \{7, \ldots, 20\}$	$B_i \in \{1, \ldots, 20\}$
$i \in \{1, \ldots, N_C\}$		

Furthermore, since we do not have any interactions between the input values, we can assume nested behaviour and re-use results at lower memberships. That is to say, if $R(\alpha)$ is the (boolean) value of requirement satisfaction at membership α, then (for example)

$$R\left(\tfrac{1}{3} < \alpha \le 1\right) = R\left(\tfrac{1}{3} < \alpha \le \tfrac{2}{3}\right) \vee R\left(\tfrac{2}{3} < \alpha \le 1\right)$$

Here, the verification requires three separate executions at membership levels α = 1, 2/3 and 1/3, with the inputs shown in Table VI. Note that in this case, we do not have to repeat calculations already considered at higher membership. In a system where the variables interact, this kind of decomposition is not possible and the full range of input values must be used at each selected α. This also reflects calculations with fuzzy quantities [13] where functions with non-interactive inputs can be calculated by consideration of end points, but functions with interactive variables require a more comprehensive sampling of the entire input space

The verification results are shown in Table IV. In this case, considerable relaxation (to α=1/3) is required before the requirements can be satisfied. The result could be used to refine the system design or to evaluate it in comparison to another design. For completeness in this example, we have also modelled the system algebraically and calculated requirement satisfaction using the X-μ representation of membership functions as shown in Fig 2. In general, algebraic analysis is not possible .

In the more general case, we would only consider a subset of requirements for relaxation, since the core functional requirements (e.g. avoid collisions between cleaners) can not be relaxed. Automatic identification of requirements eligible for relaxation is discussed in [3] and references therein.

TABLE IV RELAXED CASES AND VERIFICATION RESULTS FROM UPAAL

Relaxation level ( )	Battery Used	Number of Cleaners	Verification result
$\frac{2}{3} < \alpha \le 1$	[80,86]	3	Property NOT SATISFIED
$\frac{1}{3} < \alpha \le \frac{2}{3}$	[87,93]	4	Property NOT SATISFIED
$0 < \alpha \le \frac{1}{3}$	[94,99]	5	Property IS SATISFIED

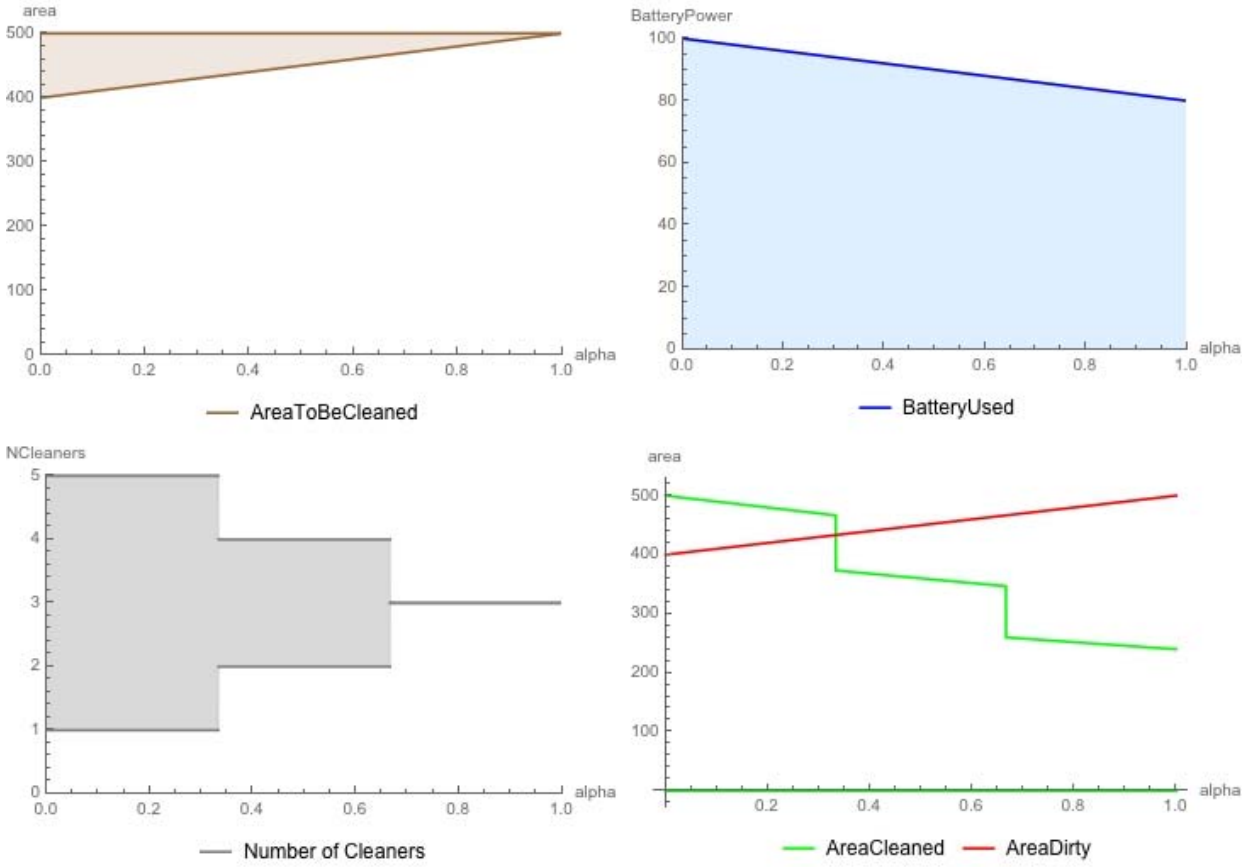

Fig. 2. Relaxed Quantities and Exact Calculation of SVS Membership Functions

V. SUMMARY

There is a clear need to verify that evolving systems are safe and will perform according to specification. The nature of evolving systems (their ability to cope with situations that may not be full understood or anticipated at design-time) is in direct conflict with this need, as verification requires a full specification of behavior. Existing (*ad hoc*) approaches generally address the verification issue by ignoring it. In this paper, we have proposed the use of RELAX as a tool for incorporating fuzziness into flexible software requirements, and the *X*-μ interpretation of fuzziness as a practical tool for processing the flexibility in requirements through the subsequent verification process. Further research is focused on exploring the use of dependency (DEP) information in decomposition of the search space, and in expanding the scope to larger examples.

REFERENCES

[1] P. Angelov, D. P. Filev, and N. Kasabov, *Evolving Intelligent Systems: Methodology and Applications*: Wiley-IEEE Press, 2010.

[2] E. T. McGee and J. D. McGregor, "Using dynamic adaptive systems in safety-critical domains," presented at the Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Austin, Texas, 2016.

[3] J. Hansel, T. Vogel, and H. Giese, "A Testing Scheme for Self-Adaptive Software Systems with Architectural Runtime Models," presented at the Proceedings of the 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems, 2015.

[4] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM,* vol. 55, pp. 69-77, 2012.

[5] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel, "RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems," presented at the Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, RE, 2009.

[6] T. P. Martin, "The X-mu representation of fuzzy sets," *Soft Computing,* vol. 19, pp. 1497 - 1509, 2014/05/31 2015.

[7] R. N. E. Anggraini and T. P. Martin, "Fuzzy Representation for Flexible Requirement Satisfaction," in *Advances in Computational Intelligence Systems*, UKCI, 2017, pp. 28-36.

[8] E. M. Fredericks, B. DeVries, and B. H. C. Cheng, "AutoRELAX: automatically RELAXing a goal model to address uncertainty," *Empirical Software Engineering,* vol. 19, pp. 1466-1501, October 01 2014.

[9] R. N. E. Anggraini and T. P. Martin, "Capturing Requirement Correlation in Adaptive Systems,," in *International Conference on Robotics, Biomimetics, Intelligent Computational Systems*, Bandung, 2018, pp. 28-36.

[10] L. A. Zadeh, "Fuzzy Sets," *Information and Control,* vol. 8, pp. 338-353, 1965.

[11] D. Sanchez, M. Delgado, M. A. Vila, and J. Chamorro-Martinez, "On a non-nested level-based representation of fuzziness," *Fuzzy Sets and Systems,* vol. 192, pp. 159-175, 2012.

[12] D. Dubois and H. Prade, "Gradual elements in a fuzzy set," *Soft Computing,* vol. 12, pp. 165 - 175, 2008.

[13] W. Dong and H. Shah, "Vertex method for computing functions on fuzzy variables," *Fuzzy Sets and Systems,* vol. 24, pp. 65-78, 1987.