FLEXIBLE REQUIREMENT SATISFACTION IN ADAPTIVE SYSTEMS

Ratih Nur Esti Anggraini

Department of Engineering Mathematics

University of Bristol

A dissertation submitted to the University of Bristol in accordance with the requirements for award of the degree of Doctor of Philosophy in the Department of Engineering Mathematics, Faculty of Engineering

September 2020

Word count 28250

ABSTRACT

An adaptive system modifies its behaviour in response to changes in its environment or in the system itself. For the system to be able to perform adaptation, the system engineer must know the conditions that the system may experience. However, it is nearly impossible to enumerate all possible conditions and adaptation. Thus, the system requirements may need to be relaxed in order to accommodate adaptive behaviour.

To enable performance toleration, the RELAX approach classifies requirements into two categories; invariant requirements, which have to be fully satisfied no matter what, and relax requirements, which can be partially satisfied to maintain flexibility. To facilitate the adaptability of the system, fuzzy branching temporal logic (FBTL) is used as the semantics of the requirements language. The approach represents the relax requirements satisfaction using fuzzy sets. As a result, instead of saying whether the relax requirement is satisfied or fails, the approach says the requirement is satisfied to a certain degree.

Moreover, RELAX provides a DEP uncertainty factor to capture the relationship between requirements. DEP indicates whether relaxing a requirement will impact the satisfaction of another requirement positively, negatively or both. However, the previous RELAX approach does not take this relationship into account while doing the relaxation.

In this approach, we use RELAX requirement language and take into account the relationship between requirement while performing the relaxation. Thus, we can achieve optimal relaxation, the minimum level of relaxation where all requirements are satisfied, and present relaxation area/s where the requirements are always satisfied to a certain degree. To verify the relaxation result, we employ a UPPAAL model checker. The verification is conducted by translating the fuzzy requirements into crisp requirements using fuzzy alpha cut so the UPPAAL can verify if the requirement is satisfied or fail at a certain degree of relaxation.

ACKNOWLEDGEMENT

First and foremost, all praises to Allah Almighty for His shower of blessings throughout my research works so that I can complete my doctoral study successfully.

I would like to express my deepest gratitude to my supervisor Professor Trevor Martin who has guided and supported me consistently during the entire period of my doctoral study. Without his constant help and motivation, this dissertation will never be finished. Apart from that, Trevor is also the kindest and warmest supervisor I could ever imagine.

I also would like to thank Indonesia Endowment Fund for Education (LPDP), Ministry of Finance of the Republic of Indonesia, for funding my doctoral study. It is my honor to be part of the LPDP awardees.

I am incredibly grateful to my lovely husband, Ario Muhammad, that has accompanied me during the up and down of this journey. He always supports and motivates me along the way. His love and encouragement are the main reason I can finish this study. My beloved kids, Muhammad Deliang al Farabi, Daisy Ramadhani Muhammad and Muhammad Dirandra al Jazari, are the best supporters of my life who always light up my day. I am so proud of you, my little angels.

Finally, I will always be indebted to my mother, Sri Widayati, who always prays for me and supports me. Also, thanks to my parent in-laws, Ali Muhammad and Sarbanun Sabaha, for the love they have given us. Thank you to my brother, Bagus Setya Rintyarna, and his family. Many thanks to all my friends that I cannot mention one by one. I appreciate all the support and love you gave to me.

AUTHOR’S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: DATE:

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Background

The growing trend of intelligent and autonomous systems (e.g., an autonomous car (Milanés, Llorca, Vinagre, González, & Sotelo, 2010) and assistive robots (Tsiakas et al., 2017)))) has triggered a growing need for highly adaptive systems. The adaptive system is a system that can modify its behaviour in response to changes in its environment and in the system itself (Cheng, de Lemos, et al., 2009). Thus, adaptations to requirements, the model itself, or its context may be needed to handle these possible changes.

In the context of the autonomous car system, the system engineer has to design the speed of the car to adapt to different road types (Zhao, Ichise, Mita, & Sasaki, 2014). To enable such adaptability, the engineer needs to know the variety of road types the car may use, e.g., motorway, local road, or private road. However, different countries may have different road types with different standards, not to mention other internal and external factors affecting the design of the car's speed adaptations. Thus, it is nearly impossible to enumerate all possible environmental conditions for which adaptations may be needed, as there will always be unexpected conditions. Consequently, there will always be uncertainty in adaptive systems (J. Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2009).

Such uncertainty can come from either internal or external sources or both (Esfahani & Malek, 2013b). Internal uncertainties (e.g., the unknown impact of system adaptations) are rooted in the system itself. By contrast, external uncertainties are the result of environmental

changes, e.g., changes in the weather or human behaviour. Even though having more information can minimize uncertainty, it is impractical and nearly impossible to enumerate everything, both internal and external, regarding the system.

Thus, a mechanism to handle the uncertainty in adaptive systems is undoubtedly essential. One approach is to develop a requirement language. Several specification languages for adaptive systems have been proposed, such as KAOS (Dardenne, Van Lamsweerde, & Fickas, 1993), A-LTL (Zhang & Cheng, 2006), or Awreqs (Souza, Lapouchnian, Robinson, & Mylopoulos, 2011). However, these languages are unable to facilitate adaptivity or uncertainty in the adaptive system simultaneously.

On the other hand, the RELAX specification language provides uncertainty factors to capture uncertainty in the adaptive system and represent adaptivity in its temporal/ordinal operators (Jon Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2010). It also introduces two types of requirements into the adaptive system, an invariant requirement (which has to be fully satisfied in all conditions) and a relax requirement (which can be partially satisfied). RELAX uses fuzzy branching temporal logic (FBTL) to capture temporal information flexibly. Hence, in this thesis, we will use RELAX to represent the requirements in the adaptive system.

One of the uncertainty factors in the RELAX specification language is DEP. DEP uncertainty factor addresses the dependency between requirements in the adaptive system. However, RELAX approach does not consider the impact of relaxing one requirement to another related requirement. Thus, we will examine the impact of relaxing one requirement to another in order to find the optimal relaxation in the adaptive system with related requirements.

Verification is essential in software and system development to assure the quality of the product (O'Keefe & O'Leary, 1993). System verification is also fundamental as it can prevent financial losses due to system failure. Therefore, a considerable amount of research on this

subject has been conducted (Beyer, 2015; Carter, He, Whitaker, Rakamaric, & Emmi, 2016; Tihana, Runeson, & Darko, 2016; Yamaguchi, Kaga, Donzé, & Seshia, 2016).

Generally, a standard method called testing has been used to conduct verification. However, the testing approach is relatively expensive because it has to investigate all possible cases (Larsen & Legay, 2016). This problem gets more acute when the system has to adapt to environmental change, under which circumstances the number of test cases gets bigger. Hence, the adaptive system needs another mechanism to perform verification.

Using formal verification can thus be valuable as a more straightforward solution (Lime & Roux, 2009; Punnoose, Armstrong, Wong, & Jackson, 2014). Model-checking, as one of the formal verification methods, verifies the system before the real system is even implemented. Early verification can avoid system failure and, accordingly, can prevent financial loss. Therefore, model checking is one of the recommended methods for performing verification in the adaptive system (Cámara & De Lemos, 2012).

Model checking, such as UPPAAL, performs verification by modelling crisp requirements and verifying properties against defined constraints. For this research, verification was conducted using computational tree logic (CTL), and the result has been produced in the form of crisp Boolean satisfaction success or failure. The problem with such a Boolean result is that it is inflexible and not best suited for the adaptive system because the dynamic adaptation and uncertainty in such a system may cause more system failure. Consequently, we have to translate verification results into a graded representation to be able to characterize the flexible satisfaction of adaptive systems (T. P. Martin & Anggraini, 2019).

The fuzzy set, however, provides a graded representation of information, meaning information belongs to a certain degree of a specified set. For example, when we talk about two men with measurable heights, we will be able to decide which one of those two men is

closer to the concept ‘tall’ than the other. By merely assessing their heights, we can decide which one will fall into which fuzzy set membership.

The concept of a fuzzy set has been applied in several different sorts of tasks, from decision-making (Xu, 2007) to knowledge representation (Shen, 2006), from management (Wong & Lai, 2011) to product categorization (Viswanathan & Childers, 1999). These fuzzy set applications have shown that the flexible nature of the fuzzy set is useful to represent information when degrees of uncertainty is inevitable. This insight has inspired us to apply the fuzzy set in our research.

In this thesis, we formalize the fuzzy requirement satisfaction by incorporating RELAX specification language and fuzzy set theory. We also propose a systematic mathematical approach to yield optimal relaxation in an adaptive system, in both linear and nonlinear cases. The UPPAAL model checker is employed to verify the relaxation result. In addition, we introduce linear regression and correlation to measure the relationship strength between two requirements in adaptive systems.

1.2. Objectives and Contributions

The main goal of this research is to propose a flexible representation of requirement satisfaction in an adaptive system with a requirement relationship. We can break down this goal into several objectives. The first is providing a mechanism to generate a graded requirement satisfaction on an adaptive system. Secondly, we propose a method to obtain optimal relaxation for related requirements. Thirdly, we provide a recommendation for the requirement threshold where the requirement is always satisfied to a certain degree. Fourthly, we recommend a method to assess requirement strength. Finally, we provide a procedure to evaluate the relaxation result.

To achieve those objectives, we propose several approaches as follows:

1. We recommend fuzzy requirement satisfaction as a more flexible way to represent requirement satisfaction in adaptive systems. This method incorporates the RELAX specification language (Jon Whittle et al., 2010) and the fuzzy set theory (Lofti A. Zadeh, 1965). The graded alpha cut representation is employed to illustrate partial requirement satisfaction and to indicate which requirement set is better than others.
2. We propose a systematic mathematical approach to relax requirement satisfaction. In this approach, we relax the requirement by considering the relationship between requirements. Thus, we will not relax it arbitrarily but will be able to obtain a minimum level of relaxation that satisfies all requirements instead. The result is optimal relaxation, which is the supremum of fuzzy requirement satisfaction for all related requirements and dependencies. In the case of an adaptive system with a linear mathematical relationship, optimal relaxation can be used to define the threshold for which a requirement is satisfied to a certain degree. However, for nonlinear cases, area/s of relaxation must be defined by obtaining the points of intersection between a requirement and nonlinear dependency.
3. We employ linear regression (Montgomery, Peck, & Vining, 2012) and correlation (Zou, Tuncali, & Silverman, 2003) to measure the strength of the relationship between related requirements. The assessment has been conducted using the Cohen effect size (Cohen, 1992).
4. We use the UPPAAL model checker to model and simulate the system (Behrmann, David, & Larsen, 2010). Reachability properties are used to verify the system model and evaluate the relaxation result of the requirement satisfaction.

In addition, during this study, we have produced several publications. The list of publications from this research is as follows:

1. **Anggraini, R. N. E.**, & Martin, T. P. (2017). Fuzzy Representation for Flexible Requirement Satisfaction. Paper presented at the UK Workshop on Computational Intelligence.
2. **Anggraini, R. N. E.**, & Martin, T. P. (2018, 8-10 Aug. 2018). Capturing Requirement Correlation in Adaptive Systems. Paper presented at the 2018 IEEE International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (Robionetics).
3. Martin, T. P., & **Anggraini, R. N. E**. (2019, June). A Graded Approach to Requirement Satisfaction for Evolving Systems. In 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (pp. 1-6). IEEE.

1.3. Research Methodology

The purpose of this research is to prove that we can use a systematic mathematical approach to find a set of flexible satisfiable requirement during the design process. In the previous work, RELAX has employed FBTL to represent a more flexible requirement satisfaction. Though it has introduced the DEP uncertainty factor to address the relationship between requirements, RELAX does not consider the impact of relaxing a requirement to the satisfaction related requirements. Thus, in this research, we combine RELAX requirement with a systematic approach to find the optimal relaxation for the related requirement in the adaptive system.

1.3.1 Research Approach

In this research, we introduce a systematic mathematical approach in relaxing requirements in the adaptive system. Our approach combines RELAX requirement language with fuzzy set theory to provide a flexible representation of requirement satisfaction.

Subsequently, we search for optimal relaxation and satisfiable area/s of relaxation by considering the impact of the relationship between requirements.

To validate our approach, we apply it to several case studies with linear and nonlinear dependency. The case studies have been used in the related studies in an adaptive system. Thus, the description will be more suitable to validate our systematic approach.

Eventually, the verification of the relaxation result is conducted by modelling the system in the UPPAAL model checker and verifying it using the TCTL query language, particularly by employing reachability property. To perform verification, we translate relaxation requirements into crisp requirement using fuzzy alpha cuts because UPPAAL only verifies in the form of Boolean result.

1.3.2 Case Studies Collection

Several case studies from the previous works in the adaptive system are used in this thesis to validate our approach. Subsequently, we modify and add some description to those case studies to make it more relevant to our research. We use it to represent the linear and nonlinear adaptive system. Following are the case studies and references we use in this work:

a. Ambient Assisted Living (AAL) System (Jon Whittle et al., 2010)
b. Smart Vacuum System (SVS) (Fredericks, DeVries, & Cheng, 2014)
c. Distributed Database System (DDS) (Özsu & Valduriez, 1999)
d. Stopping Distance (Kordani, Rahmani, Nasiri, & Boroomandrad, 2018)
e. Company BEP and Profit (Bradley, 2013)

1.4. Thesis Structures

In this section, we present a brief overview of each chapter in this thesis.

- **Chapter 2 Literature Review** gives relevant background information related to our research. First, we describe the fuzzy set theory as the more flexible way of representing

information compared to crisp Boolean representation. Then, we introduce adaptive systems and their challenges. We also discuss requirement engineering, especially in the context of an adaptive system. We present RELAX as a specification language for the adaptive system. Moreover, we talk about model checking as one of the system verification methods, and in specific, we explore the UPPAAL as the model checker used in this thesis.

- **Chapter 3 Fuzzy Requirement Satisfaction for Adaptive Systems** introduces our approach in incorporating RELAX requirement language and the fuzzy set to represent requirement satisfaction. The alpha cut is used to represent graded requirement satisfaction in the adaptive system. An ambient assisted living system is used as a case study.
- **Chapter 4 Systematic Mathematical Approach in Relaxing Related Requirements in Adaptive Systems** presents our approach to relaxing requirements when a relationship is present. In this chapter, we introduce optimal relaxation, which is the fuzzy value with optimal requirement satisfaction for related requirements. Subsequently, optimal relaxation is used to define new requirement thresholds in linear cases. Two case studies are presented to demonstrate the approach.
- **Chapter 5 Nonlinear Fuzzy Requirement Satisfaction on Adaptive System** introduces additional steps in relaxing requirement satisfaction in nonlinear cases. A few more steps are added to the systematic approach because relaxing related requirements with nonlinear dependency require a larger number of considerations. After all, at some points, a requirement may suffer another failure. Thus, we need to consider all points of intersection to restrict the relaxation area/s where requirement satisfaction is guaranteed.
- **Chapter 6 Using Correlation to Capture Relationship Strength** proposes correlation and linear regression to capture the relationship strength between two requirements in adaptive systems. Moreover, it compares the correlation value with the DEP uncertainty

factor in the RELAX language. Hypothetically, this method can be utilized to find an unknown relationship between the two requirements.

- **Chapter 7 Conclusions and Future Works** concludes our research by summarising essential findings and recommending several lines of inquiry for further work.

Chapter 2

Literature Review

2.1. Introduction

The trend towards intelligent and autonomous systems has led to the need to create increasingly adaptive software. Such types of systems differ widely, and include, for instance:

- Ambient Assisted Living includes smart homes (Botia, Villa, & Palma, 2012), mobile and wearable sensors (Chen, Gonzalez, Vasilakos, Cao, & Leung, 2011), and assistive robots (Smarr, Fausset, & Rogers, 2011).
- The autonomous vehicle, such as Google driverless car (Dethe, Shevatkar, & Bijwe, 2016)
- Autonomous Agricultural Operation, such as greenhouse climate control (Pawlowski et al., 2009) and remote sensing for agriculture (Seelan, Laguette, Casady, & Seielstad, 2003)
- Robotics, ranging from outdoor robots (De Hoog, Cameron, & Visser, 2010) to indoor robots (Chiu, Yeh, & Lin, 2009)

An adaptive system is defined as a system that can alter its behaviour in response to the changes in its environment or in the system itself or even the system's goals (de Lemos et al., 2013). The environment, in this context, refers to the setting in which the system operates; it includes all entities that can interact with the system, such as location or time (T. P. Martin & Anggraini, 2019). As the environment or the system changes, the uncertainty in the adaptive system becomes unavoidable. Therefore, a specification language facilitating both adaptivity and uncertainty is needed (Fredericks et al., 2014).

On the other hand, assuring the quality of software and guaranteeing requirement satisfaction through verification and validation (V and V) activities is essential. However, the uncertainty and dynamic nature of the adaptive system has caused the V and V activities to grow into a challenging task (Tamura et al., 2013). Consequently, a method to verify an adaptive system is indispensable. Thus, in this research, we fill this need by proposing an approach to verify an adaptive system.

This chapter provides the background for our thesis. First, we discuss the fuzzy set theory as a flexible way of representing information. Next, we introduce the concept and challenges in the adaptive system. We then give an overview of requirement engineering in general, current literature on the requirements for an adaptive system, and the RELAX specification language. Finally, we explore system verification and some model checking tools, specifically the UPPAAL model checker.

2.2. Fuzzy Set Theory

Crispness has been widely used in traditional modelling, reasoning, and computing tools. ‘Crisp’ means ‘dichotomous’, the solution to a ‘yes-no’ question. In set theory, crispness denotes whether or not an element belongs to a set. As a result, we have to provide precise data, free from ambiguity and uncertainty, to determine whether an element is a member of a set or not (Zimmermann, 2011).

However, real-world classes rarely have precisely defined membership criteria. For example, the class of vegetables includes spinach, carrot, and broccoli, and obviously excludes dogs, rocks, and water. Even though some entities such as cucumbers, peppers, or tomatoes have ambiguity concerning the class ‘vegetable’. Yet, such imprecise facts play an essential role in human thinking, particularly in the domain of pattern recognition, communication, and abstraction (Lofti A. Zadeh, 1965)

On the other hand, the fuzzy set theory provides a graded concept, in which everything belongs to a class to a certain degree (R. N. Anggraini & Martin, 2017). For example, we have a set of coffee drinks. Instead of addressing the sweetness in a crisp form such as sweet and bitter, the fuzzy set theory describes it in a more flexible way such as sweet, fairly sweet, fairly bitter, and bitter. This flexible representation is more commonly used in human language, especially to describe real-world conditions.

The fuzzy function in the fuzzy set concept maps universe *U* into a membership value between 0 and 1, as shown in Equation 2-1 (Klir & Yuan, 1995). The representation of the membership function can be seen in a tabular (see Table 2-1) or analytical expression (see Equation 2-2).

$$f: U \rightarrow [0,1] \qquad \text{Equation 2-1}$$

For the coffee drinks example, we may measure sweetness based on how many spoonsful of sugar are added. Let us assume that three spoonsful added is considered as sweet and no sugar added is considered bitter. Then, we can represent coffee sweetness in terms of a fuzzy membership, as shown in Table 2-1, using fuzzy function *f(x)*. Figure 2-1 and Figure 2-2 describe the fuzzy representation of coffee's sweetness based on the number of sugars added to the cup in continuous and discrete versions, respectively.

Table 2-1. Representation of coffee sweetness with membership function f(x)

Spoon of sugar *x*	Fuzzy value *f(x)*
3	1
2	0.67
1	0.33
0	0

U: {0,1,2,3}

f:U → [0,1]

$f(x) = \frac{x}{3}$

Equation 2-2

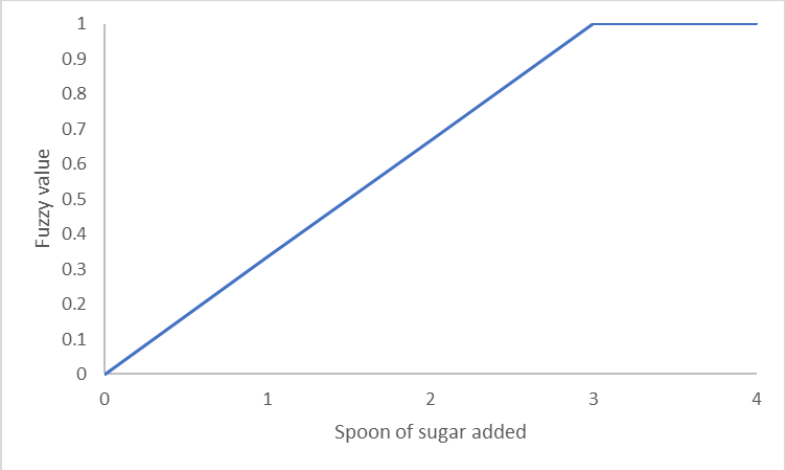


Figure 2-1. Fuzzy membership of coffee sweetness in the continuous representation

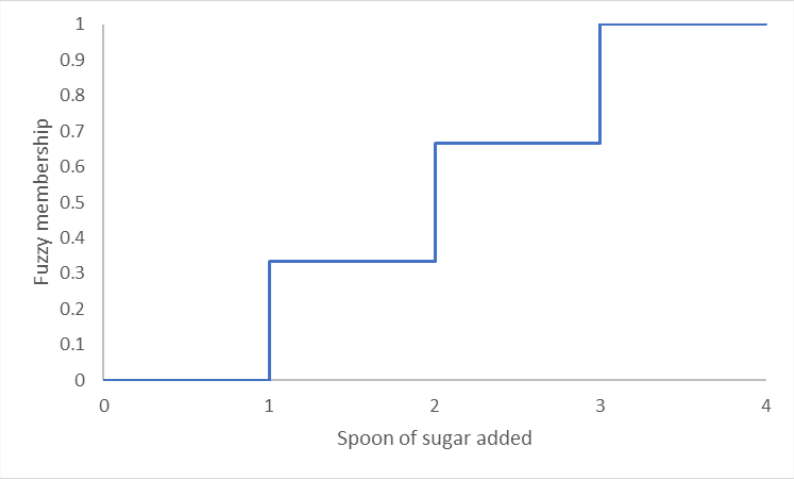


Figure 2-2. Fuzzy membership of coffee sweetness in discrete representation

As a note, we have to make a clear difference between the fuzzy and probabilistic concepts. Fuzzy deals with the degree of truth, meaning it deals with whether a fact is true, partially true, or false. By contrast, probabilistic theory relates to the degree of belief in that truth (Hájek, Godo, & Esteva, 2013).

The flexibility of the fuzzy set theory has inspired many researchers to use it in representing uncertainty (De Capua & Romeo, 2006), classification (Sassi, Yahia, & Mellouli, 2017), or scaling results (Muela, Kreinovich, & Servin, 2017). Regarding the system verification, fuzzy set theory was employed in modelling vaguely defined requirements (Morse, Araiza-Illan, Eder, Lawry, & Richards, 2017), and it has also been used to verify many different cases (e.g., forecasting (Amodei & Stein, 2009; Ebert, 2008) and service value networks (Razo-Zapata, De Leenheer, Gordijn, & Akkermans, 2012)).

2.3. Adaptive Systems

The definition of an adaptive system is debatable. Some interpret the adaptive system as the standard feedback loop in classical control theory (De Jong, 1980). Others define it as a system that modifies its parameters under specific criteria to achieve its optimal performance (Narendra & Annaswamy, 2012).

However, Zadeh explained it in a more formal definition (Lotfi A Zadeh, 1963). Let P is the performance of system A and W is its acceptable performance. Let $\gamma \in \Gamma$ indexes a family of time functions $\{T\gamma\}$ for system A so that the resultant performance is denoted Pγ. The adaptive behaviour of A is defined as follows:

Definition 2.1 The system A is adaptive with respect to $\{T\gamma\}$ and W if it performs acceptably well with every source in the family $\{T\gamma\}$, $\gamma \in \Gamma$, that is, $P\gamma \in W$. More compactly, A is adaptive with respect to Γ and W if it maps Γ into W.

Based on the above definitions, we can conclude that the adaptive system is one that tolerates performance to a certain degree because of changes in its parameters. As the system changes, the complexity of uncertainty rises. Thus, a set of an acceptable level of performance needs to be defined.

A term similar to the adaptive system is the Evolving Intelligent System (EIS). The EIS is a system that changes gradually through incremental methods or algorithms to guarantee that goals are achieved (Leite, Costa, & Gomide, 2010). Based on this definition, we can assume that the key difference between an evolving system and an adaptive system is the difference between gradual and incremental change. EISs are based on fuzzy and neuro-fuzzy set theories that permit the system functionality and structure to evolve in response to incoming data. However, our research will be focused on the adaptive system rather than the evolving system.

System adaptation, which is the system's self-modification in order to continue satisfying requirements, has become an essential topic in software engineering (de Lemos et al., 2013). However, the adaptive system is subject to uncertainty both from internal and external sources (Esfahani & Malek, 2013a). This uncertainty is one of the many challenges involved in satisfying system objectives. System adaptation may even encounter other challenges beyond these (Cheng, de Lemos, et al., 2009), such as:

- Modeling dimensions - Determining the appropriate models that support the adaptation process is challenging (Brun et al., 2013). Moreover, the model should be able to guarantee an adaptation mechanism and monitor the changes in the adaptive system (Villegas, Tamura, Müller, Duchien, & Casallas, 2013).
- Requirements – The challenges come from establishing the language to capture uncertainty in the adaptive system (Cheng, Sawyer, Bencomo, & Whittle, 2009), to manage requirement change (Sawyer, Bencomo, Whittle, Letier, & Finkelstein, 2010), and to facilitate the assurance process (Weyns et al., 2017).
- Engineering – Managing feedback control loops during the adaptive system life cycle is a challenge (Weyns et al., 2013)
- Assurances – Doing verification and validation on the adaptive system with many uncertainties poses challenges (Schmerl et al., 2017)

2.4. Requirement Engineering

Software and system requirements can be classified into functional and non-functional requirements. A functional requirement is defined as a function that a system or a system component must be able to perform. On the other hand, a non-functional requirement is not what the software can do but how it will do it ("Systems and software engineering -- Vocabulary," 2010). Or we can say that the functional requirement is the service that will be provided by the system, and the non-functional requirement is the quality constraint or criterion necessary to delivering the functional requirement (such as performance, safety, latency) (Fredericks et al., 2014).

Requirements are typically collected through a set of activities called requirement engineering (RE). These activities consist of requirement elicitation, specification, verification and validation, and management (Sommerville, 2011).). This process, traditionally, is applied at the beginning of a software development phase. As the system becomes more complex because the requirements have to be continuously changed to satisfy stakeholders' needs, RE can occur iteratively throughout the development life cycle (Pandey, Suman, & Ramani, 2010). However, this iterative process raises the risk, especially regarding the verification process and financial cost (Ramesh, Cao, & Baskerville, 2010).

One of the requirement engineering activities is requirement specification, in which requirements are documented using a specification language (Van Rooijen et al., 2017). Specifying requirements can be performed in two ways, either using natural language or formal language (Böschen, Bogusch, Fraga, & Rudat, 2016). From the stakeholder's point of view, natural language is more easily understood. Yet, it creates ambiguities that can raise problems in the design process. On the other hand, formal specification language, though it can eliminate ambiguities, can also cause resistance from some stakeholders, discouraging them from taking part in further discussion as they do not understand the syntax (Shah & Jinwala, 2015). Thus,

it is crucial to create a requirement specification language that can be used and understood effectively by both the engineer and other stakeholders.

2.4.1 Requirements of Adaptive System

One of the challenges inherent in the adaptive system is that we cannot predict all possible adaptations during runtime; thus, we cannot prepare all possible sets of requirements for all possible conditions. As a result of these uncertainties, the adaptive system is expected to have 'incomplete' requirements (Cheng, de Lemos, et al., 2009). Consequently, the requirement specifications of the adaptive system should be able to address this incompleteness.

The requirement language, such as KAOS, i* (Yu, 1997), or AwReqs (Souza et al., 2011), have been unable specifically to facilitate adaptivity and uncertainty. A-LTL, a logic developed for adaptation orientation, is able to capture adaptation and create new specifications (Zhang & Cheng, 2006). However, this new adaptation logic is still unable to deal with uncertainty in the adaptive system. Therefore, a requirement language that is able to capture uncertainty and facilitate adaptivity is necessary.

Another challenge in the adaptive system is assuring the fulfilment of requirements, even when the system changes. Cheng et al. introduced M@RT, Models at Runtime, to address adaptive system assurance (Cheng et al., 2014). It used MAPE-K feedback loop – Monitor (M), Analyser (A), Planner (P), and Executor (E) operate over Knowledgebase (K) – to control the adaptation manager. Another research project proposed statistical model checking to verify properties at runtime for the adaptive system (Iftikhar & Weyns, 2016).

In the next section, we will explore in detail a specification language for adaptive systems called RELAX. It uses natural language so that it will be easily understood by different stakeholders, from the trained to the untrained. We will also delve into some verification methods in the next sub-chapter.

2.4.2 RELAX Specification Language

The adaptive system involves inherent uncertainty due to continuous changes to the system brought about by modifications in its environment and in the system itself. Consequently, satisfying the system requirements becomes more challenging, and hence, a tolerance of requirement satisfaction is essential. A requirement language called RELAX is proposed to facilitate this toleration (J. Whittle et al., 2009).

RELAX is a requirement specification written in a structured natural language. In this approach, the requirements are written using the SHALL operator. The RELAX approach introduces two types of requirements, an invariant requirement, and a relax requirement. The invariant requirement has to be fully satisfied no matter what. On the other hand, a relax requirement can tolerate partial satisfaction.

For non-invariant requirements, the SHALL operator is followed by the ordinal/temporal operator enabling it to respond to the non-invariant requirement's flexibility. RELAX grammar is shown in Equation 2-3. The following is the parameters used in RELAX operator: p is an atomic proposition, e is an event, t is a time interval, f is a frequency, and q is a quantity.

$$\begin{aligned}\phi \ ::= \ & true \mid false \mid P \mid SHALL\ \phi \mid MAY \phi_1 OR\ MAY \phi_2 \\ & \mid EVENTUALLY \phi \mid \phi_1 UNTIL \phi_2 \mid BEFORE\ e\phi \mid \\ & AFTER\ e\phi \mid IN\ t\ \phi \mid AS\ CLOSE\ AS\ POSSIBLE\ f\ \phi \\ & \mid AS\ CLOSE\ AS\ POSSIBLE\ q\ \phi \mid \\ & AS\ \{EARLY, LATE, MANY, FEW\}\ AS\ POSSIBLE\ \phi\end{aligned} \qquad \text{Equation 2-3}$$

RELAX semantics are expressed using fuzzy branching temporal logic (FBTL) (Moon, Lee, & Lee, 2004). FBTL describes requirement satisfaction in a fuzzy set of real numbers between [0, 1] instead of saying requirements as satisfied (1) or not (0). In other words, fuzzy function $f(x): U \rightarrow [0,1]$ is used to map the degree of membership of U.

RELAX operators are shown in Table 2-2, categorized into modal, temporal, and ordinal operators. The operators in the shaded area are the ones that can be relaxed and have a fuzzy requirement satisfaction, while the unshaded temporal operators have crisp Boolean requirement satisfaction.

Table 2-2. Relax operator (Jon Whittle et al., 2010)

RELAX Operator	Informal Description
Modal Operator	
SHALL ϕ	ϕ is true in any state
MAY ϕ_1 *OR MAY* ϕ_2	In any state, either ϕ_1 or $\phi 2$ is true
Temporal Operator	
EVENTUALLY ϕ	ϕ will be true in some future state
BEFORE e ϕ	ϕ is true in any state occurring prior to event *e*
AFTER e ϕ	ϕ is true in any state occurring after event *e*
IN t ϕ	ϕ is true in any state in the time interval *t*
AS EARLY AS POSSIBLE ϕ	ϕ becomes true in some state as close to the current time as possible
AS LATE AS POSSIBLE ϕ	ϕ becomes true in some state as close to time t as possible
AS CLOSE AS POSSIBLE TO frequency ϕ	ϕ is true at periodic intervals where the period is as close to *f* as possible
Ordinal Operator	
AS CLOSE AS POSSIBLE TO quantity ϕ	There is some function Δ such that $\Delta(\phi)$ is quantifiable and $\Delta(\phi)$ is as close as possible to 0
AS MANY AS POSSIBLE ϕ	The value of quantifiable property ϕ is maximized
AS FEW AS POSSIBLE ϕ	The value of quantifiable property ϕ is as close as possible to the minimal value

In addition to the temporal and ordinal operator, the uncertainty factors which are part of RELAX provide a mechanism to capture uncertainty, as shown in Table 2-3. The environment property ENV specifies the state of the environment, which is sometimes unobservable. For the observable environment, we use MON properties to capture the information in the environment. The uncertainty factor REL defines the relationship between MON and

observable ENV. Eventually, the DEP factor is used to indicate the impact of relaxing one requirement to the satisfaction of other requirements.

Table 2-3. Uncertainty factor in RELAX language

Uncertainty Factors	Description
ENV	Define the properties of the system environment
MON	Define the properties to be monitored
REL	Define the relationship between MON and ENV properties
DEP	Identify the dependencies between requirements

The following system description, taken from Whittle et al.'s publication on RELAX (Jon Whittle et al., 2010), illustrates an example of a smart office system. We will use it to describe how RELAX language is used to express the requirements in the adaptive system.

Alice's office detects her arrival every morning and initiates a data synchronization process to ensure that Alice's Blackberry, iPhone, and desktop all maintain a consistent list of business contacts. This synchronization process is repeated every 30 min as long as Alice is in the room.

Based on the system description, the standard requirement specification could be written in SHALL statements as follows:

Table 2-4. Original requirement of smart office system

S1	The synchronization process SHALL be done when Alice entered the room and every 30 minutes after that
S2	The synchronization process SHALL distribute data to all connected devices such that all devices has the same data all the time

Those two requirements in Table 2-4 can be satisfied only in an ideal condition. However, there are situations where the requirements cannot be satisfied, such as during network outage or device malfunction, causing inconsistent data between devices. To avoid rolling back, which might cause the required data to be missing, the requirement engineer could provide adaptivity by altering the requirement, as shown in Table 2-5.

Table 2-5. Alternative requirement specification of S2 in a smart office system

S2-alt	The synchronization process SHALL distribute data to all connected devices such that all devices has the same data all the time. If there is a malfunctioning device, synchronization SHALL be carried out by communication with neighbouring devices.

Using this alternative requirement, the system will be able to adapt to a malfunctioning situation to keep satisfying the requirements. Even so, using this approach, the engineer will have to enumerate all possible situations and their solutions, which is nearly impossible. In other words, the system will not be able to work appropriately without the anticipation of all possible environmental change.

Alternatively, RELAX language provides flexibility through uncertainty factors without needing to mention the alternative requirements specifically. For example, in the case of requirements of the smart office system where, for some reason, the requirement cannot be satisfied, then we can relax the requirements as shown in Table 2-1.

Table 2-6. RELAX requirement of smart office system

S1'	The synchronization process SHALL be initiated AS EARLY AS POSSIBLE AFTER Alice enters the room and AS CLOSE AS POSSIBLE TO 30 min intervals after that. **ENV**: location of Alice; synchronization interval. **MON**: motion sensors; network sensors **REL**: motion sensors provide the location of Alice; network sensors provide synchronization interval
S2'	The synchronization process SHALL distribute data to all connected devices in such a way that AS MANY devices AS POSSIBLE are using the same data at all times. EVENTUALLY, all devices SHALL use the same data. **ENV**: number of consistent devices; time taken until the consistency is reached. **MON**: network sensors; device sensors **REL**: network and device sensors provide the number of consistent devices and time

RELAX is a semantic language built to specify the requirements of the adaptive system. The modal, temporal, and ordinal operators facilitate the adaptivity of the system. Moreover, uncertainty factors can capture uncertainty in the adaptive system. However, the ground-breaking publication describing RELAX (Jon Whittle et al., 2010) lacked a practical way of calculating requirement satisfaction. It did not show how to undertake verification on relax

requirements. Thus, in this research, we introduce the calculation of fuzzy requirement satisfaction, and the verification of those relax requirements using model checking.

2.5. System Verification

Significantly more software has started to be developed in recent years, and quality assurance has thus become an essential task in software engineering. Extensive studies on Software Quality Assurance (SQA), specifically on V&V (verification and validation), have been carried out, e.g. (Bedoya, Perez, & Marin, 2016; Suresh, 2015). The V&V methods ensure that the system works on the correct problem and solves the problem correctly, and thus the system is reliable for software quality assessment purposes.

As part of SQA, verification aims to evaluate whether the system or software requirements have been satisfied ("Systems and software engineering -- Vocabulary," 2010). The most common method in verification is testing. It is typically carried out after the system has been built (Jangra, Singh, Singh, Verma, & Management, 2011). Hence, it is relatively ineffective to verify the system since, generally, we cannot provide test cases for all possible scenarios (Larsen & Legay, 2016). Moreover, testing a system can take up to 50% of the total development effort (Reuys, Kamsties, Pohl, & Reis, 2005). Thus, some researchers suggested other verification methods, one of which is formal verification.

Formal verification evaluates the correctness of the system against its formal specification behaviours using formal mathematics methods (Edmund M. Clarke & Wing, 1996). The formal methods that are widely used include graphs (Zambon, 2010), temporal logic (Wongpiromsarn, Topcu, & Lamperski, 2015), and semantics-based verification (Schäfer, Schneider, & Smolka, 2016). However, the most popular formal verification method is model checking, which uses system states to represent system behaviours and later checks if these behaviours meet defined requirements (Edmund M Clarke, Henzinger, Veith, & Bloem, 2018).

Thus, in the next part, we will explore some model checking tools, especially the UPPAAL model checker, that will be used in this research.

2.5.1 Model Checking

Model-checking is a way of verifying the correctness of the system by modeling and verifying the model and system properties (Edmund M. Clarke & Schlingloff, 2001). The system model is usually represented using a finite state transition graph such as Petri nets (Heiner, Rohr, Schwarick, & Tovchigrechko, 2016) or timed automata (Wimmer & Lammich, 2018). In contrast, the properties are formulated in temporal logic, such as linear temporal logic (Sickert & Křetínský, 2016) or computational tree logic (Jensen, Larsen, Srba, & Oestergaard, 2016).

Computational Tree Logic (CTL) is a branching-time temporal logic, in which time is expressed using a tree-like structure (Edmund M. Clarke, Emerson, & Sistla, 1986). CTL formula can be generated with the grammar shown in Equation 2-4.

$$\phi ::= true \mid false \mid P \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \leftrightarrow \phi_2 \mid AX\phi \mid EX\phi \mid AF\phi \mid EF\phi \mid AG\phi \mid EG\phi \mid A(\phi_1 \mathrm{U} \phi_2) \mid E(\phi_1 \mathrm{U} \phi_2)$$

Equation 2-4

Where *P* is an atomic proposition, quantifier *A* means "All paths" while *E* defines "at least there exists one path". Temporal operator *X* represents the next, and *F* is the future. Also, *G* exhibits globally, and *U* means until.

Many model checking tools are available nowadays; some of them are shown in Table 2-7. Some of those model checkers are used for code analysis such as BANDERA, BLAST, and CPAChecker. BANDERA is an analyzer for the java language. BANDERA translates Java code into the finite-state model (Corbett et al., 2000). BLAST is an analyzer of C program safety properties, meaning that the verification is carried out if the program will not reach the ERROR state (Beyer, Henzinger, Jhala, & Majumdar, 2007). BLAST describes C code as a control flow automata (CFA). It used a lazy abstraction that combines on the fly abstraction

and on-demand refinement (Mong, 2004). CPAChecker is similar to BLAST, yet have a more straightforward configuration (Beyer & Keremoglu, 2011). The tool can run predicates in Single Block Encoding, Large Block Encoding as well as Adjustable Block Encoding.

The mCRL2 is a formal model checking tool that uses μ-calculus language. mCRL2 is composed of sublanguages: a data language, a process language, and a property language (Cranen et al., 2013). It has been used to verify many cases from dataflow (Cranen et al., 2013) to UML specifications (Hansen, Ketema, Luttik, Mousavi, & van de Pol, 2010).

Table 2-7. List of Model Checking Tools[1]

Model Checker	**Model-checking**		
	Type	**Modeling language**	**Property language**
BANDERA	Code analysis	Java	CTL, LTL
BLAST	Code analysis	C	Monitor automata
CPAchecker	Code analysis	C	Monitor automata
EBMC	Model-checking	SMV, Verilog	SVA
ESBMC	Code analysis	C, C++	Assertions
mCRL2	Plain, Real-time	mCRL2	μ-calculus
MRMC	Real-time, Probabilistic	Plain MC	CSL, CSRL, PCTL, PRCTL
PRISM	Probabilistic	PEPA, PRISM language, Plain MC	CSL, PLTL, PCTL
ROMEO	Real-time	Time Petri Nets, stopwatch parametric Petri nets	TCTL subset
SATABS	Code analysis	C, C++	Assertions
SPIN	Plain	Promela	LTL
UPPAAL	Real-time	Timed automata, C subset	TCTL subset

PRISM is a probabilistic model checker supporting three types of probabilistic models (discrete-time Markov chains (DTMC), Markov decision processes (MDP), and continuous-

[1] https://en.wikipedia.org/wiki/List_of_model_checking_tools

time Markov chains (CTMC)) . A model is described in PRISM language, and the properties are written in PCTL and CSL, which are a probabilistic version of CTL. The new version of PRISM is PRISM 4.0, extending with verification for (priced) probabilistic timed automata (Kwiatkowska, Norman, & Parker, 2011). The tool is open-source and can be downloaded online[2].

ROMEO is a model checker for Time Petri Nets. It employs the TCTL subset as the property language (Lime, Roux, Seidner, & Traonouez, 2009). ROMEO can perform on the fly verification and have a graphical simulation for time Petri nets (Lime & Roux, 2009).

Like ROMEO, the UPPAAL model checker also uses TCTL as the property language. However, its system modelling utilizes timed automata. We will explore more about the UPPAAL model checker in the next part of this chapter.

2.5.2 UPPAAL Model Checker

UPPAAL is a tool to model, simulate, and verify the system using timed automata as a network. This tool is free non-commercial for academia only. It supports major operating systems: Windows, Linux, and Mac OS X. UPPAAL academic website[3] contains other further information, including tutorials, case studies, and related publications.

UPPAAL consists of the states, representing action and edges connecting one state to another state, forming an automata network. The edge is equipped with several expressions:

- Select; contains a list and its type that only accessible by associated edges.
- Guard; it is to enable the edge if and only if the condition is true.
- Synchronization; it is used to connect between templates.
- Update; the edge will update the variable value in the system.

[2] http://www.prismmodelchecker.org/
[3] http://www.uppaal.org/

- Weight; it is used in probabilistic branches.

The simulator in UPPAAL can be used in two different ways: It can randomly run the system on its own as a concrete simulator, or run the system manually and do a trace to investigate how a particular state is reached. Figure 2-3 and Figure 2-4 show the manual and automatic random simulator in UPPAAL.

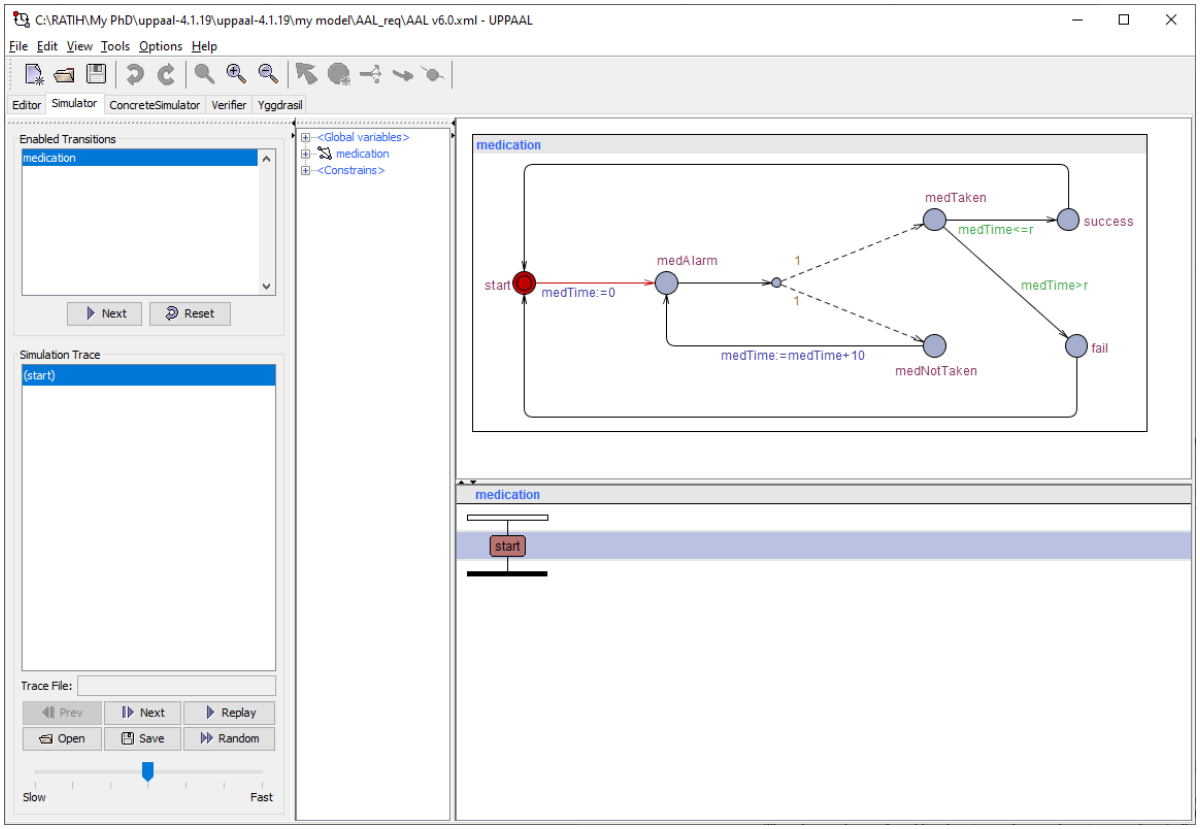


Figure 2-3. Manual simulator and simulation trace

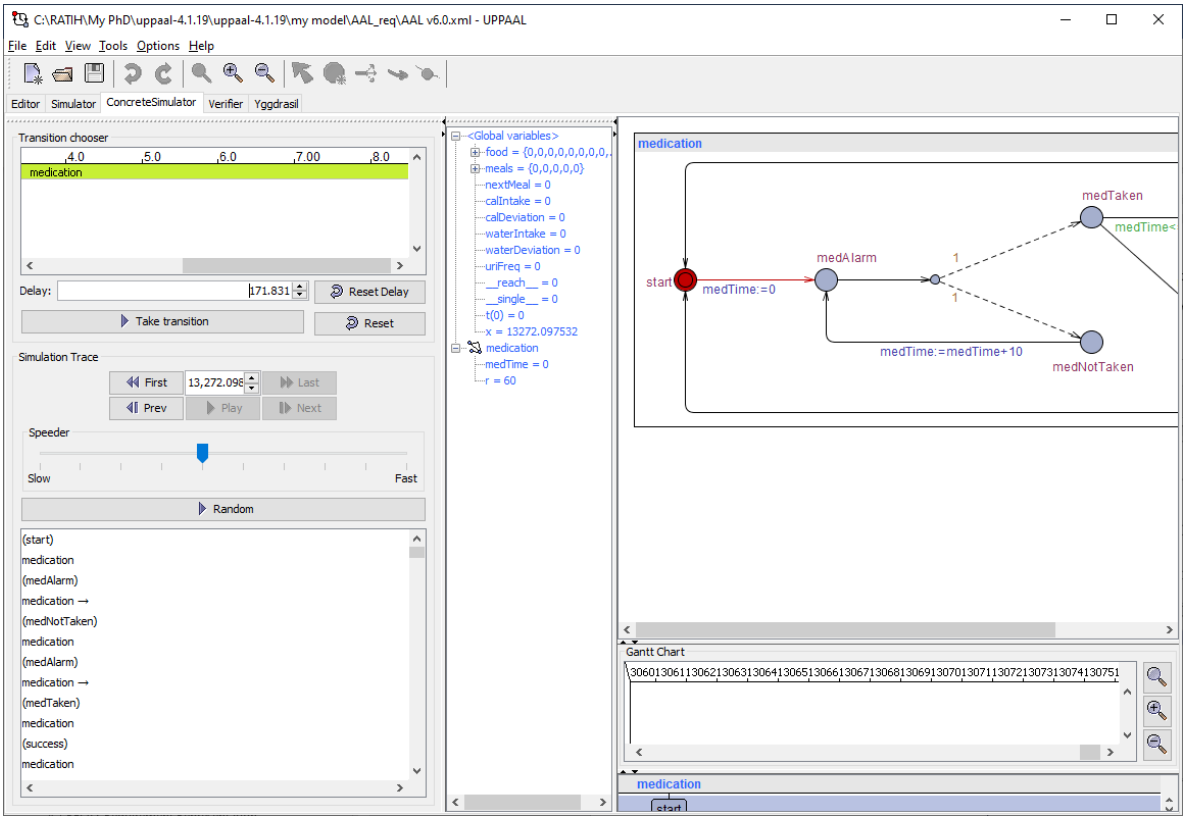


Figure 2-4. Automatic simulator with the variable values

UPPAAL uses simplified TCTL query language to verify the system (Behrmann et al., 2010). The TCTL or timed-CTL is the refinement of CTL with temporal operators to limit the scope in time (Alur, Courcoubetis, & Dill, 1990). The path formulae in the query language can check reachability, safety, and liveness properties. The path formulae TCTL quantifier as follows:

- **A** which means *'for All paths'*
- **E** which means *'along at least (there Exist) one path'*
- **[]** which means *'all states in a path'*
- <> which means *'at least one state in a path'*

Reachability states are defined if there is a path such that φ is eventually satisfied. It is verified using syntax **E<>φ**. Safety means that the system is safe because it is always in a particular condition. The syntax to verify safety property in the UPPAAL are **A[]φ** and **E[]φ**.

The liveness property implies that something will eventually happen and can be verified using **A<> φ** and **φ-->ψ**.

. In Figure 2-5, the node is the system state and an edge connects one state to another. The colored node indicates which node match the criteria in the path formulae. To give a better understanding of the query language in UPPAAL, an example is given for each path formulae in Table 2-8.

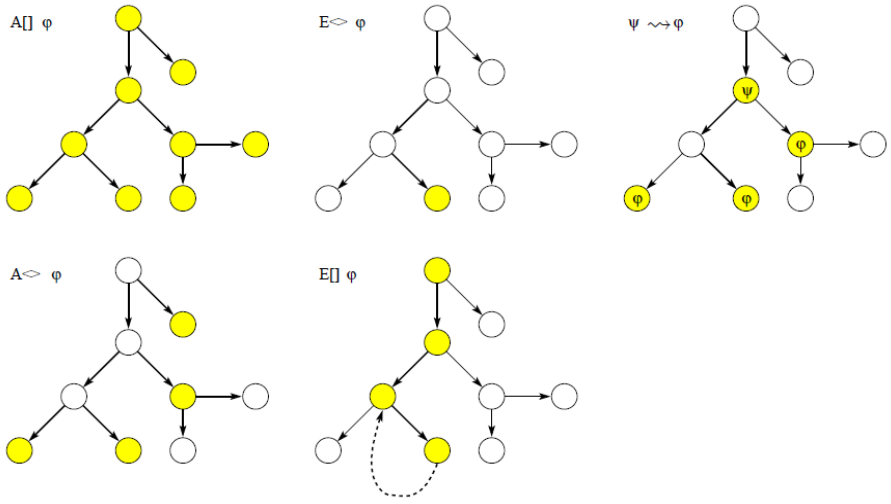


Figure 2-5. Path formulae supported by UPPAAL (Behrmann et al., 2010)

In the new version of UPPAAL, i.e., UPPAAL 4, statistical model checking is also facilitated (David, Larsen, Legay, Mikučionis, & Poulsen, 2015). A statistical model checking allows the UPPAAL to work with probability. It verifies the properties in the system with several degrees of confidence.

The UPPAAL has been used for safety verification in a timed multitask system, which is a part of the Mechatronic Standard System (MSS) (Mokadem, Berard, Gourcuff, De Smet, & Roussel, 2010). Soliman et al. built a tool to transform the function block diagram into the UPPAAL model automatically then verified the safety properties of the model (Soliman, Thramboulidis, & Frey, 2012). UPPAAL has also been used to do formal verification on a translated UML statecharts and sequence diagrams (Diethers & Huhn, 2004).

Table 2-8. Path formula explanation

Path formulae	Note
E<>φ	This formula is used to check if there is a path where the state formulae, φ, is eventually satisfied Example: E<> Success: it will be verified if at least there exist one state that matches the criteria (visiting success state)
A[]φ	This formula is used to check if the state formulae, φ, in all reachable states Example: A[] not deadlock: it will be checked if in all paths and reachable states there is no dead
E[]φ	This formula will check if there exists a maximal path such that φ is always true Example: E[] (x>3): it will be verified if there exists a path where (x>3) is always true
A<>φ	This formula will examine if φ is eventually satisfied in all paths Example: A<>(y<=10): it will be verified if, in all paths, at least one state will satisfy the criteria (y<=10)
φ-->ψ	This formula will examine if φ is satisfied then eventually ψ will be satisfied Example: Send ->Receive: it will be verified if after Send state is visited, eventually, Receive state will be visited

2.6. Summary

In this chapter, we explore the current literature on our research topic of requirement satisfaction for an adaptive system. At the beginning of this chapter, we discuss the fuzzy set theory as the flexible way of describing truth value in contrast to the more common Boolean truth value logic. Furthermore, we present the definition of the adaptive system and its research challenges.

One of the challenges in the adaptive system involves requirements. Uncertainty and adaptivity are the common characteristics of the adaptive system. Traditional requirement specification language is unable to capture uncertainty in the adaptive system or facilitate adaptivity. RELAX, as a natural language for requirement satisfaction, flexibly presents requirements and provides a mechanism to capture uncertainty without getting stuck trying to enumerate all possible situations arising from adaptivity.

Finally, we explore several model checking tools. Model-checking is one of the formal methods in system verification. The UPPAAL model checker uses timed automata to model and simulates the system. It then uses a simplified TCTL query language to verify the system.

Chapter 3

Fuzzy Requirement Satisfaction for Adaptive Systems

3.1. Introduction

In this modern world, software has become part of daily life. More and more human activities have started to involve software systems, from home appliances and vehicles to production and commerce. The trend towards ever-increasing use of software has also shifted from manual to autonomous systems that can adapt to changes in the environment (such as sensor failures, human error, or network failure) or in the system itself. This is where the self-adaptive system (SAS) is introduced.

Uncertainty is inherent in the adaptive system. Its source may be environmental, such as sensor failure or unexpected human input, or behavioural, such as when a condition triggers requirement change (Jon Whittle et al., 2010). Thus, flexible requirements can become an indispensable but simple solution to address issues arising from such uncertainty in the adaptive system (Jureta, Borgida, Ernst, & Mylopoulos, 2015). Furthermore, it can prevent failures that are common in adaptive systems. This flexibility can be achieved by requirement relaxation, a method to set a more flexible requirement threshold, and to define levels of satisfaction (R. N. Anggraini & Martin, 2017).

In this chapter, we proposed a new approach to presenting requirement satisfaction more flexibly by incorporating the RELAX specification language and fuzzy set. A graded

requirement satisfaction can illustrate how close our requirement is to being fully satisfied, instead of stating this in terms of a crisp success or failure. In addition, we demonstrate the approach using the case study of an ambient assisted living system.

3.2. Fuzzy Requirement Satisfaction

Requirement satisfaction is commonly expressed using Boolean truth values, 0 as not satisfied, and 1 as satisfied. Using such crisp values (e.g., Boolean truth values) is very inflexible. Consequently, the RELAX requirement language proposes a more effective mechanism to represent requirements as it can do so more flexibly by presenting requirement satisfaction using fuzzy values.

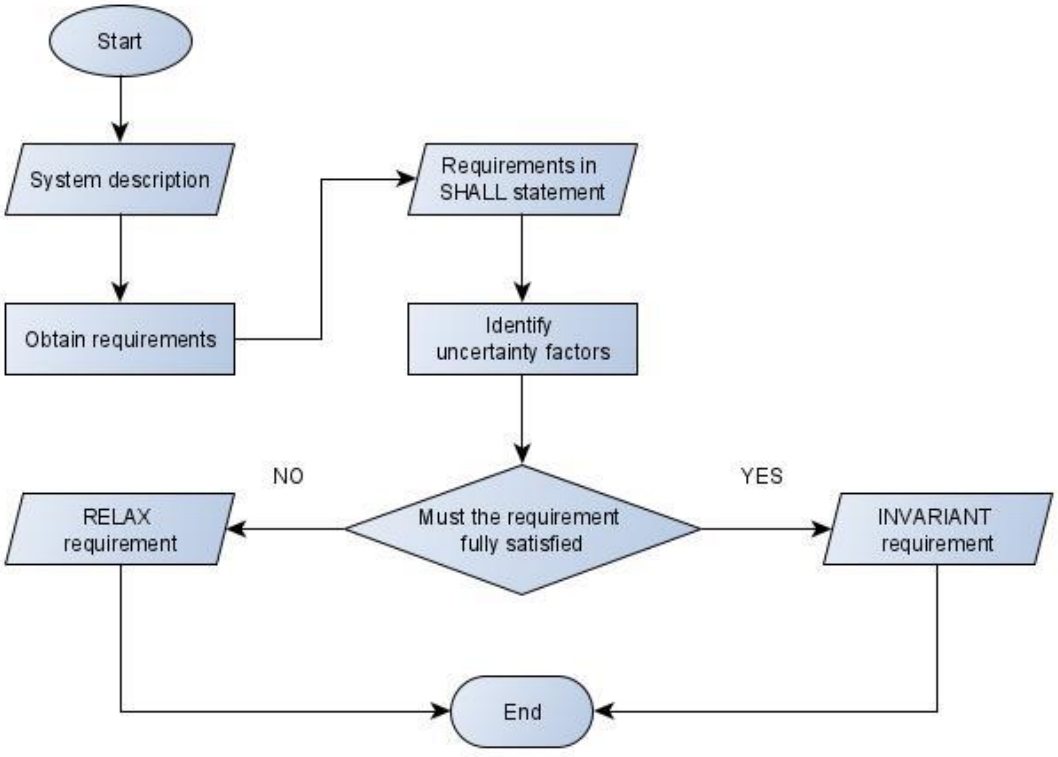
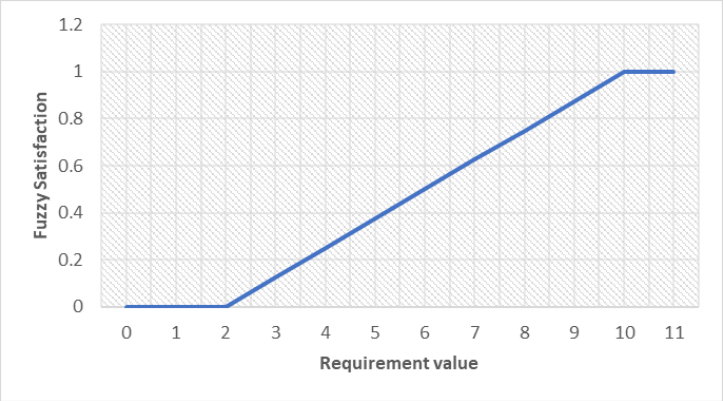
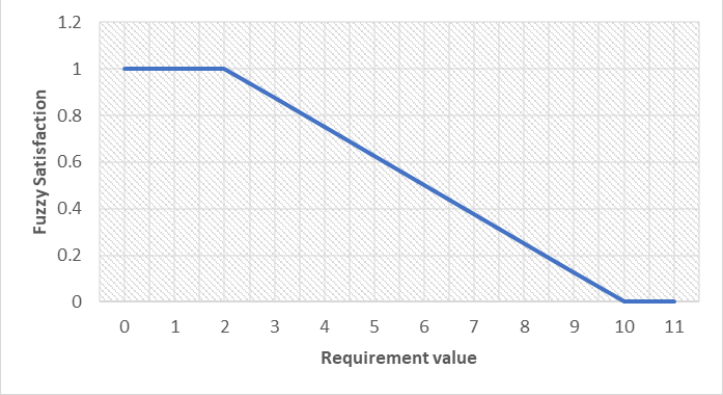
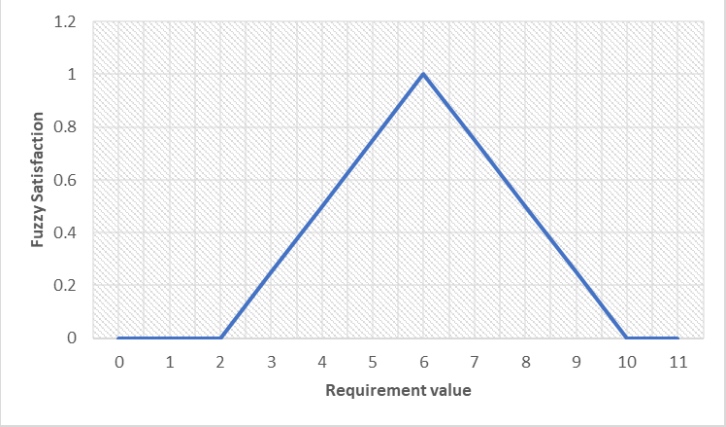


Figure 3-1. RELAX-ing process

The process of RELAX-ing requirements in the adaptive system is shown in Figure 3-1, adapted from (Fredericks et al., 2014). The system description is obtained through the requirement engineering process. Then, this description is translated into requirements by

writing it in SHALL statements. Next, the uncertainty factors need to be identified, and the requirement has to be categorized into invariant or relax requirements.

Table 3-1. RELAX operator and its fuzzy satisfaction[4]

Type	RELAX Operator	Fuzzy Illustration
RIGHT (TR)	AS MANY AS POSSIBLE, AS LATE AS POSSIBLE	
LEFT (TL)	AS FEW AS POSSIBLE, AS EARLY AS POSSIBLE	
MIDDLE (TM)	AS CLOSE AS POSSIBLE	

[4] In this illustration, we use continuous representation. However, if the requirement value can only be integer, the fuzzy set should be represented in discrete step function

The requirement that has to be satisfied all the time and must never change is classified as an INVARIANT requirement. This type of requirement should never be relaxed. Otherwise, a requirement is classified as a RELAX requirement that is open to adaptivity, and thus can temporarily be relaxed under a particular condition.

The fuzzy RELAX operators and the illustration of their requirement satisfaction are presented in Table 3-1. We categorized the fuzzy operators into three types based on the position of optimal satisfaction in the chart, i.e., RIGHT, LEFT, and MIDDLE. We create the illustration of in category in continuous representation. In this research, for the MIDDLE type of fuzzy satisfaction, we assume that the requirement is relaxed equally to the right and left.

Let *R* be the universe of requirement in an adaptive system with *n* requirements where $R = \{r_1, r_2, \dots r_n\}$ is written as a list of *SHALL* statements. Subsequently, we applied the process in Figure 3-1 to determine whether a requirement can be relaxed or is an invariant requirement. For the relax requirement, its uncertainty factors have to be defined.

The relax requirement set $R' = \{r_1', r_2', \dots r_n'\}$ will has fuzzy satisfaction. Let *F* is the set of fuzzy requirement satisfaction of *R'*. The requirement satisfaction function $f(r_i') \in F$, where $1 \leq i \leq n$, will map the requirement value into fuzzy membership value between 0 and 1, as shown in Equation 3-1.

$$f(r_i') \rightarrow [0,1] \qquad \text{Equation 3-1}$$

The original requirement threshold (θ_0) can be relaxed by the relax variable (β) to create new requirement thresholds. The lower threshold (θ_l) is the lowest value of the requirement to be considered as satisfied. The upper threshold (θ_u) is the highest value of the requirement to be considered as satisfied. The operators with a lower threshold are: AS MANY AS POSSIBLE, AS LATE AS POSSIBLE, and AS CLOSE AS POSSIBLE. Contrarily, these operators need the upper threshold: AS FEW AS POSSIBLE, AS EARLY AS POSSIBLE, and

AS CLOSE AS POSSIBLE. The lower threshold (θ_l) and the upper threshold (θ_u) can be calculated using Equation 3-2 and Equation 3-3, consecutively.

$$\theta_l = \theta_0 - \beta \quad \text{Equation 3-2}$$

$$\theta_u = \theta_0 + \beta \quad \text{Equation 3-3}$$

The fuzzy satisfaction $f(r_i')$ of the relaxed requirement r_i' can be calculated based on the type of ordinal/temporal operator in Table 3-1. Equation 3-4 shows the way to calculate the fuzzy requirement satisfaction.

$$f(r_i') = \begin{cases} 0, (type = \{TL \vee TM\} \wedge r_i' > \theta_u) OR\ (type = \{TR \vee TM\} \wedge r_i' < \theta_l) \\ 1,\ (type = TL \wedge r_i' < \theta_0) OR\ (type = TR \wedge r_i' > \theta_0) \\ \frac{(\beta+1)-|\theta_0 - r_i'|}{\beta+1},\ else \end{cases} \quad \text{Equation 3-4}$$

3.3. An Example: Ambient Assisted Living (AAL) System

3.3.1 System Description

An ambient assisted living (AAL) system is designed to monitor elderly people who live alone at home independently. This system usually focuses on improving the elderly person's quality of life while reducing the cost of healthcare (Botia et al., 2012). An AAL system combines multi-disciplinary fields ranging from smart home infrastructure to remote healthcare services (Memon, Wagner, Pedersen, Beevi, & Hansen, 2014).

The following system description is an example of the AAL system used in this research. This case study is found in Whittle et al. work, introducing RELAX specification language (Jon Whittle et al., 2010). Here we make some modifications and add other descriptions from different works (Kleinberger, Becker, Ras, Holzinger, & Müller, 2007; van den Broek, Cavallo, & Wehrmann, 2010; Zhou, Jiao, Chen, Zhang, & Cybernetics, 2011).

Table 3-2. AAL system description

Mary, a sixty-five-year-old widow, is living in her home alone and independently. She is overweight, has high blood pressure and high cholesterol levels. Thus, she needs a specific diet and regular medication. To monitor Mary's health life, an AAL system has been embedded into the smart home system. *The system has intelligent appliances. The fridge identifies the food information such as calories and food type and receives the diet plan according to Mary's condition and what she has consumed. The AAL system also integrates the medication monitor and reminder in Mary's medicine drawer.* *Another part of Mary's diet is to drink a particular amount of liquid. The liquid consumption is monitored through a sensor-enabled cup. However, she sometimes uses the cup to water the flower or just throwing it in the sink. Thus, several sensors are also installed in the flower bed and sink to monitor if the cup is used for other reasons. A toilet sensor is installed to monitor her urinary frequency as a result of her drinking activities.* *The system is also equipped with a motion sensor that will notify emergency service if there is no activity during regular waking hours. All of this is done while maintaining energy consumption to minimal use.*

Based on the system description in Table 3-2, the main requirement of the AAL system is to monitor Mary's health condition and to initiate a notification to the emergency services if a critical health condition arises. This main requirement can then be divided into several requirements structured in *SHALL* statements, as shown in Table 3-3.

Table 3-3. The AAL system requirements in SHALL statement

A1: The fridge SHALL detect and communicate with food packages
A2: The system SHALL monitor and adjust the diet plan
A3: The system SHALL monitor liquid intake
A4: The system SHALL monitor the urinary frequency
A5: The system SHALL ensure the medicine is taken on time
A6: The system SHALL raise the alarm if no activity by Mary is detected during regular waking hours

Table 3-4. The RELAX requirements of an AAL system

ID	Relax requirement $(\boldsymbol{r_i})'$	Original threshold $(\boldsymbol{\theta_0})$	Relax variable $(\boldsymbol{\beta})$
A1'	The fridge SHALL detect AS MANY AS POSSIBLE foodInfo ENV: Food locations, food item information (type, calories), food state (spoiled or unspoiled) MON: RFID readers, cameras, weight sensors REL: RFID tags provide locations/food information/food state, cameras provide food locations, weight sensors provide food information (eaten or not) DEP: A1' negatively impacts A2'	10 bags	5 bags
A2'	The system SHALL ensure Mary's calorieIntake AS CLOSE AS POSSIBLE TO the daily idealCalories ENV: Mary's daily calorie consumption MON: RFID readers and weight sensors in fridge and trash can REL: RFID readers and weight sensors provide consumed items DEP: A2' is negatively impacted by A1'	1500 cal	500 cal
A3'	The system SHALL ensure the liquidIntake AS CLOSE AS POSSIBLE TO idealIntake ENV: Mary's daily liquid intake MON: sensor-enabled cup, sink sensors, flower moisture sensors, timers REL: sensor-enabled cup, sink sensors, flower moisture sensors, timers collaboratively determine Mary's daily liquid intake DEP: A3' impacts A4' positively and negatively	1200 ml	300 ml
A4'	The system SHALL monitor Mary's urinaryFreq AS CLOSE AS POSSIBLE TO normalFreq ENV: urinary frequency MON: toilet sensor REL: toilet sensor provides information on Mary's urinary frequency DEP: A4' is impacted by A3' positively and negatively	6 times	2 times
A5'	The system SHALL ensure medTaken AS EARLY AS POSSIBLE of the schedule ENV: Mary's medication MON: medicine drawer sensor, timer, medication schedule REL: medicine drawer sensor, timer, medication schedule collaboratively determines if Mary takes her medicine on time	0 min	60 min

Applying the process in Figure 3-1, we determine which requirements are invariant and which are relax requirements. The relax requirements are presented in Table 3-4. The requirement that was not included in the RELAX table is requirement A6 - notifying the emergency services if no activity is detected during regular waking hours. This requirement is critical to the user's life, so it is categorized as an invariant requirement that always has to be fully satisfied. This is because relaxing requirement A6 could endanger Mary's life. Thus, it should never be changed or relaxed under any circumstances.

As we have seen in Table 3-4, we present the relax requirements of the AAL case study with its original threshold and relax variable. For this experiment, the value of the original threshold (θ_0) and relax variable (β) in the AAL system was determined based on some information and assumptions as follows:

a. The fridge will be stocked with 10 food bags daily. The sensor (RFID readers, cameras, and weight sensors) in the fridge will then try to obtain information about it. However, sometimes the sensors are unable to get the information because of some conditions, for example, the food is positioned improperly, or the bag is damaged.
b. Mary is a 65-year-old female, overweight (165 cm, 75 kg) with a sedentary lifestyle (little or no exercise). Based on the calorie calculator[5] , the ideal calorie intake required for her to maintain weight will be 1554 calories daily, and to get 0.5 kg weight loss, the intake should be 1054 calories per day. Thus, we decide to put 1500 calories as the original calorie threshold and relax the calorie intake by ± 500 calories.
c. The NHS suggests that the ideal fluid intake is 6-8 glasses of water (about 1200 ml)[6] daily to avoid dehydration. We relax the liquid intake by ± 300 ml.

[5] https://www.calculator.net/calorie-calculator.html
[6] https://www.nhs.uk/news/food-and-diet/six-to-eight-glasses-of-water-still-best/

d. The average urinary frequency is 6-7 times daily[7]. We relax the toilet use by ± 2 times.

e. The medicine should be consumed immediately after mealtime. An alarm will go off every 10 minutes after mealtime up to a total of 6 times until Mary takes her medicine.

3.3.2 Applying Fuzzy Requirement Satisfaction to AAL System

The RELAX requirement that is obtained above will be used to calculate fuzzy requirement satisfaction. The following example will illustrate the calculation of fuzzy satisfaction.

Example 1: Mary's fridge can detect eight food packages today

A1': The fridge SHALL detect AS MANY AS POSSIBLE foodInfo

The above description gives us information as follows:

$$A1' = 8$$

$$\theta_0 = 10$$

$$\beta = 5$$

We calculate the satisfaction using Equation 3-4.

$$f(A1') = \frac{(\beta+1)-|\theta_0 - r_i'|}{\beta+1}$$

$$f(8) = \frac{(5+1)-|10-8|}{5+1}$$

$$f(8) = {}^{4}/_{6}$$

$$f(8) = 0.667$$

Example 2: Mary randomly takes foods from the refrigerator and consumed them. Today she consumed a total of 1600 calories.

A2': The system SHALL ensure Mary's calorieIntake AS CLOSE AS POSSIBLE TO the daily idealCalories

The above description gives us information as follows:

[7] https://www.bladderandbowel.org/bladder/bladder-conditions-and-symptoms/frequency/

$$A2' = 1600$$

$$\theta_0 = 1500$$

$$\beta = 500$$

We calculate the satisfaction using Equation 3.4.

$$f(A2') = \frac{(\beta+1)-|\theta_0-r_i'|}{\beta+1}$$

$$f(1600) = \frac{(500+1)-|1500-1600|}{500+1}$$

$$f(1600) = {}^{401}/_{501}$$

$$f(1600) = 0.8$$

Example 3: Mary consumed her medicine 20 minutes late from the scheduled time.

A5: The system SHALL ensure medTaken AS EARLY AS POSSIBLE of the schedule

The above description gives us information as follows:

$$A5' = 20$$

$$\theta_0 = 0$$

$$\beta = 60$$

We calculate the satisfaction using Equation 3.4.

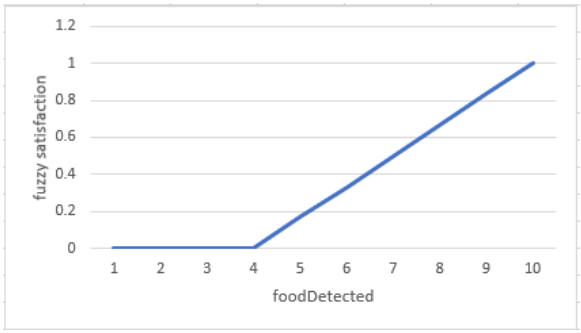
$$f(A5') = \frac{(\beta+1)-|\theta_0-r_i'|}{\beta+1}$$

$$f(5) = \frac{(60+1)-|0-20|}{60+1}$$

$$f(5) = {}^{41}/_{61}$$

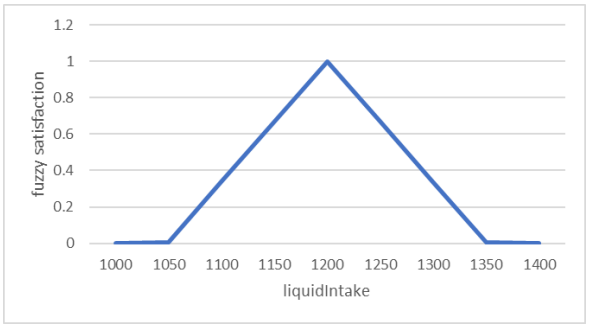
$$f(5) = 0.672$$

The translation of requirement satisfaction in fuzzy satisfaction of each requirement in the AAL case study daily is illustrated with continuous representation in Figure 3-2.

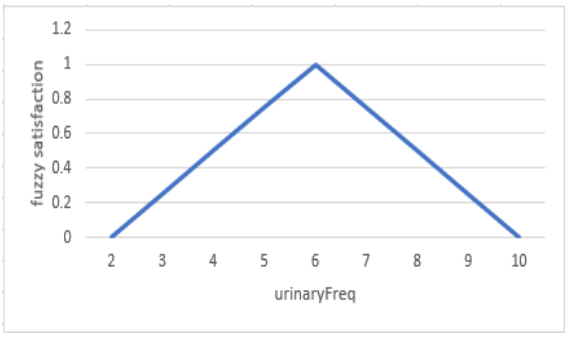


(a)

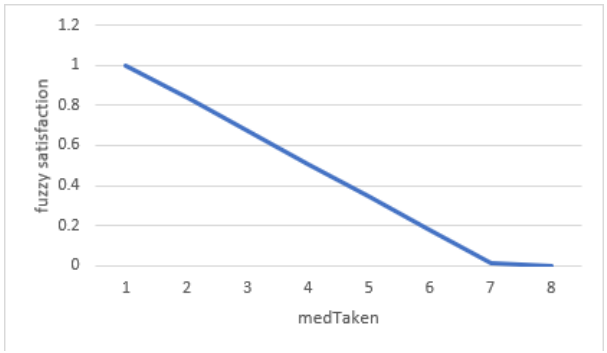
(b)



(c)



(d)



(e)

Figure 3-2. The fuzzy satisfaction of AAL system requirements:

3.4. Graded Representation of Requirement Satisfaction

Graded can be interpreted as assign a rank or score to a certain thing. Fuzzy has been used to replace Boolean yes/no with a graded membership (Trevor Martin & Azvine, 2017). Fuzzy has been widely used in grading system such as measuring students' and instructors' performance (Echauz & Vachtsevanos, 1995; Law, 1996), document grading system (S. Singh & Dey,

2005), egg classification (Omid, Soltani, Dehrouyeh, Mohtasebi, & Ahmadi, 2013) and fish product automatic grading (Hu, Gosine, Cao, & De Silva, 1998).

The fuzzy alpha cut is a method for the defuzzification of fuzzy sets (Clark, Larson, Mordeson, Potter, & Wierman, 2008). For fuzzy set *F* that map set *X* into the closed interval [0,1], the alpha cut F^{α} is defined as in Equation 3-5. We use this concept to represent graded requirement satisfaction in an adaptive system.

$$F^{\alpha} = \{x \in X | F(x) \geq \alpha\} \qquad \text{Equation 3-5}$$

Figure 3-3 is the alpha cut representation for requirement *A1': detect AS MANY AS POSSIBLE foodInfo*. In this representation, at membership 1, there is one element, which is that 10 foodInfo are detected. While the requirement is at membership 0.8333, there are two elements, which are 9 and 10 foodInfo. In other words, we can say that this representation is treating membership as an order (e.g., set {10} produces better satisfaction than set {9,10}, etc.}. The detailed values are shown in Equation 3-6.

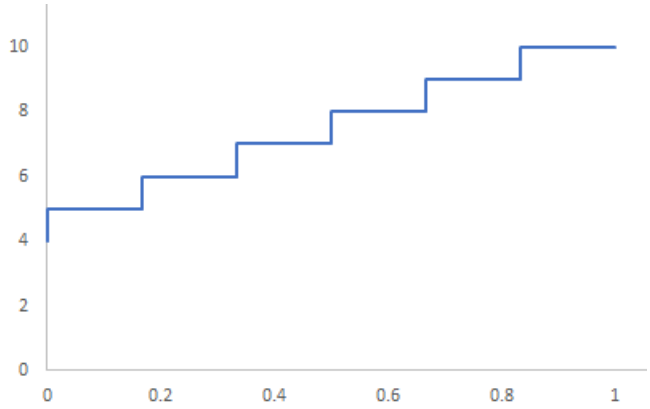


Figure 3-3. Alpha cut representation for requirement A1 of AAL case study

Using our graded representation requirement satisfaction, we can illustrate more flexibly than using crisp Boolean representation. In the context of the AAL case study, if we use the Boolean approach, and the system detects less than ten food information, it is considered to be

failed. However, with our approach, it is acceptable to have partial satisfaction as long as it does not exceed the predetermined threshold (in this case $\theta_l = 5$).

$$A1'(\alpha) = \begin{cases} \{10\}, & \frac{5}{6} < \alpha \leq 1 \\ \{9,10\}, & \frac{4}{6} < \alpha \leq \frac{5}{6} \\ \{8,9,10\}, & \frac{3}{6} < \alpha \leq \frac{4}{6} \\ \{7,8,9,10\}, & \frac{2}{6} < \alpha \leq \frac{3}{6} \\ \{6,7,8,9,10\}, & \frac{1}{6} < \alpha \leq \frac{2}{6} \\ \{5,6,7,8,9,10\}, & 0 < \alpha \leq \frac{1}{6} \end{cases}$$ Equation 3-6

Furthermore, our proposed approach can grade requirement satisfaction and tell us which requirement value is better than another. For example, the subset {9,10} of food information has better satisfaction than subset {8,9,10}. In addition, by using the alpha cut as a graded representation of requirement satisfaction, we know which subset of requirement values satisfies a certain degree of requirement satisfaction. Thus, we can see how close each set of requirements is to be FULLY satisfied.

3.5. Summary

The RELAX approach introduces two types of requirements, invariant and relax requirements. The invariant requirement is one that must be fully satisfied. On the other hand, the relax requirement tolerates partial satisfaction in a particular condition by relaxing the requirement. Relaxing a requirement means that we define a more flexible threshold to create a different satisfaction level, rather than merely using Boolean 0 and 1 value to represent requirement satisfaction.

In this chapter, we incorporate the relax requirement language and fuzzy set theory to describe the level of requirement satisfaction. The AAL system case study is used to

demonstrate how to calculate the fuzzy satisfaction value. In addition, we present the graded representation of requirement satisfaction using a fuzzy alpha cut. Besides being flexible, a fuzzy alpha cut also provides information about which requirement set is better than others. Thus, we know how close each requirement set is to be fully satisfied.

Chapter 4

Systematic Mathematical Approach in Relaxing Related Requirements in Adaptive Systems

4.1. Introduction

In chapter 3, we present the calculation of the fuzzy satisfaction of RELAX requirements. Relaxing the requirements of an adaptive system means we tolerate the requirements to be particularly satisfied. Thus, we are able to define more flexible requirement satisfaction. Moreover, the degree of membership reveals how close requirement satisfaction is to be fully satisfied.

Even though relaxing requirements can yield more flexible satisfaction representation, we should not perform relaxation arbitrarily. This is because, generally, a requirement has a relationship with other requirements (J. Whittle et al., 2009). Thus, relaxing one requirement can affect the satisfaction of related requirements.

The RELAX requirement language provides an uncertainty factor called DEP (dependency) to indicate the relationship between requirements in the adaptive system (Jon Whittle et al., 2010). The DEP uncertainty factor describes the impact of relaxing a given

requirement on the dependent requirement, either positive or negative, or both. The DEP uncertainty factor is written informally in a natural language form.

As relaxing one requirement can affect the satisfaction of other requirements, either positively or negatively, finding optimal relaxation is necessary. Optimal relaxation is the point at which the same fuzzy satisfaction for all requirements is optimally reached. This optimal relaxation can later be used to recommend a new requirement threshold.

In this chapter, we propose a systematic approach to relax requirement satisfaction, and we recommend new requirement thresholds based on optimal relaxation. Moreover, the DEP uncertainty factor is formalized and utilized to describe the effect of relaxing one requirement on another related requirement. We use two case studies, smart vacuum systems (SVS) and distributed database systems (DDS), to exemplify how to perform systematic relaxation. The UPPAAL model checker is employed to model the system and implement verification.

4.2. Proposed Approach: Systematic Relaxation for Adaptive System

This section introduces a systematic approach to relaxing requirement satisfaction on the adaptive system. The approach utilizes the RELAX specification language (J. Whittle et al., 2009) to define the requirements. The goal is to find the optimal relaxation where all requirements can be satisfied to a certain degree. Subsequently, we employ UPPAAL (Behrmann et al., 2010) to carry out the verification of the optimal relaxation. This optimal relaxation can further be used to recommend a new requirement threshold to the Requirement Engineer.

The systematic mathematical approach, as shown in Figure 4-1, extracts relax requirements from system description. Those requirements are then converted into an equation of relaxation degree. Next, mathematical dependencies are defined. Using the requirements in

the relaxation degree and mathematical dependencies, we can formulate optimal relaxation. The system will be modelled and verified using the UPPAAL model checker. We will explain each step clearly in the following sections.

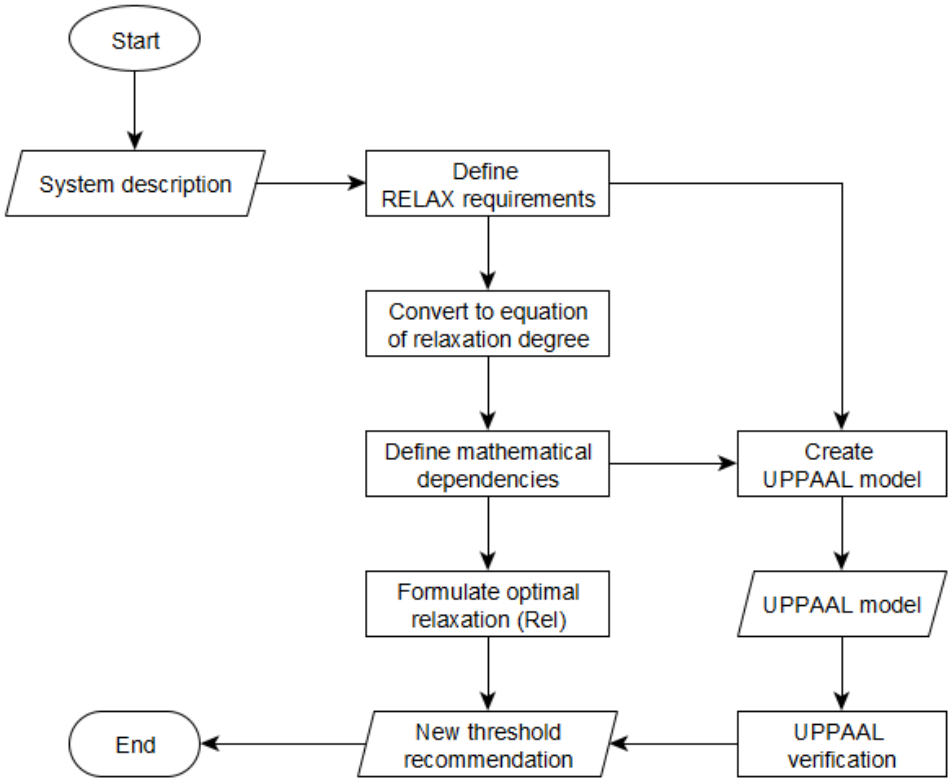


Figure 4-1. Systematic approach for verification and recommendation on RELAX requirement

4.2.1 Input

For this approach, we need a system description explaining the problem solved by the adaptive system. This usually specifies the system features (explicitly or implicitly), namely what its environment and constraints are. The system description has to be described clearly, including all assumptions needed, so that the requirement engineer can extract the requirements easily.

4.2.2 Systematic Approach

The proposed systematic approach describes procedures to relax requirement satisfaction and obtain optimal relaxation by examining the mathematical relationship between requirements. We explain the steps as follows:

1) Define RELAX requirements

The first step in the systematic approach is defining relax requirements. The requirements are obtained from the system description. The requirements especially need to state the original requirement value, RELAX variable, and the relationship between requirements.

RELAX defines the uncertainty factor called DEP (dependency) to indicate the effect of relaxing a requirement to the satisfaction of one or more related requirements. The implication concept of propositional calculus (Goldrei, 2005) can be used to portray the relationship between requirements, as stated in the DEP uncertainty factor. In the propositional calculus where θ and ω are the premise, the implication formula ($\theta \rightarrow \omega$) is further interpreted as "implies" or if θ is true then ω will also be true.

Such a concept is implemented to define the requirement relationship. Given $r_i \in R$ is the adaptive system requirements. The type of relationship (tr) defines the implication of relaxing one requirement to another requirement, which can be classified into three relationship types: plus (+), minus (-), or plus-minus (±) sign. The relationship between the requirement r_x and requirement r_y is defined as shown in Equation 4-1.

$$r_x \overset{tr}{\rightarrow} r_y \quad where\ x \neq y, r_x \in R, r_y \in R \qquad \text{Equation 4-1}$$

The type of relationship (tr) can be interpreted as follows:

- Positive (+) relationship between r_x and r_y means that relaxing requirement r_x will increase the satisfaction of the requirement r_y. This means that when we decrease the alpha cut value of r_x, the value of alpha cut in r_y will be higher. In the relax requirement, it is indicated with a positive impact on the DEP uncertainty factor.
- Negative (-) relationship meaning that relaxing r_x will decrease the satisfaction of r_y. This means that when we decrease the alpha cut value of r_x, the value of alpha cut in r_y will be

lower. In the relax requirement, it is indicated with a negative impact on the DEP uncertainty factor.

- Positive negative (±) relationship meaning relaxing r_x will both increase and decrease the satisfaction result on the requirement r_y. This means that when we decrease the alpha cut value of r_x, the value of alpha cut in r_y will be getting both lower and higher. In the relax requirement, it is indicated with both positive and negative impact in DEP uncertainty factor.

2) Convert RELAX requirements into the equation of relaxation degree

In this step, we transform each of the RELAX requirement into an equation of relaxation degree. Relaxation degree (α) is the fuzzy membership degree that is used to relax the requirement satisfaction. The translation can be done using the fuzzy alpha cut method, as shown in part 3.4 of this thesis.

In this works, we assumed that no requirement is more important than the other requirement. As there is no ranking in the requirement's importance, we use the same alpha cut value while relaxing all requirement in the adaptive system. Subsequently, we find the optimal alpha cut works for all requirement.

3) Define Mathematical Dependencies

The dependencies define the relationship between several RELAX requirements mathematically. A system has two dependencies, one to define the mathematical relationship between several requirements (*D1*) and another to establish boundaries or conditions to be achieved (*D2*).

4) Obtain Optimal Relaxation $(\boldsymbol{Rel})$

Relaxing a requirement that has a relationship to another one can alter the satisfaction value of related requirements, either positively or negatively (R. N. E. Anggraini & Martin, 2018). Consequently, we need to find the optimal relaxation where all related requirements and dependencies are satisfied to a certain degree. The supremum of relaxation degree is able to represent the optimal relaxation. (The supremum is the least upper bound of a set, which, in this case, is the set of fuzzy requirement satisfaction.)

Given D_1 and D_2 is the dependencies, and α is the relaxation degree, we can define the optimal relaxation *Rel* as the supremum of the α set as shown in Equation 4-2.

$$Rel = sup\{\alpha | D_1 \wedge D_2\} \quad \text{Equation 4-2}$$

5) Create the UPPAAL Model

We build the model of the adaptive system using the UPPAAL model checker. The model has to include a Success node, which will be visited when the dependencies are fulfilled. Otherwise, the Fail node will be visited.

6) UPPAAL Verification

The model is verified using reachability properties `E<>`φ. It will check if there is a path from the initial state where φ is satisfied or in other words, if a certain path will be visited. The two reachability properties used are `E<>System_name.Success` and `E<>System_name.Fail`. Those two properties are used to verify if the Success or Fail node will eventually be visited. Different combinations of relaxation degrees (α) will be verified.

Table 4-1 can be used to draw conclusions of the verification results.

Table 4-1. Verification result category

Reachability properties		Verification result
`E<>System_name.Success`	`E<>System_name.Fail`	
Property is not satisfied	Property is satisfied	Requirement never satisfied
Property is satisfied	Property is satisfied	Requirement sometimes satisfied
Property is satisfied	Property is not satisfied	Requirement always satisfied

4.2.3 Output

The optimal relaxation generates the optimal degree of membership that satisfies all dependencies for related requirements. Subsequently, we can translate this degree of membership into the real requirement value and recommend it as the new requirement threshold.

We use the x-mu approach (TP Martin, 2015) to get the value of the requirements threshold. *X-mu* is the inverse of membership function; thus, it maps fuzzy membership onto the universe. Let *X* denote the inverse of membership function μ; the new requirement value r_{new} at the optimal relaxation *Rel* will be mapped using Equation 4-3.

$$X_{\mu}(Rel) = r_{new} \qquad \text{Equation 4-3}$$

4.3. Case study 1: Smart Vacuum Systems

4.3.1 System Description

An SVS is a robot that is able to clean an area and to balance path plan and power preservation (Fredericks et al., 2014; Prassler, Ritter, Schaeffer, & Fiorini, 2000). The SVS robot has some sensors, such as a bumper sensor to prevent collisions and motor sensors for moderating its speed and power. The controller uses data from sensors to determine the optimal path and

preserve battery life. Uncertainty comes from the sensor data noise and the environment (the amount of dirt spread in the area and the power needed to clean the area). Thus, the SVS needs adaptation to provide acceptable satisfaction.

For this case study, consider we have a room divided into square units, each of which is dirty. A set of autonomous vacuum robots is needed to clean the room. Let us assume the room's dimensions are 20x25 (500 square units). In this case, we ignore complex issues such as route planning or the coordination between the robots. Though there are five vacuums available, the fewer vacuums used, the better. Each vacuum has 100 units of initial battery power, where each battery unit will clean 1 square unit of the room. Each vacuum needs to complete the cleaning task with no more than 80 battery units used, but this rule can also be relaxed up to the point at which almost all of the battery is used. The aim is to clean all of the room spaces, but it is acceptable to clean at least 80% of the spaces.

4.3.2 Applying Systematic Approach to SVS case study

The first step of the systematic approach is defining the relax requirements based on the system description. The SVS relax requirements are shown in Table 4-2. In the table, we specify the uncertainty factors from the RELAX approach. We also define the original threshold and relax variables. The relationship is extracted from the DEP uncertainty factor.

Table 4-2. SVS relax requirements

ID	RELAX Requirement	Original threshold	RELAX variable	Relationship
R1'	AS MANY AS POSSIBLE cleanArea ENV: room space MON: motion sensor REL: motion sensor provides the room space that has been cleaned DEP: R1 is positively impacted by R2, R1 is positively impacted by R3	500	100	$R2 \xrightarrow{+} R1$ $R3 \xrightarrow{+} R1$
R2'	AS FEW AS POSSIBLE TO numberOfVacuum ENV: vacuums MON: motion sensors, motor sensors	1	5	$R2 \xrightarrow{+} R1$

	REL: the motion and motor sensors will indicate if the vacuums are used DEP: R2 positively impacts R1			
R3'	AS FEW AS POSSIBLE batteryUsed ENV: remaining battery MON: motor sensor REL: motor sensor will determine how much battery power is left DEP: R3 positively impacts R1	80	20	$R3 \overset{+}{\rightarrow} R1$

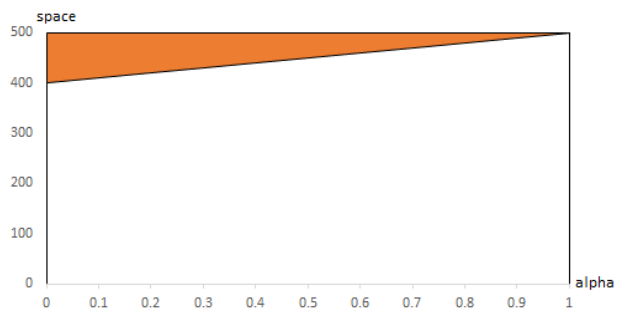
After obtaining the RELAX requirements, we translate each of them into the equation of relaxation degree (α). The result is shown in Equation 4-4, Equation 4-5, Equation 4-6, which defines *R1'*, *R2'*, and *R3'* subsequently. Figure 4-2 illustrates the satisfaction of SVS requirements based on the equation of relaxation degree.

$$R1': CleanTarget \quad T = [400 + 100\alpha, 500] \ , \ 0 < \alpha \leq 1 \qquad \text{Equation 4-4}$$

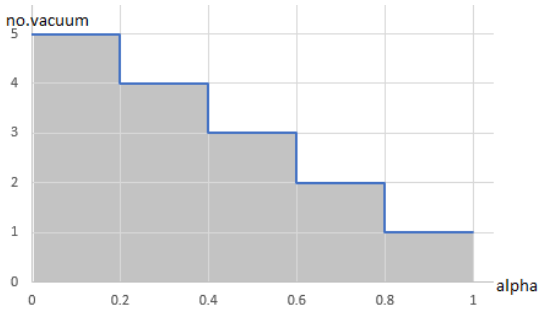
$$R2': NumberOfVacuum\ n \in N(\alpha)\ where\ \ N(\alpha) = \qquad \text{Equation 4-5}$$

$$\begin{cases} \{1\}, & \frac{4}{5} < \alpha \leq 1 \\ \{1,2\}, & \frac{3}{5} < \alpha \leq \frac{4}{5} \\ \{1,2,3\}, & \frac{2}{5} < \alpha \leq \frac{3}{5} \\ \{1,2,3,4\}, & \frac{1}{5} < \alpha \leq \frac{2}{5} \\ \{1,2,3,4,5\}, & 0 < \alpha \leq \frac{1}{5} \end{cases}$$

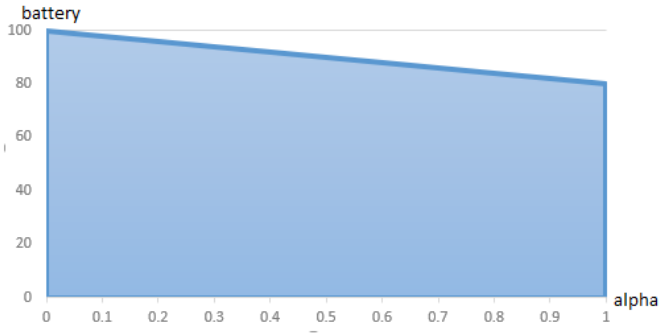
$$R3': BatteryUsed\ \ B_i \in [0{,}100 - 20\alpha],\ \ 1 \leq i \leq n,\ \ \ 0 < \alpha \leq 1 \qquad \text{Equation 4-6}$$



(a)



(b)



(c)

Figure 4-2. Relaxed quantities of SVS requirements

The mathematical dependencies, *D1* and *D2*, are then defined as shown in Equation 4-7 and Equation 4-8. The first dependency equation defines the mathematical relationship between requirements in the case study, and the second is the condition to be achieved.

$$D1: AreaCleaned \quad A = \sum_{i=1}^{n} B_i \qquad \text{Equation 4-7}$$

$$D2: T \leq A \qquad \text{Equation 4-8}$$

The optimal relaxation is further determined using the supremum of satisfied membership function for all requirements and dependencies (see Equation 4-9). Figure 4-3 illustrates the systematic approach to achieve optimal relaxation in the SVS case study, which is the point at which two lines cross each other. In this case study, we get the value of optimal relaxation is $Rel = 0.2$.

$$Rel = \sup\{\alpha | 400 + 100\alpha \leq \sum_{i=1}^{n} B_i \wedge n \in N(\alpha) \wedge B_i \in [0{,}100 - 20\alpha], 1 \leq i \leq n\} \qquad \text{Equation 4-9}$$

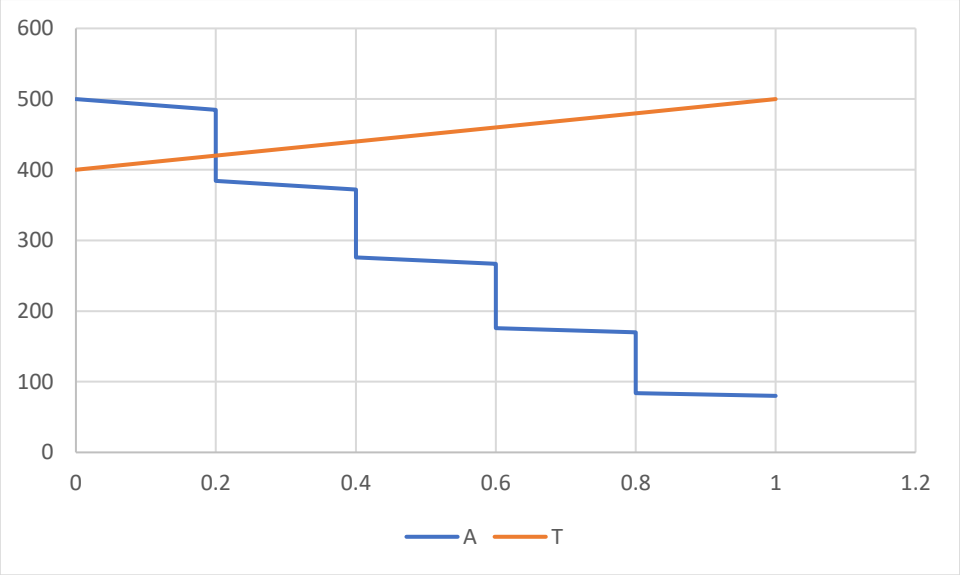


Figure 4-3. Optimal relaxation of SVS case study

To prove the correctness of the result yielded by our systematic mathematical approach, we build a model and verify it using the UPPAAL model checker. Two nodes, the Success and Fail nodes (see Figure 4-4), is used to indicate whether the dependencies are fulfilled or not. We then set various membership degrees and use

Table 4-1 to draw the conclusions.

The verification is performed by reachability properties `E<>SVS.Success` and `E<>SVS.Fail`. The *'never satisfied'* result means that the fail node is eventually be visited while the success node is not. The *'sometimes satisfied'* result means that the success and fail node are both visited. And the *'always satisfied'* result means that the success node is visited while the fail node is not.

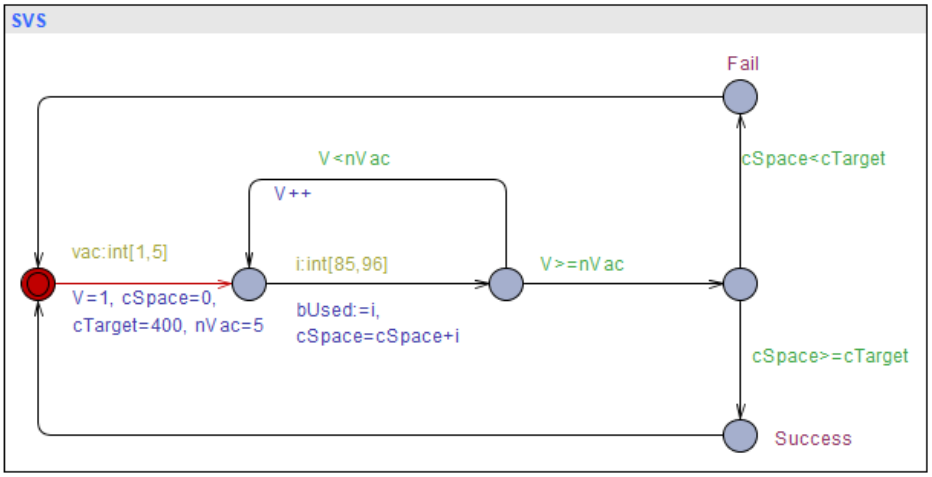


Figure 4-4. SVS model in UPPAAL

Table 4-3. Property settings and verification results on SVS case study

Alpha (α)	Battery Used	Number of Vacuum	Verification result
No relaxation	80	1	Property NEVER SATISFIED
$^4/_5 < \alpha \leq 1$	[81,84]	1	Property NEVER SATISFIED
$^3/_5 < \alpha \leq ^4/_5$	[81,88]	1,2	Property NEVER SATISFIED
$^2/_5 < \alpha \leq ^3/_5$	[81,92]	1,2,3	Property NEVER SATISFIED
$^1/_5 < \alpha \leq ^2/_5$	[81,96]	1,2,3,4	Property NEVER SATISFIED
$0 < \alpha \leq ^1/_5$	[81,100]	1,2,3,4,5	Property SOMETIMES SATISFIED
	[97,100]	4,5	Property SOMETIMES SATISFIED
	[97,100]	5	Property FULLY SATISFIED
	[85,100]	5	Property FULLY SATISFIED

The verification results of the SVS case study are shown in Table 4-3. Consistent with the systematic approach, starting at α=0.2, the `E<>SVS.Success` is satisfied. However, the

orange color indicates that only some properties are satisfied. Thus, we need to explore in greater depth exactly which combination will only visit the Success node. Based on Table 4-3, we can see that only when the vacuum number is set to five ($n = 5$) the `E<>SVS.Success` is satisfied while `E<>SVS.Fail` is not satisfied, or in other words, only this combination can deliver the requirements to always be satisfied.

Using the x-mu approach on the optimal relaxation (*Rel*) and the verification result from UPPAAL, we can recommend new requirement thresholds. For the SVS case study, Equation 4-10 shows the x-mu equation.

$$X_{\mu}(0.2) = r_{new} \qquad \text{Equation 4-10}$$

Based on the optimal relaxation and further verification on UPPAAL, we get the new requirement threshold $R1_{new} = 80\%\ clean\ space,\ R2_{new} = 5\ vacuums,$ and $R3_{new} = 85$ battery units.

4.4. Case study 2: Distributed Database Systems

4.4.1 System Description

A distributed database is a number of interrelated databases distributed in several computers over a network (Özsu & Valduriez, 1999). The purposes of building distributed database systems (DDS) are including data sharing, autonomy, and availability (Silberschatz, Korth, & Sudarshan, 1997). To achieve these purposes, the concept of replication and fragmentation has to be understood and applied (Sleit, AlMobaideen, Al-Areqi, & Yahya, 2007).

Database replication is a mechanism of creating and maintaining multiple instances of the same database throughout several duplicate databases in a distributed database system (Mazilu, 2010). Replication can ensure data availability as it save the database replicas at one or more sites (Charron-Bost, Pedone, & Schiper, 2010; Moiz, Sailaja, Venkataswamy, & Pal,

2011). Moreover, it creates better reliability during failures by activating recovery through failover scenarios (T. Singh, Sandhu, Bhatti, & Engineering, 2013). The failover scenario will automatically switch the database access from the original replica to another replica in order to ensure data availability.

In database replication, there is a primary database and one or more replicas/slaves (Wiesmann, Pedone, Schiper, Kemme, & Alonso, 2000). There are several synchronization techniques in database replication (Yan, Diao, & Jiang, 2008). In this case study, we use semi-synchronous replication (Singhal, Bokare, & Pawar, 2010), in which the data will be committed to the primary database when the log has been sent to all replicas (see Figure 4-5). This scenario could notify the replicas if there is any data update. By contrast, data synchronization is scheduled at a particular time. Data loss is detected by comparing current data in the replica with its log.

Data fragmentation is also applied in this case study. This is a method to divide and distribute data over several servers such that the fragment's combination can provide data without missing any information (Al-Sayyed et al., 2014; Özsu & Valduriez, 2011). The reason behind fragmenting the database is cost efficiency and latency since the fragment is usually placed near the clients (called local database), and only needed data are stored in these fragments. Consequently, the number of data access is minimized (Bhuyar, Gawande, & Deshmukh, 2012); thus, the latency is also minimized.

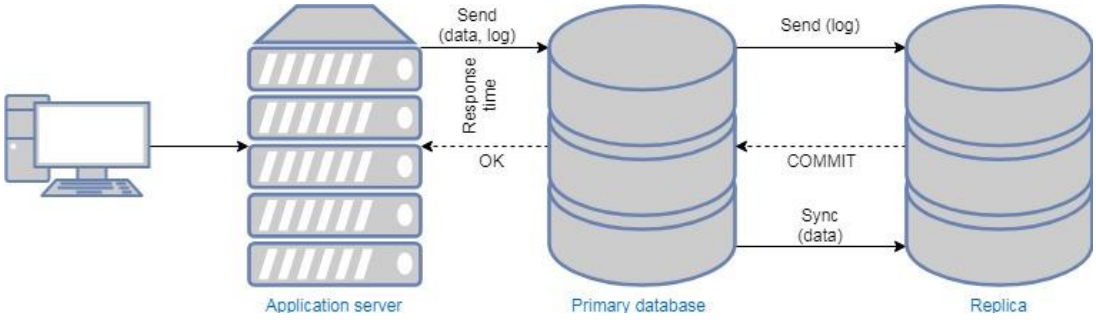


Figure 4-5. Semi-synchronous synchronization

For this case study, we assume there are five office branches in the system. To minimize network and server costs, each branch will have a local database storing the data related to that branch only. The master database and its replica/s store data from all local databases. Database synchronization to the masters is committed every hour unless there are network problems. In the case of a local database failure, the failover scenario will arrange access to one of the master databases. The replication and fragmentation process is illustrated in Figure 4-6.

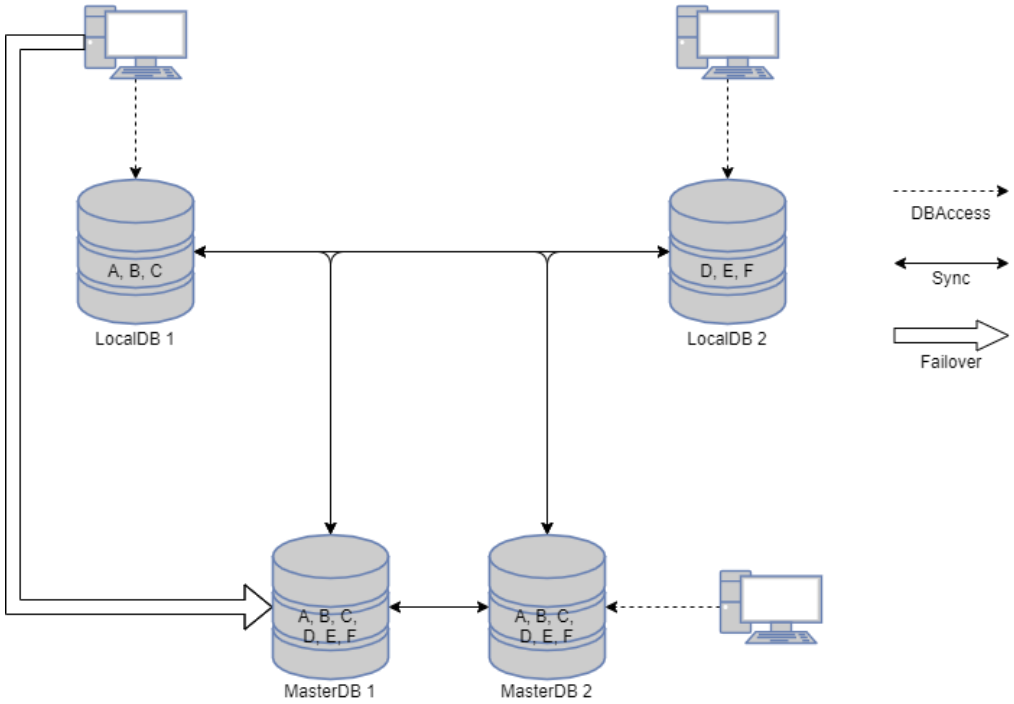


Figure 4-6. Distributed Database Systems Architecture

We specify the following problems for the DDS case study:

- With five branches, ideally, there is 20% dataFragment in each localDB. The masterDB will then save 100% instances. The synchronization is scheduled every hour with no

dataloss, and response time is expected to be 0 seconds. So, ideally, there is 200% instances on five localDBs and one masterDB.

- But as the ideal condition cannot always be achieved, we need to relax the requirements. The number of localDB can be relaxed to ± five servers in total with dataFragment that can be relaxed up to 25% in each local server. As network problems possibly occur, the synchronization process can be relaxed up to 20 times daily, response time up to 5 seconds, and dataloss up to 5%. Thus, up to 190% of instances can be tolerated. We want to minimize the number of masterDB. However, up to 5 of them are tolerable.

Table 4-4. RELAX requirements of DDS case study

ID	RELAX Requirement	Original threshold	RELAX variable	Relationship
R1	AS MANY AS POSSIBLE noInstances ENV: instances MON: all DB servers, log REL: log will provide instances that are recorded in all DB servers DEP: R1 is impacted by R2 positively and negatively, R1 is positively impacted by R3, R1 is negatively impacted by R4, R1 is positively impacted by R5	200	10	$R2 \xrightarrow{+/-} R1$ $R3 \xrightarrow{+} R1$ $R4 \xrightarrow{-} R1$ $R5 \xrightarrow{+} R1$
R2	AS CLOSE AS POSSIBLE noLocalDB TO 5 ENV: localDB MON: application server, instances REL: application server will assign instances to be saved in which localDB DEP: R2 impacts R1 positively and negatively	5	5	$R2 \xrightarrow{+/-} R1$
R3	AS FEW AS POSSIBLE noMaster ENV: masterDBs MON: replication mechanism REL: replication mechanism will arrange the replication to masterDBs DEP: R3 positively impacts R1	1	5	$R3 \xrightarrow{+} R1$
R4	AS FEW AS POSSIBLE dataLoss ENV: localDBs, masterDBs MON: log, instances REL: log will compare the instances that should be recorded in the masterDBs and localDBs DEP: R4 negatively impacts R1	0	5	$R4 \xrightarrow{-} R1$
R5	AS FEW AS POSSIBLE dataFragment ENV: localDBs MON: log, instances REL: comparing log and instances can provide how many dataFragments are in the localDBs DEP: R5 positively impacts R1	20	5	$R5 \xrightarrow{+} R1$

R6	AS MANY AS POSSIBLE noSync ENV: all DB servers MON: network connections, sync schedule REL: network connection provides information if the network between servers are connected during the synchronization schedule DEP: -	24	4
R7	AS EARLY AS POSSIBLE responseTime ENV: localDBs, masterDBs MON: timer REL: the timer will provide how long to send log from localDB to masterDBs and then commit the instances into localDB DEP: -	5	5

4.4.2 Applying Systematic Approach to DDS Case Study

Based on the description of the system, we can obtain several RELAX requirements, as shown in Table 4-4. Some of these requirements have a relationship, while the others do not. Subsequently, we translate these relax requirements into the equation of relaxation degree (see the relax quantities in Figure 4-7) and define the mathematical dependencies. The optimal relaxation (see Figure 4-8) will then be verified with the UPPAAL model (see Figure 4-9) by modifying the relaxation degree (see Table 4-5).

Equation of relaxation degree:

$R1: noInstances \quad I = 190 + 10\alpha, \quad 0 < \alpha \leq 1$

$$R2: noLocalDB\ n \in N(\alpha)\ \ where\ N(\alpha) = \begin{cases} \{5\}, & 0.8 < \alpha \leq 1 \\ \{4,5,6\} & 0.6 < \alpha \leq 0.8 \\ \{3,4,5,6,7\}, & 0.4 < \alpha \leq 0.6 \\ \{2,3,4,5,6,7,8\}, & 0.2 < \alpha \leq 0.4 \\ \{1,2,3,4,5,6,7,8,9\}, & 0 < \alpha \leq 0.2 \end{cases}$$

$$R3: noMasterDB\ \ m \in P(\alpha)\ where\ P(\alpha) = \begin{cases} \{1\}, & 0.8 < \alpha \leq 1 \\ \{1,2\}, & 0.6 < \alpha \leq 0.8 \\ \{1,2,3\}, & 0.4 < \alpha \leq 0.6 \\ \{1,2,3,4\}, & 0.2 < \alpha \leq 0.4 \\ \{1,2,3,4,5\}, & 0 < \alpha \leq 0.2 \end{cases}$$

$R4: dataLoss \quad L_i = 5 - 5\alpha, \quad 1 < i \leq n, \ 0 < \alpha \leq 1$

$R5: dataFragment \quad F_i = 25 - 5\alpha, \quad 1 < i \leq n, 0 < \alpha \leq 1$

$R6: noSync \quad S = 20 + 4\alpha, \quad 0 < \alpha \leq 1$

$R7: responseTime \quad T = 10 - 5\alpha, \quad 0 < \alpha \leq 1$

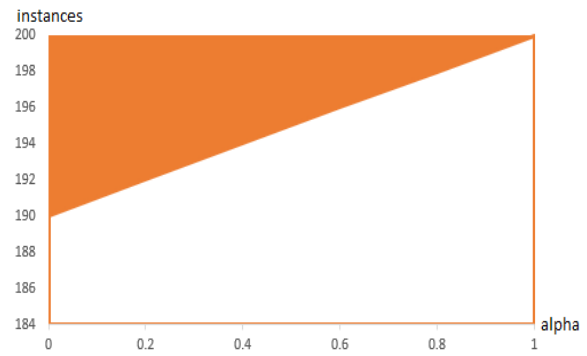
Dependencies:

$D1: DataReceived \quad D = m \times \sum_{i=1}^{n}(F_i - L_i)$

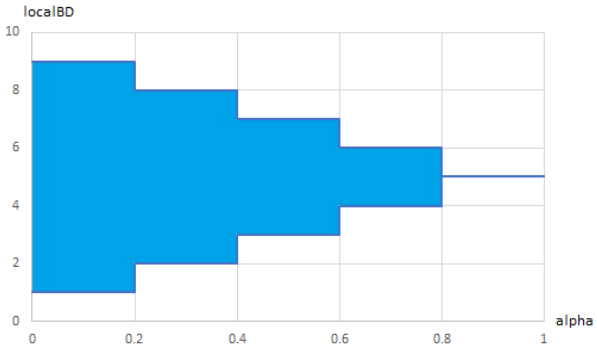
$D2: D \geq I$

Optimal Relaxation

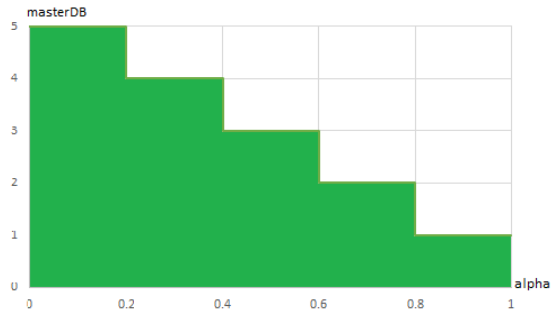
$Rel = sup\ \{\alpha | (m \times \sum_{i=1}^{n}(F_i - L_i)) \geq totalRecord \wedge n \in N(\alpha) \wedge \ p \in P(\alpha)\ ,\ 1 < i \leq n\}$



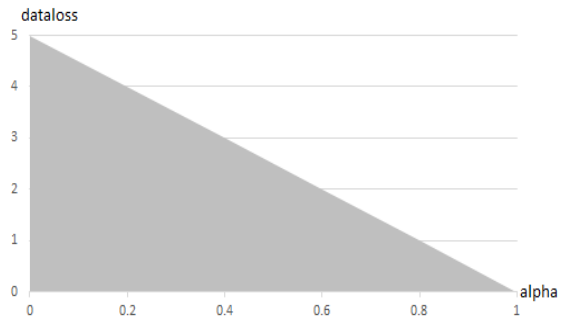
(a)



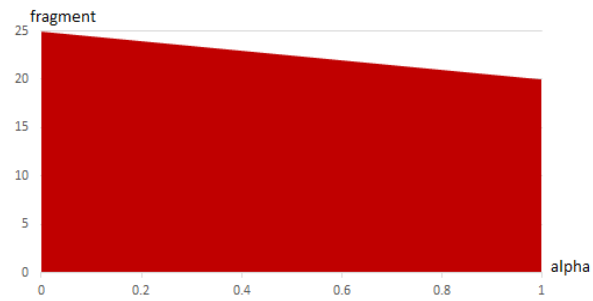
(b)



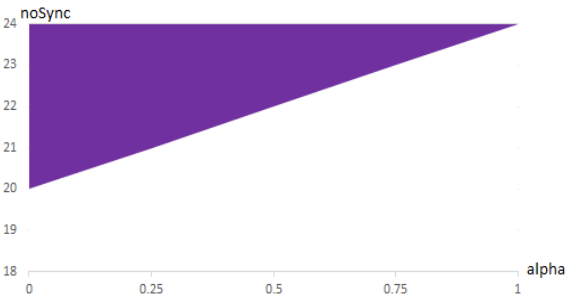
(c)



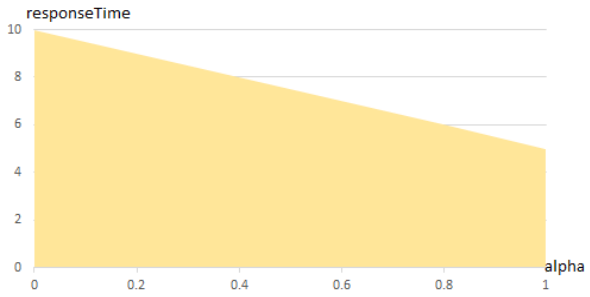
(d)



(e)



(f)



(g)

Figure 4-7. Relaxed quantities of DDS requirements

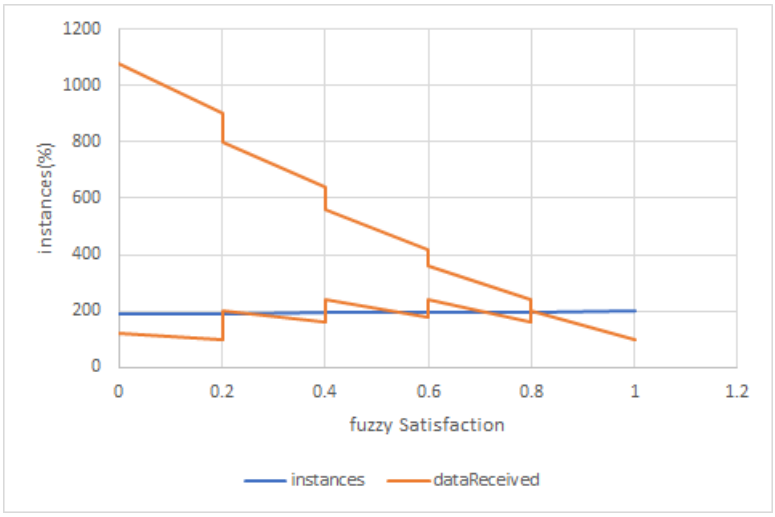


Figure 4-8. Optimal relaxation on DDS case study

UPPAAL verification

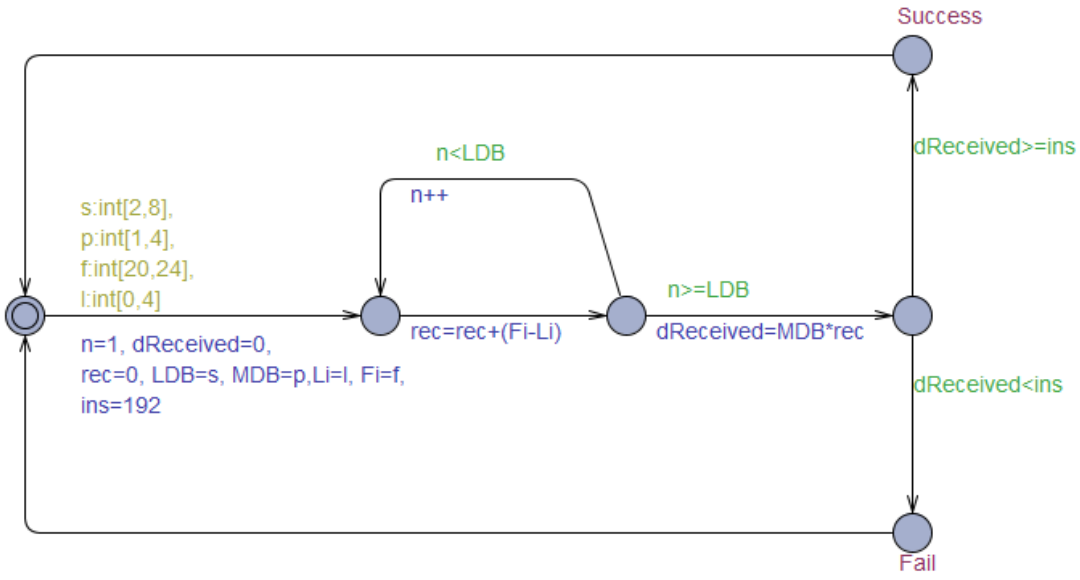


Figure 4-9. UPPAAL model for DDS case study

Based on the optimal relaxation ($Rel = 0.8$) and verification result, the combination of the new threshold is as follow: $R1_{new} = 198\ instances$, $R2_{new} = 6\ localDB$, $R3_{new} = 2\ masterDB$, $R4_{new} = 2\%\ data\ loss$, and $R5_{new} = 20\%\ dataFragment$. The complete set of satisfied requirement thresholds are shown in Table 4-5 with the green highlights.

Table 4-5. Property settings and verification results on DDS case study

Alpha (α)	localDB	masterDB	Dataloss	Fragment	Verification result	
					`E<>DDS.Success`	`E<>DDS.Fail`
No relaxation	5	1	0	20	Property IS NOT SATISFIED	Property IS SATISFIED
$^4/_5 < \alpha \leq 1$	5	1	[0,1]	[20-21]	Property IS NOT SATISFIED	Property IS SATISFIED
$^3/_5 < \alpha \leq {}^4/_5$	[4,6]	[1,2]	[0,2]	[20,22]	Property IS NOT SATISFIED	Property IS SATISFIED
	6	2	[0,2]	[20,22]	Property IS SATISFIED	Property IS NOT SATISFIED
$^2/_5 < \alpha \leq {}^3/_5$	[3,7]	[1,3]	[0,3]	[20,23]	Property IS SATISFIED	Property IS SATISFIED
	[6,7]	[2,3]	[0,3]	[20,23]	Property IS SATISFIED	Property IS NOT SATISFIED
$^1/_5 < \alpha \leq {}^2/_5$	[2,8]	[1,4]	[0,4]	[20,24]	Property IS SATISFIED	Property IS SATISFIED
	[6,8]	[2,4]	[0,4]	[20,24]	Property IS SATISFIED	Property IS NOT SATISFIED
$0 < \alpha \leq {}^1/_5$	[1,9]	[1,5]	[0,5]	[20,25]	Property IS SATISFIED	Property IS SATISFIED
	[2,9]	[2,5]	[0,5]	[20,25]	Property IS SATISFIED	Property IS NOT SATISFIED

4.5. Discussion

This chapter proposes a step by step approach to relaxing related requirements in adaptive systems. The approach takes into account the relationship between requirements. By applying this method, we can find optimal relaxation that can later be used to recommend a new requirement threshold to the system engineer. The following sections discuss the relationship between requirements in the adaptive system and some other related work.

4.5.1 Requirement Relationship

The proposed approach formalizes the DEP uncertainty factor of the RELAX requirement language, which defines the relationship between requirements in the adaptive system. We use the concept of implication to illustrate the relationship of one requirement to another.

However, not all requirements have a relationship with other requirements, as was exemplified in the DDS case study. Requirement R6 (*noSync*) and requirement R7 (*responseTime*) have no relationship defined in the case study. This could be because there is absolutely no relationship, or because the requirement engineer has failed to identify the relationship at an early stage of system development. If the latter situation is the reason, a mechanism to uncover the relationship is necessary. Correlation can be one of the alternative solutions to identify the unknown relationship between requirements (R. N. E. Anggraini & Martin, 2018).

Another issue associated with the requirement relationship is the impact of relaxation on different ordinal/temporal operators. Applying relaxation to requirements with the right or left type of ordinal/temporal operators (categorized in Table 3-1) yields a straightforward effect, either positive or negative, on other requirements. The first case study (SVS case study) illustrates the relaxation of such an operator.

However, the middle type of operator creates some complications. Relaxing requirements for such an operator will create both positive and negative impacts on the related requirements. Such a case is illustrated in the DDS case study. Figure 4-8 portrays such a situation. The more we relax requirement R2 (localDB), the result of the satisfaction of related requirements is both much better and much worse. Currently, we utilize the verification result from the UPPAAL model checker to discover which requirement set can guarantee satisfaction

and subsequently can be used as a recommendation for a set of a new requirement threshold. Thus, further work has to be conducted to explore this problem in depth.

4.5.2 Related Works

The RELAX requirement language is the basis of our work. However, in this approach, we have been more concerned about the relationship between requirements. RELAX addresses this relationship using the DEP uncertainty factor. However, RELAX does not use this factor while relaxing a requirement. Thus, in this works, we consider the impact of the relationship between requirements in the relaxation process.

Some researchers have implemented the RELAX requirement language, including the works in AutoRELAX and ambient system modelling. AutoRELAX combines KAOS, RELAX, and genetic algorithms to generate RELAXed goals (Fredericks et al., 2014). It uses a stepwise adaptation of weights to the technique to optimize the weighting scheme. On the other hand, Ahmad et al. propose requirement modelling for ambient systems using RELAX, SysML, and KAOS. (Ahmad, Bruel, Laleau, & Gnaho, 2012). This work is focused on non-functional requirements.

Another research related to our approach is a goal model called FLAGS (Baresi, Pasquale, & Spoletini, 2010). FLAGS uses the KAOS framework (Van Lamsweerde, 2009) to model the adaptive system. The approach introduces crisp goals with Boolean satisfaction and fuzzy goals with the satisfaction represented with fuzzy constraints, which is similar to invariant and relax requirements in the RELAX approach.

A requirement called awareness requirement (*AwReqs*) also uses a goal-oriented approach to model their goals (Souza et al., 2011). *AwReqs* monitor the success or failure of other requirements through MAPE feedback loops called EEAT (Robinson, 2006). This is similar to

our proposed systematic approach that assesses the impact of relaxing one requirement to the satisfaction of another one and propose optimal relaxation.

4.6. Summary

In this chapter, we introduce a systematic mathematical approach to achieve optimal relaxation of related relax requirements. The systematic approach is initiated by defining relaxed requirements, together with their original threshold, relax variable, and relationship. We formalize the relationship between two requirements that previously were expressed in the DEP uncertainty factor using natural language. Next, we translate these requirements into the equation of relaxation degree and specify the dependencies that relate requirements mathematically. Optimal relaxation is obtained through the supremum of alpha that satisfies the requirements and dependencies.

We employ the UPPAAL model checker to build the model of the system and perform verification. The result from optimal relaxation and UPPAAL verification is recommended to define the new requirement threshold. We apply the approach to two case studies: smart vacuum systems and distributed database systems.

Chapter 5

Nonlinear Fuzzy Requirement Satisfaction on Adaptive System

5.1. Introduction

In the previous chapter, we propose a systematic approach to finding optimal relaxation and recommend new requirement thresholds. We use dependencies to define the mathematical relationship between related requirements. The approach is applied to two case studies, i.e., a smart vacuum system and a distributed database system. In both case studies, we define the mathematical dependencies as a linear relationship in which we can expect that after a certain point, a requirement will be satisfied. Subsequently, we can recommend the optimal relaxation (*Rel*) to identify the new requirement threshold.

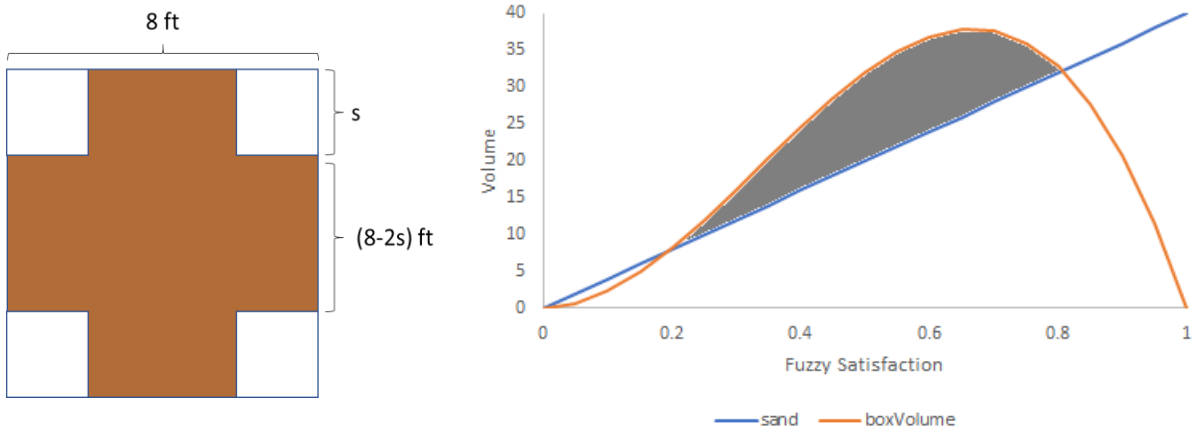
Although most cases in the adaptive system have linear mathematical dependencies, there are some cases with nonlinear mathematical relationships. In such cases, the optimal relaxation (*Rel*) cannot be recommended to define the new threshold because as we relax more, at some point, satisfaction may suffer another failure. Thus, we need to determine how far we can relax the requirement so that satisfaction will always be assured.

In this chapter, we introduce several additional steps to the systematic approach to defining the relaxation area/s in the nonlinear case study. First, we need to find the points of intersection between a linear requirement and a nonlinear dependency after formulating optimal relaxation. Next, we utilize these intersection points to determine the relaxation area/s.

We demonstrate the additional steps in two case studies: stopping distance and company BEP problems. UPPAAL is employed to model and verify the system. The result is the relaxation area/s, where the requirement is ALWAYS satisfied.

5.2. Relaxation Area on Nonlinear Case

Occasionally, the mathematical dependencies between requirements are defined in nonlinear ways, such as in the quadratic equation (Olsthoorn, Tedford, & Lawrence, 2020; Sun, Zhuang, Wu, Zhao, & Zhang, 2015). When the mathematical relationship is nonlinear, more relaxation does not guarantee better satisfaction. At some point, another dependency will be violated, or in other words, we cannot achieve satisfaction even after we find optimal relaxation. Thus, we need to figure out the area/s of relaxation where the requirement satisfaction is always satisfied.



(a) (b)

Figure 5-1. The design of the box from a square (a) and its relaxation area (b)

The problem of making a box, as illustrated in Figure 5-1, is one of the examples where further relaxation can violate dependencies. As a consequence, the satisfaction will suffer another failure. In the box-making problem, a system has to calculate the square cuts at the four corners of the square cardboard, as illustrated in Figure 5-1 (a). Subsequently, this shape will

be built as a box to hold a certain amount of sand. The requirements are to cut as small as possible squares at each corner of the cardboard while saving as much sand as possible.

Based on the description, we can define the problem as follows:

- The cardboard size is 8x8 feet. So, we can cut four square corners with the size of $s \times s$ where $0 < s < 4$ feet. Subsequently, we convert the value of s into a fuzzy set and calculate the box volume for each alpha value.
- The volume of the box can be defined $V_{Box} = (8 - 2s)^2 \times s = 64s - 32s^2 + 4s^3$. The goal is to find the optimal box volume and find the area/s where the box volume can satisfy the requirements.

The result is shown in Figure 5-1 (b) where, at a certain point of relaxation, the requirement satisfaction cannot be fulfilled. Thus, we need to find the intersection points (*x1, x2, x3* in this case) and use them to define the relaxation area (grey part in Figure 5-1. b). In this relaxation area, the requirements are always satisfied.

In this section, we propose two additional steps to the systematic approach to finding relaxation area/s on the nonlinear case study. As shown in Figure 5-2, the highlighted parts are the additional steps as explained below:

1. **Find the points of intersection**

   The points of intersection are where the nonlinear dependency ($D_1(\alpha)$), a dependency relating requirements mathematically, and main requirement ($R_1(\alpha)$) met. We obtain the points of intersections when $D_1(\alpha) = r_1(\alpha)$. Subsequently, we rearrange the equation and obtain the nonlinear equation.

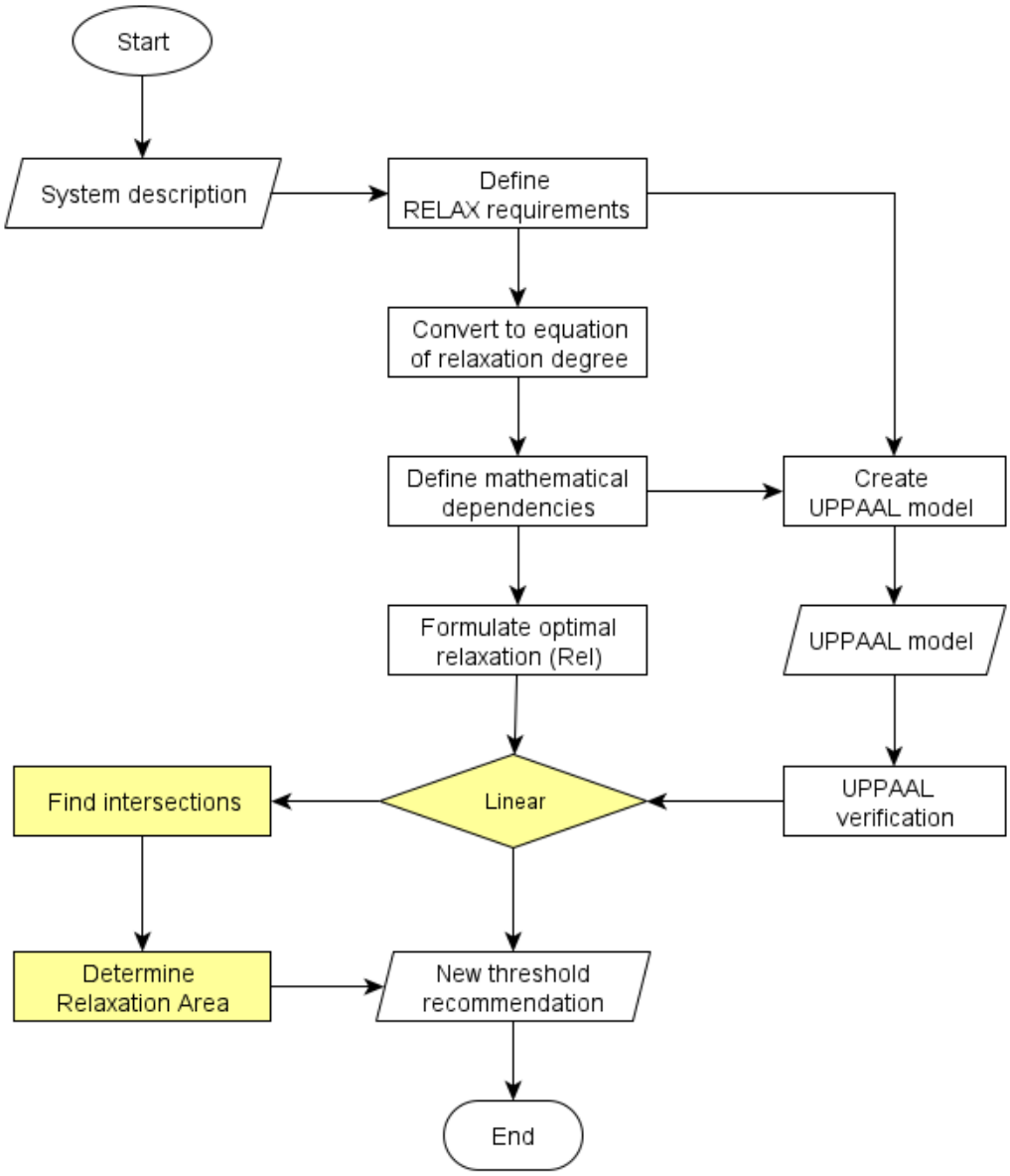


Figure 5-2. Additional steps on Finding Relaxation Area/s in Quadratic Case

For example, Equation 5-1 is the nonlinear equation form for quadratic dependency. Using the quadratic formula in Equation 5-2, we can get the points of intersection (in this case α_1 and α_2). While for other nonlinear equation, we can adjust it based on the type of the equation.

$$a\alpha^2 + b\alpha + c = 0 \qquad \text{Equation 5-1}$$

$$\alpha_i = \frac{-b \pm \sqrt[2]{b^2 - 4ac}}{2a} \qquad \text{Equation 5-2}$$

2. Determine the relaxation area/s

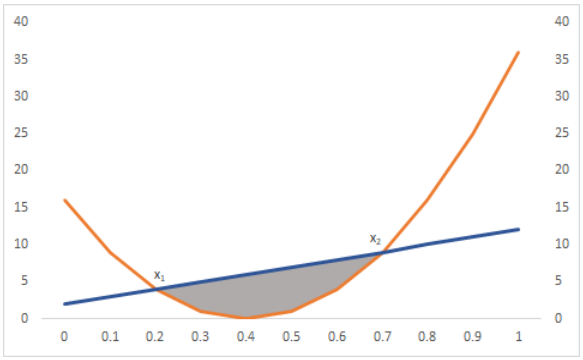
Using the points of intersection from the previous step, we can determine the relaxation area/s. Relaxation area/s is where the requirements will always be satisfied (see Equation 5-3). In this case, *D1* is the nonlinear dependency equation that defines the relationship between requirements and D2 is the constraint. Outside of the *RelArea*, the requirements are never satisfied.

$$RelArea: \{\alpha | D_1 \wedge D_2\}$$ Equation 5-3

As an example, we illustrate the quadratic case using one and two relaxation areas in Figure 5-3. In the first case, initially (without relaxation), the satisfaction is not achieved, so we need to relax the requirement. Then, after the first point of intersection, the requirement is satisfied. However, as we continue relaxing the requirement, and after the second point of intersection, the requirement suffers another failure. Thus, we can define that the relaxation area is between two points of intersection.

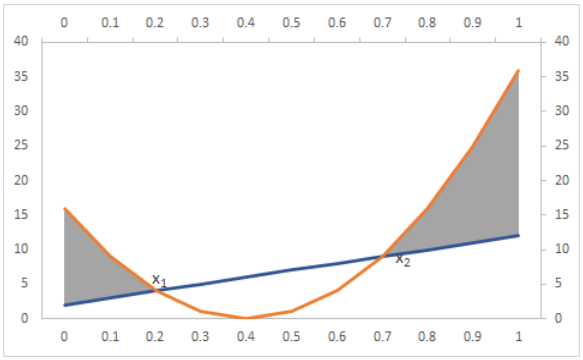
In the second case, we look at two relaxation areas, as shown in Figure 5-3 (b). The first area is between fully satisfied ($\alpha = 1$) and the first point of intersection. After this, the requirement fails to be satisfied. However, the more we relax, and after the second point of intersection, we discover the second area of relaxation. Thus, we can conclude that in these two relaxation areas, the requirement satisfaction will always be satisfied.

The number of relaxation area/s in a nonlinear adaptive system is depended on the mathematical dependencies' equation. In the next section, we will demonstrate the additional steps of a systematic approach to two case studies with quadratic and cubic dependency equations. We will first explain the system description. Then, we apply the approach and obtain the relaxation area/s for the case studies.



$RelArea: \{\alpha | \ \alpha_1 \leq \alpha \leq \alpha_2\}$

(a)



$RelArea: \{\alpha | 0 < \alpha \leq \alpha_1 \ and \ \alpha_2 \leq \alpha \leq 1\}$

(b)

Figure 5-3. Relaxation area/s of quadratic case study

5.3. Quadratic Case study: Stopping Distance

5.3.1 System Description

Estimating stopping distance is crucial for pedestrian safety (Sun et al., 2015). The stopping distance of a vehicle consists of the driver's reaction time and the vehicle's braking distance (Kordani et al., 2018; Lawson, Tabor, & IMA, 2001). The reaction (thinking) distance is the distance converted from driver reaction time. The braking distance is the distance travelled after applying the brake. The stopping distance can vary on different road condition (Lyubenov, 2011) and with a different braking system (Green, 2006)

In this case study, consider an automobile vehicle is moving at a speed of 65 KPH. For safety reasons, we want the stopping distance to be 45 m and reaction time 1 second. However, as the requirement might not be satisfied, we tolerate the stopping distance up to 50 m, speed 15 KPH, and the reaction time 6 seconds.

5.3.2 Applying the Approach to Stopping Distance Problem

We obtain the relax requirement from the system description and compose it in Table 5-1. The three relax requirements are stopping distance, the speed of the vehicle, and the reaction time of the driver. The uncertainty factor and the relationship between requirements are also defined.

Table 5-1. Relax requirement on stopping distance case study

ID	RELAX Requirement	Original threshold	RELAX variable	Relationship
R1	AS FEW AS POSSIBLE stoppingDistance ENV: distance MON: brake, speed sensor REL: brake and speed sensor will provide the distance until the car stops DEP: R1 is impacted by R2 positively, R1 is negatively impacted by R3	45	5	$R2 \overset{+}{\rightarrow} R1$ $R3 \overset{-}{\rightarrow} R1$
R2	AS MANY AS POSSIBLE speed ENV: speed MON: speed sensor REL: speed sensor provides the information of current speed DEP: R2 impacts R1 positively	65	50	$R2 \overset{+}{\rightarrow} R1$
R3	AS FEW AS POSSIBLE reactionTime ENV: braking time MON: brake, collision sensor REL: braking time is calculated between the time the collision is detected and the time the brake is applied DEP: R3 impacts R1 negatively	1	5	$R3 \overset{-}{\rightarrow} R1$

Subsequently, we define the equation of relaxation degree of each relax requirement based on its original threshold and relax variable. Next, we illustrate relaxed quantities of stopping distance requirements (see Figure 5-4). Later, the mathematical dependencies are defined where one of them is in the form of a quadratic equation.

Equation of relaxation degree

$R1: stoppingDistance \quad D = 50 - 5\alpha, \;\; 0 < \alpha \leq 1$

$R2: speed \quad V = 15 + 50\alpha, \;\; 0 < \alpha \leq 1$

$R3: reactionTime \quad R = 6 - 5\alpha, \ \ 0 < \alpha \leq 1$

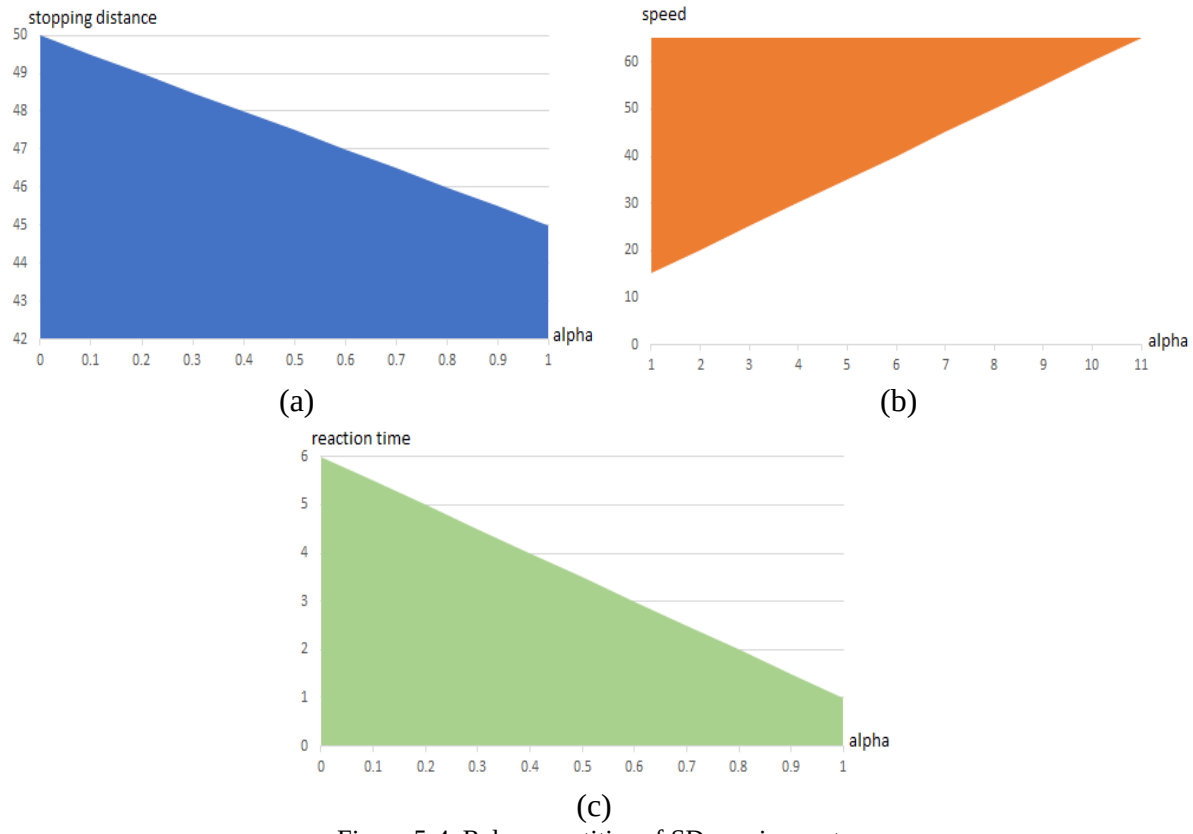


Figure 5-4. Relax quantities of SD requirements

Dependencies:

$$D1: totalDistance \ \ T = \left(R \times V \times {}^{10}/_{36}\right) + \left({}^{V^2}/_{170}\right) \qquad \text{Equation 5-4}$$

$$D2: D \geq T \qquad \text{Equation 5-5}$$

The dependencies are formulated in Equation 5-4 and Equation 5-5. The mathematical dependency relating the requirements is in the form of a quadratic equation. For this case study, we use Equation 5-4 representing the mathematical dependency between requirements (the relationship between speed *V* and reaction time *R* in this case). The left side of the equation is used to calculate the reaction distance, which will convert driver reaction time into the distance travelled during that time. The right side of the equation calculates the braking distance. We

use a braking coefficient of 170 by assuming the vehicle moves in the dry, level pavement. We translate this dependency into a relaxation degree (α) so that we can calculate the intersection later. To do so, we employ the alpha cut equation of each related requirement in the dependency *T*.

$$D1: totalDistance\ \ T = \left(R \times V \times {}^{10}/_{36}\right) + \left({}^{V^2}/_{170}\right)$$

$$T = \left((6 - 5\alpha) \times (15 + 50\alpha) \times \left({}^{10}/_{36}\right)\right) + \left({}^{(15 + 50\alpha)^2}/_{170}\right)$$

$$T = \left({}^{(900 + 2250\alpha - 2500\alpha^2)}/_{36}\right) + \left({}^{(225 + 1500\alpha + 2500\alpha^2)}/_{170}\right)$$

$$T = {}^{(-33500\alpha^2 + 43650\alpha + 16110)}/_{612}$$

$$D2: T \leq D$$

Optimal relaxation

We use the approach in Chapter 4 to formulate the optimal relaxation:

$$Rel = \ sup\left\{\alpha | stoppingDistance \geq \left(R \times V \times {}^{10}/_{36}\right) + \left({}^{V^2}/_{170}\right)\right\}$$

Find points of intersection (α_1 and α_2)

After obtaining the optimal relaxation, the next step is finding the points of intersection between the equation of linear requirement and quadratic dependencies. The lines will cross each other when $D = T$. The following is the calculation of the intersection (α_1 and α_2). The chart in Figure 5-5 illustrates the optimal relaxation and points of intersection in the stopping distance problem.

$$D = T$$

$$50 - 5\alpha = {}^{(-33500\alpha^2 + 43650\alpha + 16110)}/_{612}$$

$$30600 - 3060\alpha = -33500\alpha^2 + 43650\alpha + 16110$$

$$33500\alpha^2 - 46710\alpha + 14490 = 0$$

Based on quadratic form in Equation 5-1, we can define a = 33500, b = -46710, and c = 14490. Then, we calculate the intersections using Equation 5-2.

$\alpha_i = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, so

$$\alpha_i = \frac{46710 \pm \sqrt{(-46710)^2 - 4(33500)(14490)}}{2 \times 33500}$$

$$\alpha_1 = 0.467 \quad \text{and} \quad \alpha_2 = 0.928$$

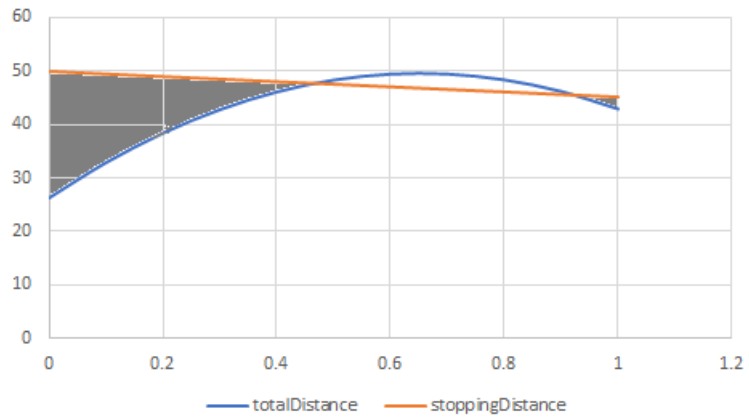


Figure 5-5. The satisfaction of mathematical dependency vs. the main requirement on Stopping Distance

Find relaxation area/s

Based on dependency $D2: D \geq T$, there are two relaxation areas in the stopping distance problem. These relaxation areas are used to recommend new requirement thresholds because, in these areas, the requirement satisfaction is always satisfied. On the other hand, outside these areas, the requirements are never satisfied. The relaxation area is defined as follows:

$$RelArea: \{\alpha | 0 < \alpha \leq 0.467 \; and \; 0.928 \leq \alpha \leq 1\}$$

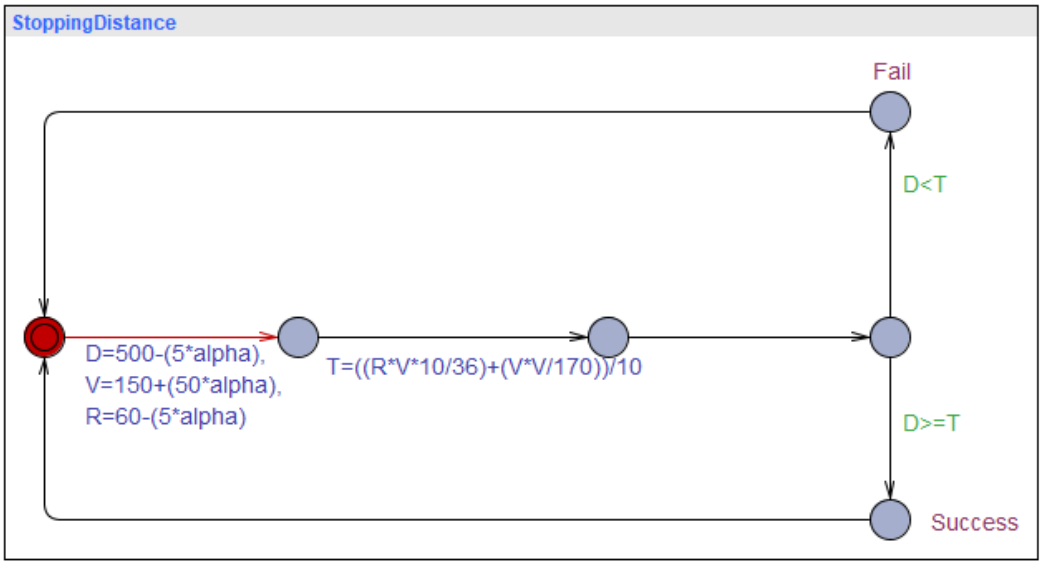


Figure 5-6. UPPAAL model for stopping distance verification

Table 5-2. Property setting and verification result on stopping distance case study

Alpha (α)	Speed V	Reaction Time R	Verification result
No relaxation	65	1	Property FULLY SATISFIED
0.9	60	1.5	Property NEVER SATISFIED
0.8	55	2	Property NEVER SATISFIED
0.7	50	2.5	Property NEVER SATISFIED
0.6	45	3	Property NEVER SATISFIED
0.5	40	3.5	Property NEVER SATISFIED
0.4	35	4	Property FULLY SATISFIED
0.3	30	4.5	Property FULLY SATISFIED
0.2	25	5	Property FULLY SATISFIED
0.1	20	5.5	Property FULLY SATISFIED
0	15	6	Property FULLY SATISFIED

UPPAAL verification

An UPPAAL model of stopping distance problems has been built to verify the approach, as shown in Figure 5-6. We have built it based on the equation of relaxation degree and dependencies. However, UPPAAL has a limitation where it does not support double value in

the simulation, so in this case, we use integer (1 to 10) to simulate the fuzzy satisfaction value then divide the dependency by 10.

The simulation result is shown in Table 5-2. The UPPAAL verification result proves that in quadratic case, more relaxation does not guarantee the requirement satisfaction. At some point in the relaxation process, the mathematical dependency can be violated, which means the satisfaction is not met. In our case study, when the requirement is not relaxed, the simulation result is fully satisfied. However, when the satisfaction starts to be relaxed (at $\alpha = 0.9$), the requirement becomes unsatisfied. This condition continues up to $\alpha = 0.5$, but then starting from $\alpha = 0.4$ the properties is satisfied again. The reason is that the second point of intersection is somewhere between $\alpha = 0.5$ and $\alpha = 0.4$.

These verification results support our proposed additional step to handle the nonlinear case. More relaxation does not ensure requirement satisfaction. Thus, finding the area/s of relaxation is significant as it can be used as the recommendation of a new requirement threshold.

5.4. Cubic Case study: Company BEP and Profit

5.4.1 System Description

The break-even point (BEP) in business is the point where the cost and revenue are equal, meaning no profit nor loss (Kampf, Majerčák, & Švagr, 2016). BEP is useful to determine the number of outputs that start yielding a profit. BEP is analytical tools to provide a view on the relationship between sales, cost, and profits.

In this case study, we utilize a system to calculate the BEP and profit of a company. Assumed for the product Q, the demand function of Q is $P(Q) = 90 - Q$, so the total revenue $TR(Q) = (90 - Q)Q$. The total cost function is $TC(Q) = 0.5Q^3 - 15Q^2 + 175Q$ (Bradley, 2013). The system needs to find BEP and the number of products that make a profit.

5.4.2 Applying the Approach to BEP Problem

We obtain the relax requirements based on the system description, as shown in Table 5-3. Next, we translate the relax requirements into the equation of relaxation degree. Figure 5-7 shows the relaxed quantities of those two requirements. Then, we formulate the dependencies and optimal relaxation.

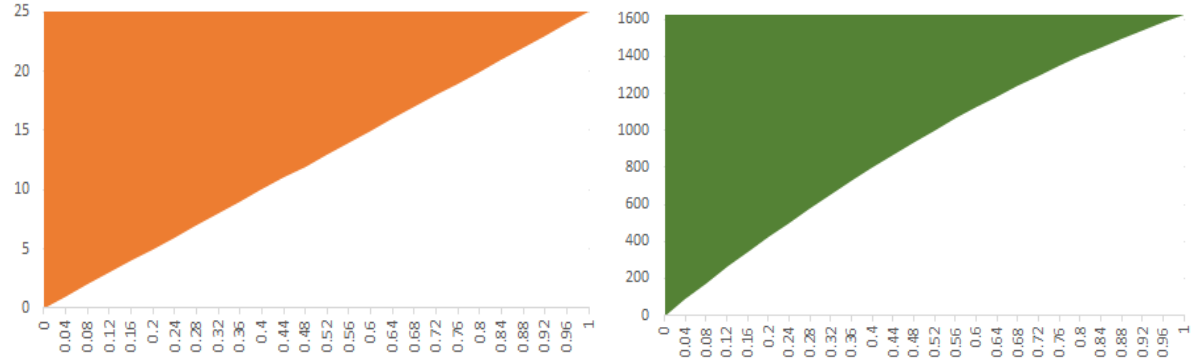
Table 5-3. RELAX requirements of BEP and profit problem

ID	RELAX Requirement	Original threshold	RELAX variable	Relationship
S1	AS MANY AS POSSIBLE totalRevenue DEP: R1 is impacted by R2 positively	1625	1625	$R2 \overset{+}{\rightarrow} R1$
S2	AS MANY AS POSSIBLE product DEP: R2 impacts R1 positively	25	25	$R2 \overset{+}{\rightarrow} R1$

Equation of relaxation degree

$S1: totalRevenue \quad TR = (90 - Q)Q = 2250\alpha - 625\alpha^2, \quad 0 < \alpha \leq 1$

$S2: product \quad Q = 25\alpha, \quad 0 < \alpha \leq 1$



(a) (b)

Figure 5-7. Relax quantities of BEP problem requirements

Dependencies:

$D1: TC = 0.5Q^3 - 15Q^2 + 175Q$

$TC = 7812.5\alpha^3 - 9375\alpha^2 + 4375\alpha$ Equation 5-6

$D2: TR \geq TC$ Equation 5-7

Optimal relaxation

We used the approach in Chapter 4 to formulate the optimal relaxation:

$$Rel = \ sup\{\alpha | TR \geq (0.5Q^3 - 15Q^2 + 175Q)\}$$

Find points of intersection

The next step is to find the points of intersection, which show the BEP of the company. The two lines cross each other at $TC = TR$. The chart in Figure 5-8 illustrates the points of intersection and relaxation area in the case of the company BEP problem.

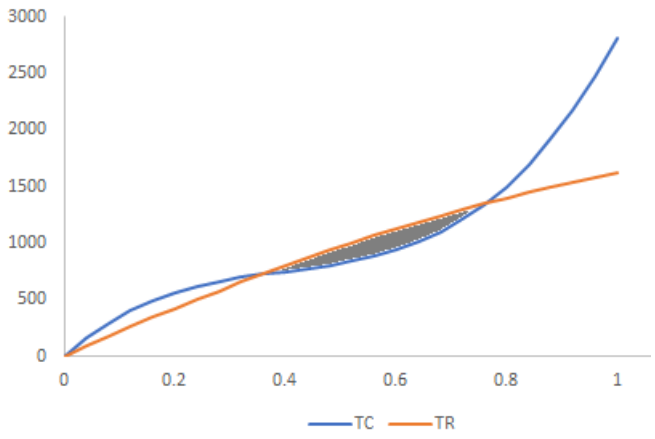


Figure 5-8. The satisfaction of mathematical dependency vs. the main requirement on BEP problem

The following is the calculation to find the point of intersection.

$$TC = TR$$

$$7812.5\alpha^3 - 9375\alpha^2 + 4375\alpha \ = 2250\alpha - 625\alpha^2$$

$$7812.5\alpha^3 - 8750\alpha^2 + 2125\alpha = 0$$

In this case, there are 3 points of intersection, and the values are: $\alpha_1 = 0,\ \alpha_2 = 0.356$ and $\alpha_3 = 0.764$. At these three points, the company is reaching BEP. In other words, these three points are the turning point from loss to profit or vice versa.

Find relaxation area/s

Based on dependency $D2: TR \geq TC$, there is one relaxation area in the BEP problem. In this relaxation area, the requirement is always satisfied so that the company will make a profit. On the other hand, outside the relaxation area, the requirements are never satisfied. Consequently, the company will suffer loss instead. The relaxation area of the company BEP cases study is defined as follows:

$$RelArea: \{\alpha | 0.356 \leq \alpha \leq 0.764 \}$$

5.5. Discussion

This chapter aims to prove that, in some cases, adaptive systems have nonlinear mathematical dependencies. Although most cases work in linear ways, we have to prepare a mechanism to handle nonlinear cases. These types of case studies cannot use optimal relaxation as a means to recommend new requirement thresholds as, at some points, after optimal relaxation is achieved, the satisfaction can suffer another failure. Thus, we need to find the area/s of relaxation where the requirement satisfaction is always satisfied.

Before the area/s of relaxation can be defined, we need to find the points of intersection between the primary requirement and the nonlinear dependency. A system can have one relaxation area (e.g., in the BEP problem) or more (e.g., in the stopping distance problem). The number of relaxation areas in the nonlinear adaptive system depends on the dependencies and requirement equation. This relaxation area/s can then be recommended to the system engineer as a new requirement threshold where the requirement is always satisfied. Subsequently, the system with guaranteed requirement satisfaction can be built based on these new requirement thresholds.

However, in this research, we also identify the issues in the UPPAAL model checker arising from its limitation[8]. One UPPAAL limitation comes from the range of integer variables (It has to be between -32767 and 32767). This is why we cannot perform verification in the company BEP case study, as the value exceeds the variable limitation of UPPAAL. Furthermore, UPPAAL cannot have double data types in the structure. Thus, we have to translate double values into integers to be able to proceed with verification.

5.6. Summary

In this chapter, we present additional steps to find the area/s of relaxation in nonlinear case studies. These steps are essential because, in nonlinear cases, more relaxation does not guarantee satisfaction. At some point, the dependencies will be violated again, meaning requirement satisfaction will suffer another failure. Thus, we introduce two additional steps to the systematic approach to handling this problem.

First, we need to find the points of intersection between primary requirement and nonlinear dependency. Subsequently, these points are used to determine the area/s of relaxation. This area is where the requirement is always guaranteed to be satisfied. We demonstrate this systematic approach with additional steps using two case studies, stopping distance and company BEP.

[8] https://people.cs.kuleuven.be/~danny.weyns/software/ActivFORMS/Current-restrictions-Uppaal.pdf

Chapter 6

Using Correlation to Capture Relationship Strength

6.1. Introduction

An adaptive system is famous for its adaptivity in response to the changing environment or system settings enabling it to satisfy particular objectives. The adaptations modify requirements, models, and contexts (R. N. Anggraini & Martin, 2017). Many challenges arise from the dynamic nature of the adaptive system, one of which is the need for a particular requirement language designed to meet particular challenges created by the adaptive system (de Lemos et al., 2013). One such language is the RELAX requirement language.

The RELAX requirement language supports the adaptivity inherent in the adaptive system and the uncertainty that this creates (J. Whittle et al., 2009). The temporal and ordinal operators in the relax requirements enable the system to tolerate requirement satisfaction. Relaxing a requirement means that we tolerate the requirement satisfaction being partially satisfied. However, we cannot merely arbitrarily relax a requirement because some requirements are related to other requirements. Thus, relaxing one requirement can affect the satisfaction of other requirements.

RELAX provides the DEP uncertainty factor to describe the relationship between two requirements (Jon Whittle et al., 2010). There are two types of dependency in the DEP factor:

- Positive DEP means that relaxing a requirement will support the ability of an adaptive system to satisfy the related requirement
- Negative DEP means that relaxing a requirement will impair the ability of the system to satisfy associated requirements.

For instance, when the DEP describes "requirement A'" negatively impacts requirement B'", it means that relaxing requirement A' will decrease the system's ability to satisfy requirement B'. However, this DEP uncertainty factor does not describe how significant the impact of relaxing a requirement will be for the satisfaction of other requirements. Thus, in this chapter, we propose linear regression and Pearson correlation to fill this need and allow us to assess the relationship strength between requirements in the adaptive system.

In this chapter, we recommend an approach to capturing relationship strength. First, we introduce the concept of linear regression and correlation. Then, we explain our method for assessing relationship strength between two relax requirements. Two case studies are utilized to describe how to use our approach to analyse the relationship strength between two requirements. Finally, we discuss the result of correlation analysis compared to the DEP uncertainty factor.

6.2. Linear Regression and Correlation

Linear regression is a method used to estimate the relationship between variables by describing it on the slope of the regression line (Montgomery et al., 2012). The linear regression model of independent variable x_i and dependent variable y_i for $i = 1, \dots, n$ is shown in Equation 6-1. The intercept α is where the regression line cross *y-axis*, and β is the slope showing how steep the line is, and ε_i is a random error component.

$$y_i = \alpha + \beta x_i + \varepsilon_i$$ Equation 6-1

A correlation coefficient is a way to describe the strength of the relationship between two continuous variables (Bobko, 2001; Zou et al., 2003). The value is between -1 and +1. A negative correlation means that each increase in one variable will decrease the other, 0 means both variables are not correlated, and a positive value means that every increase in one variable will increase the other correlated variable. In other words, the closer the correlation coefficient value is to -1 or +1, the stronger the relationship between the two variables, either negative or positive.

Pearson correlation coefficient (*r*) can be employed to measure the linear relationship between two random variables (Pearson, 1896). It can be computed on sample data. Suppose we have *n* dataset with the first sample dataset $(x_1, \dots, x_n)$ and the second sample dataset $(y_1, \dots, y_n)$, the Pearson correlation coefficient *r* can be calculated using Equation 6-2.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$ Equation 6-2

In Equation 6-2, $\bar{x}$ and $\bar{y}$ are the mean values of x_i and y_i, respectively, and can be calculated using Equation 6-3.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$ Equation 6-3

After calculating the correlation coefficient, we need to assess the result. Cohen's effect size is widely used to interpret the result of Pearson correlation *r* (Cohen, 1992). The guidelines for using effect size to evaluate relationship strength is described in Table 6-1. The small size of an effect implies that there is a relationship but seeing it would require meticulous study. In contrast, the impact of a substantial relationship would be visible to the naked eye, as it were.

Table 6-1. Effect Size Guide for Pearson *R*

Pearson *r*	Effect size
0.1	Small
0.3	Medium
0.5	Large

6.3. The Approach to Capture Relationship Strength

We explain the approach to capturing the relationship strength in this section. Figure 6-1 presents a flowchart of how we obtain relationship strength. The input of this approach is the relax requirements. We give an overview of each step in detail as follows.

1. **UPPAAL modelling and simulation**

   The system is modelled and simulated using the UPPAAL model checker. The simulation is conducted with defined random variables. The normally distributed random number generated by UPPAAL is used in the simulation.

2. **Translate the result into fuzzy satisfaction**

   The results of the simulation are then translated into fuzzy requirement satisfaction using Equation 3-4.

3. **Plot simulation result**

   The next step is to plot the simulation results on a scatter plot. A scatter plot shows the relationship, if any, between two sets of data. After the data have been plotted, we can obtain the regression line.

4. **Linear regression and correlation analysis**

   In this step, we obtain the regression equation and the Pearson coefficient. Linear regression is used to predict the relationship between a dependent variable and an independent variable. Independent variables are the output when we change the

dependent variable (controlled inputs) value. By contrast, the Pearson correlation coefficient is used to measure the linear correlation between two variables.

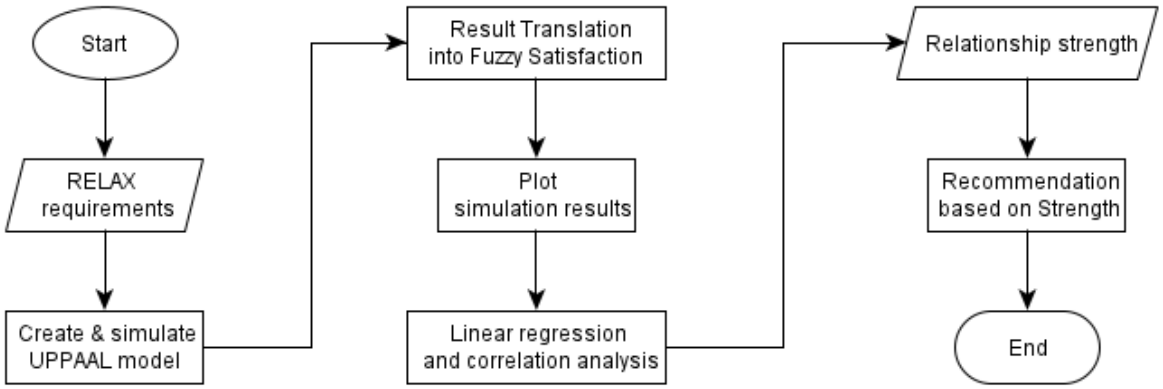


Figure 6-1. The approach to capturing relationship strength

The output of our approach is the relationship strength. We use the effect size to assess the relationship strength between two requirements based on the value of the Pearson correlation coefficient. The guide to the effect size is in Table 6-1. The stronger the relationship between two requirements, the more significant the impact of the increase/decrease of one requirement on another requirement.

In some cases, the requirement engineer may be unable to define the relationship between requirements. The result of relationship analysis is hypothetically can be used to define the unknown potential relationship between one requirement to another. Then eventually, we can use the relationship strength analysis result as a recommendation to the requirement engineer on the impact of relaxation. For example, when the relationship is strong, the requirement engineer has to take into account the relationship as relaxing one requirement can impact significantly on the satisfaction of another requirement, either positively or negatively. On the other hand, for the weak relationship strength, the requirement engineer can just ignore the

relationship while doing the relaxation as this process has nearly no impact on another requirement satisfaction.

6.4. Case Studies

6.4.1 Case study 1: Smart Vacuum Systems

In this section, we utilize the smart vacuum systems case study that has been used in Chapter 4. However, we make several modifications to the problem. Here, we will observe the relationship between the clean area and the remaining battery. The room size is 50 units. Some of the areas in the house have more dirt than the others. The robot vacuum needs different amounts of energy to clean a particular area, depending on how dirty it is. (The energy may vary from 1-3 battery units, and we set the robot vacuum randomly to simulate different levels of dirtiness). The robot has a limited battery (100 battery units). It is able to charge itself by moving from its original place to the charging point when a certain minimum number of battery units are left. (20 battery units are needed to travel from the farthest area to the charging station).

Based on this description, we create the relax requirements, as shown in Table 6-2. The DEP uncertainty factor indicates that there is a dependency between S1' and S3'. However, it does not explicitly describe how strong the impact is. Thus, in this approach, we will analyze how strong the relationship between these two requirements is by employing linear regression and correlation approaches.

Table 6-2. SVS Relax Requirement

S1': SVS SHALL achieve AS MANY clean AS POSSIBLE
S3': The system SHALL have remaining battery AS MANY AS POSSIBLE
DEP: S3' positively impacts S1'

Next, we model and simulate the problem in the UPPAAL model checker. The model is shown in Figure 6-2, and the local declarations are in Table 6-3. To model a different amount of dirt, we use normally distributed random numbers [1,3] in our simulation. The scatter plot, as shown in Figure 6-3, illustrates the simulation results (blue dots) and the regression line (red line).

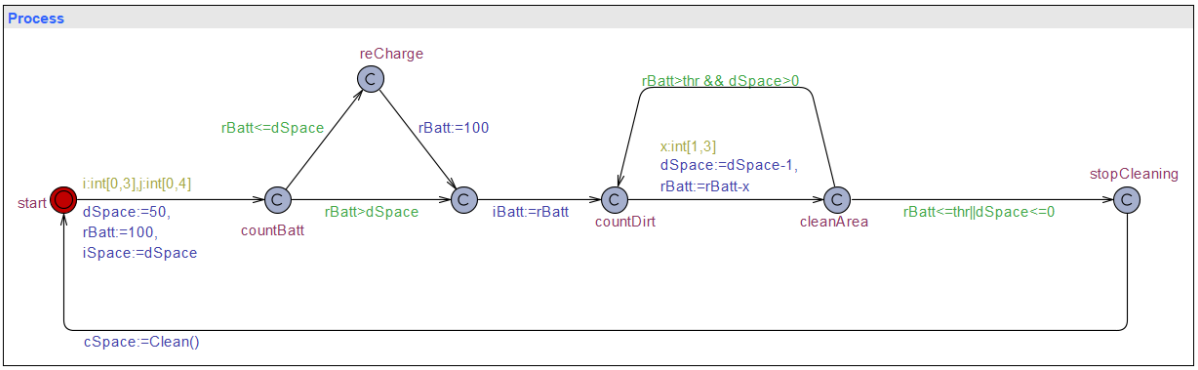


Figure 6-2. SVS model

Table 6-3. Local declarations on the SVS model

```
// Place local declarations here.
int rBatt,dSpace, iBatt, iSpace;
double cSpace;
int thr:=7;
int sChoice[4]:={20, 40, 60, 80};
int bPower[5]:={20, 40, 60, 80, 100};

double Clean(){
    double percentClean;
    percentClean:=((iSpace-dSpace)/iSpace)*100;
    return percentClean;
}
```

After this, we obtain the equation of the regression line (Equation 6-4). The linear regression equation shows that at $x = 0$, the value of $y = 0.9505$ and the slope indicates a negative relationship where every unit increase in x will decrease y by average 0.8568. Using Cohen's effect size in Table 6-1, the value of the Pearson correlation coefficient (Equation 6-5) shows that S3 *'has a large negative impact on S1'* meaning the higher the fuzzy satisfaction of the remaining battery, the lower the fuzzy satisfaction of clean space.

$$y = -0.8586x + 0.9505$$ Equation 6-4

$$r = -0.706003265$$ Equation 6-5

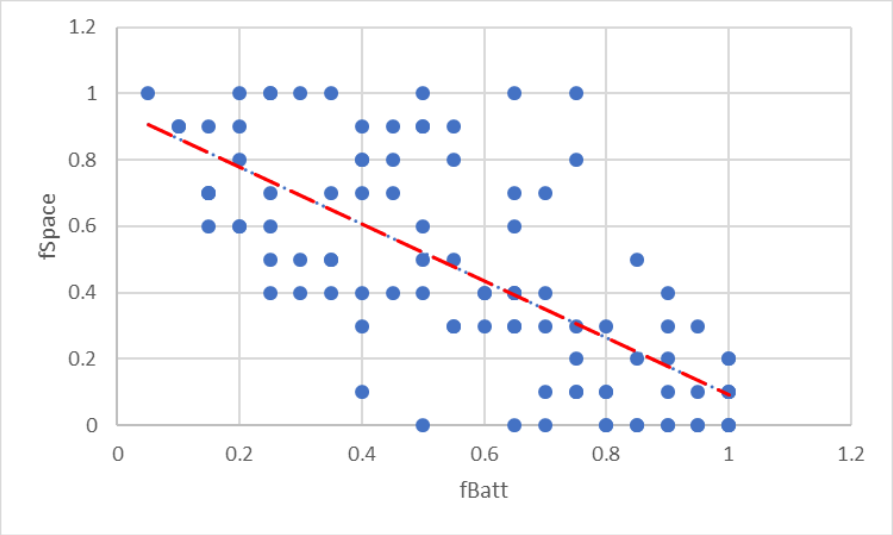


Figure 6-3. Simulation results in the scatter plot

6.4.2 Case study 2: Ambient Assisted Living System

Using the case study of ambient assisted living system in Chapter 3, we apply our method to measure the relationship strength between two requirements. We refer to the relax requirements and its DEP uncertainty factor in Table 3-4. Then, we model and simulate the system in the UPPAAL model checker.

Figure 6-4 is the model of requirement A1: foodInfo. This model will simulate the fridge in detecting food information. We use probabilistic branching and assign the weight to 49:1 to simulate whether the food sensor can extract the food information or not. The success of the system depends on how much food information (food calorie in this case) is detected. The local declaration in the foodInfo template is shown in Table 6-4.

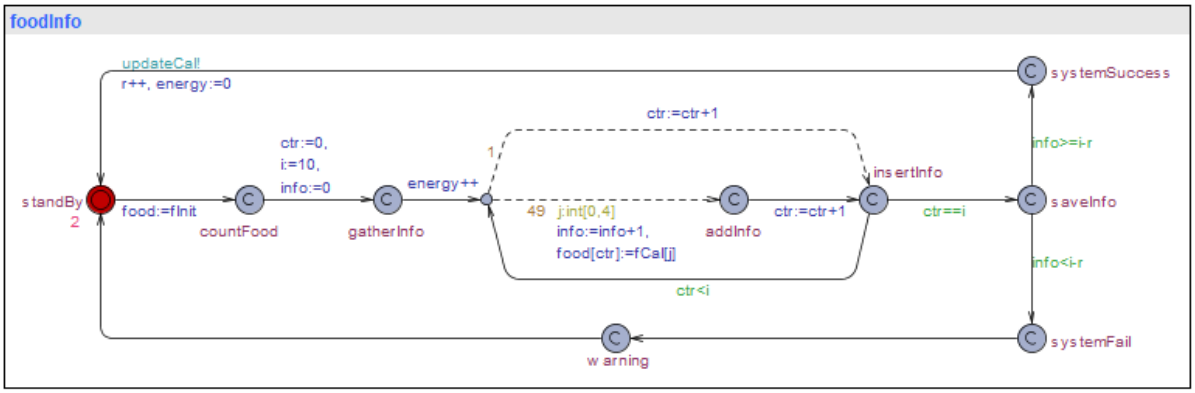


Figure 6-4. The model of requirement A1 foodInfo

Table 6-4. Declarations in foodInfo template

```
int info, ctr, i;
int fInit[10];
int fCal[5]:={100, 200, 300, 400, 500};
int r:=5;   //relax variable
```

Figure 6-5, the model of requirement 2: calorieIntake, simulates the number of calories consumed. We assume there are three mealtimes and two snack times, with a meal alarm set to sound if Mary has not taken her meal. Then the system calculates the calories consumed and how many calories need to be taken in at the next meal. The AAL system measures all the consumed calories, and each day determines whether the diet is a success or a failure. The variable declarations and functions used in modelling the calorie intake are shown in Table 6-5.

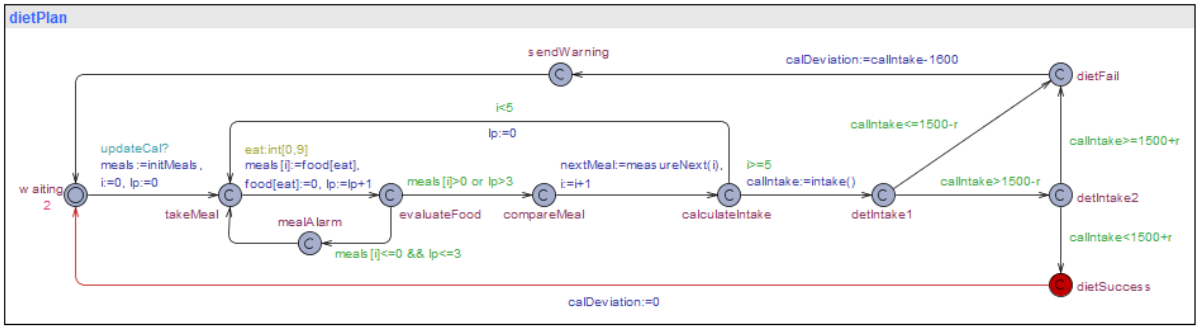


Figure 6-5. The model of requirement A2: totalCalories

Table 6-5. Declarations in calorieIntake template

```
int initMeals[5];
int idealMeals[5]:={400, 200, 400, 200, 400};
int i, previousMeal, lp, r:=500;

int measureNext(int j){
    if(j==0) previousMeal:=400;
    else{
        previousMeal:=nextMeal;
    }
    if(j<4){
        nextMeal:=idealMeals[j+1]+previousMeal-meals[j];
    }else nextMeal:=0;
    return nextMeal;
}

int intake(){
    int k:=0;
    calIntake:=0;
    for(k=0;k<5;k++){
        calIntake:=calIntake+meals[k];
    }
    return calIntake;
}
```

Figure 6-6 is the model of requirement A3: liquidIntake. Each time Mary spills the liquid in her cup, the system will detect whether she uses the cup to water the flowers, is discarding it in the sink, or is just drinking from it. The branching probabilistic weight of 95:4:1 is used to simulate drinking, discarding water in the sink, or watering flowers each day. Table 6-6 declares the variables and functions used to calculate water intake.

Figure 6-7 is the model of requirement A4: urinaryFreq. The system will determine whether Mary's frequency of urination is still normal or not. The variable declarations are shown in Table 6-7. Furthermore, the model of requirement A5: medTaken is shown in Figure 6-8. After mealtime, the medication alarm is activated every 10 minutes for an hour until Mary

takes her medicine. This requirement is considered to be satisfied if Mary takes medicine not later than the relaxed time. The variable declarations are in Table 6-8.

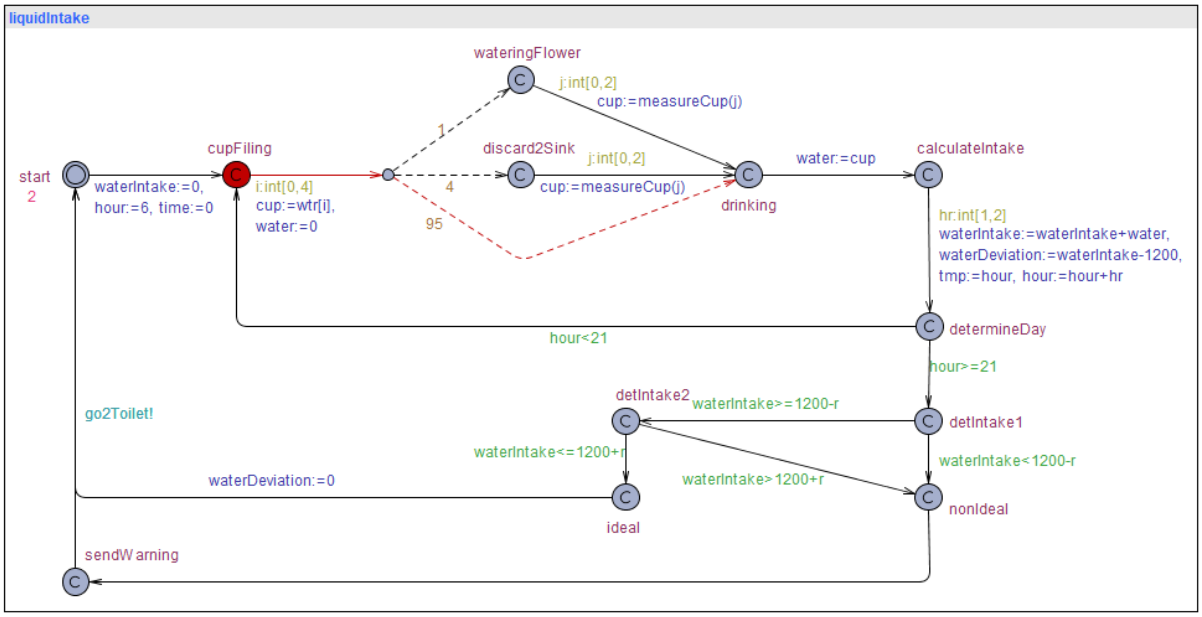


Figure 6-6. The model of A3 liquidIntake

Table 6-6. Declarations in liquidIntake template

```
int wtr[5]:={100,150,200,250,300};
int dsChoice[3]:={100,150,200};
int dsc;
int time, water, cup, hour:=6, tmp;
int r:=300;  //relax variable

int measureCup(int j){
    int cupIntake;
    dsc:=dsChoice[j];
    if(cup>=dsc) cupIntake:=cup-dsc;
    else cupIntake:=0;
    return cupIntake;
}
```

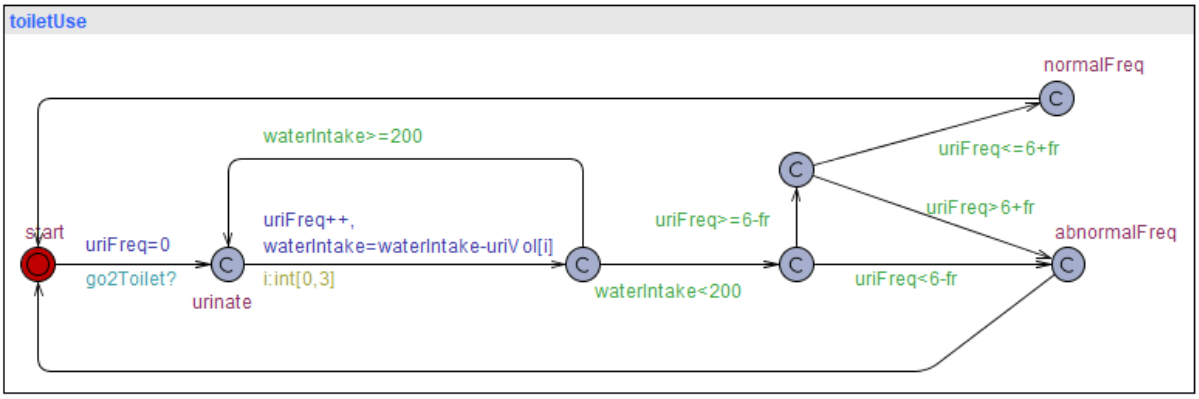


Figure 6-7. The model of A4. urinaryFrequency

Table 6-7. Declarations of urinaryFrequency

```
int uriVol[4]:={100, 150, 200, 250};
int fr=2;  //relax variable
```

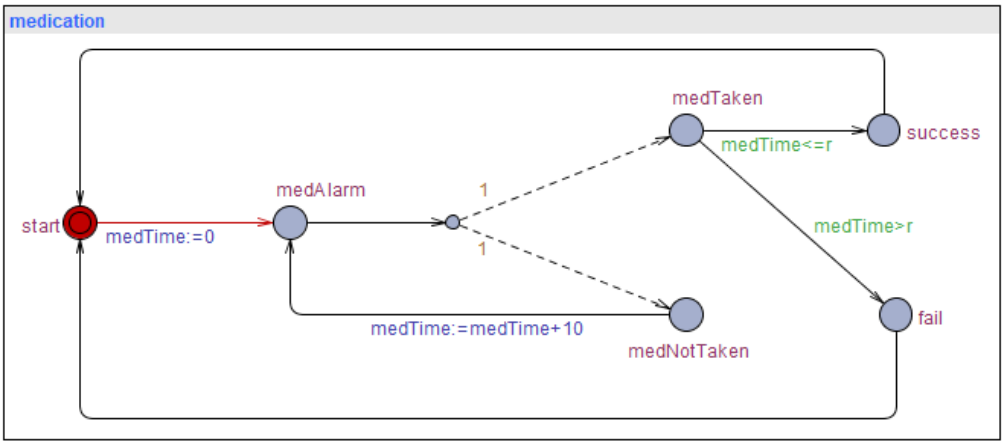


Figure 6-8. The model of A5: medTaken

Table 6-8. Declarations of medTaken

```
int medTime;
int r=60;  //relax variable
```

6.4.2.1 *Problem 1: A1' foodInfo vs A2' calorieIntake*

The summary of requirements A1' and A2' are in Table 6-9. After simulating the model, we translate the result into fuzzy satisfaction and present it in a scatter plot (see Figure 6-9). The red line in the plot indicates the linear regression line between both requirements.

Table 6-9. The requirements A1' and A2'

A1': The fridge SHALL detect AS MANY AS POSSIBLE foodInfo
A2': The system SHALL ensure Mary's calorieIntake AS CLOSE AS POSSIBLE TO the daily idealCalories
DEP: A1 negatively impacts A2

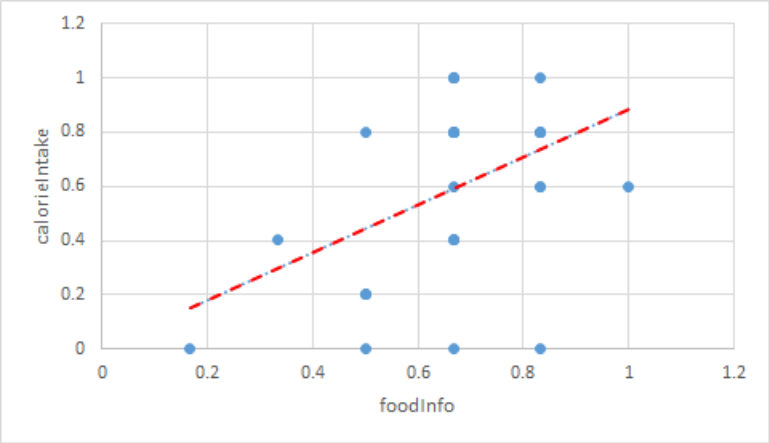


Figure 6-9. Fuzzy satisfaction foodDetected vs. calorieIntake

The regression equation in Equation 6-6 shows the positive slope, which means every unit increase in x will increase y by average 0.8794 points. Thus, we can say that the requirements have a positive relationship. Next, we calculate the Pearson correlation coefficient and use the Cohen effect size to determine the relationship strength between A1' and A2'. The correlation result in Equation 6-7 shows that both requirements have a medium positive relationship strength meaning every increase in the satisfaction of foodInfo will also increase the satisfaction of calorieIntake.

$$y = 0.8794x + 0.008 \quad \text{Equation 6-6}$$

$$r = 0.453217 \quad \text{Equation 6-7}$$

6.4.2.2 Problem 2: A3' liquidIntake vs. A4' urinaryFrequency

Table 6-10. The requirements of A3' and A4'

A3': The system SHALL ensure the liquidIntake AS CLOSE AS POSSIBLE TO idealIntake
A4': The system SHALL monitor Mary's urinaryFreq AS CLOSE AS POSSIBLE TO normalFreq
DEP: A3 impacts A4 positively and negatively

Table 6-10 summarizes the requirements for our case. This time, we use it to analyze the result of a positive and negative relationship. The fuzzy satisfaction of A3': liquidIntake vs. A4': urinaryFrequency in scatter plot is displayed in Figure 6-10. The regression line (yellow line) is flat.

The regression line equation (see Equation 6-8) exhibits a tiny negative slope. It means that the change in liquidIntake has nearly no effect on the satisfaction of urinaryFrequency. The Pearson correlation coefficient (see Equation 6-9) is also very small. Based on the Cohen effect size, it indicates that there is no correlation between A3' and A4'.

$$y = -0.0045x + 0.6241 \qquad \text{Equation 6-8}$$

$$r = -0.004683644 \qquad \text{Equation 6-9}$$

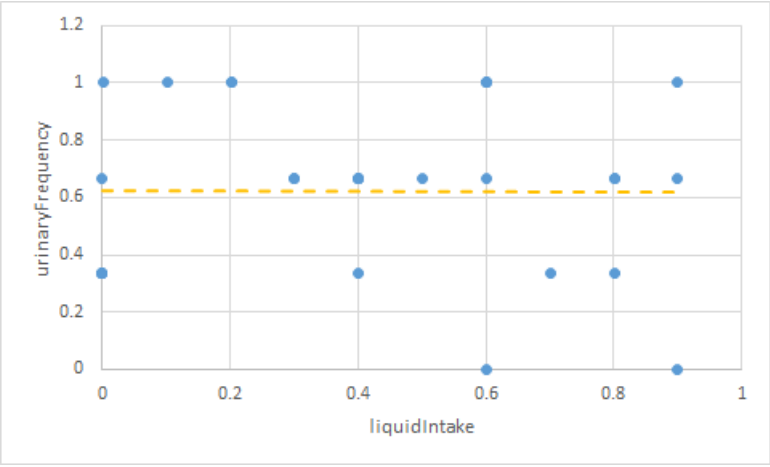


Figure 6-10. Fuzzy requirement satisfaction of A3': liquidIntake vs. A4': urinaryFrequency

6.4.2.3 *Problem 3: A4' urinaryFrequency vs. A5': medTaken*

Table 6-11. The requirement of A4' and A5'

A4': The system SHALL monitor Mary's urinaryFreq AS CLOSE AS POSSIBLE TO normalFreq
A5': The system SHALL ensure medTaken AS EARLY AS POSSIBLE of the schedule
DEP unknown

Table 6-11 summarizes requirement A4': urinaryFrequency and A5': medTaken with no DEP uncertainty factor declared. Figure 6-11 shows the fuzzy satisfaction result of both requirements (blue dots) and its linear regression line (green line). The slope of the regression line, as shown in Equation 6-10, indicates no relationship because the slope is too small to be considered as correlating. The Pearson correlation coefficient (see Equation 6-11) amplifies this statement.

$$y = -0.0182x + 0.8528 \qquad \text{Equation 6-10}$$

$$r = -0.026679706 \qquad \text{Equation 6-11}$$

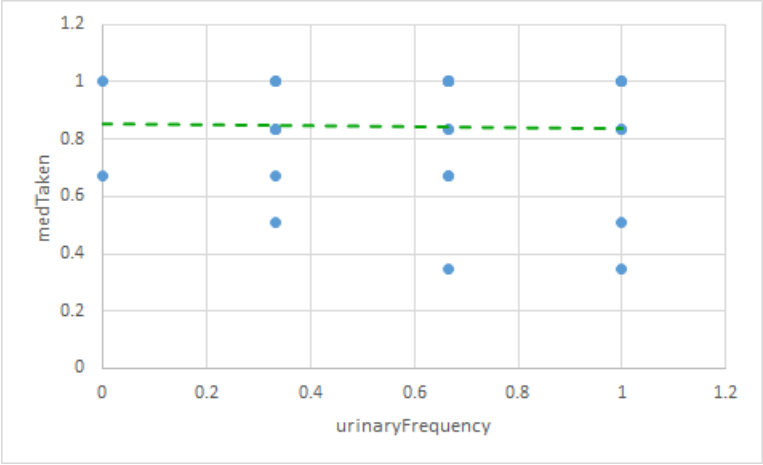


Figure 6-11. Fuzzy requirement satisfaction of A4': urinaryFrequency vs. A5': medTaken

6.5. Discussion

6.5.1 Correlation vs DEP uncertainty factor

The concepts of linear regression and correlation express how a positive relationship between two variables means that an increase in one variable will raise the value of another variable. On the contrary, the DEP uncertainty factor states that a positive impact means that relaxing one variable (tolerating a decrease to fuzzy requirement satisfaction) will increase the satisfaction of another requirement. Thus, we can conclude that the two concepts have opposite effects.

The two case studies we have undertaken in the previous sections have confirmed this statement. In the first case (the SVS case study), the DEP uncertainty factor states that S3' positively impacts S1'. However, the correlation analysis shows the opposite result, a large negative correlation. The DEP uncertainty factor in Problem 1 in the second case (the AAL case study) states that the two requirements have a negative relationship. Nonetheless, the correlation result shows a medium positive correlation instead. These case studies prove that linear regression and correlation theory has the opposite concept to the DEP uncertainty factor.

Interestingly, in Problem 2 in the AAL case study, when the DEP uncertainty factor expresses a positive and a negative impact at the same time, the correlation analysis reveals no correlation between the requirements. It might be because the operator AS CLOSE AS POSSIBLE makes the relaxation move in both positive and negative directions so that the correlation result turns out to be no correlation. We need, therefore, to explore this problem in depth in further study.

In addition, we can also use linear regression and correlation to capture the possible relationship between two requirements when there is no DEP uncertainty factor declared. The last problem in the second case study is an example of how to capture an unknown relationship

between the two requirements. Although the example delivers a no correlation result, it is logically true because it makes sense that there is no relationship between urinary frequency and the time when the medicine is taken.

Based on the concepts above and the examples in the case studies, we can conclude that the correlation concept has the opposite result to the uncertainty factor DEP. Consequently, the following insights can be expressed:

- When DEP states that requirement A impacts requirement B negatively, the correlation between the two requirements is positive. Therefore, relaxing one requirement will decrease the satisfaction of the other related requirement.
- On the other hand, when the DEP uncertainty factor states that requirement A impacts requirement B positively, then their correlation will be negative. It means that relaxing one requirement will increase the satisfaction of the other requirement.
- If the DEP uncertainty factor declares a positive and negative relationship at the same time, the correlation analysis will generate no correlation between the requirements.
- In the case of no relationship being stated, correlation analysis can be used to obtain the possible relationship between two requirements using the simulation data.

Our findings in this chapter can be used to formulate a recommendation to the requirement engineer to help them consider the impact of relaxing a requirement on the satisfaction of a related requirement. As for unrelated requirements, hypothetically, this approach can capture an unknown relationship between the two requirements. It can be used as an indication of the potential relationship between the two requirements to help the requirement engineer notice this relationship.

6.5.2 Related works

Using linear regression and correlation analysis is relatively simpler to support a requirement engineer in deciding on whether a relaxation has to be conducted or not based on the effect size of the requirement relationship. Thus, they can decide earlier during the design phase.

Some other works have been conducted to support decision making in the adaptive systems. To assist the decision making of self-adaptation, Bencomo et.al proposes Dynamic Decision Networks (DDN) adopting probabilistic Bayesian Networks (Bencomo, Belaggoun, & Issarny, 2013). This approach works during design time and runtime with different sources of uncertainty. It also supports reasoning partial satisfaction using probabilities (Bencomo & Belaggoun, 2013).

Another research used a requirement-aware model (RaM) combined with Partially Observable Markov Decision Processes (POMDPs) to support the decision on self-adaptation (Bencomo & Paucar, 2019). The approach was applied to reason the partial satisfaction in non-functional requirements.

6.6. Summary

In this chapter, we introduce linear regression and correlation to assess the relationship strength between two requirements in the adaptive system. We use two case studies (smart vacuum systems and ambient assisted living systems) to illustrate how to capture relationship strength. We employ the UPPAAL model checker to simulate the system, then translate the result into fuzzy satisfaction and analyse its relationship strength.

In addition, we compare the result of correlation analysis with the DEP uncertainty factor in the RELAX language. We find that the two concepts have the opposite results. In other

words, when DEP states the positive impact on one requirement of relaxing another requirement, correlation analysis will indicate a negative relationship or vice versa. Our proposed approach, hypothetically, can also be used to detect whether there is a potential relationship between two requirements when no relationship is declared.

Chapter 7

Conclusions and Future Works

This chapter summarizes the works carried out in this thesis. Section 7.1 reviews the result of our research on flexible requirement satisfaction in adaptive systems. Suggestions for further research direction are discussed in section 7.2.

7.1. Conclusions

Chapter 1: Introduction

This opening chapter discusses the reasons for our research. The adaptive system is facing several challenges, one of which is caused by adaptation and uncertainty. It is nearly impossible to enumerate all conditions which could arise in order to prepare adaptations for them, and this results in more uncertainties. RELAX requirement language facilitates adaptation using temporal/ordinal operator and capture uncertainties using uncertainty factors. Though it has a DEP uncertainty factor to indicate a relationship between requirements, it does not show the effect of relaxing one requirement to another. Thus we are interested in exploring this relationship deeper in this research.

We also discuss the objective of our research, which is to provide a flexible representation of requirement satisfaction. Therefore, we introduce a fuzzy requirement satisfaction. Furthermore, we propose a systematic approach to requirement relaxation in linear and nonlinear cases by considering the relationship between requirements. In addition, we use correlation and linear regression to illustrate relationship strength. The UPPAAL model checker is utilized to perform verification to the result of the systematic approach.

Chapter 2: Literature Review

In this chapter, we provide a background for our work. First, we present the fuzzy set theory as a more flexible way of representing information compared to crisp Boolean representation. Then, the definition of adaptive systems and the challenges faced by such systems are reviewed. Next, we discuss requirement engineering, especially for the adaptive system. We also introduce the RELAX specification language that is incorporated in this research. Lastly, we present model checking tools as one way to perform system verification and specifically explore the UPPAAL model checker as the verification tool used in this research.

Chapter 3: Fuzzy Requirement Satisfaction for Adaptive Systems

In this chapter, we introduce fuzzy requirement satisfaction, which is calculated by integrating a fuzzy set and RELAX specification language. We categorize the fuzzy satisfaction into three types of graphs (LEFT, RIGHT, and MIDDLE) based on the relax operator. Then, we formalize the fuzzy requirement satisfaction so that it can be used in all types of case studies.

We demonstrate our approach in an Ambient Assisted Living (AAL) System case study. The requirements are extracted from the system description. Then, we determine whether a requirement is invariant or relax. We apply the approach to the relax requirement. We employ fuzzy alpha cut to present graded requirement satisfaction. In this way, we not only can represent flexible requirement satisfaction but can also discover how close a particular set of requirements is to be fully satisfied.

Chapter 4: Systematic Mathematical Approach in Relaxing Related Requirements in Adaptive Systems

This chapter presents our systematic approach to relaxing requirement satisfaction, especially on related requirements. The following summaries can be extracted from Chapter 4:

- We formalize the relationship between two requirements by incorporating the implication concept into the previously known DEP uncertainty factor in RELAX requirement language
- Mathematical dependencies are introduced in this chapter. It is used to describe the relationship between requirements in the adaptive system mathematically.
- We propose optimal relaxation (*Rel*), which is the supremum of fuzzy requirement satisfaction that satisfies all related requirements and dependencies. The optimal relaxation (*Rel*) can then be used to recommend the new requirement thresholds to the requirement engineer
- We use reachability properties $E<>\varphi$ in the UPPAAL model checker to verify the result on the systematic approach. We classify the verification result into always satisfied, sometimes satisfied, and never satisfied.

To demonstrate the proposed approach, we use two case studies, Smart Vacuum Systems, and Distributed Database System. The result indicates that relaxing one requirement can impact the satisfaction of other related requirements. Thus, when we relax a requirement, we need to consider related requirements.

This approach can work well for relaxing requirements with linear mathematical dependencies. However, it can be more challenging to do this for a system with nonlinear mathematical dependency. Thus, we propose two additional steps for nonlinear cases in Chapter 5.

Chapter 5: Nonlinear Fuzzy Requirement Satisfaction on Adaptive System

This chapter covers the additional steps involved in handling the relaxation of requirements in adaptive systems with nonlinear mathematical dependency. Applying more relaxation to requirements in the nonlinear case does not guarantee satisfaction. Thus, we need to find the

points of intersection between the mathematical dependency and requirement to be able to determine the area/s of relaxation.

The following conclusions can be drawn from Chapter 5:

- In the nonlinear case, relaxing requirements does not assure satisfaction. At some point, the satisfaction can be violated even after the optimal relaxation is achieved.
- The relaxation area is the area where requirement satisfaction is always achieved. To determine the area/s, we need to find points of intersection. This area can be used as a suggestion to the requirement engineer/software designer to adjust requirement thresholds, so that requirement satisfaction is always guaranteed.

The approach with additional steps is applied to the stopping distance problem and the company BEP problem. The system is modelled with the UPPAAL model checker. The verification result is consistent with the systematic approach.

Chapter 6: Using Correlation to Capture Relationship Strength

Chapter 6 proposes a method to capture relationship strength by utilizing correlation and linear regression. We model the case studies in the UPPAAL model checker. The simulation is conducted with defined random variables. Then, we evaluate the result with linear regression and correlation. We can extract the following conclusions from Chapter 6:

- Linear regression is used to predict the relationship between the dependent and independent variables. The relationship strength is measured using the Pearson correlation coefficient, and the Cohen size effect is used to assess the result.
- The correlation and linear regression concept have the opposite result of the DEP uncertainty factor.

We demonstrate the approach with two case studies: a smart vacuum system and an ambient assisted living system. The result shows that the approach is hypothetically able to detect an unknown relationship between two requirements in the adaptive system.

7.2. Future Works

The investigations described in this work are related to representing flexible requirement satisfaction in an adaptive system. Based on the current result, we can suggest several further directions, as follows:

1. Our systematic relaxation works limitedly on the adaptive system with defined mathematical dependencies that describe the relationship between requirements mathematically. However, if there are no mathematical dependencies, the approach cannot be used. Further research is needed to find a method to find optimal relaxation when the dependencies are not defined.
2. In this work, the relaxation is performed with the same alpha value for all requirements because we assume that all requirements have the same importance. In future work, we may consider a different ranking on the requirement of the adaptive system, so the relaxation may also need to be adjusted based on this ranking.
3. Exploration of defining fuzzy relaxation for a categorical requirement is needed. For example, in the AAL case study, initially, Mary's diet prescribed for carbohydrates is potatoes. However, this diet can be relaxed to allow corn and later relaxed to allow her to eat rice. We can work on this issue in future research.
4. In our work, we assume that the fuzzy set has a symmetric shape at the middle type of fuzzy satisfaction. An investigation into the asymmetrical fuzzy set can become a further essential study.

5. Our approach to capturing relationship strength is currently applied to artificial data. In a further study, we could use real data from an adaptive system to see if this approach can be used to capture the relationship between two requirements.

References

Ahmad, M., Bruel, J.-M., Laleau, R., & Gnaho, C. (2012). Using RELAX, SysML and KAOS for ambient systems requirements modeling. *Procedia Computer Science, 10*, 474-481.

Al-Sayyed, R. M., Al Zaghoul, F. A., Suleiman, D., Itriq, M., Hababeh, I. J. J. o. S. E., & Applications. (2014). A new approach for database fragmentation and allocation to improve the distributed database management system performance. *7*(11), 891.

Alur, R., Courcoubetis, C., & Dill, D. (1990). *Model-checking for real-time systems.* Paper presented at the Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium on e.

Amodei, M., & Stein, J. (2009). Deterministic and fuzzy verification methods for a hierarchy of numerical models. *Meteorological Applications, 16*(2), 191-203.

Anggraini, R. N., & Martin, T. (2017). *Fuzzy Representation for Flexible Requirement Satisfaction.* Paper presented at the UK Workshop on Computational Intelligence.

Anggraini, R. N. E., & Martin, T. P. (2018, 8-10 Aug. 2018). *Capturing Requirement Correlation in Adaptive Systems.* Paper presented at the 2018 IEEE International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (Robionetics).

Baresi, L., Pasquale, L., & Spoletini, P. (2010). *Fuzzy goals for requirements-driven adaptation.* Paper presented at the 2010 18th IEEE International Requirements Engineering Conference.

Bedoya, A. E., Perez, Y. M., & Marin, H. A. P. (2016). A review on verification and validation for embedded software. *IEEE Latin America Transactions, 14*(5), 2339-2347. doi:10.1109/TLA.2016.7530431

Behrmann, G., David, A., & Larsen, K. G. (2010). A tutorial on uppaal. 2004. *Department of Computer Science, Aalborg University, Denmark*.

Bencomo, N., & Belaggoun, A. (2013). *Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks.* Paper presented at the International Working Conference on Requirements Engineering: Foundation for Software Quality.

Bencomo, N., Belaggoun, A., & Issarny, V. (2013). *Dynamic decision networks for decision-making in self-adaptive systems: a case study.* Paper presented at the 2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS).

Bencomo, N., & Paucar, L. H. G. (2019). *Ram: Causally-connected and requirements-aware runtime models using bayesian learning.* Paper presented at the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS).

Beyer, D. (2015). *Software verification and verifiable witnesses.* Paper presented at the International Conference on Tools and Algorithms for the Construction and Analysis of Systems.

Beyer, D., Henzinger, T. A., Jhala, R., & Majumdar, R. (2007). The software model checker Blast. *International Journal on Software Tools for Technology Transfer, 9*(5), 505-525. doi:10.1007/s10009-007-0044-z

Beyer, D., & Keremoglu, M. E. (2011). CPAchecker: A Tool for Configurable Software Verification. In G. Gopalakrishnan & S. Qadeer (Eds.), *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings* (pp. 184-190). Berlin, Heidelberg: Springer Berlin Heidelberg.

Bhuyar, P., Gawande, A., & Deshmukh, A. (2012). Horizontal fragmentation technique in distributed database. *International Journal of Scientific and Research Publications, 2*(5), 1-7.

Bobko, P. (2001). *Correlation and regression: Applications for industrial organizational psychology and management*: Sage.

Böschen, M., Bogusch, R., Fraga, A., & Rudat, C. (2016). *Bridging the Gap between Natural Language Requirements and Formal Specifications.* Paper presented at the REFSQ Workshops.

Botia, J. A., Villa, A., & Palma, J. (2012). Ambient Assisted Living system for in-home monitoring of healthy independent elders. *Expert Systems with Applications, 39*(9), 8136-8148. doi:https://doi.org/10.1016/j.eswa.2012.01.153

Bradley, T. (2013). *Essential mathematics for economics and business*: John Wiley & Sons.

Brun, Y., Desmarais, R., Geihs, K., Litoiu, M., Lopes, A., Shaw, M., & Smit, M. (2013). A design space for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II* (pp. 33-50): Springer.

Cámara, J., & De Lemos, R. (2012). *Evaluation of resilience in self-adaptive systems using probabilistic model-checking.* Paper presented at the 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS).

Carter, M., He, S., Whitaker, J., Rakamaric, Z., & Emmi, M. (2016). *SMACK software verification toolchain.* Paper presented at the 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C).

Charron-Bost, B., Pedone, F., & Schiper, A. J. L. (2010). Replication. *5959*, 19-40.

Chen, M., Gonzalez, S., Vasilakos, A., Cao, H., & Leung, V. C. (2011). Body area networks: A survey. *Mobile networks and applications, 16*(2), 171-193.

Cheng, B. H., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., . . . Cukic, B. (2009). Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems* (pp. 1-26): Springer.

Cheng, B. H., Eder, K. I., Gogolla, M., Grunske, L., Litoiu, M., Müller, H. A., . . . Rumpe, B. (2014). Using models at runtime to address assurance for self-adaptive systems. In *Models@ run. time* (pp. 101-136): Springer.

Cheng, B. H., Sawyer, P., Bencomo, N., & Whittle, J. (2009). *A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty.* Paper presented at the International Conference on Model Driven Engineering Languages and Systems.

Chiu, M.-C., Yeh, L.-J., & Lin, Y. (2009). The design and application of a robotic vacuum cleaner. *Journal of Information and Optimization Sciences, 30*(1), 39-62.

Clark, T. D., Larson, J. M., Mordeson, J. N., Potter, J. D., & Wierman, M. J. (2008). *Applying fuzzy mathematics to formal models in comparative politics*: Springer.

Clarke, E. M., Emerson, E. A., & Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS), 8*(2), 244-263.

Clarke, E. M., Henzinger, T. A., Veith, H., & Bloem, R. (2018). *Handbook of model checking* (Vol. 10): Springer.

Clarke, E. M., & Schlingloff, B.-H. (2001). Chapter 24 - Model Checking. In *Handbook of Automated Reasoning* (pp. 1635-1790). Amsterdam: North-Holland.

Clarke, E. M., & Wing, J. M. (1996). Formal methods: state of the art and future directions. *ACM Comput. Surv., 28*(4), 626-643. doi:10.1145/242223.242257

Cohen, J. (1992). A power primer. *Psychological bulletin, 112*(1), 155.

Corbett, J. C., Dwyer, M. B., Hatcliff, J., Laubach, S., Pasareanu, C. S., & Zheng, H. (2000). *Bandera: Extracting finite-state models from Java source code.* Paper presented at the Software Engineering, 2000. Proceedings of the 2000 International Conference on.

Cranen, S., Groote, J. F., Keiren, J. J., Stappers, F. P., de Vink, E. P., Wesselink, W., & Willemse, T. A. (2013). *An Overview of the mCRL2 Toolset and Its Recent Advances.* Paper presented at the TACAS.

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

David, A., Larsen, K. G., Legay, A., Mikučionis, M., & Poulsen, D. B. (2015). Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer, 17*(4), 397-415.

De Capua, C., & Romeo, E. (2006). *A Fuzzy Approach to represent Measurement Data affected by both A-type and B-type Uncertainty.* Paper presented at the Instrumentation and Measurement Technology Conference, 2006. IMTC 2006. Proceedings of the IEEE.

De Hoog, J., Cameron, S., & Visser, A. (2010). *Autonomous multi-robot exploration in communication-limited environments.* Paper presented at the Proceedings of the Conference on Towards Autonomous Robotic Systems.

De Jong, K. (1980). Adaptive system design: a genetic approach. *IEEE Transactions on Systems, Man, and Cybernetics, 10*(9), 566-574.

de Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., . . . Wuttke, J. (2013). Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In R. de Lemos, H. Giese, H. A. Müller, & M. Shaw (Eds.), *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers* (pp. 1-32). Berlin, Heidelberg: Springer Berlin Heidelberg.

Dethe, S. N., Shevatkar, V. S., & Bijwe, R. (2016). Google driverless car. *IJSRSET, 2*(2).

Diethers, K., & Huhn, M. (2004). Vooduu: Verification of Object-Oriented Designs Using UPPAAL. In K. Jensen & A. Podelski (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004. Proceedings* (pp. 139-143). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ebert, E. E. (2008). Fuzzy verification of high-resolution gridded forecasts: a review and proposed framework. *Meteorological applications, 15*(1), 51-64.

Echauz, J. R., & Vachtsevanos, G. J. (1995). Fuzzy grading system. *IEEE Transactions on Education, 38*(2), 158-165.

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

Esfahani, N., & Malek, S. (2013b). Uncertainty in Self-Adaptive Software Systems. In R. de Lemos, H. Giese, H. A. Müller, & M. Shaw (Eds.), *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers* (pp. 214-238). Berlin, Heidelberg: Springer Berlin Heidelberg.

Fredericks, E. M., DeVries, B., & Cheng, B. H. (2014). AutoRELAX: automatically RELAXing a goal model to address uncertainty. *Empirical Software Engineering, 19*(5), 1466-1501.

Goldrei, D. (2005). *Propositional and Predicate Calculus*: Springer.

Green, D. (2006). *A comparison of stopping distance performance for motorcycles equipped with ABS, CBS and conventional hydraulic brake systems.* Paper presented at the International Motorcycle Safety Conference.

Hájek, P., Godo, L., & Esteva, F. (2013). Fuzzy logic and probability. *arXiv preprint arXiv:1302.4953*.

Hansen, H. H., Ketema, J., Luttik, B., Mousavi, M., & van de Pol, J. (2010). Towards model checking executable UML specifications in mCRL2. *Innovations in Systems and Software Engineering, 6*(1), 83-90. doi:10.1007/s11334-009-0116-1

Heiner, M., Rohr, C., Schwarick, M., & Tovchigrechko, A. A. (2016). MARCIE's secrets of efficient model checking. In *Transactions on Petri Nets and Other Models of Concurrency XI* (pp. 286-296): Springer.

Hinton, A., Kwiatkowska, M., Norman, G., & Parker, D. (2006). PRISM: A Tool for Automatic Verification of Probabilistic Systems. In H. Hermanns & J. Palsberg (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems: 12th International Conference, TACAS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006. Proceedings* (pp. 441-444). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hu, B.-G., Gosine, R. G., Cao, L., & De Silva, C. W. (1998). Application of a fuzzy classification technique in computer grading of fish products. *IEEE Transactions on Fuzzy Systems, 6*(1), 144-152.

Iftikhar, M. U., & Weyns, D. (2016). Towards runtime statistical model checking for self-adaptive systems. *CW Reports*.

Jangra, A., Singh, G., Singh, J., Verma, R. J. I. J. o. I. T., & Management, K. (2011). Exploring testing strategies. *4*, 297-299.

Jensen, J. F., Larsen, K. G., Srba, J., & Oestergaard, L. K. (2016). Efficient model-checking of weighted CTL with upper-bound constraints. *International Journal on Software Tools for Technology Transfer, 18*(4), 409-426.

Jureta, I. J., Borgida, A., Ernst, N. A., & Mylopoulos, J. (2015). The requirements problem for adaptive systems. *ACM Transactions on Management Information Systems (TMIS), 5*(3), 17.

Kampf, R., Majerčák, P., & Švagr, P. (2016). Application of break-even point analysis. *NAŠE MORE: znanstveno-stručni časopis za more i pomorstvo, 63*(3 Special Issue), 126-128.

Kleinberger, T., Becker, M., Ras, E., Holzinger, A., & Müller, P. (2007). *Ambient Intelligence in Assisted Living: Enable Elderly People to Handle Future Interfaces*, Berlin, Heidelberg.

Klir, G., & Yuan, B. (1995). *Fuzzy sets and fuzzy logic* (Vol. 4): Prentice hall New Jersey.

Kordani, A. A., Rahmani, O., Nasiri, A. S. A., & Boroomandrad, S. M. (2018). Effect of adverse weather conditions on vehicle braking distance of highways. *Civil Eng. J, 4*(1), 46-57.

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of Probabilistic Real-Time Systems. In G. Gopalakrishnan & S. Qadeer (Eds.), *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings* (pp. 585-591). Berlin, Heidelberg: Springer Berlin Heidelberg.

Larsen, K. G., & Legay, A. (2016). *Statistical model checking: past, present, and future.* Paper presented at the International Symposium on Leveraging Applications of Formal Methods.

Law, C.-K. (1996). Using fuzzy numbers in educational grading system. *Fuzzy sets and systems, 83*(3), 311-323.

Lawson, D., Tabor, J. J. T. M., & IMA, I. A. I. J. o. t. (2001). Stopping distances: an excellent example of empirical modelling. *20*(2), 66-74.

Leite, D., Costa, P., & Gomide, F. (2010). *Granular approach for evolving system modeling.* Paper presented at the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems.

Lime, D., & Roux, O. (2009). Formal verification of real-time systems with preemptive scheduling. *Real-Time Systems, 41*(2), 118-151. doi:10.1007/s11241-008-9059-0

Lime, D., Roux, O. H., Seidner, C., & Traonouez, L.-M. (2009). Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches. In S. Kowalewski & A. Philippou (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems: 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings* (pp. 54-57). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lyubenov, D. (2011). Research of the stopping distance for different road conditions. *Transport problems, 6*, 119-126.

Martin, T. (2015). The x-mu representation of fuzzy sets. *Soft Computing, 19*(6), 1497-1509.

Martin, T., & Azvine, B. (2017). *Graded concepts and associations.* Paper presented at the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE).

Martin, T. P., & Anggraini, R. N. (2019). *A Graded Approach to Requirement Satisfaction for Evolving Systems.* Paper presented at the 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE).

Mazilu, M. C. (2010). Database replication. *Database Systems Journal, 1*(2), 33-38.

Memon, M., Wagner, S. R., Pedersen, C. F., Beevi, F. H. A., & Hansen, F. O. (2014). Ambient assisted living healthcare frameworks, platforms, standards, and quality attributes. *Sensors, 14*(3), 4312-4341.

Milanés, V., Llorca, D. F., Vinagre, B. M., González, C., & Sotelo, M. A. (2010). *Clavileño: Evolution of an autonomous car.* Paper presented at the 13th International IEEE Conference on Intelligent Transportation Systems.

Moiz, S. A., Sailaja, P., Venkataswamy, G., & Pal, S. N. J. I. J. o. C. A. (2011). Database replication: A survey of open source and commercial tools. *13*(6), 1-8.

Mokadem, H. B., Berard, B., Gourcuff, V., De Smet, O., & Roussel, J.-M. (2010). Verification of a timed multitask system with UPPAAL. *IEEE Transactions on Automation Science and Engineering, 7*(4), 921-932.

Mong, W. S. (2004). Lazy abstraction on software model checking.

Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to linear regression analysis* (Vol. 821): John Wiley & Sons.

Moon, S.-i., Lee, K. H., & Lee, D. (2004). Fuzzy branching temporal logic. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34*(2), 1045-1055.

Morse, J., Araiza-Illan, D., Eder, K., Lawry, J., & Richards, A. (2017). *A fuzzy approach to qualification in design exploration for autonomous robots and systems.* Paper presented at the Fuzzy Systems (FUZZ-IEEE), 2017 IEEE International Conference on.

Muela, G., Kreinovich, V., & Servin, C. (2017). *Scaling-invariant description of dependence between fuzzy variables: Towards a fuzzy version of copulas.* Paper presented at the Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS), 2017 Joint 17th World Congress of International.

Narendra, K. S., & Annaswamy, A. M. (2012). *Stable adaptive systems*: Courier Corporation.

O'Keefe, R. M., & O'Leary, D. E. (1993). Expert system verification and validation: a survey and tutorial. *Artificial Intelligence Review, 7*(1), 3-42.

Olsthoorn, J., Tedford, E. W., & Lawrence, G. A. (2020). The Cooling Box Problem: Convection with a quadratic equation of state. *arXiv preprint arXiv:2003.01761*.

Omid, M., Soltani, M., Dehrouyeh, M. H., Mohtasebi, S. S., & Ahmadi, H. (2013). An expert egg grading system based on machine vision and artificial intelligence techniques. *Journal of food engineering, 118*(1), 70-77.

Özsu, M. T., & Valduriez, P. (1999). *Principles of distributed database systems* (Vol. 2): Springer.

Özsu, M. T., & Valduriez, P. (2011). *Principles of distributed database systems*: Springer Science & Business Media.

Pandey, D., Suman, U., & Ramani, A. (2010). *An effective requirement engineering process model for software development and requirements management.* Paper presented at the 2010 International Conference on Advances in Recent Technologies in Communication and Computing.

Pawlowski, A., Guzman, J., Rodríguez, F., Berenguel, M., Sánchez, J., & Dormido, S. J. S. (2009). Simulation of greenhouse climate monitoring and control with wireless sensor network and event-based control. *9*(1), 232-252.

Pearson, K. (1896). Mathematical contributions to the theory of evolution. III. Regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character, 187*, 253-318.

Prassler, E., Ritter, A., Schaeffer, C., & Fiorini, P. J. A. R. (2000). A short history of cleaning robots. *9*(3), 211-226.

Punnoose, R. J., Armstrong, R. C., Wong, M. H., & Jackson, M. (2014). *Survey of Existing Tools for Formal Verification* (SAND2014-20533; Other: 551829 United States 10.2172/1166644 Other: 551829 SNL English). Retrieved from http://www.osti.gov/scitech/servlets/purl/1166644

Ramesh, B., Cao, L., & Baskerville, R. (2010). Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal, 20*(5), 449-480.

Razo-Zapata, I. S., De Leenheer, P., Gordijn, J., & Akkermans, H. (2012). *Fuzzy Verification of Service Value Networks.* Paper presented at the CAiSE.

Reuys, A., Kamsties, E., Pohl, K., & Reis, S. (2005). *Model-based system testing of software product families.* Paper presented at the International Conference on Advanced Information Systems Engineering.

Robinson, W. N. (2006). A requirements monitoring framework for enterprise systems. *Requirements engineering, 11*(1), 17-41.

Sassi, I. B., Yahia, S. B., & Mellouli, S. (2017). *Fuzzy classification-based emotional context recognition from online social networks messages.* Paper presented at the Fuzzy Systems (FUZZ-IEEE), 2017 IEEE International Conference on.

Sawyer, P., Bencomo, N., Whittle, J., Letier, E., & Finkelstein, A. (2010). *Requirements-aware systems: A research agenda for re for self-adaptive systems.* Paper presented at the 2010 18th IEEE International Requirements Engineering Conference.

Schäfer, S., Schneider, S., & Smolka, G. (2016). *Axiomatic semantics for compiler verification.* Paper presented at the Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs.

Schmerl, B., Andersson, J., Vogel, T., Cohen, M. B., Rubira, C. M., Brun, Y., . . . Baresi, L. (2017). Challenges in composing and decomposing assurances for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems III. Assurances* (pp. 64-89): Springer.

Seelan, S. K., Laguette, S., Casady, G. M., & Seielstad, G. A. (2003). Remote sensing applications for precision agriculture: A learning community approach. *Remote sensing of environment, 88*(1-2), 157-169.

Shah, U. S., & Jinwala, D. C. (2015). Resolving ambiguities in natural language software requirements: a comprehensive survey. *ACM SIGSOFT Software Engineering Notes, 40*(5), 1-7.

Shen, V. R. (2006). Knowledge representation using high-level fuzzy Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 36*(6), 1220-1227.

Sickert, S., & Křetínský, J. (2016). *Mochiba: Probabilistic LTL model checking using limit-deterministic Büchi automata.* Paper presented at the International Symposium on Automated Technology for Verification and Analysis.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (1997). *Database system concepts* (Vol. 5): McGraw-Hill New York.

Singh, S., & Dey, L. (2005). A rough-fuzzy document grading system for customized text information retrieval. *Information processing & management, 41*(2), 195-216.

Singh, T., Sandhu, P. S., Bhatti, H. S. J. I. J. o. C., & Engineering, C. (2013). Replication of Data in Database Systems for Backup and Failover-An Overview. *2*(4), 535.

Singhal, R., Bokare, S., & Pawar, P. (2010). *Design and Implementation of efficient semi-synchronous replication solution for disaster recovery.* Paper presented at the on-line), Recent Advances in Software Engineering, Parallel and Distributed Systems (retrieved from http://www. wseas. us/e-library/conferences/2010/Cambridge/SEPADS/SEPADS-18. pdf).

Sleit, A., AlMobaideen, W., Al-Areqi, S., & Yahya, A. (2007). A dynamic object fragmentation and replication algorithm in distributed database systems. *American Journal of Applied Sciences, 4*(8), 613-618.

Smarr, C.-A., Fausset, C. B., & Rogers, W. A. (2011). *Understanding the potential for robot assistance for older adults in the home environment*. Retrieved from

Soliman, D., Thramboulidis, K., & Frey, G. (2012). Transformation of function block diagrams to UPPAAL timed automata for the verification of safety applications. *Annual Reviews in Control, 36*(2), 338-345.

Sommerville, I. (2011). Software engineering 9th Edition. *ISBN-10, 137035152*.

Souza, V. E. S., Lapouchnian, A., Robinson, W. N., & Mylopoulos, J. (2011). *Awareness requirements for adaptive systems*. Paper presented at the Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Waikiki, Honolulu, HI, USA.

Sun, R., Zhuang, X., Wu, C., Zhao, G., & Zhang, K. (2015). The estimation of vehicle speed and stopping distance by pedestrians crossing streets in a naturalistic traffic environment. *Transportation Research Part F: Traffic Psychology and Behaviour, 30*, 97-106. doi:https://doi.org/10.1016/j.trf.2015.02.002

Suresh, Y. (2015, 29-31 Oct. 2015). *Software quality assurance for object-oriented systems using meta-heuristic search techniques.* Paper presented at the 2015 International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT).

Systems and software engineering -- Vocabulary. (2010). *ISO/IEC/IEEE 24765:2010(E)*, 1-418. doi:10.1109/IEEESTD.2010.5733835

Tamura, G., Villegas, N. M., Müller, H. A., Sousa, J. P., Becker, B., Karsai, G., . . . Wong, K. (2013). Towards Practical Runtime Verification and Validation of Self-Adaptive Software Systems. In R. de Lemos, H. Giese, H. A. Müller, & M. Shaw (Eds.), *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers* (pp. 108-132). Berlin, Heidelberg: Springer Berlin Heidelberg.

Tihana, G. G., Runeson, P., & Darko, H. (2016). Quantitative analysis of unit verification as predictor in large scale software enginering. *Software Quality Journal, 24*(4), 967-995.

Tsiakas, K., Abellanoza, C., Abujelala, M., Papakostas, M., Makada, T., & Makedon, F. (2017). *Towards designing a socially assistive robot for adaptive and personalized cognitive training.* Paper presented at the Proceedings of the Robots.

van den Broek, G., Cavallo, F., & Wehrmann, C. (2010). *AALIANCE ambient assisted living roadmap* (Vol. 6): IOS press.

Van Lamsweerde, A. (2009). *Requirements engineering: From system goals to UML models to software* (Vol. 10): Chichester, UK: John Wiley & Sons.

Van Rooijen, L., Bäumer, F. S., Platenius, M. C., Geierhos, M., Hamann, H., & Engels, G. (2017). *From user demand to software service: using machine learning to automate the requirements specification process.* Paper presented at the 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW).

Villegas, N. M., Tamura, G., Müller, H. A., Duchien, L., & Casallas, R. (2013). DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II* (pp. 265-293): Springer.

Viswanathan, M., & Childers, T. L. (1999). Understanding how product attributes influence product categorization: development and validation of fuzzy set-based measures of gradedness in product categories. *Journal of Marketing Research, 36*(1), 75-94.

Weyns, D., Bencomo, N., Calinescu, R., Cámara, J., Ghezzi, C., Grassi, V., . . . Malek, S. (2017). Perpetual assurances for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems III. Assurances* (pp. 31-63): Springer.

Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., . . . Göschka, K. M. (2013). On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II* (pp. 76-107): Springer.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., & Bruel, J.-M. (2010). RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements engineering, 15*(2 %@ 1432-010X), 177-196. doi:10.1007/s00766-010-0101-0 %U http://dx.doi.org/10.1007/s00766-010-0101-0

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., & Bruel, J. M. (2009, Aug. 31 2009-Sept. 4 2009). *RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems.* Paper presented at the 2009 17th IEEE International Requirements Engineering Conference.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., & Alonso, G. (2000). *Database replication techniques: A three parameter classification.* Paper presented at the Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000.

Wimmer, S., & Lammich, P. (2018). *Verified model checking of timed automata.* Paper presented at the International Conference on Tools and Algorithms for the Construction and Analysis of Systems.

Wong, B. K., & Lai, V. S. (2011). A survey of the application of fuzzy set theory in production and operations management: 1998–2009. *International Journal of Production Economics, 129*(1), 157-168.

Wongpiromsarn, T., Topcu, U., & Lamperski, A. J. I. T. o. A. C. (2015). Automata theory meets barrier certificates: Temporal logic verification of nonlinear systems. *61*(11), 3344-3355.

Xu, Z. (2007). Some similarity measures of intuitionistic fuzzy sets and their applications to multiple attribute decision making. *Fuzzy Optimization and Decision Making, 6*(2), 109-121.

Yamaguchi, T., Kaga, T., Donzé, A., & Seshia, S. A. (2016). *Combining requirement mining, software model checking and simulation-based verification for industrial automotive systems.* Paper presented at the 2016 Formal Methods in Computer-Aided Design (FMCAD).

Yan, H., Diao, X.-c., & Jiang, G.-q. (2008). *Research on data synchronization in oracle distributed system.* Paper presented at the 2008 International Seminar on Future Information Technology and Management Engineering.

Yu, E. S. (1997). *Towards modelling and reasoning support for early-phase requirements engineering.* Paper presented at the Proceedings of ISRE'97: 3rd IEEE International Symposium on Requirements Engineering.

Zadeh, L. A. (1963). On the definition of adaptivity. *Proceedings of the IEEE, 51*(3), 469-470.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, 8*(3), 338-353. doi:http://dx.doi.org/10.1016/S0019-9958(65)90241-X

Zambon, E. (2010). Using Graph Transformations and Graph Abstractions for Software Verification. In H. Ehrig, A. Rensink, G. Rozenberg, & A. Schürr (Eds.), *Graph Transformations: 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27–October 2, 2010. Proceedings* (pp. 416-418). Berlin, Heidelberg: Springer Berlin Heidelberg.

Zhang, J., & Cheng, B. H. (2006). Using temporal logic to specify adaptive program semantics. *Journal of Systems and Software, 79*(10), 1361-1369.

Zhao, L., Ichise, R., Mita, S., & Sasaki, Y. (2014). *An ontology-based intelligent speed adaptation system for autonomous cars.* Paper presented at the Joint International Semantic Technology Conference.

Zhou, F., Jiao, J. R., Chen, S., Zhang, D. J. I. T. o. S., Man,, & Cybernetics, P. C. (2011). A case-driven ambient intelligence system for elderly in-home assistance applications. *41*(2), 179-189.

Zimmermann, H.-J. (2011). *Fuzzy set theory—and its applications*: Springer Science & Business Media.

Zou, K. H., Tuncali, K., & Silverman, S. G. (2003). Correlation and simple linear regression. *Radiology, 227*(3), 617-628.