Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Integrating Data Mining into Contextual Goal Modeling to Tackle Context Uncertainties at Design Time

Arthur J. R. Farias

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientador
Prof.a Dr.a Genaína Nunes Rodrigues

Brasília
2017

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Integrating Data Mining into Contextual Goal Modeling to Tackle Context Uncertainties at Design Time

Arthur J. R. Farias

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Prof.a Dr.a Genaína Nunes Rodrigues (Orientador)
CIC/UnB

Prof. Dr. Li Weigang
CIC/UnB

Dr. Raian Ali
Bournemouth University

Prof. Dr. Bruno Luiggi Macchiavello Espinoza
Coordenador do Programa de Pós-graduação em Informática

Brasília, 24 de Novembro de 2017

Acknowledgements

I would like to thank CAPES for the financial support received, and my supervisor, Prof.a Dr.a Genaína Nunes, for the time devoted, and for the patient, yet persistent guidance throughout this work. I would also like to thank the Dependability Group from UnB's Software Engineering Lab for the several useful insights.

Abstract

Understanding and predicting all context conditions the self-adaptive systems will be exposed to during its life time and implementing appropriate adaptation techniques is a very challenging mission. If the system cannot recognize and adapt to unexpected contexts, this can be the cause of failures in self-adaptive systems, with possible implications of not being able to fulfill user requirements or even resulting in undesired behaviors. Especially for dependability attributes, this would have fatal implications. The earlier the broad range of high level context conditions can be specified, the better adaptation strategies can be implemented and validated into the self-adaptive systems. The objective of this work is to provide (automated) support to unveil context sets at early stages of the software development life cycle and verify how the contexts impact the system's dependability attributes. This task will increase the amount of potential issues identified that might threaten the dependability of self-adaptive systems. This work provides an approach for the automated detection and analysis of context conditions and their correlations at design time. Our approach employs a data mining process to suitably elicit context sets and is relying on the constructs of a contextual goal model (CGM) for the mapping of contexts to the system's behavior from a design perspective. We experimentally evaluated our proposal on a Body Sensor Network system (BSN), by simulating a myriad of resources that could lead to a variability space of 4096 possible context conditions. Our results show that our approach is able to elicit contexts that would significantly affect a high percentage of BSN assisted patients with high health risk profile in fulfilling their goals within the required reliability level. Additionally, we explored the scalability of the mining process in the BSN context, showing it is able to perform under a minute even for simulated data at the size of over five orders of magnitude. This research supports the development of self-adaptive systems by anticipating at design time contexts that might restrain the achievability of system goals by means of a sound and efficient data mining process.

Keywords: Self-adaptive systems, Context uncertainty, Data mining, Design-time, Goal models

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The consistent specification of all stakeholder needs is rarely enough to ensure the quality of a self-adaptive system (SAS), demanding the designer the full specification of the contexts in which such requirements shall be executed. Consequently, assuring that a system successfully functions for all environmental conditions represents a great challenge for the software engineering of self adaptive systems. Although there are some work that support the elicitation of typical contexts [4, 5, 6], the subjective nature of the problem and the lack of information at initial stages make it quite difficult to, anticipatively, identify all of them. To illustrate, we can mention the conflicts among stakeholders that often lead to a system with requirements that are neither alternative nor consistent with each other, having their operation triggered by specific contextual conditions [5].

Like any other system, in order to guarantee the expected operation of the service provided by a SAS, dependability attributes such as reliability, availability, safety, security and maintainability must be taken into consideration [7], specially in safety-critical systems, which cannot afford to fail. The system's dependability and users' satisfaction are heavily impacted by context variability, and the assurance of such aspects is put in risk if the impact of the contexts are unknown, or even if the context awareness of the system is compromised. At the same time, a superficial domain and environment knowledge might lead to the insertion of dormant faults in the implementation, imperceptible from a design perspective. Hence, such faults, when active, are potential causes of failure, i.e., deviation from the correct service state. This fact emphasizes the need of a dependability assessment method that considers the different contexts and user profiles a system might be exposed to at runtime.

The dependability of a self-adaptive system can also be limited by the presence of uncertainty, a concept that haunts the assurance of many aspects of the software engineering process. The unpredictability caused by the limitation in accounting for all environmental conditions at design time is a common type of uncertainty for self-adaptation [8]. The uncertainty in contexts can be found beyond the elicitation stage, it can be found on the monitoring of such contexts.

The success of the monitoring routine depends on the availability of the system's components and how these resources affect the performance of the system. The availability of the resources is subject to changing environment conditions as the system gets executed [9]. As a result, uncertainty is a worrying concept for the dependability aspect, causing degradation in the quality of service or even failures in self-adaptive systems, possibly leading the system to have undesired behaviors.

There are some tools and techniques that assist the assessment of dependability at design time. For instance, one is able to estimate the reachability of the stakeholder needs through the analysis of the system's models, scanning its transitions and probabilities. However, these techniques usually ignore important runtime factors, such as context variability and distinct user profiles, that directly impact dependability attributes in positive or negative ways. The user's satisfaction is another important quality-of-service metric that is hardly ever considered in a statical analysis, even though, in a real scenario, a SAS is supposed to operate correctly under different context conditions, satisfying distinct users profiles.

1.1 Problem Definition

Due to the difficulty in predicting environmental conditions still at design time, there are very few works that try to tackle the derived uncertainty at such stage [10], particularly w.r.t. context variability conditions. On the other hand, many state-of-the-art contributions have been proposed to deal with it through control and runtime adaptation [8], more precisely on runtime context data, after the system is implemented. Although the runtime approaches may tackle the problem within some acceptable degree of accuracy, the inspection of all possible situations has a negative impact on the system's performance. Moreover, it seems to be a risky strategy for safety-critical systems, since they take a while to learn from the environment and may not be able to adapt fast enough to avoid an interruption, putting the dependability of the application in check. Relying only on a reactive approach to deal with such kind of uncertainty can also be troublesome when it comes to the cost of changes and repairs, given that these costs usually increase with the elapsed project time. A software design flaw spotted at runtime, for example, tends to be expensive and time consuming to fix.

The relation between design-time and runtime variability, especially for SAS, still requires improvement [4]. Information about reachability and variability, collected at design time, are not fully reused at runtime. Nevertheless, the generation of all possible situations, exclusively at runtime, poses a risk to the system's performance and dependability, which indicates the need of a design-time assessment. Such a gap is particularly critical when considering runtime aspects, reinforcing the need to adopt a preventive approach for context elicitation and dependability analysis.

To the best of our knowledge, there is no approach that applies data mining to unveil context conditions at design time, when planning the self-adaptation strategies accounting for its dependability. For complex systems, it is unfeasible to make a combinatorial exploration of every monitored attribute and map all possible contexts afterwards, defining, for example, if-then-else rules for every possible scenario. It is likely that such approach will result in a state explosion problem. Despite the challenge in discovering context conditions at design time due to the lack of data to exercise the method, we argue that analyzing context conditions from the dependability perspective is paramount and should be done as early as possible in the system development cycle. Differently from runtime analysis, at design time there is no sufficient data provided yet. The use of prototypes helps one to overcome this limitation, and, at the same time, supports the understanding of the differences of end-users in context [5]. It can potentially help to reduce failures caused by invalid adaptations executions.

Considering the current scenario of this area, the research gap, and all the limitations described in the previous paragraphs, we summarize the objective of this work in the following research question.

RQ: Is it possible to support the decision making process for dependable SAS, using a data mining process, specifically classification and association rules algorithms, to anticipate the identification of contexts in a way that time and space limitations are surpassed?

1.2 Proposed Solution

In this work, we propose an approach that aims to identify, at design time, contexts that are potential sources of uncertainty, i.e., contexts that can prevent the full satisfaction of some requirements or trigger unknown behavior that should be disallowed from the perspective of dependability. Through a data mining process we determine context sets based on requirements knowledge that would otherwise be neglected and only be identified at runtime in the presence of context failure. Considering goals are first class citizen of self-adaptive systems and goal models are a common technique on how to document such goals [11], our context set discovery process also considers the possible impacts on goals' fulfilment caused by inter-contexts combinations.

In order to abstract away the relationship between contexts and goals, we use Contextual Goal Model (CGM) [12] to map the contexts to the system's behavior from a design perspective. This way, we focus on the goals' accomplishment by modeling the system adaptation in face of context variations including context failure ranges. Additionally, a qualitative and quantitative context analysis is then used to update the contextual constraints into their respective nodes of the model. The CGM is a useful artifact to carry the knowledge generated by our proposal, allowing not only the analysis of the desired behavior of the system but also the development

of an adequate adaptation plan for different context. Thus, we minimize the gap between the design-time and runtime model by keeping the CGM always up-to-date through a feedback loop process.

1.3 Evaluation

We experimentally evaluate our approach on a simulated prototype version of a Body Sensor Network system (BSN) [2]. We simulated twelve relevant BSN resources, that are abstract representations of computational or environmental aspects, e.g. sensors, vital sensed data, storage module. The resources, by a mere combinatorial approach, represents a variability space of 4096 possible combinations that may influence the current context. We generated a database representing 100,000 patient records using a Monte Carlo method. Our approach was able to efficiently identify 17 new relevant contexts not initially considered in the BSN case study [2]. Our results reveal the overwhelming fact that almost 84% of high risk patients profiles and 73% of normal risk patient profiles would be unattended or failed, which allows us to infer the efficacy of our elicited contexts to significantly identify critical situations on the BSN case study.

We also validated the scalability of the data mining process by dividing the analysis in two different scenarios: (i) a verification concerning the load scalability, i.e., comparing the generation time growth of the prediction model with the capability of the data mining method to process larger and heavier datasets, and (ii) a verification concerning the functional scalability, i.e., observing the generation time growth of the prediction model in face of the ability to expand the scope of analysis by adding new attributes to the dataset, object of data mining. We have noticed that the data mining process should not be an obstacle to the scalability aspect. The method is able to perform under a minute even for simulated data at the size of over five orders of magnitude.

1.4 Organization

The remaining parts of this manuscript are organized as follows: In Chapter 2 we provide a brief introduction to contexts, dependability, contextual modeling and some insights about data mining. Chapter 3 presents major related work. Chapter 4 presents a succinct characterization of the Body Sensor Network system, case study of our approach. In Chapter 5, we present the core of our proposal, followed by Chapter 6, where we report the evaluation of our approach. Finally, Chapter 7 concludes along with future work.

Chapter 2

Background

2.1 Contexts in Self-Adaptive Systems

The word *context* carries an important and pervasive concept, not restricted to the computational field. This fact sometimes mislead the communication of the meaning of context, since it is common that people tacitly understand its definition but apply in different ways in their researches. For the purpose of our research, there are a few suitable definitions. The first one can be found in [13] where the author states:

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." (Dey, 2001, p. 3).

Another good definition was proposed by Ali et al. in [12], where contexts are "monitorable pieces of information about the environment in which systems operate". In order to better understand the such definition, we need to know what environment means. Finkelstein and Savigni [14] defines environment as "whatever over which we have no control", e.g. environmental conditions, user characteristics or availability of resources. The context analysis presented in [12] relies on the refinement of contexts described at a high level of abstraction to a formula of observable facts. Such methodology is suitable for adaptive and self-adaptive systems, since their adaptation are mostly defined by verifiable facts. Different contextual information can be monitored through means like sensors, and a context may be represented by the combination of specific resources states. We call this process *context operationalization*.

The definitions above are specially adequate for our approach, once the enumeration of contexts is an important step of the process. The focus of our work is to elicit context focusing at the satisfaction of a required dependability level. Many attributes of dependability are entangled with the operation of the system's devices at an operational level. Our approach assumes as a context, facts that are often mistaken as resource failures. In fact, they are actually situations of an entity, with a certain probability to happen that are relevant to the interaction between the

user and the application. Our approach is adjustable to such specific contexts, but respects the overall concept mentioned before.

2.2 Assurance for Self-adaptive Systems

The *assurance* concept we adopt in the scope of this work is the one proposed by the IEEE Standard Glossary of Software Engineering Terminology, which states that assurance is "a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements" [15]. Such definition encompasses not only quality attributes like cost-benefit, versatility, flexibility, resiliency and energy efficiency, but also broader software aspects such as safety, reliability and dependability [16, 17].

Before the introduction of self-adaptiveness in software systems, the assurance for software systems was conducted at design and development time through verification, validation, test, measurement, conformance to standards and certification [18]. Nowadays, the most acceptable strategy to provide assurance at runtime is utilizing self-adaptation mechanisms and their capabilities of self-management. However, the dynamic environment to which self-adaptive systems are exposed creates obstacles that hinders the assurance provision in SAS [17]. To illustrate, we can cite the uncertainty present on contexts monitoring, that can affect the ways in which the information about the environment can be gathered. Through our method, we propose to provide assurance by the early identification of possible sources of uncertainty by means of unforeseen contexts, and quantify the impact on self-adaptive systems' ability to provide assurance evidence.

2.2.1 Dependability

Dependability is concept that is usually adopted to reference characteristics inherent to the expected behavior of a system, i.e., the capability of a system to avoid, tolerate and adapt to failures. In order to guarantee a minimum quality-of-service level, there are some dependability attributes that need to be considered when developing a system. Such statement is even more valid for self-adaptive systems due to the dynamic environment to which it is exposed. According to Avizienis et al. [7], among the aspects that compose dependability we have the following attributes:

- **Availability:** readiness for correct service;
- **Reliability:** continuity of correct service;
- **Safety:** absence of catastrophic consequences on the user(s) and the environment;
- **Integrity:** absence of improper system alterations;

- **Maintainability:** ability to undergo modifications and repairs.

Before we advance in the characterization of dependability, the distinction between the concepts that resemble to systems' undesired traits needs to be enlightened. Basically, there are three definitions that are commonly mistaken: *fault*, *error* and *failure*. The order of enumeration was intentional to show a causality chain linking them. As Avizienis et al. state, a service is a sequence of the system's external states, thus, a service *failure* means that at least one external state of the system deviate from the correct service state. We call such deviation an *error*, while the adjudged or hypothesized cause of an error is called a *fault*, that can be internal or external of a system.

The present work aims at assisting the analyst in a preventive way, eliciting as many contexts as possible and providing an adaptation plan at design time. As complement, the system must provide, at runtime, a reactive approach to deal with the contexts that were not previously unforeseen. Summarizing, still based on the concepts defined in [7], through the anticipated identification of context and the impact analysis of their combinations, our work assists the implementation of: (i) fault prevention techniques, i.e., means to prevent the occurrence or introduction of faults, and (ii) fault forecasting techniques, means to estimate the present number, the future incidence, and the likely consequences of faults.

The notion of *contextual failures* will be introduced in the next chapters, and to take the best out of it, it is necessary to briefly describe the background that supported the creation of such definition, specifically regarding its domain. According to Avizienis [7] the domain of a failure may be seen from two distinct viewpoints, they are:

- **Content failures:** The content of the information delivered at the service interface deviates from implementing the system function;
- **Timing failures:** The time of arrival or the duration of the information delivered at the service interface deviates from implementing the system function.

When both content and timing problems are observed in a failure, it may fall into two classes:

- **Halt failure:** as the name suggests, the service is halted, i.e., the system activity, if there is any, is no longer perceptible to the users; a special case of halt is **silent failure**, when no service at all is delivered at the service interface;
- **Erratic failures:** otherwise, i.e., when a service is delivered (not halted), but is erratic (e.g., babbling).

For the purpose of this work, there are context violations that belong to a data domain, and violations that belong to the time domain. Both types can lead to a interruption of the service provided like a halt failure, but also can lead to behaviors typical of erratic failures.

2.3 Contextual Goal Models

When an application is planned to operate in a dynamic environments, the notion of context must be the cornerstone of the system development. In order to build a proper self-adaptive system blueprint, it is required to take into consideration not only the requirements and means to achieve them, but also the contextual information that may be related to the system's operation. A contextual goal model (CGM) is a suitable specification for this purpose, since it is able to represent in a simple structure the requirements to meet, the ways to meet requirements, and factors that can affect the quality and behavior of a system. In order to better absorb the global semantics of a contextual goal model specification, firstly, we need to understand its elements. According to the definitions presented in [12], a CGM can be composed by:

- **Actor:** an actor is an entity that has goals and can decide autonomously how to achieve them. An actor can be of different types such as human actors, software actors, or organizational actors.
- **Goal:** goals are a useful abstraction to represents stakeholders' needs and expectations and they offer a very intuitive way to elicit and analyze requirements.
- **Task:** a task is an atomic part responsible for the operationalization of a system goal, i.e., an operational means to satisfy stakeholders' needs.
- **Context:** a context is a partial state of the world that is relevant to an actor's goals. A context is inherently partial and volatile, it is also strongly related to goals, for it changes the current goals of a stakeholder and the possible ways to satisfy them.
- **Resource:** an entity data or physical device that is generated or required by an actor. For instance: a sensor, a vital sign measurement, the storage space available in a disk component, etc.
- **AND Decomposition:** an AND-decomposition is a refinement link that decomposes an actor's goal or task into sub-goals or sub-tasks, where all decomposed goals/tasks must be fulfilled/executed in order to satisfy its parent entity.
- **OR Decomposition:** an OR-decomposition is a refinement link that decomposes an actor's goal or task into sub-goals or sub-tasks, where at least one decomposed goals/tasks must be fulfilled/executed in order to satisfy its parent entity.
- **Means-end:** a relation that indicates a means to fulfill an actor's goal through the execution of a task.

At this point, it is important to make a distinction between resources and contexts. We assume that a sensor that measure the environmental conditions is a system resource. Addition-

ally, some specific behaviors of this particular sensor can be taken as contexts. For example, the availability or unavailability of a given resource, if it is relevant to the accomplishment of a goal, characterize two different contexts of operation.

It is possible to achieve a system goal through different ways within a CGM. Each different way might contain different conjunction of contexts, and each conjunction shapes the system to fulfill a requirement in a different quality level. Each conjunction of contexts is known as a *context of a goal model variant* [12]. Self-adaptive systems implement runtime routines to choose which variant to adopt when more than one variant is applicable in the actual scenario. Often, users prioritization is the criteria adopted to support this decision. However, this decision process rarely is a simple task due to two major problems: the potentially large number of goal model variants and the potentially large number of nodes in each variant. The first makes the prioritization very time consuming, while the latter makes hard for the user to distinguish the differences between variants [12]. Our method allows one to combine the users prioritization or users profiles as quality measures (e.g. softgoals) with data mining techniques, aiming to identify the most efficient variants in terms of a specific requirement.

2.4 Data Mining

To create a solid method of context elicitation, bypassing both time and space limitations, we intend to merge the advantages of using artificial intelligence over monitored data, characteristic of runtime approaches, with the perks of having a robust modeling process at design time. The core of our approach relies on the data mining, analysis, and discovery of new contexts that can possibly lead the system to inconsistent states. The object of the data mining process, at design time, can be provided by historical data from previous executions, by a domain driven simulated dataset through Monte Carlo, or by a prototype version, in case there are no runtime data available.

The science of learning is present in the fields of statistics, data mining and artificial intelligence, intersecting with areas of engineering and other disciplines [19]. On self-adaptive systems, this kind of technique composes the core of the application, making possible for the system to adapt to new scenarios by learning from environmental data. In a typical operation of a self-adaptive system, we have an outcome measurement, usually quantitative or categorical, that we wish to predict based on a set of resources. We have a training set of data, coming from sensors, in which we observe the outcome and resources measurements for a set of objects (such as users of a medical monitoring service). Using this data we build a prediction model, or learner, which will enable one to predict the outcome for new unseen objects. A good learner is one that accurately predicts such an outcome. Afterwards, the prediction model is used to support the system to formulate the adaptation plans.

2.4.1 Association Rules

Countless techniques may be used to assist the data mining process. Among the several categories of data mining methods, there is the *Association Rules Method*, which uses a rule-based mechanic that allows us to discover relations between variables in large databases.

According to the definition proposed by Agrawal et al. [20], the association rule mining process works as follows: Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals, called *items*; Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$; Associated with each transaction is a unique identified, called TID. A transaction T contains X, a set of some items in I, if $X \subseteq T$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \bigcap Y = 0$.

In order to find useful rules from the set of all possible rules, constraints that measure the significance and interest of sch rules must be applied. The most common constraints are minimum thresholds on *support* and *confidence*. The rule $X \Rightarrow Y$ holds in the transaction set D with *confidence* c if c% of transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set D if s% of transactions in D contain $X \bigcup Y$.

Another important concept for association rules is the *Lift*. It help us to identify which rules are useful and which are not. The lift of a rule is defined by the equation:

$$lift(X \Rightarrow Y) = \frac{supp(X \bigcup Y)}{supp(X) \times supp(Y)} \tag{2.1}$$

If the rule have a lift of 1, the probability of occurrence of the antecedent and that of the consequent are independent of each other, i.e., no rule can be drawn involving those two events. On the other hand, a lift higher than 1 indicates the degree to which those two occurrences are dependent on one another, meaning that these rules are potentially useful for predicting the consequent in future data sets.

We have adopted the *Apriori* algorithm [21] in our approach. It is a derivation of the association rule category, where the rules are somewhat more general than the ones previously described, allowing a consequent to have more than one item.

2.4.2 Classification Methods

The classification routine tackles the problem of identifying to which of a set of categories a new observed fact belongs. It is done based in a training set of data containing observations (or instances) whose category membership is known, i.e. it is an instance of supervised learning. There are a few ways to measure the quality of classification methods [22]. We have adopted the popular and well-defined measurements precision, recall, and their harmonic mean f-measure, to evaluate our approach. In order to explain the terminology and the application of there metrics, we list a series of important concepts regarding the classification context.

- **Condition Positive (P):** the number of real positive cases in the data;
- **Condition Negative (N):** the number of real negative cases in the data;
- **True Positive (TP):** the object classification represents a hit.
  (correctly classified as true)
- **True Negative (TN):** the object classification represents a correct rejection.
  (correctly classified as false)
- **False Positive (FP):** the object classification represents a false alarm.
  (classified as true, but actually it is false)
- **False Negative (TN):** the object classification represents a miss.
  (classified as false, but actually it is true)

The precision and recall are defined as:

$$Precision = \frac{tp}{tp + fp} \qquad\qquad Recall = \frac{tp}{tp + fn}$$

RIPPER

In our work, we are applying two classification techniques to assist the context elicitation and the measurement of the impact on system's dependability. The first one is the *JRip* (implements RIPPER algorithm) [23], a propositional rule learner that create rules for every class in the training set and then prune these rules. The discovered knowledge in this class of algorithm is represented in the form of IF-THEN prediction rules and are specially useful to define the operation thresholds of some resources. The RIPPER algorithm works as follows [23, 24]: (i) after the initialization of a list RS, the algorithm divides iteratively the training set into growing and pruning sets in a *building stage*, having the stopping condition defined by the description length of the ruleset and error rate. The *grow phase*, a sub stage of the building phase, consists in growing one rule by greedily adding conditions to it until the rule is completely accurate. The algorithm tries every possible value of each attribute and selects the condition with the highest information gain. At the second sub stage of the building stage, in the *prune phase*, the algorithm prunes each rule and allow the pruning of any final conditions. Once the stopping criteria is met, and the initial ruleset $\{R_i\}$ is generated, the procedure reaches the (ii) *optimization stage*, in which the algorithm generates and prunes two variants of each rule R_i from randomized data in the previous phases. One variant is generated from an empty rule while the other is generated by greedily adding antecedents to the original rule. The smallest possible description length for each variant and the original rule is computed. The variant with the minimal description

length is selected as the final representative of R_i in the ruleset. At last, (iii) the rules that would increase the description length of the whole ruleset are deleted, while the resultant ruleset is added to RS.

Decision Tree

The second classification method we use is the *J48* classifier algorithm, which implements a *Decision Tree* [1], a tree-like graph used to support the decision making process using the depth-first strategy. J48 is an open source Java implementation of the C4.5 decision tree algorithm, extension of the ID3 algorithm [25]. The exemplifying dataset presented in Table 2.1, proposed by Quinlan [1], compiles some weather measurements of a series of Saturday mornings. The attribute *Class* represent the feasibility of a given activity, the value P means positive, i.e., the planed activity is feasible in that given weather, while the value N, that stands for negative, represents the impossibility of the activity to occur. After the execution of the C4.5 algorithm we come to a tree similar to the one depicted in Figure 2.1.

	Attribute				
No.	**Outlook**	**Temperature**	**Humidity**	**Windy**	**Class**
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

Table 2.1: An example of a training set [1]

Basically, C4.5 relies on the concept of *information entropy* to build a decision tree from a set of training data, guided by the objective of reducing the impurity in data as much as possible, i.e., having the most instances of a subset belonging to the same class. Entropy is a measure of the uncertainty associated with a random variable. The original entropy of a given dataset can be achieved from a set of samples S through the formula, with C representing the set of desired class:

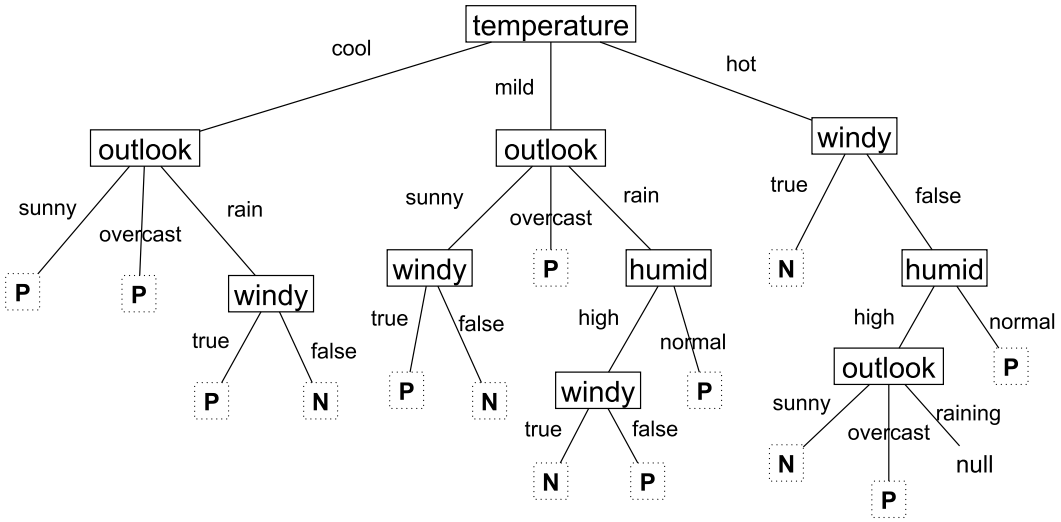


Figure 2.1: Decision tree based on the training set presented in Table 2.1 [1].

$$E[D] = -\sum_{i=1}^{|C|} P(c_i) log_2 P(c_i) \tag{2.2}$$

To illustrate, the entropy of a detaset D that has a 30% positive examples (P(*positive*)=0.3) and 70% negative examples (P(*negative*)=0.7) is:

$$E[D] = -0.3 \times log_2 0.3 - 0.7 \times log_2 0.7 = 0.8813 \tag{2.3}$$

Analogously, the entropy of an attribute can be calculated by making the target attribute, with n values, the root of the tree. Thus, after the partition of a dataset D into n subsets, it is possible to calculate the entropy of an attribute with the following equation:

$$E_{attribute}[D] = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} E[D_i] \tag{2.4}$$

The training data set is composed by samples already classified (attribute value and the class it belongs to). At each node of the tree, the algorithm chooses the attribute of the data that best splits its samples into subsets. The splitting criterion is based on the *information gain* value, which is the difference of the dataset entropy and the entropy of the selected branch. The C4.5 recurs on the the smaller sublists using the information gain to make the classifying decisions. The information gained by selecting a given attribute to partition the data can be obtained by:

$$gain(Attribute) = E[D] - E_{Attribute}[D] \tag{2.5}$$

2.4.3 Data Mining for Context Discovering

Combining the aforementioned algorithms with a requirements engineering process that considers contextual information, the learning premise can be extended to self-adaptive systems and their resources. If we assume, for example, that a patient having a heart attack configures the context 'emergency', it is possible to use Apriori to discover which resources were involved in the activation of such context, for instance, heart beat sensor and accelerometer. Later on, JRip is used to find rules that describe the activation of the context in terms of resource values. To illustrate, one could discover through a pruned rule that an individual's pulse above 140 beats per minute indicates a risk of activating the 'emergency' context. Lastly, all the contexts and resources can be mapped into a decision tree structure, giving a global perspective of the system to the analyst, showing how such entities interact to successfully achieve the system goals. In the next chapters we will specify the entire process and demonstrate how such combination can take place in a sound manner. The information gain values are crucial for our work because they represent, in a certain way, the quantification of a node's sensitiveness. The hierarchical view provided by the decision tree helps one to visualize the most important categories in a higher level, assisting the prioritization and assurance of systems' dependability attributes in detriment of others, for different context configurations. From a context elicitation point of view, the dependability attributes of a system, as the object of analysis, might be more sensitive to the variability of contexts in upper nodes, indicating that the identification of such context is paramount to observe a correct operation of the system.

2.5 Theoretical Overview

In this chapter we have presented the theoretical foundations needed to best assimilate the proposal. We begun with some context definitions and their implications in self-adaptive systems. We have learned that a context is any information that can be used to characterize the situation of an entity. It was possible to notice that the identification and characterization of a context is paramount to the provision of assurance for SAS, i.e., all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. Dependability is a core definition for assurance provision, as it encompasses several attributes related to quality of service and non-functional requirements such as reliability, availability, safety, security and maintainability. In this chapter we also have explained some points about contextual goal modeling, a goal-oriented specification technique suitable to represent self-adaptive systems and their elements. Finally, we presented the machine learning algorithms used in our method: (i) apriori, as an association rule technique, (ii) RIPPER, as a propositional rule learner, and (iii) J48, an open source Java implementation of the C4.5 decision tree algorithm. In the next chapter we will present the major related work and how they are compared to our method.

Chapter 3

Related Work

3.1 Context elicitation

Knauss *et al.* [5] contribute with a study on how to derive contexts from stakeholder needs. The authors explore the usefulness of several existing elicitation techniques such as interviews, prototyping, scenarios, goal-based approaches, and focus groups for the identification of contextual requirements at design time. Their work brings some insights on the topic, including the fact that (i) conflicts among stakeholders indicate the need for contextual requirements, (ii) viewpoints are valuable to identify context related to requirements and to analyze contextual requirements, and (iii) prototypes are particularly helpful to understand the context of conflicting requirements in details, which motivated us to use prototyping techniques to represent a real system operation.

Hong *et al.* created a methodology for the elicitation of requirements in context-aware applications [6]. The proposed method links context-awareness features with the target context by capability matching. They divided the notion of contexts into three categories: the first category is *computing context*, referring to the hardware configuration used, e.g. processors available, devices accessible, and bandwidth; the second category is *user context*, which represents all the human factors, e.g. user's profile, calendars; the final category is *physical context*, that encompasses the non-computing-related information provided by a realworld environment, such as location, time, lighting, noise levels, etc. Although we also categorize the unveiled contexts in three classes, differently from their work, we propose a classification based on the data format in which the context will be perceived by the autonomous controller of the SAS.

Another research on requirements elicitation considering context variability, Gómez *et al.* [26] present APP STORE 2.0, an app store which applies data mining techniques to exploit crowd-sourced information at runtime aiming at the elicitation of requirements, and improvement of the overall quality of the delivered service. The APP STORE 2.0 involves users in a quality feedback loop, creating value based on the end-users, while the end-users indirectly benefit

from APP STORE 2.0 with better apps. Our proposal, focusing on dependability elements, considers that the elicitation of new contexts, at design time, can reveal possible uncertainties to the adaptive system, i.e., disallowed behaviors latent in the design that can jeopardize the requirements fulfillment at runtime.

3.2 Specification and adaptation to contextual changes

Ali *et al.* proposed a framework [12] that, starting from contextual requirements, involves reasoning and goal modeling to support the adaptation of a system to different contextual conditions. The choice and prioritization of the adequate goal model variant is a difficult task for complex systems due to the potentially large number of goal model variants, and the potentially large number of nodes in each variant. Such a limitation motived us to use data mining in this sense. Moreover, through the feedback loop proposed in our work, one is able to verify another question raised in [12], concerning the influence on context caused by the actions that the system takes to meet its requirements and the problems it may lead to.

Villegas *et al.* present DYNAMICO [27], a reference model for governing control and context relevance in self-adaptive systems, in which they have defined, discussed, and implemented the preventive approach in the context of self-adaptation. The authors claim that separation of concerns, dynamic monitoring, and runtime requirements variability are critical for satisfying system goals under highly changing environments. In order to guarantee these aspects, DYNAMICO is composed of three types of feedback loops: (i) the control objectives feedback loop, (ii) the target system adaptation feedback loop, and (iii) the dynamic monitoring feedback loop. In our work that a similar analysis can be done, still at design time, to identify the resources and correlated contexts that, while available, guarantee the satisfaction of the probabilistic requirements.

3.3 Uncertainty definition for self-adaptive systems

Esfahani and Malek developed a study about uncertainty in self-adaptive systems field [9] claiming that, although uncertainty is sometimes taken as a second-order concept, it is not possible to remove uncertainty of a system by focusing exclusively on its normal behavior. The work contributes with a classification and detailed description of several software aspects that can be considered as sources of uncertainty such as uncertainty in the objectives, uncertainty due to model drift, uncertainty in contexts, etc. Mahdavi-Hezavehi *et al.* [28] propose a classification framework for architecture-based approaches tackling uncertainty in self-adaptive systems with multiple quality requirements. The work helps us to understand the current state of research

regarding uncertainty. Another work that classifies and describes uncertainties in the context of adaptive systems was proposed by Ramirez *et al.* [8].

Whittle *et al.* claim that a more rigorous treatment of requirements explicitly relating to self-adaptivity is needed, more specifically the uncertainty-related aspects [29]. They present RELAX, a requirements language to explicit the presence of uncertainties in self-adaptive systems. Cheng *et al.* [30] take a step further and combine the RELAX specification with goal modeling to develop requirements of an adaptive system. Additionally, they use a threat modeling variation to explore environmental uncertainty factors. The aforementioned researches give us some specific targets when the subject is uncertainty mitigation. Our work corroborates the fact that the use of data mining is a sound alternative to guide analysts to anticipate such uncertainties, avoiding possible time and space limitation, inherent of the combinatorial exploration in complex systems.

3.4 Dependability in context-based systems

From the dependability perspective, the closest in nature to our work combines contextual requirements and goal models: Mendonça *et al.* [31] propose a method to capture contextual failures and use that to enrich the representation of dependability requirements. The approach mitigates the assumption about the certainty of success of a task to reach its goal by adding contextual information about the quality of alternative tasks. However, due to the combinatorial if-then-else rules required from expert domain knowledge for fine grained individual analysis of contexts, the approach requires too much effort from the domain expert to guarantee an accurate context analysis. Differently from our work, the following researches do not consider the context elicitation as a means to reach dependability, however they help us in delineating the scope of our work, considering the research gaps in dependability assessment for adaptive systems. Grassi *et.al* [32] present an approach that support the assessment of performance and dependability attributes through a model transformation chain that maps a "design oriented" model to an "analysis oriented" model. Mahdavi-Hezavehi *et al.* [33] proposed a systematic literature review concerning to architecture-based methods for handling multiple quality attributes (QAs) in SAS. The authors say that performance and cost are the most frequently addressed set of QAs. Lemos *et al.* [34] reunited some researches about the challenges in the provision of assurances for SAS, the majority of the them are caused by the high degree of uncertainty introduced by runtime changes.

3.5 Tackling uncertainty at design time

A few works propose to deal with uncertainty still at design time. Among them, Horkoff *et al.* present an iterative methodology that guides the resolution of uncertainties necessary to achieve desired levels of goal satisfaction, assuming the model as the source of such uncertainty [10]. Hassan *et al.* created a method that is also worth mentioning [35]. Their method consists in allowing designers to make explicit links between the possible emergence of undesired surprises, risks and design trade-offs. The objective is to provide designers of self-adaptive systems with a basis for multi-dimensional what-if analysis to revise and improve the understanding of the environment and its effect on non-functional requirements and thereafter decision-making. Our major contribution in comparison to other design time approaches is the use of data mining, still at design time, to analyze unexplored relations between resources that can be possible sources of new contexts, verifying the impact of such contexts in the satisfaction of functional or non-functional requirements with a view to dependability attributes.

3.6 Tackling uncertainty with AI methods

Many state-of-the-art contributions which involves artificial intelligence for dealing with uncertainties at runtime are present in the literature. To tackle this unpredictability at execution time, Knauss *et al.* proposed ACon [36]. The framework is based on machine learning and data mining techniques and it is meant to provide an adaptation of contextual requirements at runtime. The framework aids a self-adaptive system to adjust itself to overcome unexpected context variability and infrastructure failures through a feedback-loop.

Esfahani *et al.* propose a similar approach [37], it uses machine learning to make a feature-oriented adaptation, focusing on features instead of contextual requirements. The authors state that domain expert's knowledge, represented in feature-models, adds structure to on-line learning, which in turn improves the accuracy and efficiency of adaptation decisions. They deal with uncertainty through control and runtime adaptation.

Welsh *et al.* [38] merges the uncertainty mapping with runtime resolution for the realization of requirements-aware systems through REAssuRE. The authors claim that the combination of requirements awareness with requirements monitoring and self adaptive capabilities should help optimize goal satisfaction even in the presence of changing run-time context. They include claims to support the reasoning over uncertainty. REAssuRE is able to reason about how design-time assumptions affect goal realization strategies, as evidence for or against design-time assumptions is gathered by claim monitoring.

Sharifloo et al. [39] argue that design-time uncertainty on how the context might change may mean that a DSPL lacks adaptation rules or configurations to properly reconfigure itself at

runtime. To cope with this limitation, they propose a feedback approach through an adaptive system model that combines learning of adaptation rules with evolution of the DSPL configuration space.

Our method leverage the use of data mining, prototyping and contextual goal modeling to assist the elicitation of contexts related to dependability attributes, dealing with possible sources of uncertainty at early stages of the software development lifecycle, adopting a preventive-like behavior.

3.7 Final Considerations About the Related work

The related work presented in this chapter corroborate with the statement that an early identification of possible context conditions and the quantification of their impact on the system's behavior is critical to support the development of a dependable software. However, the lack of information at design time makes the identification of such conditions a great challenge for the software engineering of self-adaptive systems. In the scope of our work, such a challenge is tightly related to the concept of uncertainty, that begins at the elicitation stage and extends itself to runtime through the monitoring of the current context. Throughout this chapter, we have listed some related work that propose alternatives to the problem described above, regarding the challenges of context elicitation, dependability, specification, and adaptation of SAS to contextual changes, as well as uncertainty modeling and management in self-adaptive systems. Table 3.1 summarizes the most important properties and characteristics of each related work and how they differ from each other and from our method. In the next chapter we will describe our running example w.r.t. its components, features, and objectives.

Work by:	**Appl. Stage**	**Goal-oriented**	**Dependability-oriented**	**Context elicitation**	**Techniques applied**
Hong *et al.*, 2005 [6]	Design	No	No	No	Reasoning
Cheng *et al.*, 2009 [30]	Design	Yes	No	No	Goal modeling, RELAX, threat modeling variation
Grassi *et.al*, 2009 [32]	Design	No	Yes	No	Model transformation, reasoning
Villegas *et al.*, 2010 [27]	Design	No	No	No	Reasoning, feedback loops
Welsh *et al.*, 2011 [38]	Runtime	Yes	No	No	Claims
Esfahani *et al.*, 2013 [37]	Runtime	Yes	No	No	Machine learning
Horkoff et al., 2014 [10]	Design	Yes	No	No	Reasoning
Knauss *et al.*, 2014 [5]	Design	Yes	No	Yes	Combined requirements elicitation techniques
Mendonça *et al.*, 2014 [31]	Design	Yes	Yes	Yes	Goal modeling, reasoning
Hassan *et al.*, 2015 [35]	Design	No	No	No	Multi-dimensional what-if analysis
Knauss *et al.*, 2016 [36]	Runtime	No	No	Yes	Machine learning, data mining
Gómez *et al.*, 2017 [26]	Runtime	No	No	Yes	Data mining, path analysis, topic modeling, feedback
Present work	**Design**	Yes	**Yes**	**Yes**	**Data mining, goal modeling**

Table 3.1: Comparative table of the related work and their properties

Chapter 4

Running Example: Body Sensor Network

Without loss of generality, we illustrate the concepts of our approach throughout this work using the example of the Body Sensor Network (BSN). Figure 4.1 shows how a BSN is organized. Wireless sensors are connected to a person. There may be a central node (Control Sensor) responsible for preprocessing the data collected, filtering redundancy, or translating communication protocols. The other sensors are the following: Accelerometer (Acc.), ECG (for heart rate and electrocardiogram curve), Oximeter (for blood oxidation and blood oxidation curve, called SPO2), and Temperature (Temp.).

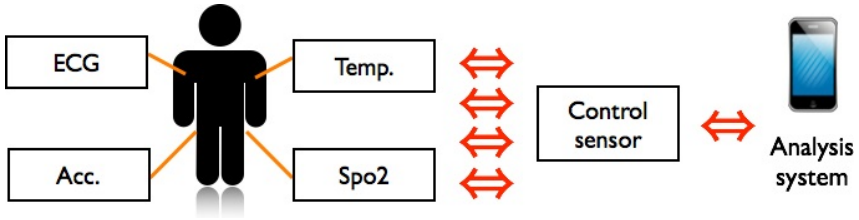


Figure 4.1: Body Sensor Network visual representation [2].

The main objective of a BSN system is to continuously monitor the vital signs of an individual adjusting its configuration according to the patient's health risk status. The individual's health risk can be classified into *low risk*, *normal*, or *high risk* category. The patient's risk status maps to a quality goal of BSN. Each health risk requires a minimum quality level to be considered trustworthy, while each different configuration of BSN provides a different level of reliability to the operation. Figure 4.2 illustrates the feature model that represents our case study. It takes into consideration a configuration in which all resources, that is, sensors and their information type are present. However, one or more resources might be unavailable at certain moments depending on the reliability of each device.

Patients with different profiles should be able to wear the system, and, over time, these profiles or even the subsidiary medical knowledge may evolve. Hence the quality goals needs to adjust according to each profile. The ranges of the sensors involved in the operationalization of

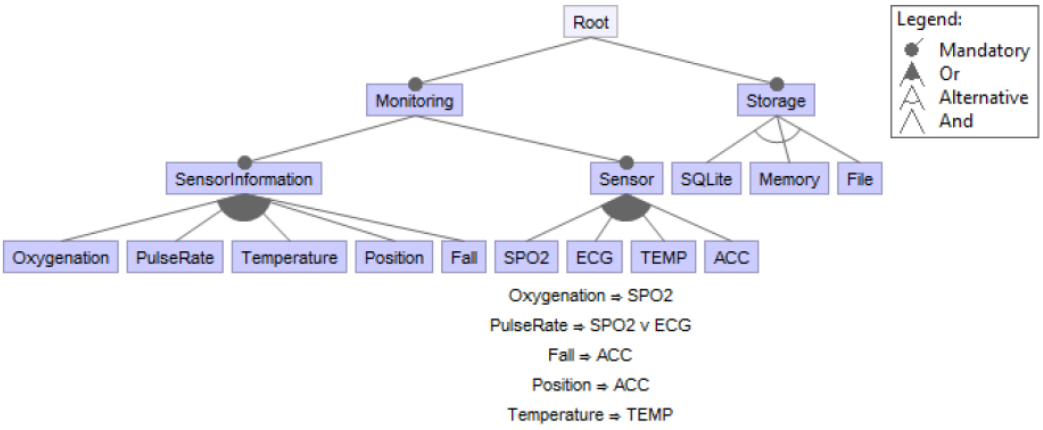


Figure 4.2: Body Sensor Network Feature Model [2].

the patient status was defined by a health expert and declaratively expressed in the aforementioned work [2]. Table 4.1 shows how the sensors' values relate to the patient's health risk. If at least one of the resources' measurements is at *high* range, the patient health status is defined as *high risk*. If the previous scenario does not happen, and if at least one of the resources' measurements is at *normal* range, the patient health status is defined as *normal risk*. On the other hand, if none of the aforementioned scenarios is active, the patient health status is defined as *low risk*.

Sensor Information	**Sensor Information Ranges**
Oxygenation:	$100 > \text{low} > 94 > \text{normal} > 90 > \text{high} > 0$
Pulse Rate:	$\text{high} > 120 > \text{low} > 80 > \text{high} > 0$
Temperature:	$50 > \text{high} > 38 > \text{normal} > 37 > \text{low} > 35 > \text{normal} > 30 > \text{high} > 0$
Fall:	if (Fall = 'yes') $\longrightarrow$ Patient's status = high risk

Table 4.1: Context operationalization for patient's status

The CGM presented in Figure 4.3 depicts the goals to be achieved by the Body Sensor Network, which is meant to detect emergencies on a patient based on its monitored data following works [2, 40]. The root goal is "G1: Detect Emergency", which is performed by the actor Body Sensor Network. The root goal is divided into two subgoals: "G2: Patient status is monitored" and "G3: Sampling rate is adjusted". G2 is refined to "G4: Vital signs are processed", and further, G4 is divided into other two subgoals: "G5: Vital signs are monitored" and "G6: vital signs are analysed". Such goals are then further decomposed, within the boundary of the BSN actor, to finally reach executable tasks. The only context present in the first version of the CGM is the aforementioned "Patient Status", represented by the IC (Initial Context) label and mapped to subtree of goal G3. The IC may assume three possible values, *low*, *normal* and *high risk*. The execution of the task T3 will depend on the current context value, that is, the higher the patient's risk, the lower the sampling interval must be.

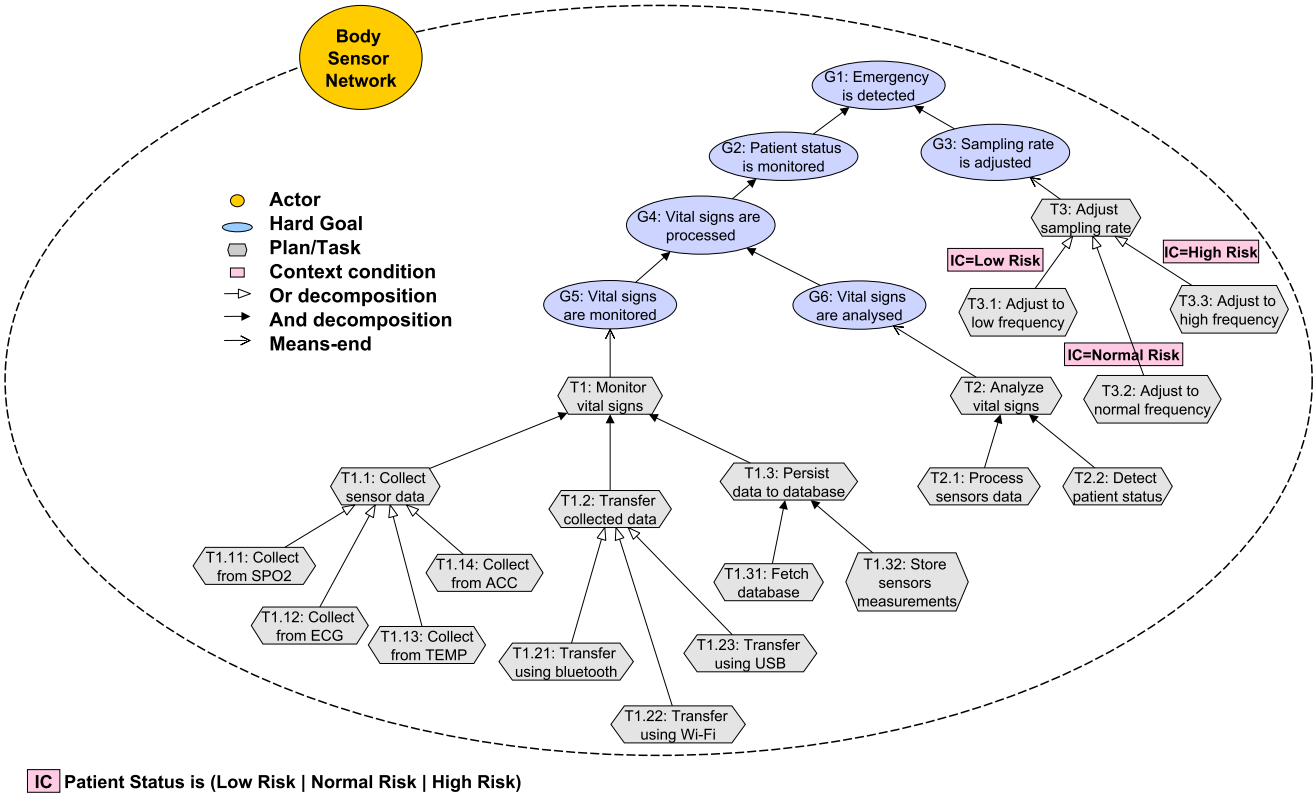


Figure 4.3: CGM for the BSN system

Considering the relationship between context and requirements, context can influence decisions about different aspects of the system [12]. Using the BSN case study to introduce these concepts, we have:

- **Requirements to meet:** if the context "patient is in a high risk state" is present, the processing unit has to speed up the sampling rate of the patient's vital signs, increasing the frequency of data gathering to guarantee the required satisfaction level. On the other hand, if the context "patient is in a low risk state" applies, the processing unit can slow down the sampling rate and even deactivate some resources to save battery.
- **Ways to meet requirements:** The BSN has several variants to identify the current patient health status, basically, it can be identified by any combination of the vital signs: blood oxygenation, pulse rate, temperature and fall. Each variant requires a set of valid contexts, such as "sensor SPO2 is available" and "sensor ECG is deactivated".
- **Quality of each way:** As we will notice on next chapters, different configurations of the BSN contexts (e.g. set of available sensors) provide different reliability levels. Knowing that, it is reasonable for a high risk state patient to require a configuration with a higher reliability level, while a patient in a low state risk can be more flexible in terms of observed contexts.

4.1 BSN Outline

Throughout this chapter he have described BSN, a medical device composed by four distinct sensors (accelerometer, electrocardiogram sensor, oximeter, and temperature sensor) whose objective is continuously monitor the vital signs of an individual adjusting its configuration according to the patient's health risk status, that can be classified into *low risk*, *normal*, or *high risk* category. The patient status is defined after the analysis of a set of conditions (oxygenation, pulse rate, temperature, fall detection), taking as base a operationalization list defined by a domain expert such as the one presented in Table 4.1. The structure and behavior of the BSN are depicted in Figure 4.3, which shows six goals, twenty tasks and a single context condition that triggers the system's adaptation. We have seen that such context can influence decisions about different aspects of the system, especially the (i) requirements to meet, (ii) ways to meet requirements, and (iii) quality of such ways. In the next chapter we will describe the approach itself, detailing how the contexts are categorized, and how the data mining process is integrated with the CGM to find points of interests that can potentially turn into new contexts.

Chapter 5

A Learning Process to Unveil Contexts for Dependability at Design Time

Our method aims at assisting the development process of dependable self-adaptive systems, by reducing as much as possible, at design time, the effects of unknown context variability. Figure 5.1 depicts the process. Based on the CGM structure in addition to the operationalization values of sensed information, we propose an analysis process using an adequate subset of data mining algorithms to extract a list of relevant contexts and their related variables, tasks and/or goals. As object of the mining process at design time, we create an execution dataset from a domain based simulation. Such operational information can be provided by: (i) external sources, e.g. execution logs, test cases, data sheets of similar systems; (ii) internal sources, e.g. data generated by a Monte Carlo simulation; or (iii) prototype version. Knowing the contexts and their operationalization, we reapply the data mining methods to provide a quantitative impact of the contexts interaction into the goal achievability, given some contextual constraints. A context analysis is then made in order to update the CGM by inserting the contextual constraints into their respective nodes, i.e., the tasks that have their operation modified by the specific context variability.

5.1 Contextual Requirements in Goal Modeling

In our work, we consider the system behavior is fulfilled by means of the goals tree structure of the CGM. Each goal may be further refined by child subgoals, which are ultimately realized by leaf-tasks. As such, each leaf-task of the CGM dynamically realizes its following parent goals following a finite state machine previously defined in [3]. The concept of context dependency and how it is related to the system resources and their variables is encompassed by our approach, and we propose an extension of the aforementioned state machine for such purpose.

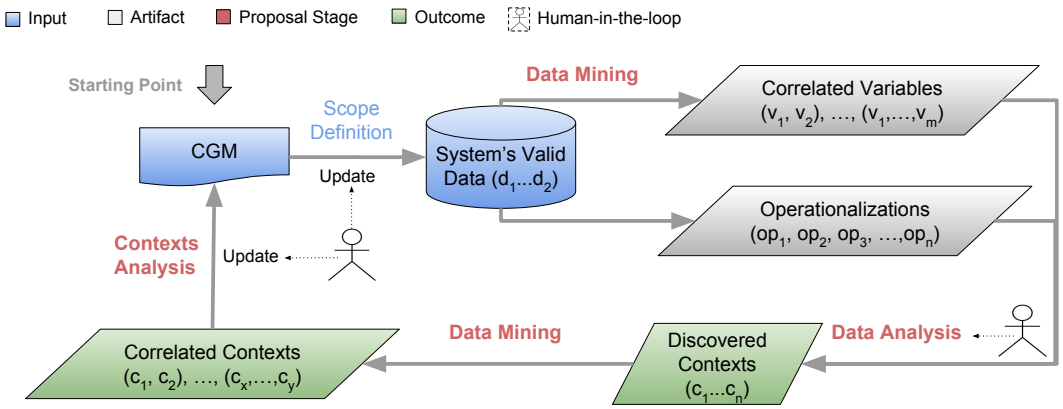


Figure 5.1: Process overview of our method

While some contexts in the CGM are directly related to the fulfilment of system goals, other contexts focus on meeting quality constraints [41]. We have adapted a finite state machine from GODA [3], a goal-oriented dependability analysis framework, to represent the operationalization of a CGM's task in face of different kinds of contexts, described in Figure 5.2. Our proposal introduces the path highlighted in red of the state machine, while the other paths had been already covered in [3]. From Figure 5.2 it can be noticed that, if there is a context that should be considered, its knowledge anticipation is paramount to the fulfilment of a task and therefore to its parent goals. An unsatisfied context hinders not only the execution of a task, but may also have a cascading error effect, compromising the fulfilment of the overall systems goal. In our work, we analyze not only the context implication from the local perspective of a task, but also from the global perspective of the tasks and their interactions towards the system's goal fulfilment.

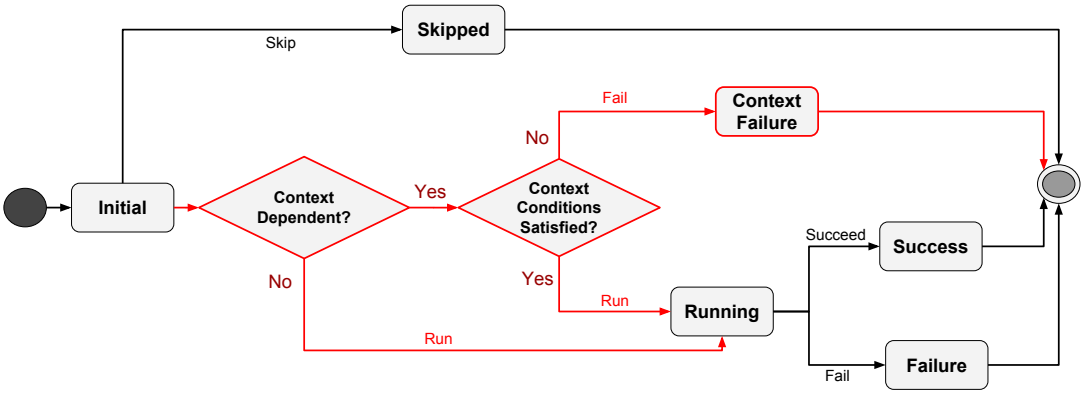


Figure 5.2: Overall behavior of CGM leaf-tasks (adapted from Mendonça et al. [3])

Initially, the system may try to execute or simply skip to the next task, changing its state to *Skipped*, following the task's behavior rule. If the system attempts to execute a task, it is verified whether the task operation depends on any kind of context configuration. If the task is

independent of context, the system proceeds to *Running* state. Otherwise, we check whether the context is satisfied or not in order to proceed to *Running* or *Context Failure*. Even if the system satisfies the context constraints and proceeds to a running state, the system may still have a minor probability of failure (*Failure* state). This chance is related to cyber-physical limitations and operational uncertainty, for instance, a routing device failing to send a message due to a layer of dust in its antenna. However, if the system does not present any operational issue, the task executes successfully.

If the task is not skipped, and fails to satisfy its context conditions we say that a *context failure* happened. Following Avizienis et al. domain failure classification [7], context failures may characterize either a *time* or a *data* or both domains. A time domain failure happens in our approach when, for example, a resource is unavailable at the time it is requested. A data domain failure happens when the requested data is out of the specified range of operation, characterizing, for example, a data integrity issue. We consider that safety may be also another dependability attribute related to both domain failure types for the context since a failure in the context characterizes a failure in the environment [7], which could render catastrophic consequences. Table 5.1 summarizes the levels of domain failure and their related dependability attributes.

Domain	**Attribute**	**Consequence**
Time	Availability Safety	If all sensors are not available then the patient's data cannot be collected
Data	Integrity Safety	If all monitored data are out of range then the patient status cannot be defined

Table 5.1: Domain failure classification and related dependability attributes

A context may assume one out of multiple possible values at a given moment and the environmental values that define each context may also vary through time. We call this *context variability*. Based on the context failure classification adopted, our method is able to deal with context variability by discovering new patterns and adjusting the valid ranges required for the operationalization of the contexts along time. On the other hand, our method also acknowledges the concept of *context information unavailability*, that indicates a situation in which the system is incapable of identifying the current context value due to the lack of information provided by the monitoring resources. Hence, context information that falls into this case are labeled as unavailable. The distinction between a context variability situation and a context information unavailability situation is made by the data analyst, with the pattern occurrence rate taken as indicative parameter.

In our method, the context conditions are divided into different categories following the context grammar defined in Listing 5.1. The types of context conditions depend on the resources involved in their operationalization and how they interact to do so. The conditions can be clas-

sified into: C_{range}, C_{fuzzy}, and $C_{boolean}$ classes. The operation submitted to a C_{range} condition expects a numeric value (integer or float) between the range specified in the task or goal to be considered apt to proceed to the running state (e.g. Oxygenation = $\{oxyg \in \Re \mid 0 \leq oxyg \leq 100\}$). There is also a possibility of an upper or lower bound range (e.g. *Battery level* ≥ 15). A task or goal under a C_{fuzzy} condition expects a specific enumeration that represents the context in a given moment (e.g. *Patient State* = "Low Risk" **or** "Normal Risk" **or** "High Risk"). Details of the fuzzy context approach can be found in [31]. Finally, the $C_{boolean}$ class is a context condition in which a variable can assume boolean values only (e.g. *USB is available* = "TRUE" **or** "FALSE"). Listing 5.1 specifies the context grammar addressing such context types adopted in our method.

Listing 5.1: Context grammar and syntax rules

```
<Context> ::= <Id> | <Expr>
<Expr>  ::= <Expr> <op_v> <Value> | <Expr> <op_r> <Range> | <Expr> <op_e> <Expr>
          |'(' Expr ')'
<op_r> ::= '<' | '<=' | '>' | '>='
<op_v> ::= '=' | '!='
<op_e> ::= '&' | '|'
<Value> ::= <Range> | <Fuzzy> | <Bool>
<Range> ::= <Int> | <Float>
<Fuzzy> ::= <Var>
<Bool> ::= ('TRUE'|'FALSE')
<Var> ::= ('a'..'z'|'A'..'Z')
<Int> ::= <Digit>|<Int>
<Float> ::= <Int>'.'<Int>
<Id> ::= <Var><Int> | <'!'><Var><Int>
<Digit> ::= '0'..'9'
```

In order to adjust the domain information to our application, we categorize the data into a resources type table describing the variables present in the CGM in accordance with our context grammar, described in Listing 5.1. Table 5.2 presents an excerpt of a type table and some illustrative variables applicable to the BSN domain. The column *Variable* represents the measurable aspects of a given resource that are involved somehow in the accomplishment of tasks and goals. The leaf nodes of the BSN's feature model (Figure 4.2) represent well some of those variables. The *Type* column in Table 5.2 refers to the computational abstraction and classification of an environmental resource. For instance, the availability of some devices (e.g. SPO2, ECG, TEMP, ACC, WiFi) are represented as a boolean data type. Meanwhile, to describe the individual's vital signs (e.g. Temperature, Pulse Rate, Oxygenation), a valid numerical range is defined following context operationalization of the BSN previously presented in Table 4.1.

There is also the fuzzy data type, which may represent, for instance, the strength of the WiFi signal or the patient health status in a higher level of abstraction.

Variable	Type	Value	Description
SPO2	Boolean	TRUE	SPO2 sensor is available
Temperature	Range	$0 \leq T \leq 50$	Patient's temperature is within the valid range
Wi-Fi Signal	Fuzzy	Strong	Wi-Fi signal level is strong
Patient Status	Fuzzy	Low Risk	Individual has a low health risk

Table 5.2: Example of a resource table and description of its variables

Independently of the context class, a failure of a context can be represented by the character '!', placed before the corresponding id. However, for each class there is a specific interpretation. For the boolean class, the reasoning is straightforward, i.e., $C_{boolean}$ means TRUE, while $!C_{boolean}$ means FALSE. The interpretation works the same way, for instance, $C_{boolean}$ might indicate the availability of a resource, while $!C_{boolean}$ indicates its unavailability. The process is analogue for the C_{range} class. Even though the context constraint consists in the expectation of a numerical value belonging or not to an specific range (e.g. $0 \leq oxyg \leq 100$), the practical implication converges to a binary decision, i.e., C_{range} represents the range constraint being respected (e.g. *oxyg* = 98), while $!C_{range}$ represents the violation of such constraint (e.g. *oxyg* = 103). Lastly, a C_{fuzzy} class can assume one out of multiple values (e.g. *Bluetooth signal strength* = "weak" or "normal" or "strong"). The failure of such context, i.e. $!C_{fuzzy}$, represents the absence of one of the predefined context possibilities, implying the unavailability of the resource or variable analyzed.

5.2 Data Mining Process

In a nutshell, having the system's CGM as input, the core of our approach relies on the data mining, discovery and analysis of new contexts that can lead the system to inconsistent states. In this section we provide a coarse grained stepwise perspective of our context mining process through Algorithm 1.

Starting the algorithm, it receives a CGM as input parameter. In addition, the resources syntax table, such as the illustrated Table 5.2, is also taken as input parameter, and contains a list of the system's resources with the respective variables name, data type and the possible value. In case of the actual runtime data or the data generated by the prototype is not being used, in line 3, the dataset is created, object in which will be executed the data mining process. Via the Monte Carlo method, the first step of the process is to generate the dataset (simdata) from a probability distribution of choice based on domain information[1]. For BSN, the Gaussian distribution was

[1] In case runtime data or data coming from similar systems is available, this step can be skipped.

Algorithm 1 ContextsMining

Input: ResourcesSyntaxTable resources_table, CGM cgm
Output: ContextList, ProbabilityReport

```
1:  CorrelatedVariables varlist ← NULL
2:  OperationalizationRules operlist ← NULL
3:  Dataset simdata ← resources_ranges.monteCarlo()
4:  for all n in cgm do
5:      varlist.push(simdata.associationRule(n))
6:      simdata ← simdata.filter(varlist)
7:      operlist.push(simdata.JRip(n), n.id)
8:  end for
9:  ContextList contexts←processRules(operlist)
10: for all records in simdata do
11:     records.replace(contexts)
12: end for
13: for all n in cgm do
14:     if contexts.getContextId(n.id) != NULL then
15:         cgm ← cgm.insert(contexts.getContextId(n.id))
16:     else
17:         return
18:     end if
19:     ContextList context_relations ← simdata.classify()
20:     ProbabilityReport probability_list ← simdata.getQuantitativeAnalysis(context_relations)
21: end for
```

adopted to represent most of the variables[2]. In line 4, we begin the CGM tree traversal to visit each CGM node (goal or task) and verify via data mining how the CGM nodes and its variables interact to successfully execute the parent tasks/goals, that is, which variable configurations of the lower tasks lead to system's goals fulfilment and which do not. We run through the CGM using a post-order depth-first search (DFS). In line 5 we apply Apriori [21] as an association rule method to detect correlated resources, which are updated into *varlist*. The adoption of Apriori in our approach is paramount to optimize the search space of the possible resources combinations that could be part of a new arising context (line 6). The Apriori helps to isolate the only variables involved in a particular routine. Thus, to discover contexts in this particular area, there is no need to carry unrelated variables to the next mining routine, bringing unnecessary complexity to the process. After defining the search space through the previous method, we apply a ruler method to quantify those correlations and generate the operationalization list (*operlist*). We use JRip algorithm [23] for such functionality (line 7). JRip is particularly useful to analyze resources represented by numerical values, like sensor measurements. Additionally, the current

[2]It should not be a threat to validity since there are plenty of well known probabilistic distributions available in the literature [42] suitable to create different arranges of datasets, each one semantically related to the respective domain.

node's id is also aggregated to the *operlist*. Through the aforementioned process, one will assist the CGM's update by simplifying the identification of the nodes in which the contexts conditions shall be inserted.

In line 9 we create a context list which corresponds to a contextual syntax table formed by a context id associated to the corresponding rules in *operlist*. Then, we update in line 11 the simulated dataset by replacing any occurrence of the rule by its corresponding context id present in *contexts*. This routine significantly aids visualizing the results on further steps.

Approaching the end, in lines 13 and 14, the CGM starts to be traversed again, verifying if the visited node has any associated context. If the node has a related context, the CGM is updated in line 15 and the contexts ids are inserted into their corresponding node. This part belongs to the semantic association step of the data mining process and relies on human interaction to link the discovered contexts with the corresponding goals through the CGM.

The next step is then to discover how different context combinations influence the success or failure of a CGM task or goal. This routine can be executed in parallel with the previous one (line 15), since the input artifact here is the dataset itself, modified only in line 11. In line 19, a classifier routine runs through either J48 [1] or JRip for such purpose. This step, combined with the one in line 20, provides a quantitative impact of the contexts interaction into the goal achievability, given the contextual constraints. This knowledge is useful to develop better adaptation plans, focusing on the system's most common and problematic operations.

Figure 5.3 exemplifies how the algorithm works for the transformation of the runtime or simulated data into useful information to define contexts with a model having the task T1.3 realized by tasks T1.31 and T1.32 following Figure 4.3. In the first step of illustrating the dynamics of the algorithm process, represented by (1) in Figure 5.3, is to apply the Association Rule method over the dataset in stage (a) of Figure 5.3 in order to discover the correlation between variables. It would be possible to verify, as described in (b), that variables like *Disk_Availability*, related to the leaf task T1.31, and *Disk_Space*, related to the task T1.32, work together to successfully achieve the parent task T1.3. The application of the Apriori method (1) enhances the efficiency of the JRip procedure (2), filtering only the attributes that are somehow related to the target node. Noticing the clear relation between the two variables and task T1.3, the three variables (T1.3, *Disk_Availability* and *Disk_Space*) are isolated and the ruler method JRip is applied over the specific attributes on step (2). The returned rules detail how these two variables relate with each other and how they cooperate to complete task T1.3. A possible rule generated by JRip could be translated to: "every time the storage device is unavailable, the disk space is unknown" or "every time the disk space is below 100MB, the task T1.3 fails". The rules and relations that imply in tasks' or goals' failures are counted during the process, generating a probability report as presented in artifact (c), pointing out to the likelihood of an event to occur. After the CGM traversal, the data analyst has enough material to study and come up with the most sensitive

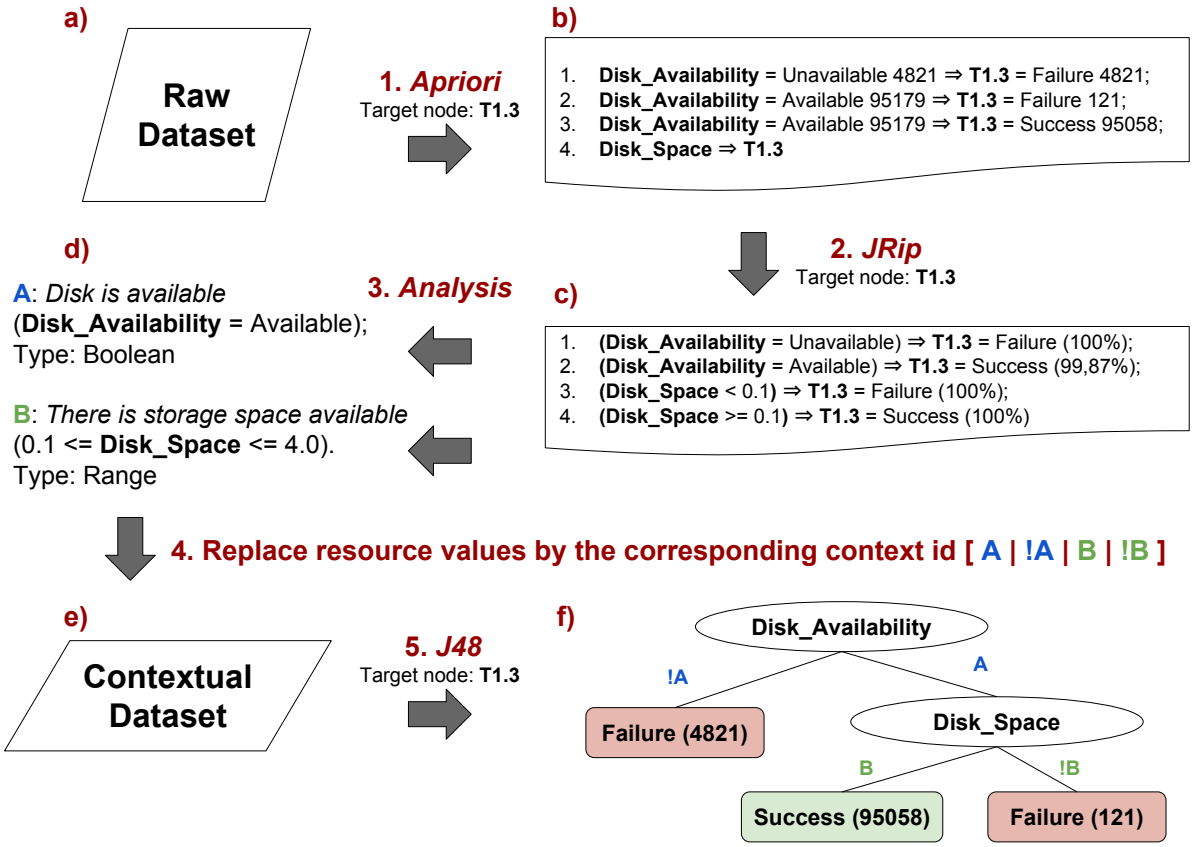


Figure 5.3: Data mining process for the persistence module described in Figure 5.4

variables and possible bottlenecks of the system. After the analysis in (3), the contextual syntax table (d) is produced to express the relation types of the newly discovered contexts. To illustrate, a record of a syntax table could be expressed as: B (*context id*); Boolean (*context type*); "There is storage space available" (*context description*); **If** Disk_Space $\geq$ 100MB **and** Disk_Space $\leq$ 4GB **then** B is true, **else** B is false (*contextual operationalization rules*). The contexts ids are placed by the analyst in the corresponding refinements of the CGM. Meanwhile, in step (4) the rules that represent a given context in the dataset are also replaced by the respective context id. For example, in every record of the simulated dataset where the disk has no storage available, i.e., the Disk_Space value is below the defined threshold, the corresponding resource value is replaced by the tag '!B'. The same happens to every other context present in the context list. At last, having a dataset only in terms of contexts ids (e), it is possible to advance to step (5) and apply J48 to create a decision tree and discover how the possible context variants behave in terms of different aspects of the system in step (5) of Figure 5.3. The decision tree in (f) indicates how the variables Disk_Space and Disk_Availability relates to the contexts 'A' and 'B' to fulfil the target node T1.3. Analyzing the decision tree (f), one can notice a red leaf box represented by *Failure (4821)*, which signalizes the fail rate of T1.3 due to unavailability of the

storage device, including almost 4.8% of the total amount. The second red box, *Failure (121)*, indicates that the task T1.3 fails due to lack of storage space in approximately 0.12% of the executions. On the other hand, the green box labeled as *Success (95058)* shows the success rate of task T1.3 when both contexts are satisfied, representing approximately 95.06% of the records. Figure 5.4, illustrates the refinement of task T1.3 (an excerpt of Figure 4.3), which persists the data of the BSN sensors information either on a database or on a file system storage.

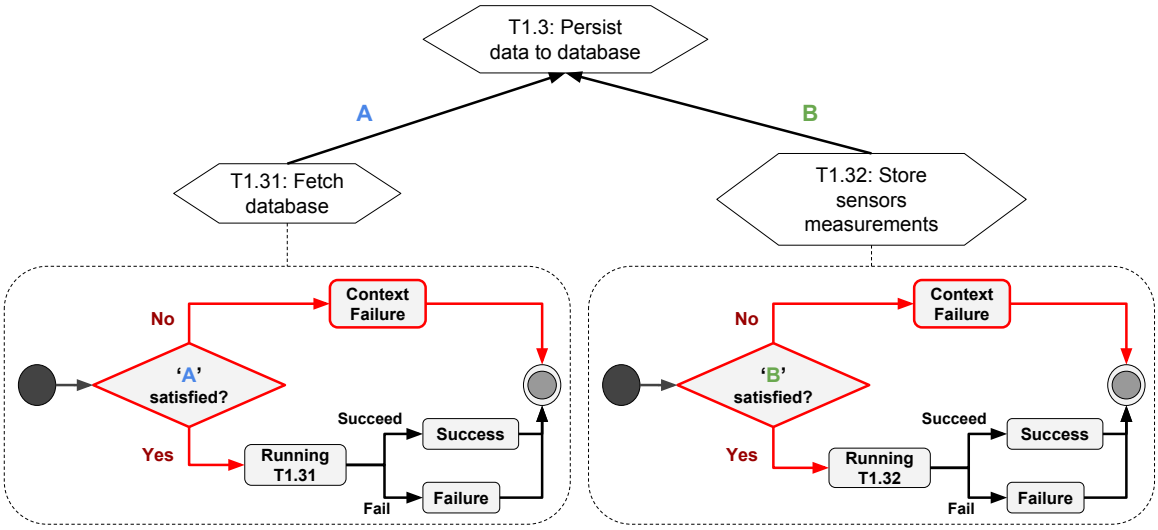


Figure 5.4: BSN's CGM excerpt with associated variables (Adapted from Figure 4.3)

5.3 Proposal Overview

We have started this chapter explaining the general structure of the feedback loop that depicts our approach. The CGM structure combined with the operationalization values of sensed information are the basis of the analysis process. We use an adequate subset of data mining algorithms to extract a list of relevant contexts and their related variables, tasks and/or goals. We have proposed an extension for the finite state machine from GODA, in a way that each task is subject to a context verification before its execution. The violation of a context constraint can be considered either a time domain failure, or a data domain failure. In the scope of our work, the contexts were categorized into C_$range$, C_$fuzzy$, and C_$boolean$. Later on the chapter we have presented the algorithm of context mining, and offered a brief example of the method applicability. In the next chapter we evaluate our method in terms of reliability and scalability of the data mining process, and w.r.t. the method's contribution itself.

Chapter 6

Evaluation

In this chapter we evaluate our approach by means of an adaptation of the Goal-Question-Metric (GQM) methodology [43]. The adaptation of the GQM plan consists in the addition of a new column containing some general results. The questions that are relevant to evaluate our approach and its results are divided into two major parts: one regarding the algorithms used in the data mining process itself and the other regarding the efficacy of the method as a whole. The research questions are detailed in Table 6.1.

Goal 1: Data mining process		
Question	Metric	Results
1.1 How reliable are the answers provided?	Precision and Recall Rates	Precision: 0.969 Recall: 0.968
1.2 How does the data mining scale over the amount of records and simulated resources?	Execution time	Upper bound: 10 seconds (100,000 records)

Goal 2: Method's contribution		
Question	Metric	Results
2.1 Can our approach substantially identify new contexts related to dependability for the BSN?	Number of new contexts.	17 new contexts
2.2 What is the impact of the newly identified contexts on the BSN goals' satisfaction ?	% reliability satisfaction for levels of patient health risks.	Low Risk 100%; Normal Risk 27%; High Risk 16%.

Table 6.1: GQM plan

6.1 Experimental Setup

We evaluate our approach on the BSN case study, where the *Patient Status* is taken as the initial context and act as a trigger to activate the adaptation plans. Our approach intends to derive new

contexts from the combination of this specific context and the system goals. The patient status can be defined by processing the data gathered from sensors in different configurations.

For the operationalization of the patient status, there are three possible risk states: *low*, *normal* or *high*. The objective of the BSN is to adjust the sampling rate (data gathering of patient's measurements) based on the patient's status and its required reliability, according to [44, 2]. The initial adaptation rules are defined as follows: if the patient presents a high risk status, his vital signs shall be checked every minute in order to ensure his safety. If the patient status is processed as normal risk, the sensor data collecting rate must be five minutes. At last, if the patient is in a low risk state, a verification of his vital signs every fifteen minutes is enough to guarantee his safety.

To demonstrate the applicability of our approach, we developed the BSN prototype, after building its CGM previously presented in Figure 4.3, having only the patient status as its initial context. The prototype was developed in Python and the experiments were processed in an Intel Core i7 5500U, 2.4GHz, 6GB DDR3. The prototype assisted in simulating the data sent by the sensing devices to the processing unit. The processing unit works as an autonomous controller, processing the simulated measurements defining a health status according to the policies. The adaptation plans adopted after the processing step are sent to the effectors, that manages the user's vital signs sampling rate according to his status. Table 6.2 shows the variables simulated by the BSN prototype and the respective data type.

Simulated Resource	Domain	Data Type
SPO2	Physical Availability	Available / Unavailable
	Logical Availability	Available / Unavailable
	Oxygenation	integer: gauss(97,3)
ECG	Physical Availability	Available / Unavailable
	Logical Availability	Available / Unavailable
	Pulse Rate	integer: gauss(100,10)
TEMP	Physical Availability	Available / Unavailable
	Logical Availability	Available / Unavailable
	Temperature	integer: gauss(36,0.8)
ACC	Physical Availability	Available / Unavailable
	Logical Availability	Available / Unavailable
	Fall	Yes / No
Bluetooth	Signal Level	Strong / Average / Weak / Unavailable
Wi-Fi	Signal Level	Strong / Average / Weak / Unavailable
USB	Physical Availability	Available / Unavailable
Storage Device	Physical Availability	Available / Unavailable
	Storage Space	float:[0.00 - 4.00]
Battery	Charge Level	Integer:[0 - 100]
Patient Status	Health Risk	Low Risk / Normal Risk / High Risk / Unmapped

Table 6.2: Example of the resources on BSN simulation

After setting the prototype resources and the operation logic, we applied the Monte Carlo Method. The operation range of each resource is analyzed and taken as parameter to support its simulation. Then, following the adequate distribution for each variable, the database that will be used on data analysis stage is created. For instance, one could use the Monte Carlo method allied with the Gaussian distribution to simulate the individual's heart beat data, mimicking a working pulse sensor over time. The variable value associated to each resource arise from a simulation of 100,000 records. A record is a line of the dataset, that contains the resources values in a given moment, like a snapshot of the system status at the time. In this case, the necessary elements to identify the patient's health status. Each record is composed by a set of different vital signs measurements of a fictitious person based on a well known patient guide [45], following the work reported in [44] and valid sets of resources configurations available online[1]. The undesired behaviors of the simulated sensors were classified in two groups: unavailability (unexpected interruption of the component's operation) and outlier detection (monitored data out of expected range). This classification helps the mining process to identify how the contexts influence the occurrences of undesired behaviors. For the purpose of replicating our evaluation, all the artifacts of this experiment are available at a GitHub repository[2].

6.2 Goal 1: Data mining process

Question 1.1: How reliable are the answers provided? Weka [24] was the selected tool to assist the analysis of our case study. Before starting a data mining procedure we must firstly verify whether the techniques applied are adequate and the data preprocessing was done correctly or not. We focus on three pillars: (i) accuracy, that represents the prediction model efficiency in correlating an outcome with the attributes in the provided data; (ii) reliability, that represents how well the prediction model will behave in face of different datasets; and (iii) usefulness, that tells the applicability of a given method in extracting useful information from a dataset. Summarizing, the data mining stage begins with the validation of both, the generated data and the data mining techniques that we should apply. To measure the accuracy of the data mining process, we take into consideration metrics as precision and recall. As means to quantify the reliability of the answers provided we executed the prototype ten times, generating ten different datasets to be analyzed. In order to measure usefulness, we applied five known classification methods (J48, JRip, One-R, Decision Table and Bayes Net) for each dataset.

We chose the operationalization of *Patient State context* as the prediction object for the classifying process. Each dataset has one hundred thousand records. For this test, the chosen resources to represent the record was: <*Oxygenation*, *Pulse Rate*, *Temperature*, *Position*, *Fall*,

[1] https://code.google.com/p/spl-model-checking/

[2] https://github.com/ArthurJRF/UnB-Dissertation-Artifacts.git

Patient State>. The training and testing phases were based on a ten fold Cross-Validation, a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it. As we can see by the results demonstrated in Table 6.3, all tested methods performed well in terms of precision and recall. Although the adopted algorithms (J48 and JRip) performed very similarly in terms of precision and recall in comparison with the Decision Table and Bayes Net, the J48 and JRip provide us fundamental tools for our method, i.e. the decision tree and the operationalization rules. Our false-positive(fp) and false-negative(fn) results are described in terms of precision ($tp/tp+fp$) and recall ($tp/tp+fn$). The minimum support and minimum confidence level defined for Apriori algorithm was 0.55 and 0.9 respectively. Picking the appropriate values for support and confidence is a tricky job, given that it depends on the domain knowledge. Since we are dealing with a medical application, we empirically established a balance of support and confidence value in order to get an adequate set of new rules. For instance, if the support and confidence values are too high, very few or no rules will be returned by the association rule method, and one might lose useful insights. On the other hand, if the values are low, the algorithm will probably return many rules, however it is likely that the majority of them will not aggregate any helpful information.

		J48	**JRip**	One-R	Decision Table	Bayes Net
Precision	Mean	**0.969**	**0.969**	0.850	0.969	0.969
	StdDev	**5.6765E-04**	**5.1640E-04**	1.595E-03	5.6765E-04	5.1640E-04
Recall	Mean	**0.968**	**0.968**	0.828	0.968	0.968
	StdDev	**5.6765E-04**	**6.7495E-04**	1.912E-03	5.6765E-04	6.7495E-04

Table 6.3: False positive and false negative analysis of the generated prediction models

Considering that we are dealing with a medical system, the low rate of false negative and false positive is crucial for a good classifier. Running this classification routines as a preliminary validation, we verified a very low rate of false positives and negatives, proportionally speaking. We also can state that we achieved satisfactory marks (the worst prediction model has precision and recall values above 96%), considering the non-deterministic property of some variables. Such marks indicate that the utilization of these data mining routines is significantly more valuable than a random guess or a naïve approach to extract information from the dataset.

Question 1.2: How does the data mining scale over the amount of records and simulated resources? A scalability test was applied utilizing the J48 and JRIP, since they have rendered more suitable for the BSN domain following the analysis in Question 1.1. We noticed that there are two possible ways to affect the time consumption of the data mining process. The first one is related to the increasing number of records that shall be analyzed simultaneously. Figure 6.1a presents the outputs of our scalability test. We defined the amount of six simulated variables responsible for the operationalization of the *Patient Status context* (Oxygenation, Pulse Rate,

Temperature, Position, Fall and Patient State), and varied the number of records, measuring the average time of ten replications for each configuration.

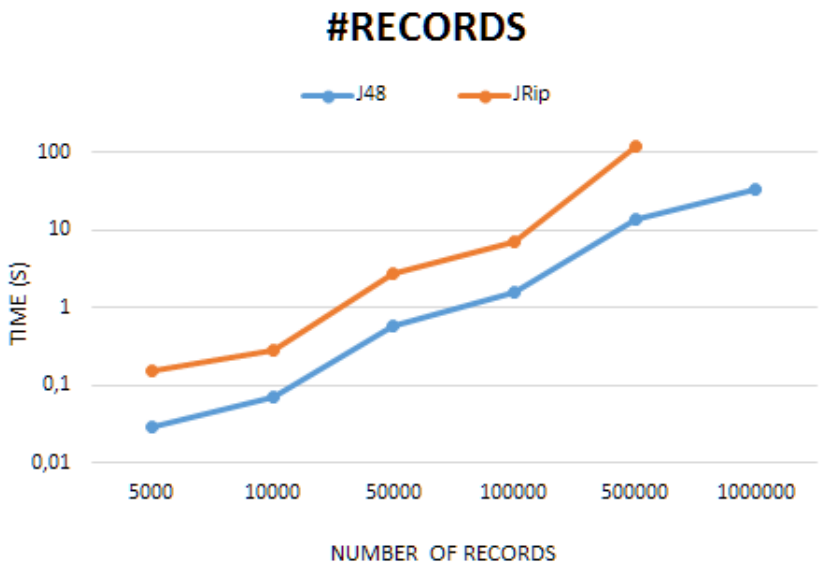
The second parameter that may influence the performance of the mining process is the number of resources (e.g. sensors, battery, storage) in each record. Figure 6.1b shows a scalability test on the number of resources necessary to operationalize a single context, i.e., the number of different sensors measurements involved in the generation a context. Since we focused on sensitiveness to resource variability on this part of the analysis, we fixed the dataset size to 100,000 records, and we varied the number of resources, conducting the experiment ten times for each configuration. We chose tasks T1.1, T1.2 and T1.3 in Figure 4.3 to illustrate this analysis, since these tasks encompass the majority of resources and represent the core of system's operation.

Results in Figure 6.1 show that, in terms of time scalability, the mining process is slightly more sensitive to variation on records amount than to variation on resources amount, since the time variation coefficient of the former is higher in comparison to the latter. Nevertheless, based on the obtained results, the data mining time process shall not be a problem, considering the BSN domain and the considerable amount of 100,000 records exercised. In fact, considering that even for records as big as six orders of magnitude, our analysis showed that the results were processed under two minutes. Even though the JRip processing time extrapolates 100 seconds for 10^6 records, the actual result is approximately four minutes. The capability to afford this processing time will depend on the application.

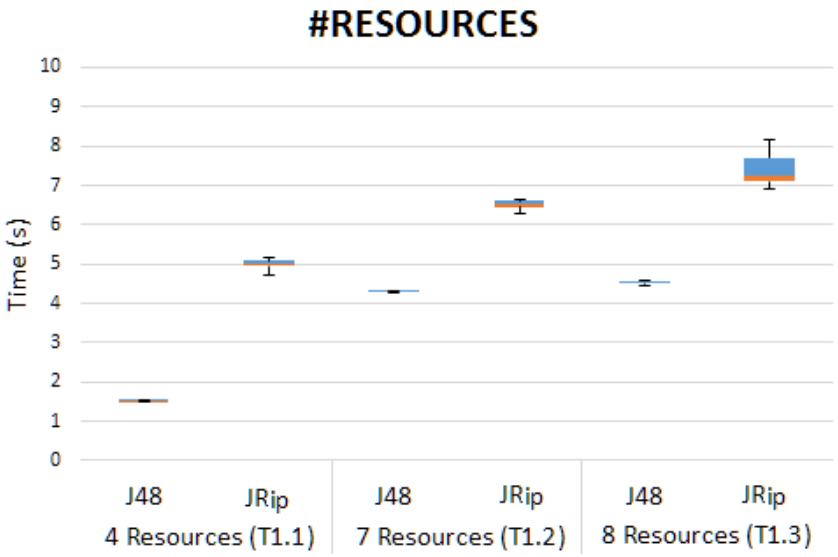
6.3 Goal 2: Method's contribution

Question 2.1: Can our approach substantially identify new contexts related to dependability for the BSN? Figure 6.2 presents the improved CGM, i.e. the BSN with new context constraints discovered in the process. The new elicited contexts are those from C1 to C17. Therefore, our method supported the identification and operationalization of 17 new contexts for BSN not present in previous published results of the BSN [2, 44]. The initial conception of the BSN had only the initial context, that after the method became C18: *Patient Status*, ruler of the system adaptation. Given this improvement, it is possible to state that we successfully met the initial intention of giving a starting point to the software architect in the identification of the contexts to which the system may be exposed, taking into account the ranges that could render undependable contexts. In the next question we explore how important such contexts were to validate if the BSN fulfils its goals for varying patients' health risks.

Question 2.2 What is the impact of the newly identified contexts on the BSN goals' satisfaction? In real scenarios, the achievement of the main goal is useless if the quality constraint is not satisfied. On the BSN, its major quality constraint is its reliability, where each set of



(a) Number of Records *vs* Time



(b) Number of Resources *vs* Time

Figure 6.1: Prediction model building time for different amount of records and resources

resources configurations has a reliability level. And each patient's health risk state must have a one or few possible resource configuration. Following reported data[3], we set the minimum reliability level required for each patient state as: 95% for low risk, 96% for normal risk and 97% for high risk. Considering the reliability reported data ranges from approximately 95% to 99%, our set of reliability ranges means that at most 5% probability of failure for the resources is acceptable for the low risk patient, at most 4% probability of failure is acceptable for the

[3]https://code.google.com/p/spl-model-checking/

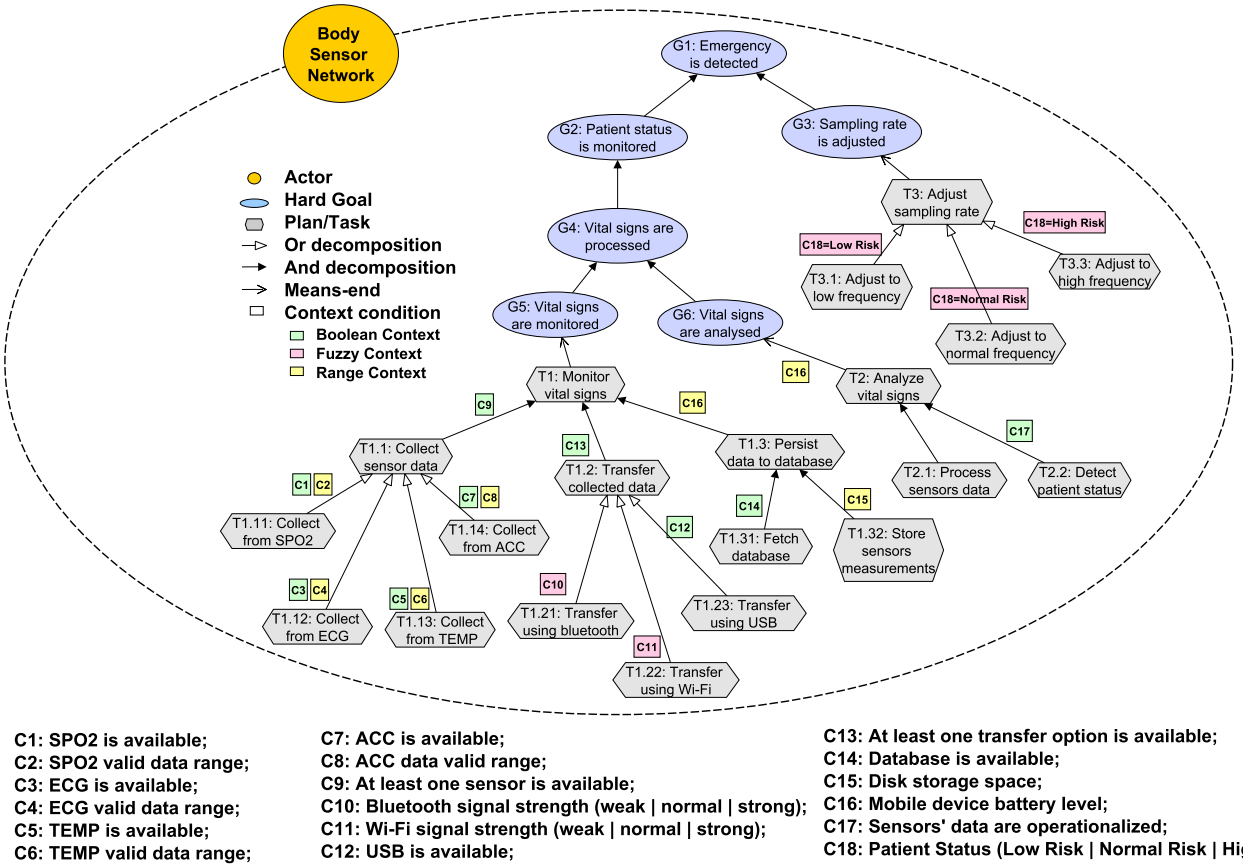


Figure 6.2: CGM for the BSN system with detected contexts

medium risk and at most 3% probability of failure is acceptable for the high risk patient. We should note that our choice for such reliability ranges is based on the reported reliability ranges and should not represent a threat to validity since it could be adjusted to vary in accordance with the domain expert requirement. In the BSN case study, such resources are reported as features following a Software Product Line approach. For example, a configuration <!SP02, TEMP, ECG, ACC, Pos, !Oxy, Fall, PulseRate, Temperature>, that has 95.31% of reliability, means that all resources and variables but SPO2 and Oxygenation are present and its reliability level would suffice to satisfy the quality constraint of a patient in a low risk state. However, it wouldn't meet our threshold for the required reliability level of a normal or high risk state patient.

In Figure 6.3 we report the results on the percentage of patients' health risk state that are either satisfied or not considering the elicited contexts for the BSN following our method for 100,000 patient records where 8.8% of the records fall in the profile of high risk, 25.7% of normal risk and 62.8% of low risk. Moreover, 2.7% of the records identified critical operational failure in the system, which would render a crash of the BSN for all patients' health risk. Results

depicted in Figure 6.3 show the alarming fact that only the low risk profiles would be completely satisfied, while almost 84% of high risk profiles and 73% of normal risk profiles would be violated (not attended). In other words, the high percentage of violations on both high and normal risk states allows us to infer that our elicited contexts do have a significant impact to identify critical situations, which was not previously taken into consideration on the BSN works. Figure 6.3 also shows the non-identified state of some quality constraints, regarding its satisfaction or violation. The amount of non-identified states corresponds to the 4.62% of the total executions and it is related to the uncertainty of the prediction model, inherent to the data mining method.

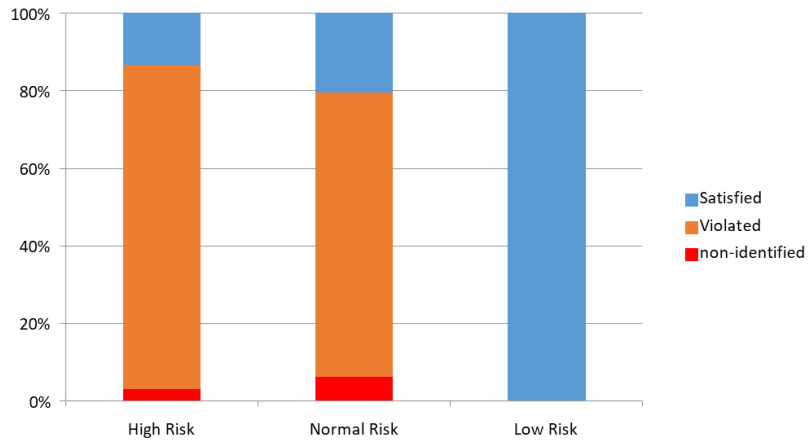


Figure 6.3: Impact on the BSN goal satisfaction for patient's high, normal and low risk states under the contexts discovered through our data mining approach over 100,000 records.

Through the decision tree presented in Figure 6.4, as an outcome of the J48 algorithm, we are able to see the previous analysis from a different perspective, aggregating some knowledge that was not explicit before. The orange box, labeled *Oper. Failure*, illustrates an unavailable context information, indicating that the system had no information enough to characterize the context as one of the three possible patient states. The leaf nodes were colored differently to distinguish the context configurations that, for a specific patient risk, lead to the fulfilment of the reliability threshold (green) from those that does not achieve the required level (red). The numbers in the leaf nodes refer to the amount of records found in the dataset in such context configuration. The nodes with two numerical values, refer to the true positive amount (node's left-hand side) and false positive amount (node's right-hand side), generated by the prediction model. The false negative values are related to the amount of non-identified states mentioned before.

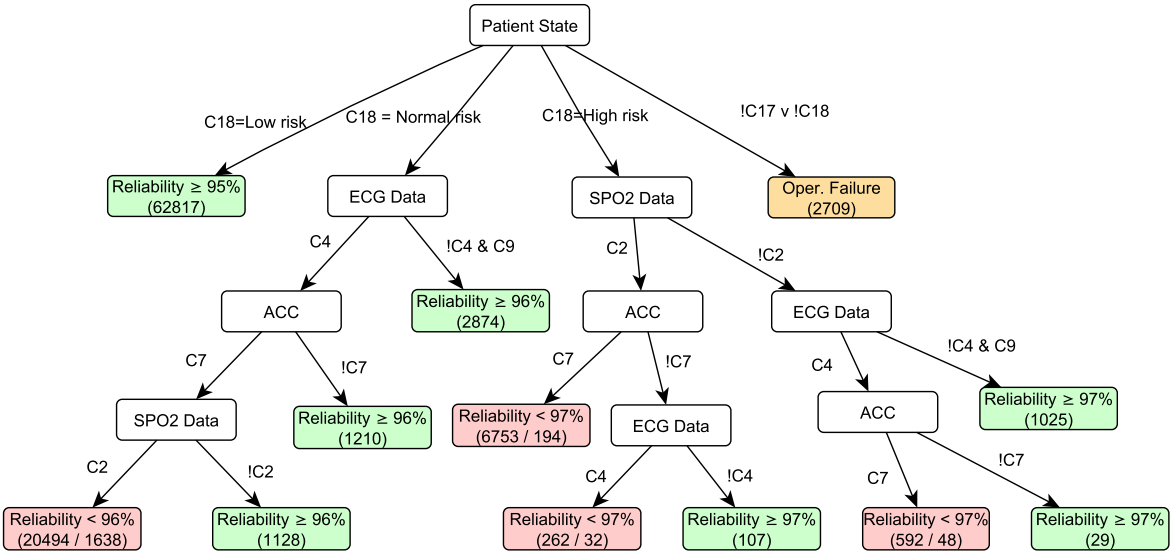


Figure 6.4: Decision tree displaying the influence of some contexts in the achievement of quality constraints

6.4 Discussion

Although there are some tools and methods (e.g. model checkers) that provide us the reachability value of every modeled path in a system, these methods disregard fundamental aspects of runtime behavior that are tightly related to the fulfillment of self-adaptive system goals, such as context conditions or distinct user profiles. The data mining process proposed in our work comes to fill this gap, defining the scope of interest and limiting the system analysis to the practical scope of its use. The method manages the analysis considering distinct user profiles, context variability, and the presence of new contexts.

The decision tree, presented in Figure 6.4, is used to show the contrast between the initial reliability analysis, through a probabilistic model checker, and the characteristics of the system in a real scenario, considering different user profiles, each one requiring a distinct reliability level to be satisfied. Statically, one could estimate the reliability of a system ignoring the user profiles or contexts inherent to runtime. However, such analysis is almost useless when it is necessary to analyze the system behavior in a given profile of use and/or under a specific context of operation. That is the real gain of applying a data mining process over the prototype operation.

A model checking process or even a SAT solver could provide us a generic reachability value of a given goal, disregarding contexts and user profiles. On the other hand, using the BSN case study to illustrate, through our method we are able to identify that, for a patient diagnosed with a severe heart disease, in a high risk profile in most part of the time, the ECG sensor is the most relevant device, and its dependability shall be prioritized. On the other hand, for a patient

with occasional blood oxygenation falls, that is in a low risk state most part of the time, the ECG sensor is the most sensitive. Despite the fact that the individual reliability of each sensor is 99.9%, the reachability of the goals and the satisfaction level of each patient will depend on the current context configuration, i.e., the active sensors and their corresponding valid data.

The quantitative analysis and the reliability values found in the present work are based on a parametric formula embedded on the BSN prototype. Each context represents a variable that can be replaced in the formula. For each record, the prototype verifies the current configuration, replaces the variables in the formula, and calculate the reliability. The current reliability is then compared with the required threshold for that specific patient status. The verification is made for each record, and the data to generate the decision tree is provided. The decision tree encompasses the reliability distribution according to the user profile, defining the contexts and resources that are relevant to that user and the configurations that are not, being useful to create a dependable adaptation policy for each situation.

Before we conclude, there is another topic that is worth discussing and still needs some clarification: the relevance of context elicitation for decision making in face of uncertainty. As mentioned before, not identifying some contexts still at early stages of software development can make the system prone to uncertainty. In possession of the new contexts and the analysis of their impact in the system's goals, the analyst has a sound source of information to support the decision making process and to adequately plan the system adaptation. In this sense, some action shall be taken to maintain the quality of the service provided and avoid some undesired events. To illustrate such actions we will take the example previously defined in Figures 5.3 and 5.4 and the uncertainty taxonomy proposed by Ramirez et al. [8] to demonstrate how the present work supports the decision making process and the policy specification in different stages of operation: requirements, design and runtime. We explain how to address each of these stages as follows.

Requirements: We have learned that the disk availability and storage space contexts are determinant to the success of the root goal. We also noted some missing requirements in the persistence module. It can be natural to conclude by pure reasoning that, in order to store data into a database, the storage device must be available (context A). However, it can be tricky to have the same insight for other situations. The process showed us that the size of the data can oscillate through the iterations, hence, to avoid the system to run out of space (failure of context B), it is necessary to define a threshold (e.g. 100 MB) to trigger a contingency routine that, for instance, frees the memory by erasing old data. The data mining process combined with the CGM enables the creation of an objective criteria, minimizing ambiguity and dubious interpretations of the stakeholder needs.

The elicitation of contexts relying on a combinatorial analysis is not a feasible alternative. The same for SAT solvers. Although the context conditions can be described by propositional logic formulas, their representation would not be intuitive. For instance, while it takes only one variable to express the patient risk state (e.g. *Patient State* = "Low Risk" **or** "Normal Risk" **or** "High Risk") in a decision tree, a more complex notation is required through a boolean representation $[(x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3)]$ where x_1, x_2 and x_3 correspond to Low Risk, Normal Risk and High Risk respectively. Such representation would not be of any use to the designer in complex systems. Another reason why SAT solvers are not suitable to this kind of analysis is due to the nature of some operations. For instance, in some cases, a weak Wi-Fi signal can be enough to satisfy a given goal. However, for the same configuration in a different moment, a weak Wi-Fi could not be enough. Via data mining we can represent such situations using the true positives and false positives rates. On the other hand, in a boolean satisfiability problem such fact could lead to mathematical contradiction (e.g. $x_i \wedge (\neg x_i)$) due to the non-deterministic nature of the operation.

Design: The knowledge unveiled by the data mining process over the prototype data enables one to identify unexplored design alternatives and adjust the system implementation accordingly. The method can also assist the analyst in discarding irrelevant design decisions with respect to requirements. The analysis revealed that, if the context A fails, there is no point in execute the following task (T1.32) and try to store the data. It is more reasonable to retry the access to the storage device within an interval Δt, without worrying about the storage mode (SQL, memory or file) since they make no difference from the requirements perspective.

Evidently, the analyst relies on the SLA (Service Level Agreement) to calculate the trade-off and decide whether is worth to tackle the contextual failures. For instance, if the failure tolerance of the system is ρ = 0.2%, a contextual fault tolerance would have to be adopted to avoid disk unavailability, that has a fail rate of 4.82%. On the other hand, individually speaking, it would not be necessary to tackle the contextual failure related to lack of storage space, since its fail rate is equal to 0.12%.

Another use for the combination data mining / contextual goal modeling is the constant verifiability of the design regarding the satisfaction of the requirements. The majority of inadequate implementations can be exposed through a simulation, by forcing certain variables combinations that are very unlike to occur in a real scenario. Such execution variants can initiate unknown behaviors that should not be allowed. On the other hand, many unpredicted events are often labeled as exception paths, when in fact, they are alternative ways to achieve a requirement. All these knowledge can be useful for the designer at modeling stage.

Runtime: The unpredictability of the environment is, perhaps, the hardest uncertainty source to tackle. It is a fact that many events and conditions in the environment cannot be anticipated. To this matter, our proposal can be useful in the sense of giving a starting point to the analyst, guiding the conjecture of the possible causes for each discovered context. For instance, the failure of context A indicates the unavailability of the storage device. Based on that, the analyst can apply other elicitation techniques, such as wide-field ethnography, allied to the requirements information to come up with possible reasons (e.g. power outage, data center flood, fire, etc.) that can lead the storage device to the degraded state.

Finally, the method can be useful even at runtime for decision planing. Although problems such as sensing failure, noise, imprecision or inaccuracy are usually caused by physical limitations, the method proposed in our work assists the identification of such bottlenecks and, abstracting the causes, can base the response policy for whenever such problems take place. To illustrate, the rate of false positives and false negatives returned by the data mining process can be a strong indicator of divergence between a measured value and its real value.

6.5 Threats to validity

Construct validity – In order to assure we provide a reliable and sound input data for our evaluation, we relied on a reported sound case study (BSN) and its published available data. In addition, the generated input information for the patients record used well-established Monte Carlo simulation method on the reported input data. Despite all the care we took to avoid the generation of unrealistic data, we cannot guarantee that the generated data will perfectly represent a real scenario. Further study must be done in order to verify such representation.

Internal validity – In our evaluation, based on the published information of the BSN we were able to model not only the contextual goal model that reflects the BSN architecture goals but also to vary the sensed data and perceive a significant impact on the outcomes. Therefore, the modeling structure showed itself efficient to adequately evaluate our approach and identify new relevant contexts for the BSN. However, unveiling all the contexts involved in a system's operation is inherently NP-complete, which could represent a threat to the completeness of the elicited contexts and consequently to the overall dependability of the system.

External validity – Although our approach is not tailored to be domain specific, we do reckon the limitation of the evaluation since it was applied in the specific case of the BSN. Further evaluation of the approach must be performed to evaluate the actual applicability of our approach for generalization purposes.

Chapter 7

Conclusion and Future Work

In this work we have illustrated a preventive approach that assists the unveiling of contexts at early stage of software development. The concept of context is becoming a important part of the requirements definition for self-adaptive systems through the use of so called contextual requirements. Missing out to identify important context conditions early on in the development life cycle leads to missing requirements that might become very important for the satisfaction of user needs. Moreover, failing in predicting contexts, at design time, might be catastrophic for the system's dependability when it is being executed, hindering the assurance of attributes such as reliability, availability, safety, security and maintainability.

In order to avoid a combinatorial exploration, typical of design time verification approaches, the use of data mining methods over the system's prototype was quite useful to reduce the search space formed by the countless combinations of environmental and computational resources. We evaluated the proposal on the previously published BSN case study and, we were able to verify a significant increase on the mapped contexts amount, before the implementation stage. As a result, the new and relevant elicited contexts were indispensable to prevent relevant system goals to render fulfilled.

Further research is necessary to generalize the applicability of our approach to other systems as well, expanding the reach to other domains. For future analysis, we plan to quantitatively analyze the impact and and criticality of new contexts from the system's dependability perspective. Self-adaptive systems can highly benefit from this kind of analysis, guiding its adaptation by the reports generated by our method. Some improvements for this proposal and potential new contributions regarding the early applicability of learning methods to assure dependability are planned for future work, more specifically:

- **Support the assurance for real-time self adaptive systems:** the ability of the process described in our work to discover hidden patterns in data can potentially assist the identification and quantification of the impact that some contexts have on the satisfaction of real-time constraints in self-adaptive systems. Such study could help us to validate prop-

erties defined still at design time, contributing to minimize the research gap related to the model checkers' limitation in analyzing the system's behavior considering contextual conditions.

- **Integration with GODA:** complementing the previous item, we intend to integrate our method with GODA framework and then, taking contexts into consideration, estimate the reachability of the system goals in face of context variability via model verification.
- **Analyze user profiles as contexts:** we want to verify the feasibility of adapting our method to enlighten design decision with a view to users' satisfaction, instead of focusing exclusively on dependability aspects. Since the users can achieve their objectives through different paths in distinct contexts, we want to verify the applicability of a prediction model to, based on the users' profiles knowledge, estimate the context conditions in which the user satisfaction would be maximized.
- **Unsupervised learning to improve the method:** finally, we aim at the exploration of different learning techniques, especially unsupervised such as clustering, to enhance the context unveiling process and improve the pattern discovery capability.

Reference List

[1] Quinlan, J.R.: Induction of decision trees. Mach. Learn. 1(1), 81–106 (Mar 1986) vii, viii, 12, 13, 31

[2] Pessoa, L., Fernandes, P., Castro, T., Alves, V., Rodrigues, G.N., Carvalho, H.: Building reliable and maintainable dynamic software product lines: An investigation in the body sensor network domain. Information & Software Technology 86, 54–70 (2017) vii, 4, 21, 22, 35, 38

[3] Mendonça, D.F., Rodrigues, G.N., Ali, R., Alves, V., Baresi, L.: GODA: A goal-oriented requirements engineering framework for runtime dependability analysis. Information & Software Technology 80, 245–264 (2016) vii, 25, 26

[4] Muñoz-Fernández, J.C., Knauss, A., Castañeda, L., Derakhshanmanesh, M., Heinrich, R., Becker, M., Taherimakhsousi, N.: Capturing ambiguity in artifacts to support requirements engineering for self-adaptive systems. In: Joint Proceedings of REFSQ-2017 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2017), Essen, Germany, February 27, 2017. (2017) 1, 2

[5] Knauss, A., Damian, D., Schneider, K.: Eliciting contextual requirements at design time: A case study. In: 4th IEEE International Workshop on Empirical Requirements Engineering, EmpiRE 2014, Karlskrona, Sweden, August 25, 2014. pp. 56–63 (2014) 1, 3, 15, 20

[6] Hong, D., Chiu, D.K.W., Shen, V.Y.: Requirements elicitation for the design of context-aware applications in a ubiquitous environment. In: Proceedings of the 7th International Conference on Electronic Commerce, ICEC 2005, Xi'an, China, August 15-17, 2005. pp. 590–596 (2005) 1, 15, 20

[7] Avizienis, A., Laprie, J., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Sec. Comput. 1(1), 11–33 (2004) 1, 6, 7, 27

[8] Ramirez, A.J., Jensen, A.C., Cheng, B.H.C.: A taxonomy of uncertainty for dynamically adaptive systems. In: 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012, Zurich, Switzerland, June 4-5, 2012. pp. 99–108 (2012) 1, 2, 17, 43

[9] Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers. pp. 214–238 (2010) 2, 16

[10] Horkoff, J., Salay, R., Chechik, M., Sandro, A.D.: Supporting early decision-making in the presence of uncertainty. In: IEEE 22nd International Requirements Engineering Conference, RE 2014, Karlskrona, Sweden, August 25-29, 2014. pp. 33–42 (2014) 2, 18, 20

[11] Filieri, A., Maggio, M., Angelopoulos, K., D'Ippolito, N., Gerostathopoulos, I., Hempel, A.B., Hoffmann, H., Jamshidi, P., Kalyvianaki, E., Klein, C., Krikava, F., Misailovic, S., Papadopoulos, A.V., Ray, S., Sharifloo, A.M., Shevtsov, S., Ujma, M., Vogel, T.: Software engineering meets control theory. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 71–82. SEAMS '15, IEEE Press, Piscataway, NJ, USA (2015) 3

[12] Ali, R., Dalpiaz, F., Giorgini, P.: A goal-based framework for contextual requirements modeling and analysis. Requir. Eng. 15(4), 439–458 (2010) 3, 5, 8, 9, 16, 23

[13] Dey, A.K.: Understanding and using context. Personal and Ubiquitous Computing 5(1), 4–7 (2001) 5

[14] Finkelstein, A., Savigni, A.: A framework for requirements engineering for context-aware services. In: In Proc. of 1 st International Workshop From Software Requirements to Architectures (STRAW 01). pp. 200–1 (2001) 5

[15] Ieee standard glossary of software engineering terminology. ieee std 610.12-1990 (1990) 6

[16] Weyns, D., Bencomo, N., Calinescu, R., Cámara, J., Ghezzi, C., Grassi, V.M., Grunske, L., Inverardi, P., Jézéquel, J.M., Malek, S., Mirandola, R., Mori, M., Tamburrelli, G.: Perpetual assurances for self-adaptive systems (2016) 6

[17] de Lemos, R., Garlan, D., Ghezzi, C., Giese, H., Andersson, J., Litoiu, M., Schmerl, B., Weyns, D., Baresi, L., Bencomo, N., Brun, Y., Camara, J., Calinescu, R., Cohen, M.B., Gorla, A., Grassi, V., Grunske, L., Inverardi, P., Jezequel, J.M., Malek, S., Mirandola, R., Mori, M., Müller, H.A., Rouvoy, R., Rubira, C.M.F., Rutten, E., Shaw, M., Tamburrelli, G., Tamura, G., Villegas, N.M., Vogel, T., Zambonelli, F.: Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-Adaptive Systems III, vol. 9640. Springer (2017) 6

[18] Cheng, B.H.C., Eder, K.I., Gogolla, M., Grunske, L., Litoiu, M., Müller, H.A., Pelliccione, P., Perini, A., Qureshi, N.A., Rumpe, B., Schneider, D., Trollmann, F., Villegas, N.M.: Using models at runtime to address assurance for self-adaptive systems. In: Models@run.time - Foundations, Applications, and Roadmaps [Dagstuhl Seminar 11481, November 27 - December 2, 2011]. pp. 101–136 (2011) 6

[19] Hastie, T., Tibshirani, R., Friedman, J.H.: The elements of statistical learning: data mining, inference, and prediction, 2nd Edition. Springer series in statistics, Springer (2009) 9

[20] Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993. pp. 207–216 (1993) 10

[21] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile. pp. 487–499 (1994) 10, 30

[22] Powers, D.: Evaluation: from precision, recall and f-measure to roc, informedness, markedness & correlation. Journal of Machine Learning Technologies 2, 37–63 (01 2011) 10

[23] Cohen, W.W.: Fast effective rule induction. In: Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995. pp. 115–123 (1995) 11, 30

[24] Frank, E., Hall, M.A., Holmes, G., Kirkby, R., Pfahringer, B.: WEKA - A machine learning workbench for data mining. In: The Data Mining and Knowledge Discovery Handbook., pp. 1305–1314 (2005) 11, 36

[25] Salzberg, S.L.: C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. Machine Learning 16(3), 235–240 (Sep 1994) 12

[26] Gómez, M., Adams, B., Maalej, W., Monperrus, M., Rouvoy, R.: App store 2.0: From crowdsourced information to actionable feedback in mobile ecosystems. IEEE Software 34(2), 81–89 (2017) 15, 20

[27] Villegas, N.M., Tamura, G., Müller, H.A., Duchien, L., Casallas, R.: DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers. pp. 265–293 (2010) 16, 20

[28] Mahdavi Hezavehi, S., Avgeriou, P., Weyns, D.: Chapter 3 - a classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements. In: Managing Trade-Offs in Adaptable Software Architectures, chap. 3Managing Trade-Offs in Adaptable Software Architectures, pp. 45–77. Morgan Kaufmann (01 2017) 16

[29] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.: RELAX: incorporating uncertainty into the specification of self-adaptive systems. In: RE 2009, 17th IEEE International Requirements Engineering Conference, Atlanta, Georgia, USA, August 31 - September 4, 2009. pp. 79–88 (2009) 17

[30] Cheng, B.H.C., Sawyer, P., Bencomo, N., Whittle, J.: A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009. Proceedings. pp. 468–483 (2009) 17, 20

[31] Mendonça, D.F., Ali, R., Rodrigues, G.N.: Modelling and analysing contextual failures for dependability requirements. In: 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014. pp. 55–64 (2014) 17, 20, 28

[32] Grassi, V., Mirandola, R., Randazzo, E.: Software engineering for self-adaptive systems. chap. Model-Driven Assessment of QoS-Aware Self-Adaptation, pp. 201–222. Springer-Verlag, Berlin, Heidelberg (2009) 17, 20

[33] Mahdavi-Hezavehi, S., Durelli, V.H.S., Weyns, D., Avgeriou, P.: A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems. Information & Software Technology 90, 1–26 (2017) 17

[34] de Lemos, R., Garlan, D., Ghezzi, C., Giese, H., Andersson, J., Litoiu, M., Schmerl, B., Weyns, D., Baresi, L., Bencomo, N., Brun, Y., Camara, J., Calinescu, R., Cohen, M.B., Gorla, A., Grassi, V., Grunske, L., Inverardi, P., Jezequel, J.M., Malek, S., Mirandola, R., Mori, M., Müller, H.A., Rouvoy, R., Rubira, C.M.F., Rutten, E., Shaw, M., Tamburrelli, G., Tamura, G., Villegas, N.M., Vogel, T., Zambonelli, F.: Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-Adaptive Systems III, vol. 9640. Springer (2017) 17

[35] Hassan, S., Bencomo, N., Bahsoon, R.: Minimizing nasty surprises with better informed decision-making in self-adaptive systems. In: 10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015. pp. 134–145 (2015) 18, 20

[36] Knauss, A., Damian, D., Franch, X., Rook, A., Müller, H.A., Thomo, A.: Acon: A learning-based approach to deal with uncertainty in contextual requirements at runtime. Information & Software Technology 70, 85–99 (2016) 18, 20

[37] Esfahani, N., Elkhodary, A.M., Malek, S.: A learning-based framework for engineering feature-oriented self-adaptive software systems. IEEE Trans. Software Eng. 39(11), 1467–1493 (2013) 18, 20

[38] Welsh, K., Sawyer, P., Bencomo, N.: Towards requirements aware systems: Run-time resolution of design-time assumptions. In: 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011. pp. 560–563 (2011) 18, 20

[39] Sharifloo, A.M., Metzger, A., Quinton, C., Baresi, L., Pohl, K.: Learning and evolution in dynamic software product lines. In: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2016, Austin, Texas, USA, May 14-22, 2016. pp. 158–164 (2016) 18

[40] Nunes, V., Mendonça, D., Rodrigues, G., Alves, V.: Towards compositional approach for parametric model checking in software product lines. In: International Workshop on Architecting Dependable Systems (WDAS'13). SBC (2013) 22

[41] Guimarães, F.P., Rodrigues, G.N., Ali, R., Batista, D.M.: Planning runtime software adaptation through pragmatic goal model. Data & Knowledge Engineering pp. – (2017) 26

[42] Casella, G., Berger, R.L.: Statistical inference. Duxbury/Thomson Learning, 2nd ed edn. (2002) 30

[43] van Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal Question Metric (GQM) Approach. John Wiley & Sons, Inc. (2002) 34

[44] Rodrigues, G.N., Alves, V., Nunes, V., Lanna, A., Cordy, M., Schobbens, P., Sharifloo, A.M., Legay, A.: Modeling and verification for probabilistic properties in software product lines. In: 16th IEEE International Symposium on High Assurance Systems Engineering, HASE 2015, Daytona Beach, FL, USA, January 8-10, 2015. pp. 173–180 (2015) 35, 36, 38

[45] Beth israel hospital - mit database patient guide, `http://physionet.org/physiobank/database/mghdb/patient-guide.shtml` 36