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

Enhancing Runtime Monitors of Cyber-Physical Systems using Negative Selection

João Paulo C. de Araujo

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientadora
Prof.a Dr.a Genaína Nunes Rodrigues

Brasília
2022

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

Enhancing Runtime Monitors of Cyber-Physical Systems using Negative Selection

João Paulo C. de Araujo

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Brasília, 11 de Novembro de 2022

Acknowledgements

Thanks to God, and my savior Jesus Christ, for giving me a purpose and meaning in life. What is the point of searching for academic excellence? Or at least of trying to develop a good final project? In five hundred of years, no one will ever remember of the young kid from Brasilia who tried to do his best, pulling dozens of all-nighters and leaving his social life behind, neither of his work, for it will be totally outdated. I ask again, what is the meaning of all of it? I found that meaning in something greater than me. By looking at life through this materialistic lenses, I saw myself in complete desperation, lacking for something that nothing in this world would never satisfy, since any satisfaction I'd find was only temporary. I've found rest only in the words of a Jew who said: "Everyone who drinks this water will be thirsty again, but whoever drinks the water I give them will never thirst." (John 4:13,14) He said that only him could provide me with everything I was searching for, namely, a purpose, and he provided it in the most unimaginable way: by excelling in all areas of his life here on earth and exchanging his perfect resume for my failed one in the cross, through sheer grace! He was everything I cannot be and achieved everything I cannot achieve. Now, I don't need to live by seeking for pleasures or finding significance through my performance, for I already have the ultimate pleasure in him and his perfect curriculum was graciously given to me. My life now is dedicated to follow him and to grow more likely to his character. If there is any aspect of this work that is good in any form, it is because of his kindness in providing me with wisdom and knowledge.

I also want to thank my wife Luiza, for being the one person who, even though did not know a thing about programming, patiently and diligently listened to detailed descriptions of the problems I've faced during the execution of this project. Thanks to Genaína, my professor adviser, who was attentive and available, answering my emails almost in real time, always receiving me with a smile in her face. Thanks to Arthur Rodrigues, a colleague who became a friend by walking the second mile to help me. Thanks also to my family for all the supporting me altruistically through the entire process.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Recentemente, os sistemas Ciber-Físicos (CPS) têm se tornado cada vez mais relevantes em nossa vida cotidiana, o que demanda maiores níveis de confiabilidade. A área de Runtime Verification endereça esse ponto por meio de monitores em tempo de execução que verificam a satisfatibilidade de propriedades do sistema formalizadas. Quando uma violação é identificada, o sistema pode alertar o usuário ou executar alguma rotina predefinida para manter a operação regular do sistema. Ao final, o analista pode diagnosticar o problema e realizar as medidas necessárias para a correção

No entanto, a complexidade inerente a estes domínios suscita alguns desafios, como por exemplo a imprevisibilidade de certos acontecimentos, dificuldades em representar os processos cibernéticos ou físicos, e o conhecimento incompleto dos contextos do ambiente. Tudo isso pode fazer com que os monitores se comportem de forma inesperada, ou que eles não monitorem algum aspecto que deveria ser considerado.

Procurar inspiração em processos de outros campos é uma atividade muito comum na Informática. O Algoritmo de Seleção Negativa, por exemplo, é uma técnica de base imunológica com múltiplas aplicações bem sucedidas em CPS, principalmente no campo do diagnóstico de falhas para a identificação de comportamentos anômalos. A explicabilidade do algoritmo pode trazer benefícios expressivos para a concepção e verificação de CPS, ajudando a compreender os padrões de violação de propriedade, e assim melhorar a verificação do sistema.

Neste trabalho, propõe-se uma metodologia que visa aumentar a confiabilidade de CPS. Isto é feito através de um diagnóstico sistemático das violações das propriedades do sistema baseado em dados gerados por um protótipo. O algoritmo de Seleção Negativa (NSA) serve como método de redundância analítica para isolar e identificar fatores que contribuem para a violação de propriedade no sistema. É possível que, através do arrazoamento sobre tais fatores, as razões pelas quais as violações de propriedade acontecem, os monitores possam ser refinados e mecanismos tolerantes a falhas possam ser criados permitindo, assim, o desenvolvimento de aplicações mais seguras e melhores.

Palavras-chave: Sistemas Cyber-físicos, Especificação de Propriedades, Seleção Negativa, Diagnóstico de Falhas, Verificação

Abstract

Recently, Cyber-physical systems are becoming more relevant in our day-to-day lives, thus demanding higher reliability. The Runtime Verification field addresses this by relying on runtime monitors to verify the satisfiability of formalized system properties. When a violation is spotted, the system can alert the user, or run some predefined process in order to maintain the regular operation. Afterwards, the analyst can take a closer look and perform the proper correction measures.

Nevertheless, the complexity inherent of these domains raises some challenges like unforeseen events, difficulties in depicting either the cyber or the physical processes, and the incomplete knowledge of the environment contexts, for example. All of that may cause the monitors to misbehave or miss out on some important aspects that should be considered when monitoring a property.

Seeking inspiration in processes from other fields is a very common activity in Computer Science. The Negative Selection Algorithm, for example, is an immuno-based technique with multiple successful applications in the field of CPS, primarily in the field of fault diagnostics for the identification of anomalous behavior. The algorithm's explainability may bring expressive benefits for the design and verification of CPS by helping understand the property violation patterns, and thus enhance the system verification.

In this work, we propose a methodology that aims at increasing the reliability of CPSs. This is achieved by a systematical diagnosis of system properties violations based on data generated by a prototype, performed in the early stages of development. An immuno-inspired algorithm called Negative Selection (NSA) serves as an analytical redundancy method to isolate and identify the cause for property violation in the system. We believe that, by reasoning about why the property violations happen, the runtime monitors may be refined, fault-tolerant mechanisms may be added, and, thus, safer and better applications might be written.

Keywords: Cyber-physical Systems, Property Specification, Negative Selection, Fault Diagnosis, Verification

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Definition

Cyber-physical systems are systems that integrate software components with physical processes [6]. They are a definite reality in our day-to-day lives, especially in the recent years. It is fair to say that the development of autonomous systems and the Internet of Things have taken us quite near to the future visions we had a few decades ago. Flying cars, for example, were supplanted by unmanned aircraft and autonomous drones, for instance, although these technologies are now used for a variety of applications rather than just passenger transportation. The usage of CPSs is also increasing amongst sophisticated applications, whose domains include self-driving vehicles, smart homes, smart cities, and more.

Nevertheless, the complexity inherent of these domains raises some challanges. Unforeseen events, for example, might make the CPS unreliable at runtime, which could have disastrous effects. This is a fact, particularly for safety-critical systems, which are subject to strict safety rules, time limitations, and performance requirements, and whose malfunctions might potentially endanger the user's life. Such systems demand the provision of assurances to guarantee that the system's goals are met during its entire lifespan, from conception to operation.

The Runtime Verification field [7] tries to address this problem. One of its techiniques relies on deriving a set of properties from the specification that formally describe how the system should behave. Then, runtime monitors are developed as a means to verify the satisfiability of each property while the CPS is being executed. The so called observers are state machines capable of performing the verification task by reading the log files, or the signals sent between the system modules. When a violation is spotted, an error state is reached. When that happens, the system can alert the user, or run some predefined process in order to maintain the regular operation. Afterwards, the analyst can take a closer look and perform the proper correction measures.

Nevertheless, in some cases, it might not be possible to wait for the CPS to be deployed and realize the verification afterwards, since it may potentially be costly or even harmful. A possible solution is to develop a prototype of the system before its deployment [8]. In this case, the CPS would be modeled in a simulation environment, while still in the early stages of development, with the Runtime Verification monitors acting as observers to analyze the properties during the simulation. Several simulations may be executed, with varying configurations to account for the different scenarios that may happen in runtime. Such scenarios may include failures in components, physical processes that may impact the CPS' behavior, crashes in the system and so on.

However, the CPS verification task is a difficult undertaking yet to be perfectly achieved [9]. One of the reasons is that, when applications are first designed, the knowledge about the environment in which they will be deployed may be incomplete, and also subject to continuous change over the application's lifetime [10]. This fact may affect the quality aspects of CPSs operation both at a physical and at a logical level, such as a freezing temperature affecting a sensor's capacity to deliver reliable data. Besides that, the complex relation between the cyber and physical aspects of the CPS contributes to the challenging task of monitoring such systems. While the physical process are modeled by time-continuous equations, the discrete behavior of software components can be described as state machines [11]. Other complexities are related to the dynamic behavior of the physical components, i.e. deterioration and malfunctioning, for instance.

All of that may cause the observers to misbehave or miss out on some important aspects that should be considered when monitoring a property. In face of a faulty scenario, the system analyst might wonder "what caused the property violation?" or "which components or behaviors are related to violations of the property?". The intricate relation between the cybernetic and physical natures of CPS creates obstacles when obtaining the set of configurations or behaviors that lead the system to anomalous behavior. Trying to address this problem by debbuging the prototype can be a very time-consuming task, whithout any guarantees of solving it.

Hence there is a need for an approach that increases the reliability of Cyber-physical systems by enhancing the runtime monitors. This technique should support the system analyst in the understanding of the context variabilities that provoke the violation of the system's properties, in a way that accounts for the complexities of the CPS. In summary, the goal of this study can be condensed in the following research question:

RQ: How to enhance runtime monitors accurately and systematically to account for the complex interaction between the physical and cybernetic nature of CPS?

1.2 Proposed Solution

Seeking inspiration in processes from other fields is a very common activity in Computer Science. Nature, especially biology, has long served as a fruitfull source of methodologies like artificial intelligence approaches. Artificial Neural Networks (ANN) are inspired by the functioning of the nervous system [12], Evolutionary Algorithms (EA) were based on the model of natural evolution [13], and the biologic immune system led to the creation of Artificial Immune Systems (AIS) [14]. These new developing soft computing approaches have already been employed in a variety of disciplines since the early 1990s.

The Negative Selection Algorithm [14], for example, is an immuno-based technique with multiple successful applications in the field of CPS, primarily in the field of fault diagnostics for the identification of anomalous behavior. This one-class method works by developing detectors aimed at classifying data that corresponds to abnormal behavior. A carefull study of these detectors may lead to explanations about the patterns that generate such behaviors and the isolation of the fault and the enabling of the development of fault-tolerant mechanisms that increase the runtime montioring of CPSs. This algorithm may bring several benefits for the field of CPS verification. By defining as abnormal behavior situations where the formalised system properties are not met, the patterns that define violations could be assessed and, hence, the system properties could be enhanced.

In this work, we propose a methodology that aims at increasing the reliability of CPSs. This is achieved by a systematical diagnosis of system properties violations based on data generated by a protoype, performed in the early stages of development. The immuno-inspired algorithm called Negative Selection (NSA) serves as an analytical redundancy method to indicate the features or behaviors that account for property violations in the system. We believe that, by reasoning about why the property violations happen, the runtime monitors may be refined, fault-tolerant mechanisms may be added, and, thus, safer and better applications might be written.

The methodology is comprised of two main phases. In the first phase, a prototype of the system is implemented based on the specification of the system and a set of properties derived from it. The observers are also implemented in the prototype for the verification of the properties satisfiability during the simulantions. After that, several simulations are performed with varying configurations, input data and parameters in order to account for the context variability that may arise at runtime.

In the second phase we perform the analysis of property violations in the data extracted from the prototype. Initially, a feature engineering process takes place to derive a set of boolean features to describe the behavior of the CPS. Then, based on the intended behavior of the observers, the rows are labeled as self or nonself, meaning regular data traces or faulty traces that incur in violation. Next, each row is formatted as binary strings. Then, the NSA is executed on the self data: new binary strings are generated, which go through a censoring process, in which they are tested against the self data, based on a similarity function, and are discarded if matched. The resulting set contains detectors specialized in matching nonself data which are carefully examined so that the patterns discovered can be comprehended. This process shares some similarities with the Analytical Redundancy Method [15], since the pre-knowledge of a healthy system are the properties being verified against the prototype data. Hence, it allows for the reasoning about the causes of such violations, which can be used by the Analyst to enhance the sytem's specification, properties and the observers.

The Biological immune system (BIS) not only provides the NSA technique, but also acts as a metaphore for the entire methodology. The initial phase can be related to the Innate Immunity response, since it is the first line of defense of the body. The prototype with the "naive" observers face the simulated runtime scenarios without a refined knowledge, just as the nonspecific process of the human body. The antigens are seen here as the execution traces that incur in property violation. The operation data from the simulation follows to the second phase of our approach, which can be compared to the Adaptive Immunity response of the BIS. The biological system produces T cells, which are matured in the Thymus by a censoring process called Negative Selection, in which the lymphocytes are discarded if they strongly bind to proteins of the body. These matured white blood cells are then used by the body to combat pathogens. Analogously, the framework we are proposing uses the Negative Selection algorithm to create a set of random detectors, which are "matured" by being tested against the operation dataset from the prototype. These detectors, in turn, are used to enhance observer automata capable of acting with a more refined knowledge of the environment.

1.3 Evaluation

We experimentally assess our strategy by designing a version of the Body Sensor Network (BSN) [16]. Its Contextual Goal Model specifies a set of modules and resources, like sensors, battery and patient's sensed data, that will allow us to represent it as a Cyber-Physical system. A system prototype of the CPS will be implemented in a simulation environment called OpenModelica [17], which is a well known graphic modeling tool for the development of CPS prototypes in the Modelica language [18]. The simulations were performed by using a dataset of 1,000 randomly generated patient data. The feature engineering process and the Negative Selection Algorithm were both implemented in the Python programming language, which was also used in the causality analysis of the detectors.

The NSA was evaluated regarding its generalization to unseen data. In this matter, all of the execution segments that the model identified as property violations found were, in fact, violations, accounting for a precision of 100%. Nevertheless, some patterns were not detected by the algorithm, resulting in a recall rate of 0.9958. In addition, the analysis of the efficiency of the technique was performed based on the number of patterns found. The generated nonself detectors resulted in the discovery of 49 unveiled patterns related to faulty segments, which were grouped into 11 clusters based on similarity.

Our approach was also compared with other Machine Learning algorithms. Two popular state-of-the-art semi-supervised anomaly detection algorithms, namely the One-Class SVM and the Isolation Forest [19], were run on the same data, achieving a precision of 41.8% and 29.2%, respectively. This shows the strength of the NSA for this task. Besides that, the algorithms Decision Tree, Random Forest and Gradient Boosting [20, 21] were also tested, with similar performance when compared with the NSA. The advantage of the Negati Selection Algorithm lies on the generated strings that are capable of matching with nonself data. Those strings can be further analyzed to understand the patterns that may cause vaiolations in the system. In the meanwhile, the Random Forest and the Gradient Boosting provide only the importance of each feature, but not the relationship between them in such cases. On the other hand, the Decision tree outputs a tree from which patterns can be visualized, although it provided way less patterns in comparison with the proposed methodology.

1.4 Organization

The rest of the document is organized as follows. Chapter 2 introduces some concepts that are useful for the understanding of the framework. In Chapter 4 introduces the case study that will be followed along the work. Chapter 5 futher details the proposed approach. Chapter 3 shows some works that are strongly related to our proposed methodology. Chapter 6 presents the results of the work after the evaluation, while Chapter 7 concludes and provides the directions for future works.

Chapter 2

Background

2.1 Cyber-Physical Systems (CPS)

Cyber-Physical Systems (CPS) can be defined as systems that integrate physical processes and software components [6]. This is done by embedded computers and networks which are capable of monitoring and controlling physical processes, tipically with feedback loops, which affects the physical environment while they are operating [22].

It's not new for physical and digital processes to be integrated into the same system. The notion of engineered systems that combine computer softwares with physical processes have been refered to as "embedded systems" for some time [6]. Car electronics, games, weapons systems, household appliances, and toys can be mentioned as a few successful examples. In order to delineate the distinction, Schatz [23] states that, even though CPSs encompasses embedded systems, the cyber-physical integration in CPSs happens both on a local and a global scale. Lee [6] and Jadzi [24] goes in a similar direction by distinguishing CPSs and embedded systems in terms of networking and outsourcing of computational power. Jadzi [24] goes beyond by alluding the data exchange as its most important feature, while defining CPSs in terms of embedded systems that are "able to send and receive data over a network." In turn, Dowdeswell et. al [25] adds that, in this sense, CPSs can be seen as seamless entities that form entire systems.

A CPS comprises three main modules [24]: a control unit, which relates to the computational aspect; a set of sensors and actuators, which refer to the physical processes and are controled by the control unit; and a collection of microcontrollers, that interfaces the two. Besides that, a communication interface is also required for the data exchange with other CPSs, a central unit or a cloud.

Aleksandrowicz et al. [1] further divide the Cyber part of the CPS as a four-layered architecture, as depicted in Figure 2.1. First, the analog/mixed-signal (AMS) layer focuses on the design's aging behavior and simulates the low level components, particularly sensors and actuators. Second, the digital hardware layer abstracts the AMS layer's signals as binary values. Thirdly, the architectural layer sees the CPS as a collection of computing units with varying capacities, a communication network connecting those computational units, and a collection of high-level-of-abstraction activities that should be completed on the system. Finally, the behavioral layer addresses the CPS's functional behavior and tasks by representing the behavioral aspects of the system's functioning implemented in either software or hardware.

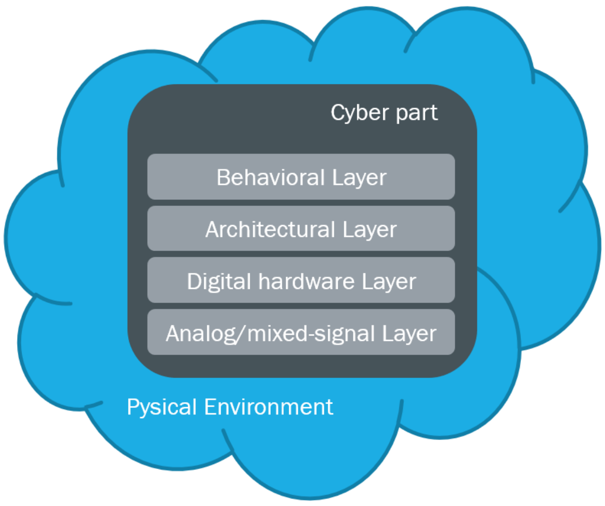


Figure 2.1: The stack of layers of the Cyber nature of CPS [1] (with adaptations)

Automotive systems, avionics systems, defense systems, manufacturing systems, process control systems, traffic control systems, robotics, smart medical devices, smart household applications, and marine systems are just a few of the application areas where CPSs have been utilized and developed [26]. Banerjee et al. [22] review the literature and group a non-exaustive list of representatives: the usage of physiological sensors that allow for continuous health monitoring, the quick identification of medical situations, and the administration of treatments; Data centers (DCs) that count on renewable energy for cooling reasons; smart buildings that sense tenant absence and turn off the cooling unit to improve energy efficiency; and unmanned aerial vehicles (UAVs) that surveils an area based on a picture of the landscape.

Another distinctive aspect of CPSs worth mentioning is their complexity. Schatz [23] divides it into three dimensions: (i) the "cross-dimension", which relates to process of computing and physical natures being related to different domains, technologies, organizations and so on; (ii) the "live-dimension", referring to the support of critical systems which cannot be turned off, and thus require updates at runtime; and (iii) the "self-dimension", which includes autonomous capabilities, like adaptation, healing monitoring and others. Bolbot et al [26], however, segments the CPSs complexity as structural, dynamic and organisational, according to the literature of system design and development. The first, called structural complexity, refers to systems with many components and unexpected interactions between them. Next, the dynamic complexity occurs when it is difficult to understand how a system behaves and changes through time. Finally, the organizational complexity relates to the configuration of the team in charge of the complex system's design and operation.

2.2 System Verification

The inherent complexity permeating cyber-physical systems create the need of evidence to prove that the system is capable of meeting its requirements during its entire lifetime [10]. This evidence, also known as assurances, can be defined as "the collection, analysis and synthesis of evidence for building arguments which demonstrate that the system satisfies its functional and non-functional requirements during operation". This is specially important for safety-critical systems because of its safety restrictions and strict guidelines.

System verification is an approach for the provision of assurances that can be used in cyber-physical system. Instead of relying on identification of hazards, it focuses on stablishing that the CPS satisfies certain properties [27]. These properties are derived from the system's specification, which prescribes what the system should and should not do. In this case, faults are related to the nonfulfillment of properties, and the system is considered "correct" only when all properties are satisfied.

Formal methods are considered by the state-of-the-art approaches as the most promising way of providing such guarantees [10]. Nevertheless, some uncertainties might be faced only during the system's operation, which makes the off-line solutions insufficient. To address this matter, Weyns et al. [28] introduce the concept of perpetual assurances, which consists of a continuous process performed by both humans and the system to derive and incorporate new evidences. To this end, two types of assurance must be consistently updated [10]. First, the evidence that concerns the parts of the system that are little affected by uncertainties and, thus, can be acquired by traditional off-line methods. Second, the evidence regarding the modules that are considerably impacted by uncertainties. This type of assurance should be synthesized during the handling of the uncertainty, at runtime.

The methods that provide the perpetual assurances can be divided in three categories: the human-driven, the system-driven and the hybrid approaches. Next, one sample strategy from each group will be described.

- **Formal Proof:** A human-driven approach that relies on mathematical calculations performed by automatic theorem provers to demonstrate a set of related theorems based on a formal description of a system. It requires from the developer a thorough domain knowledge of the system and a vast mathematical experience, but provides a reliable and unambiguous evidence.

- **Model Checking:** A hybrid method that verifies if a property is satisfied by checking all the reachable states in the system. It is typically run offline, but can also be applied online, and can both verify properties in the entire system (a high level abstraction) or just in some specific module.
- **Runtime Verification:** A system-driven lightweight procedure that relies on gathering data from an active system to determine whether specific properties are being violated.

2.2.1 Model Checking

One of the techniques used in the area is called Model Checking, in which a timed-automata model of the system goes through a reachability analysis in order to verify the satisfiability of the properties [27]. It works by exploring the whole range of states that are possible to occur in the system in a brute-force manner to check that the specified properties are satisfied

If a state is encountered that violates the property under consideration, the model checker provides a counterexample that indicates how the model could reach the undesired state. The counterexample describes an execution path that leads from the initial system state to a state that violates the property being verified

The process of model checking comprises three phases:

- **Modeling phase:** the system is modeled according to its specification in terms of the description language of the tool that will run it. Such models must accurately and unambiguously reflect the system's intended behavior. They are typically modeled using finite-state automata, meaning that they myst have a finite collection of states and of transitions. States include information about variable values, previously run statements, and so on. Transitions illustrate how a system transitions between states.
- **Running phase:** In this phase, the model checking tool is executed given the model and the properties to be verfied. The checker exhaustively checks all the reachable system states to determine the validity of the properties.
- **Analysis phase:** After running the checker, if no violations are found, the process is concluded. If that is not the case, the cause of the result must be identified. It

might be a modeling error or a property specification error. In both cases, upon making the required fixes, a new verification is required.

2.2.2 Property Specification

In order to run the system verification process in a correct manner, the properties must be specified in a precise and unambiguous way [27]. Differently from the system model, which describes the behavior of the system, properties determine what the system should and should not do.

To allow for thorough verification, properties must be stated precisely and unambiguously. A property specification language is often used for this. In order to represent significant properties of Cyber-physical systems, temporal logic is often used. This language is essentially an extension of regular propositional logic with operators that relate to the behavior of the system across time. It enables the specification of a wide range of relevant system properties, including functional correctness, reachability, safety, liveness, fairness, and real-time properties.

However, there are some intrinsic problems that permeate the probess of property specification. These problems are related to the difficulty in ensuring that a given specification corresponds to a software engineer's understanding about the system. To address that issue, Autili et al. [29] proposed a property specification framework and a pattern catalog. Their work aims at the simplification of the process of system property specification. Initially, the property is written by using a Structured English Grammar, which consists of phrases in plain English limited to temporal logics denotable terms. Then, the property is mapped to instance patterns in the pattern catalog. Finally, the corresponding temporal logic formula is derived from the pattern.

2.2.3 Runtime Verification (RV)

Verification is task performed in a CPS that relates to the assessment performed to check the compliance of the system with the specification [30]. It can be performed while the system is being run (runtime) or after its execution through the reading of the system's logs [7]. Leucker defines RV as a discipline in Computer Science that "that deals with the study, development, and application of those verification techniques that allow checking whether a run of a system under scrutiny (SUS) satisfies or violates a given correctness property" [31].

Colombo et al. [7] describe divide this activity into 7 steps:

- **1.** Selecting a language for the specification of the property;

- **2.** Defining the behavior of the system
- **3.** Development of a monitor capable of listening the events of the system related to the desired property;
- **4.** Deriving an algorithm from the monitor;
- **5.** Implementing the algorithm into the system;
- **6.** Analysing the results of the monitoring;
- **7.** Performing a response.

One of the ways that this monitors can be implemented is through Observers. This components are capable of collecting information from the CPS, by reading the system's logs or the signals that are sent by the modules of the system. They help the system analyst to identify faults, security issues, measure statistics and so on. This technique consists of the translation of the TCTL formulas to finite-state automata that are added the model to be verified. These state machines are engineered to reach error states only when the observed property in the system model is violated [32].

Observers can be derived from the properties of the system with the help of a specification pattern catalog, as the one proposed by Vogel et al. [33]. In their work, patterns in the properties are used to automatically map them to timed-automata. They have designed templates that account for all the specification patterns described by Autili et al. [29], which can be translated into UPPAAL [34] autotama for the execution of the Model Checking process, allowing for an end-to-end verification. This approach is utilized in the checker to allow the verification of properties that go beyond simple reachability [35].

End-to-end Verification

An end-to-end verification process, that goes from the CGM specification to runtime monitoring, can really benefit the provision of assurances for Cyber-physical systems, since the guarantees provided would help increase the reliability of such systems. This can be achieved by initially modeling the core architectural elements, along with their behavior, then by specifying a set of properties of interest, and then, by conducting a model checking process in a verification tool like UPPAAL [34], and finally by implementing the observers into the deployed to assert the system's correctness and the satisfiability of the properties in runtime. Figure 2.2 illustrates the modeling of the system and the derivation of properties and observers until the Model Checking process.

First, the model is derived directly from the CGM, the input for this process. The intended behavior of the leaf-tasks are modeled as timed automata, according to the

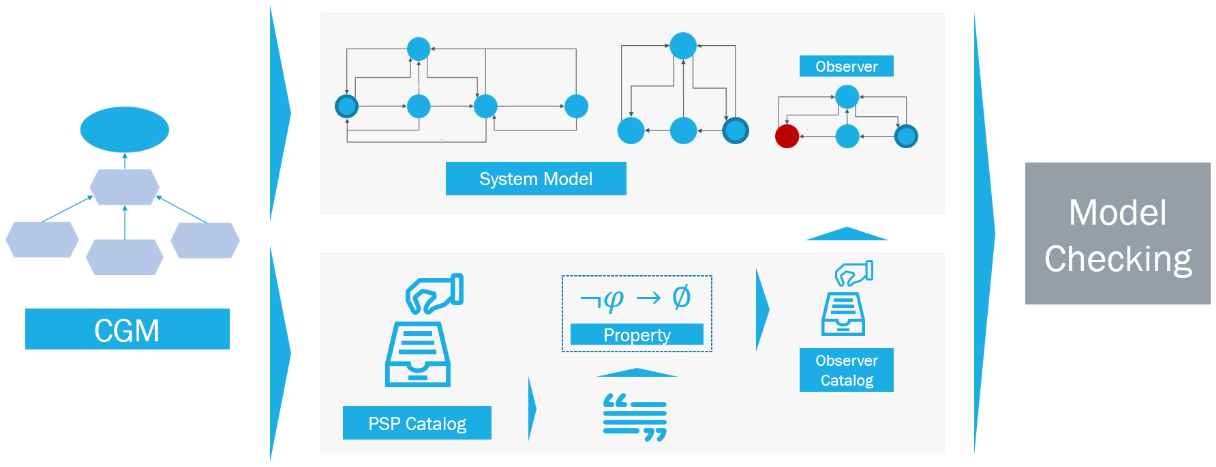


Figure 2.2: Verification Process until Model Checking

system architecture. The automata denote module templates, meaning that they might be reused if two instances share the same behavior. One of the possible modeling strategies is to use guard conditions and different locations in the model to portray the progression of the module's behavior, and thus, its life cycle. Hence, the progress and fulfillment of the CGM task's behavior may be depicted in the UPPAAL model as reachable states or locations, enabling the usage of reachability properties when verifying the correctness of the system.

Second, with the formalized model in hand, the next activity is to specify the set of properties that will be verified. In this sense, the Autili et al. [29] property specification pattern catalog and framework (PSPFramework) can be used to define the CGM properties. Property specification patterns have been effectively utilized to bypass the pragmatic hurdles of specification formalism, while still maintaining syntactic correctness and a true reflection of the system's intuition. As described earlier, this catalog comprises a list of property patterns, or categories, each relying on a set of structured English phrases, composed of terms that are denotable in temporal logics.

Each system property is specified in the PSPFramework in a top-down manner. A property category is first chosen from the catalog patterns list by the software developer, who then refines it with the specific scope and complementary attributes, like time or probability constraints. As a consequence, a sentence in structured English is created, which will then be mapped to an appropriate logic formalism. The mapping rule takes into account the property category and scope defined earlier to match a logic formula template from the catalog, which is written in a target language, like CSL or TCTL. Finally, the given time and probability constraints are inserted into the formula using an attribute assessment mechanism. The result is a formally expressed property, ready to be verified. The resultant property set may be written in Timed Computational Tree Logic

(TCTL) [36], since that is the language used by UPPAAL when verifying the formalized model's real-time features.

The Observer Technique can also be leverage as an additional step towards providing evidence that the system behaves as expected. The translation is performed by using an Observer Template catalog [33], as describe earlier, designed for model checking real-time systems in UPPAAL. Analogously to the property specification process, the property pattern and its corresponding scope are mapped to Observer UPPAAL template models in the catalog. As a result, each monitorable property will have a corresponding observer automata in the system model.

Afterwards, the generated formulas are be verified for reachability, safety or liveness in the modeled system, by a Model Checking process. The TCTL model-checking problem is to determine for a particular timed automaton TA and TCTL formula ϕ whether TA $\models \phi$ [27]. A tool named UPPAAL [34] can be used to solve this problem by algorithmically assessing if the specified set of properties hold true for each possible state of the system model. If that is not the case, the model, the system specification and the system properties must be revised for a reverification.

Finally, after the assurances are provided in design time, the observers are implemented as algorithms as a means to monitor the CPS during its execution. The algorithm is deployed with in the system and reads the signals sent by the modules in order to verify the satisfaction of the designed properties in runtime. Whenever a violation occurs, an error state is reached, triggering a suitable response.

2.3 Biological Immune Systems (BIS)

As mentioned previously, the Artificial Immune System is a model inspired by its biological counterpart in order to solve computational problems. Therefore, there is a need to introduce the main concepts and processes found in the Biological Immune System so that the analogies and metaphors implemented by the AIS might be better understood. This subsection focuses on providing the reader with the foundational knowledge on the BIS.

2.3.1 Main Concepts

Immunity can be defined as the ability to respond to unknown substances [14]. In this sense, the Biological Immune System (BIS) comprises a set of structures and mechanisms which are capable of distinguishing the body's cells from foreign substances and responding adequately. It includes specific organs, cells and molecules. It is a complex, fault-tolerant and distributed multilevel defense mechanism composed of two main layers, namely the innate immunity and the adaptive immunity [2].

The primary goal of the BIS is to protect the body from potentially harmful material [37]. It is composed by a multilevel defense mechanism that is capable of distinguishing between molecules from the body itself and foreign ones, selecting a specific response to a threat, and enacting an inflammatory reaction in order to maintain the health and safety of the body. All of these activities are capable of evolving through time since they rely on aspects of learning and memory, which are present in the BIS.

The innate immunity is the body's first line of defense. Inherited from the host's progenitors, it is responsible for a quick or immediate response against infections. It is achieved through physical and chemical barriers or through cellular responses. The barriers are considered nonspecific mechanisms that work as shields against pathogens by blocking them from entering the body. They comprise the skin, mucous membranes on the body's openings and the secretions of both. The low pH of the skin, for instance, inhibits the proliferation and growth of bacteria, while the antimicrobial substances present in saliva and tears keep the antigens from invading through the membranes [14]. The cellular response, on the other hand, focus on perceiving the pathogens that were able to surpass the barriers and activating a variety of cellular responses, which include: the ingestion of the substance (phagocytosis), the induction of an inflammatory response and the triggering of the adaptive immunity for a tailor made response.

The adaptive immunity is an immunological mechanism that is capable of "specifically recognize and selectively eliminate foreign microorganisms and molecules" [14]. In contrast with the innate immunity, it has a high level of specificity when dealing with the antigens, meaning that the response is customized and based on the particularities of the foreign substance. The downside is that the response can take days to be performed. Nevertheless, the information from previous infections is persisted in order to achieve a faster response when a similar antigen is detected. In the literature, the adaptive immunity responses is divided into two distinct, but overlapping, categories: the humoral immunity and the cellular immunity.

The first kind of response, called humoral immunity, relies on the interaction between the antigen and B lymphocytes, a specific type of white blood cell also known as B Cell. These cells are created in the bone marrow and, when activated, are able to produce antibodies, which bind to the antigen during the immunological response as a means to destroy it. This can only happen if there is a match between the antibody and the surface of the foreign material. Humans are thought to have 10^7 to 10^8 different antibodies with distinct chemical compositions [38] that account for the possible variations of antigens that one may find during the course of a lifetime.

The second kind of adaptive response is called cellular immunity and is mediated by lymphocytes called T cells. These cells are also produced in the bone marrow, but are matured in an organ named Thymus. They are responsible for killing tumor cells and cells from the body that were infected by the pathogen (altered self-cells).

2.3.2 Overview of the Immune Response

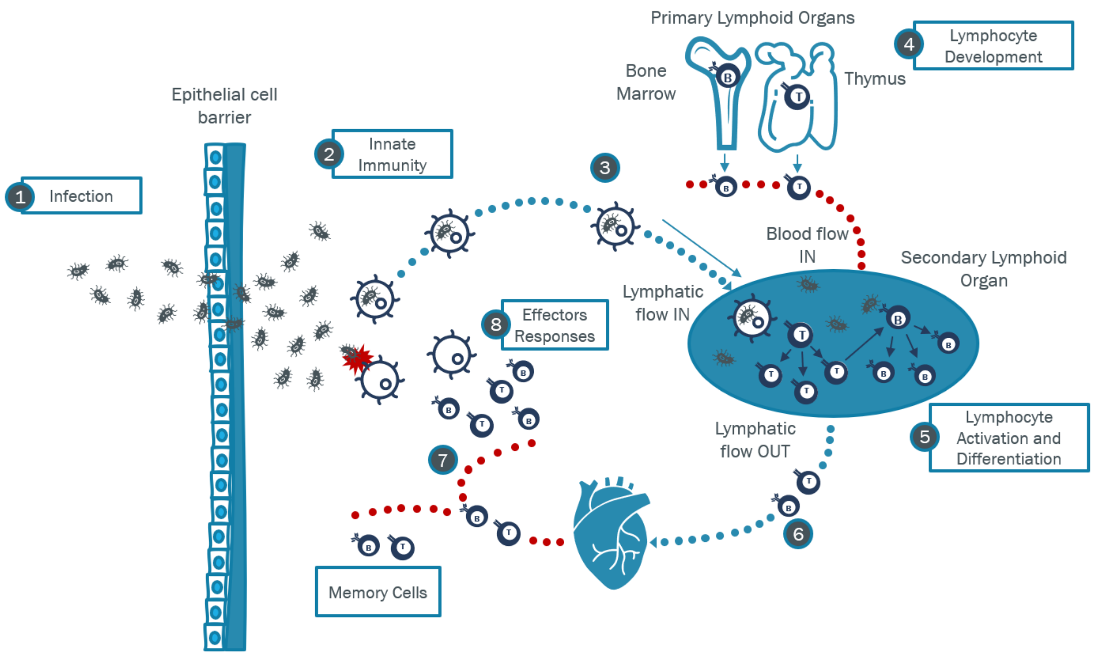


Figure 2.3: Main Processes of an Immune Response [2] (with adaptations)

Figure 2.3 shows an overview of the main activities that may happen during an immune response. The whole process starts with an infection caused by an antigen that was capable of passing through the physical and chemical barriers of the body (1). After that, when the pathogen is detected by the front line phagocytic cells, a hormone-like protein called cytokine is released by them as to induce a local inflammatory response (2). Besides that, these cells are capable of engulfing the antigen and transporting it through the lymphatic vessels with the objective of enacting the adaptive immune response (3).

In the meanwhile, both B and T lymphocytes are derived in the Bone marrow via a process called hematopoiesis. The receptors of these cells undergo a pseudo-random genetic rearrangement process during their creation that account for the variety of cells, and thus the ability to bind with unseen substances [39]. While the B cells flow directly through the blood flow into secondary lymphoid organs, the T cells stop at the Thymus to be matured, and then follow the same path as the B cells (4). The maturation of the T cells is a censoring process in which the lymphocytes are tested against proteins of the body. If a T-cell strongly binds to some self-protein, it is discarded. Hence, only the T-cells that did not have a strong bind are allowed to flow through the bloodstream and be used against the pathogen. This process is called Negative Selection and aims at avoiding autoimmune responses.

The adaptive immune response starts at a secondary lymphoid organ, with the arrival of the phagocytic cell carrying the antigen. In this step, a mechanism called Clonal Selection is performed (5). The B and T lymphocytes with a high level of engagement with the pathogen proliferate and mutate (somatic hypermutation) as a means to grow in number and to improve the affinity with the foreign substance. Afterwards, these differentiated cells leave the lymphoid organ and are pumped throughout the circulatory system by the heart (6) until reaching the local of the inflammatory response (7). Finally, the specialized lymphocytes act on the antigens in order to destroy the residual of the invasion (8). These cells are kept as memory cells so that, in the case of a future similar threat, allow for a faster response.

2.4 Artificial Immune Systems and the Negative Selection Algorithm

In 1986, Farmer et al. [38] sought inspiration in the properties and theories of this biological system to propose a computational model, named Artificial Immune System (AIS). It was built in such a way that the immunological concepts and processes are distilled into algorithms that can be simulated by a computer in order to solve all sorts of problems from the real world. The idea behind it is that, since this system is able to protect us, it might also be computationally useful [40]. Nevertheless, by that time, the resources available were not powerful enough to allow its usage in real systems [41].

Recently, the advances in technology enabled complex tasks to be performed in a fast and efficient way. This provided the researchers the tools needed to implement and further improve the application of the immunological concepts into a wide range of problems. For example, AIS techniques have been used in problems like anomaly detection, optimization, classification and clustering [37]. The Negative Selection, along with the Clonal Selection, Immune Network Theory and the Danger Theory are amongst the most researched approaches in the literature [3].

One of the fundamental skills of the BIS is the ability to differentiate between the body's own cells and the foreign material. It is called self/nonself discrimination [3] and helps to protect the body from attacking itself whilst building a strong defense against foreigners.

As described in the previous subsection, T-cells are created in the bone marrow, where they have their receptors differentiated by a pseudo-random genetic rearrangement process. Later, in the Thymus, they are maturated by being tested with self cells. Lymphocytes that have a strong binding with self cells are discarded, in a censoring fashion. The result of the process is a set of diverse T cells specialized in binding with nonself cells.

In 1994, Forrest et al. introduced an algorithm inspired by this process that is used in the field of change/anomaly detection [42]. They have abstracted the concepts of self and nonself so that the process of Negative Selection could be used in more general problems. They have used a file authentication system as an example to instantiate one of the possible applications. In their work, data from legitimate users trying to access a file were thought as the body's own cells, while non-authorized accesses were seen as nonself material, or pathogens. Both types of data were translated into binaty strings, and T-cell detectors were generated as binary strings of the same size which, during the censoring phase, did not match the self data. A change was considered whenever there was a match between some detector and the new data in the monitoring phase.

Since then, even though the main idea of the algorithm was kept, several implementations and adaptations were made to improve its performance and allow for the application it in different scenarios. Dasgupta [3], in a recent work, reviewed the literature on this matter and classified the algorithms found based on the following characteristics: data type, data representation, distance function, detector size and initialization.

- **Data Type:** Refers to the type of the data that is used by the algorithm. It can be binary or real-valued.

- **Data Representation:** Related to how the data is structured or formatted. If the data type is binary, it can be shaped as a string or a grid. If it is a real value, the formats are grid and vector.

- **Distance function:** Also called "affinity" or "matching" Function, it is a function that identifies how strong is the bind between the self and nonself data. In the case of binary strings, the most common are the r-chunk, r-continuous bits and hamming distance with its variations. When talking about real valued types, the functions are usually the distance between vectors, like the manhattan, euclidean or minkowski distance. This section will provide more details about this topic latter on.

- **Detector Initialization:** Describes how the detectors are generated. In both data types it can be in a random, semi random or adaptive fashion. Further details are provided below.

- **Detector Size:** It is usually fixed, but in some real valued algorithms the detectors' size may vary in size in the generation process.

Even though different implementations may fall in different buckets, Ji and Dasgupta [43] have distilled the three aspects that must be present in an algorithm so that it may be considered a Negative Selection Algorithm. They are:

1. The goal is to identify the self-set counterpart.
2. Change/anomaly detection is performed by using some form of detector.
3. The algorithm makes use of only the self-samples during the detector generation.

Some discussion has been made over the Data Type and Representation characteristics since they limit all the others [43] [3] [14]. The advantages of using bit strings are that (1) any data can be presented as a binary string, (2) it facilitates the analysis of the result and (3) categorical data are well represented in this form, even though they have a scalability issue that comes with the increased string size.

All in all, to achieve the goals of this work, the binary data type will be used as the string representation. Even though the Binary Negative Selection (BNSA) is well-known for its performance and completeness issues [44], modeling the behavior of the CPS as boolean features allows for simplicity and this format provide a great level of interpretability, since the binary detectors can be reverse mapped to the original features, thus, unveiling the discovered patterns.

The next subsections provide a more in depth view of the algorithm for the binary data type, by showing its overall structure, detailing how the comparisons with the monitored data are made and how the detectors are generated.

2.4.1 Algorithm Description

The Negative Selection Algorithm (NSA) can be divided in two steps: the generation of the detectors, and the actual process of detection of the nonself. These steps are similar to most supervised algorithms, in which there is a training and a test phase [3].

Figure 2.4 helps to shed some light on both phases. On the left-hand side, the generation of the detectors is illustrated. In this "training" step, a set of random candidates is generated by some predefined process and undergo a censoring process based on the self samples. The candidates that match the self samples, measured by the distance function, are discarded, while the ones that do not match are added to the nonself detector set. On the right-hand side of the figure, the detection, or "test", phase is shown. The nonself detector set obtained in the first step is tested against the data that is being monitored.

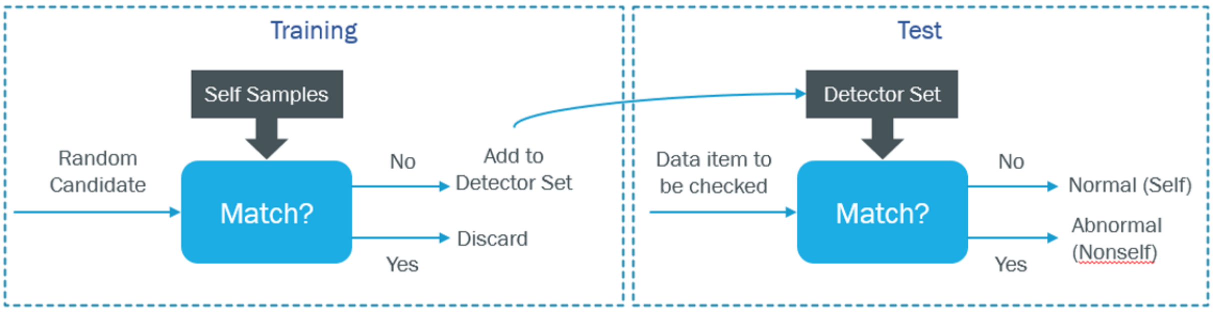


Figure 2.4: The two steps of the NSA [3] (with Adaptations)

The same distance function is utilized here to check whether the new data matches with some of the detectors in the set. In the case of a match, the data is considered abnormal.

For a complete understanding of the inner workings of the algorithm, two important pieces are missing: a description of how the random detector candidates are generated and the definition of the matching function that will account for the affinity measurement.

2.4.2 Distance Function

First, the Distance function, or matching rule, is an operation that relies on the comparison of characters (or bits) between two strings and provides a score telling how similar (or different) the strings are. Three rules are the most widely used in the literature [45]: the Hamming Distance, the R-Contiguous and the R-Chunk matching rules.

- **Hamming Distance (HD):** This function measures the number of characters that differ between two strings [14]. Let X be a string of lenght n such that $X = x_1x_2x_3...x_n$ and let D be a dectector with the same lenght, so that $D = d_1d_2d_3...d_n$. The Hamming Distance can be formally defined as:

$$HD = \sum_i (\overline{x_i \oplus d_i})$$

  where $\oplus$ is the XOR operation. In this case, a match between X and D is said to have happened when the HD score is below a predefined threshold. Figure 2.5 shows how the comparison is made. The characters at each position are compared and, if they are different, a unit is added to the final score. Therefore, two strigs must have a low score to be considered similar.

- **R-Contiguous:** Let again X be a string of lenght n such that $X = x_1x_2x_3...x_n$ and let D be a dectector with the same lenght, so that $D = d_1d_2d_3...d_n$. Let also r be an integer, such that $0 > r >= n$. This rule defines a match between X and D whenever the two strings have at least r consecutive identical characters starting at

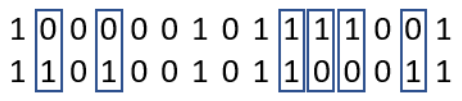


Figure 2.5: Illustration of the Hamming Distance Matching Rule

any position. This rule was mainly used in the first implementations of the NSA, in which the detectors were created in a generate-and-test fashion [45].

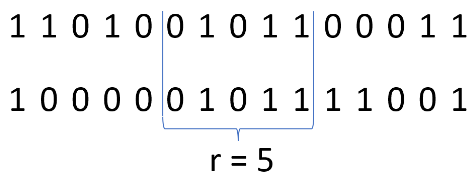


Figure 2.6: Illustration of the R-Contiguous Matching Rule

Figure 2.6 illustrates this concept. A window of size r slides searching for a region where the substrings match. If at least one region is found, then X and D are considered a match.

- **R-Chunk:** Let X be a string of lenght n such that $X = x_1x_2x_3...x_n$ and let D be a dectector of size m so that $D = d_1d_2d_3...d_m$, with $m \leq n$. Similarly to the R-Continuous rule, the string and the detector are considered a match if, at a position p, all bits of D are identical to the bits X in a window of size m, with $0 \leq p \leq n-r$. Hence, the detector is characterized by a chunk of size r and a starting position p, and can be uniquely identified as $t_{p,D}$. The practical difference between this rule and the previous one is that this function allows for detectors of any size, which improves the self-space coverage [14].

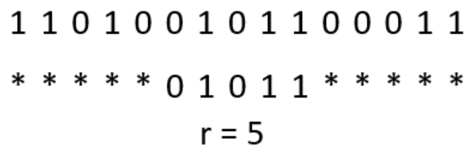


Figure 2.7: Illustration of the R-Chunk Matching Rule

Figure 2.7 shows an example of this rule. The detector $t_{6,01011}$ has size 5 and was a match in the sixth position of the string. The "*" represents irrelevant positions meaning that any character can be matched.

The R-chunk is said to enhance the accuracy and performance of the NSA [45] because the same generated string can be used as a detector in several positions. Therefore, one can say that lower sized detectors comprise an optimal detector set, since more abnormal data can be detected. Nevertheless, as cited by Wierzchon and Chmielewski [46], a study performed by Stibor showed that strings generated with low values of r are less likely to become detectors. This probability highly increases in the middle range, and is close to 1 for large string sizes. Hence, there is a sweet spot when trying to find the size of the string, which is usually for middle values of r, that aligns accuracy and coverability with efficiency. In our work, the R-chunk will be used as the distance function in the NSA due to its advantages and since the matched region is isolated, allowing for a better interpretability of the pattern found.

2.4.3 Detector Generation

Both Ayara et al. [47] and Dasgupta and Niño [14] provide a thorough detailing of the different methods found in literature for the generation of the detectors set for binary data. The most basic approach is the exhaustive detector generation, which was introduced in the original NSA paper [42]. The idea is to exhaustively generate random candidates until a big enough set of detectors is achieved. It was reported to be very time-consuming, since the amount of candidates grows exponentially with the size of the self-set [4].

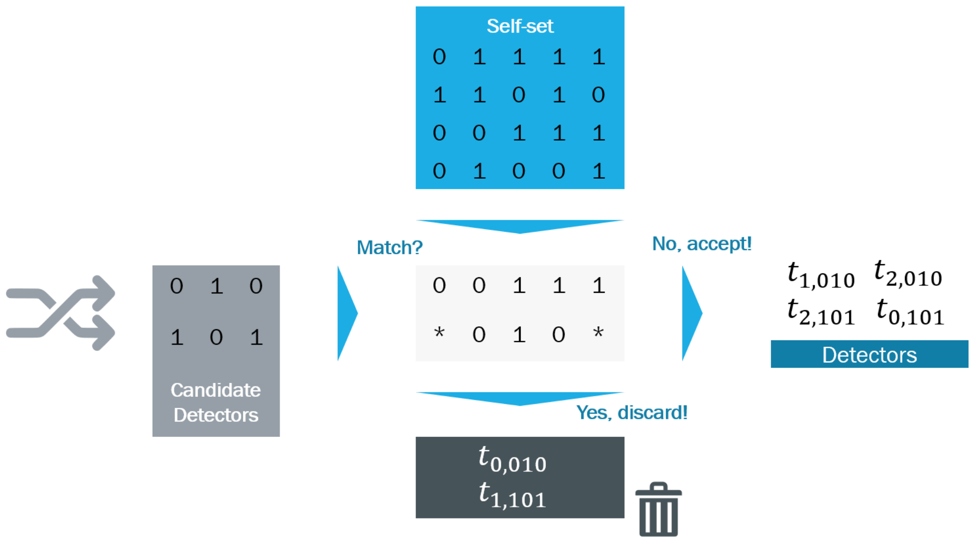


Figure 2.8: Example of the Exhaustive Detector Generation Process [4] (with adaptations)

This generate-and-test method can be further explained with Figure 2.8 in which random binary strings are generated and compose the Candidate Detectors set. Then, each candidate is tested for a match with the self-set S by using one of the functions described earlier. In the figure, if the compared strings have at least 2 matching contiguous bits, the candidate is rejected, otherwise it is accepted as a valid detector and joins set R.

The problem with the exhaustive detector generation is that a great number of candidates are rejected during the censoring, making it inneficient [44], and costly in terms of the computational use of resources [47]. To tackle that, two other approaches arose: the linear and the greedy algorithms, both based on the r-countinuous distance method.

The linear time algorithm is a two-phase process named after its complexity introduced by D'haeseleer et al. [44]. Initially, all the strings that are unmatched by the self-set have their recurrence counted. Then, this counting recurrence is used to naturally number the unmatched strings and allow for the picking of random detectors, according to the desired size of the detector's set. Even though this algorithm runs in linear time according to the detector's set and self-set sizes, the recurrence counting requires the storage of all the possible matches two strings can have by using the r-contiguous distance. This means that, although its complexity is linear in terms of time, it is exponential with regard to space.

The Greedy algorithm is yet another algorithm introduced by D'haeseleer et al. [44], which tries to provide a better coverage of the string space without increasing the amount of detectors. It does so by slightly modificating the construction of the array of possible matches. It also relies on two arrays, one that stores the candidates picked by the algorithm and other keeps track of the strings that still were not matched with any picked detector. New detectors are generated based on the unmatched strings that have the highest recurrence value [14]. This algorithm provides an optimal set of detectors but has a higher time complexity when compared with the Linear algorithm. This happens because of the upDateType of the two arrays that happens whenever a new detector is generated. Time complexity is kept, though.

Even though these two algorithms are proven to provide a better detector generation, they were designed specifically for the R-contiguous distance function. For this reason, the Generate-and-test method will be used in our work as the main algorithm for the generation of such detectors.

Chapter 3

Related work

This chapter discusses relevant work related to the themes discussed in this document. The first one is related to verification methods in Cyber-Physical Systems. The second topic deals with the usage of the Negative Selection Algorithm in fault diagnosis of CPSs.

3.1 Verification Methods in Cyber-Physical Systems

Formal Methods are widely applied the analysis of CPS because of their expanding application in many safety and financial-critical sectors. Model Checking, for example, is particularly popular since it addresses multiple aspects found in CPSs [30].

Nevertheless, the complexity found in CPSs can bring problems to the design of such systems. Representing both the cyber and physical aspects of any CPS challenges the task of modeling such systems, since we are combining discrete and continuous models. Aside from the complexity of representing the real environment, a CPS has only a limited understanding of its surroundings. Sensing constraints induced by sensor blindspots and sensor interference complicate the modeling process [48]. Hence, oversimplified models may be invalidated for not anticipating failures dependant on the two layers [30].

Akella et al. [49], for instance, addresses this problem by trying to simplify the physical model. This was done by discretizing the events that cause flow change and to representing the CPS as a deterministic state model with discrete flow values inside its physical components. Model checking is then utilized to formally test insecure interactions between all conceivable behaviors of the specified CPS. Botaschanjan et al. [50] provide a sucessfull attempt to close this gap in automotive systems by combining the model checking verification of the lower layers of the system with the simulation of the upper layers.

Formal methods can also be used to perform anomaly detection. Jones et al. [51] uses formal methods to an anomaly detection framework to determine whether or not a particular CPS is under attack. They build STL properties based on the normal behavior of the system and flag the abnormal behavior. Then, a one-class SVM model is used to detect deviations. Our approach share some similarities with Jone's. Nonself data, i.e. anomalies, are flagged according to system properties and a one-class machine learning algorithm (NSA) is performed to understand the patterns based on the self data. Although in our work the properties are specified by the stakeholders and undergo a process of formalism. Besides that, we leverage the detectors generated to gain insight on the violations, which is used to enhance the specification. Webster [52] also integrates model checking and simulation in order to ensure safe operations for autonomous unmanned aircrafts. They argue that the high level of assurance provided by the formal method would increase the accuracy of the simulation model.

Another issue with model checking is that some of the system properties of the CPSs may not be thoroughly verified and tested during the design and building phases. An alternative is performing the verification task after the system is deployed, in what is called Runtime Verification. This technique is considered to be a lightweight dynamic formal method in the sense that it is performed during the CPS's execution and relies on the verification of real data for the assurance of safety properties [7]. One of the ways in which it can be realized is by using Observers, which are a reification of the property in the form of state machines. They are responsible for reading the signals and messages that are shared among the modules of the system, checking the system's logs and so on, in order to perform statistics, identify faults and so on. They can be derived from the specified system properties through pattern catalogs [33].

In the proposed framework, the model checking process will be integrated with simulation. This will be done by implementing a prototype in Modelica [18], which is a tool that allows for writing algorithms and physical equations, for the modeling of the cyber and physical aspects of the system. Observers will also be modeled as timed automata for the model checking phase, as well as implemented as components in the prototype. This will allow for a seamless transition from runtime and design time.

3.2 Fault Analysis and the Negative Selection

Another method for increasing the reliability of CPSs is fault detection, diagnosis and isolation (FDDI), which is conceptually similar to the assessment of property violations. Both are looking for techniques to determine whether and why the system is not operating as planned. In this sense, understanding which approaches are being utilized in the FDDI field may help us to better evaluate our methodology.

Zhang et al. [53] suggests a model-based detection, isolation, and system reconfiguration techinque for induction motor drives dependant on extended EKF, as well as strong current and DC-link voltage observers, which allows the systems to continue its operation even under sensor faults. Poon et al. [54] also relies on models to present a fault detection and identification (FDI) technique for switching power converters utilizing a model-based state estimator methodology, which boosts fault tolerance and awareness in power electronics systems. Garcia et al. [55] provides a real-time monitoring and preventive fault diagnosis method for solar panel strings in real-world installations, through the identification and parametric separation of fault symptoms using Voc-Isc curve analysis. In this strategy, identification and isolation occur with sufficient leeway to notify and stop the deterioration phenomena and its cumulative impact, which would potentialy result in the formation of irreversible failure via automated disconnection.

However, the success of traditional fault detection systems is determined by a number of elements, including prior knowledge of defective responses, an accurate system model, and a vast amount of data-history patterns. Furthermore, typical fault detection algorithms are often developed for a single system, thereby addressing a restricted number of defect types [56].

These limitations motivated the adoption of new approaches, which make use of machine learning techniques. Chopdar and Koshti [57], for instance, address the problem of faults in the growing number of power grid transmission lines via an Artificial Neural Network method that efficiently detects and classifies faults to keep the performance of the power system. The model was trained with data taken from simulation software, which reduced the demand for historical data.

Nevertheless, there is still a critical necessity for the development of efficient fault detection systems that are less domain-specific. In this sense, Artificial immune systems (AIS) have recently captured the interest of the scientific community, specially for the purpose of anomaly detection. The Negative Selection Algorithm (NSA), which is inspired by the self/nonself discrimination process performed by the body during an immune response [2], has emerged as a viable alternative [56].

Gupta and Dasgupta [3] provide a detailed assessment of the literature, demonstrating how the approach has evolved through time, highlighting the most notable variations and comparing with similar alternatives. They come to the conclusion that NSA outperforms most other approaches for nonlinear representation, and it can perform better than neural-based models in computing time.

Govender and Mensah [58] propose the usage of the NSA in modern manufacturing settings, in order to minimize the production lossess related to equipment malfuctioning. They simulate an automobile hub pressing machine using Matlab that has a programmable logic controller (PLC) connected to sensors, switches and safety curtains, and a module hosting the AIS. The self-set training data comes from the execution of the system under error free operations and are represented in the Hamming space as binary strings. Nonself detectors are randomly generated in the censoring phase of the algorithm by using the Hamming distance for the comparisons. Finally, the detectors are used in runtime to determine the anomalies in a hierachical fashion. They state that the representation of the data as binary strings allow for the reverse map of the resulting detectors back into the real nature of the equipment, even though the paper does not further explore this aspect.

Our approach makes use of the NSA in a similar way. The Hamming space is leveraged as a means to reverse map the detectors into system states during the pattern analysis process. However, we consider this idea paramount for the increase of the reliability of CPSs, since the explainability it provides enables the identification of design failures, the creation of fault tolerant mechanisms and the refinement of the system properties.

Two other works are worth mentioning in this section. The first, Alizadeh et al. [59] suggested a wind turbine defect diagnostic system that relies on real-valued NSA to boost nonself-space coverage. V-detectors are used for representing the detectors, the Euclidean distance is utilized for matching, and data are represented as spheres. The detection phase proceeds as default, but the isolation phase relies on the training of new instances of the NSA for each pre-defined fault, which are employed in a hierarchical manner to better describe identified faults. The second was proposed by Ren et al. [60] a different algorithm for fault discovery and isolation. In this paper, V-detectors are created in such a way that the detector's coverage is increased while the overlap is reduced. The detection phase contains no surprises, but the isolation phase relies on an algorithm that attempts to match the defect at hand with a collection of pre-identified fault representatives.

Both frameworks use more advanced Negative Selection algorithms, that rely on real-valued data representation, while our approach still represents the data as binary strings. Nevertheless, even with the well-knwon shortcommings of the binary NSA, the explainability provided by the detectors eliminates the need for any prior knowledge of the system's probable faults.

3.3 Final Considerations on Related Works

Our work lies in the line that unites design time and runtime. we aim at enhancing the CPSs reliability by improving the process of specification of system properties. In that sense, a model checking process is performed based on a timed-automata model of the system and a set of system properties derived from the specification. To account for the complexities found in the relation between computing and physical process [30], a prototype of the system will be implemented in a tool designed to simulate Cyber-physical Systems, named Modelica. It provides both a cyber and a model interface, which allows for the simulation of physical equations integrated with algorithms [61].

Additionally, observer automata will be derived from the properties and implemented in the prototype. This will enable the usage of runtime verification techniques while still in design time. A dataset will be extracted from simulation containing the execution logs, which will then be used in the Negative Selection Algorithm [3] for the discovery of patterns that are related to the violation of the specified properties. The patterns will be analyzed and used to enhance the properties, the system specification itself and make room for the development of fault-tolerant mechanisms and runtime monitors. We believe that these measures will account for the enhancing of the overall reliability of the system.

Chapter 4

Running Example: Body Sensor Network

Without loss of generality, the fundamentals of our methodology will be exemplified by a Cyber Physical System from the healthcare domain throughout this work. This chapter provides an overview of the system and its inner workings.

4.1 Overview

The Body Sensor Network (BSN) [62] is a pervasive platform designed to monitor and evaluate individual patient health statuses using a network of sensors and a centralized processing unit [16]. The system works as follows: the patient wears several physiological sensors, responsible for frequently measuring her vital signs, like the temperature, blood pressure, blood oxigenation and so on. These sensors are wirelessly liked to the Central node, a Personal Digital Assistant (PDA), which is capable of fusing the multi-sensor data and filtering the clinically significant events [63]. When an emergency is identified, the Central node sends out an alarm, either locally or remotely, so that the patient may receive the necessary treatment.

Figure 4.1 depicts the sensors attached to the patient as well as the wireless data transfer to the Central node. In our work, the set sensors comprises a thermometer, an oximeter, a heart rate monitor and an ABP for measuring both the diastolic and systolic artery blood pressure.The patient's health risk can be classified into three levels: low, normal, and high. When a high risk is detected, the Central node signals the individual or a third party about the emergency. The ranges of the sensors were defined by a health expert [62] and their relation with the risk levels are displayed in Table 4.1. The patient's health state is considered as *high risk* if at least one of the sensors measurements is in the *high* range. If that is not the case, and one or more resources gauge is within textitnormal

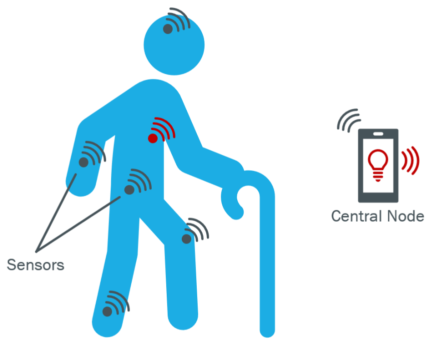


Figure 4.1: Visual representation of the BSN

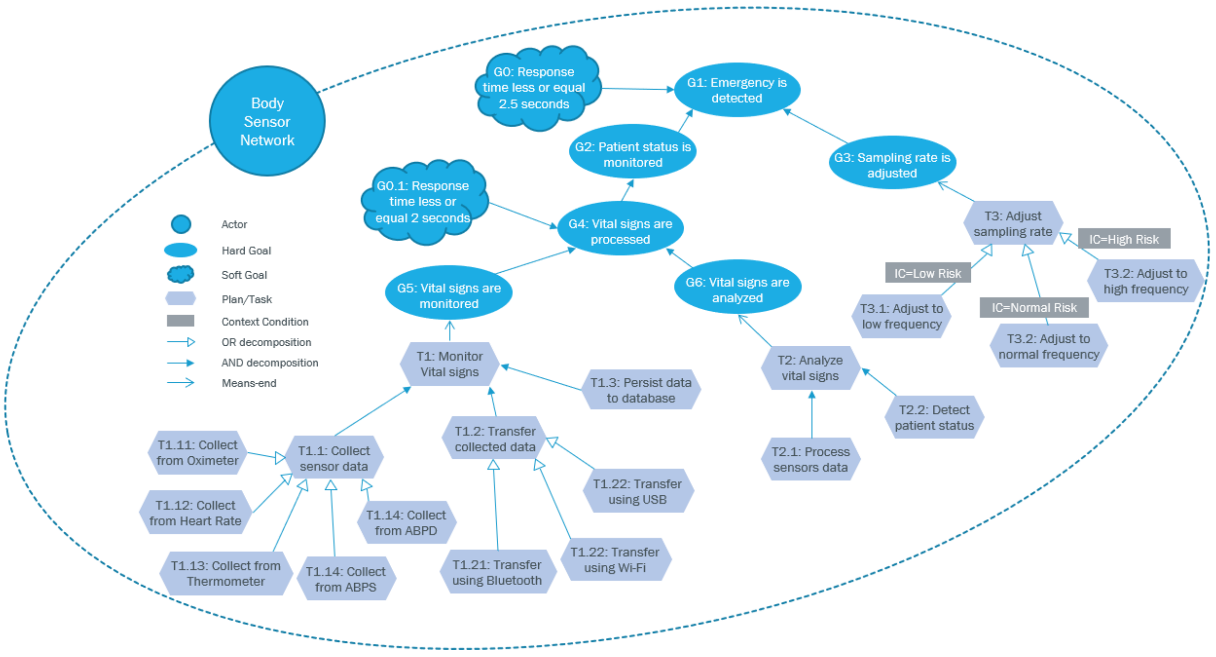
limits, the patient's health state is classified as *normal risk*. Otherwise, the patient's health state is classified as *low risk*.

Sensor	**High**	**Moderate**	**Low**	**Moderate**	**High**
Oximeter	[0, 55]	(55, 65]	(65, 100]	-	-
Heart Rate	[0, 70]	(70, 85]	(85, 97]	(97, 115]	(115, 300]
Thermometer	[0, 35]	(35, 36.5]	(36.5, 37.5]	(37.5, 38.3]	(38.3, 50]
ABPD	-	-	[0, 80]	(80, 90]	(90, 300]
ABPS	-	-	[0, 120]	(120, 140]	(140, 300]

Table 4.1: Ranges of the Sensors

4.2 Contextual Goal Model

Figure 4.2 shows the Contextual Goal Model (CGM) that represents the Body Sensor Network. It was inspired by the BSN specification from Rodrigues et al. [5], with a few added sensors and a new soft goal to indicate the time restriction of goal G2. This model provides a powerful way of understanding the needs of the stakeholders besides figuring out the motives behind the development of the CPS, by laying out user goals and ways to meet them [64]. The main goal of the BSN [62], also called root goal, is "G1: Emergency is detected". It is fulfilled by two other goals: "G2: Patient Status is monitored" and "G3: Sampling rate is adjusted". In turn, the goal G2 is achieved when "G4: Vital signs are processed" is achieved. G4 is divided into two other goals: "G5: Vital signs are monitored" and "G6: Vital signs are analyzed". Following the decomposition of the goals, a series of executable tasks are elicited to satisfy them. The context condition *IC* can be set to "low risk", "normal risk" and "high risk" and is utilized in the fulfillment of T3.



Body Sensor Network
G0: Response time less or equal 2.5 seconds
G1: Emergency is detected
G2: Patient status is monitored
G3: Sampling rate is adjusted
G0.1: Response time less or equal 2 seconds
G4: Vital signs are processed
G5: Vital signs are monitored
G6: Vital signs are analyzed
T3: Adjust sampling rate
IC=High Risk
IC=Low Risk
IC=Normal Risk
T3.1: Adjust to low frequency
T3.2: Adjust to high frequency
T3.2: Adjust to normal frequency
Actor
Hard Goal
Soft Goal
Plan/Task
Context Condition
OR decomposition
AND decomposition
Means-end
T1: Monitor Vital signs
T1.3: Persist data to database
T2: Analyze vital signs
T2.2: Detect patient status
T2.1: Process sensors data
T1.11: Collect from Oximeter
T1.1: Collect sensor data
T1.2: Transfer collected data
T1.22: Transfer using USB
T1.12: Collect from Heart Rate
T1.14: Collect from ABPD
T1.13: Collect from Thermometer
T1.14: Collect from ABPS
T1.21: Transfer using Bluetooth
T1.22: Transfer using Wi-Fi

Figure 4.2: Contextual Goal Model of the BSN [5]

4.3 System Properties

Another important aspect of the BSN that need to be addressed for the verification process is the definition of the system properties. Rodrigues et al. [5], by using a CGM for the BSN very similar to the one adopted here, derived a series of properties for the BSN in Timed Computational Tree Logic (TCTL) [65] to verify the satisfiability of the goals modeled through Model Checking. Their work bring the sufficient TCTL-like properties to ensure the correct behavior of the CPS by satisfying all the goals specified in the CGM.

Properties P1 and P2 relate to general aspects of the system behavior by accounting for problems that are common in distributed systems, like deadlocks and the execution of all modules within a cycle, respectively. P3 is related to the root goal and addresses the soft goal G0, which presents a time bound for the execution of a cycle. P4, P5 and P6 were designed to assess Goal G3 in relation to the context variability that it possess. Property P7, in turn, relates to G2 and was slightly modified here with the addition of the soft goal G0.1. This was done to better clarify some of the processes that are performed during the data processing step of our approach. Besides that, for the verification of the property using an obsever suring the simulation, a time bound needed to be added so that the error state could be reached whithin a cycle. Finally, properties P8, P9 and P10 ensure that the goals G4 and G5 are met and that the data is whithin the expected range.

Moreover, Vogel et al. [66] modeled some of the properties defined above as timed-automata observers by using a pattern catalog. Their work intends to use these observers during the Model Checking process, but in our work we will implement them as state machines in the BSN prototype for the verification of the properties during the simulation. The process of creation of the observers based on the properties is thoroughly described in their paper, and referred in Section 2.2.2, whilst the observers that were utilized here can be found in their GitHub repository [33].

In our work, a slightly modified version of property **P7** will be implemented, since it will suffice to showcase the strength of the approach. The property **P7** will be modified by adding a time restriction, as a way to better account for the different possibilities of the approach. Hence, the property reads as follows: "Whenever a sensor node has collected data, in within at most 2 seconds, the Central Node will process it." It was chosen because it focuses on the reliability of the system, since its satisfaction guarantees that the collected data will be processed in a reasonable time. Besides that, the Oximeter was the sensor chosen to reify the analysis.

Chapter 5

Enhancing Runtime Monitors of Cyber-Physical Systems using Negative Selection

In this work, we provide a paradigm to aid in the creation of Cyber-physical systems that uses the Negative Selection Algorithm to reveal operational circumstances of the CPS that may effect the fulfillment of non-functional requirements and real-time properties. Our proposed technique has two primary objectives. The first is to aid the analyst in the verification process during the early phases of system development by discovering patterns in simulated execution data linked to property violations. The secondary goal, a byproduct of the first, focuses on increasing the ability to confidently and systematically monitor and analyze CPSs. The methodology combines simulation frameworks with learning techniques to help address the complexity associated with CPS verification.

This chapter is structured as follows: First, we provide an overview of the framework, explaining in high level each step. Next, each step is further detailed in the subsequent sections.

5.1 Overview

The framework comprises two main steps, as depicted in Figure 5.1. The fist step uses as input the Contextual Goal Model (CGM) specification of the CPS to be developed, the system properties that were specified based on the CGM and the observers derived from them. With that in hand, we implement a prototype of the system, based on the expected behavior and environment conditions. The prototype will be composed of the modules of the system, the physical processes that may impact the operationalization, and some "naive" observers, for the purpose of monitoring the system properties during the execution. Next, the prototype is executed, and the operation dataset is extracted, containing the traces of execution, the consumption of resources and the context conditions at each moment of the execution.

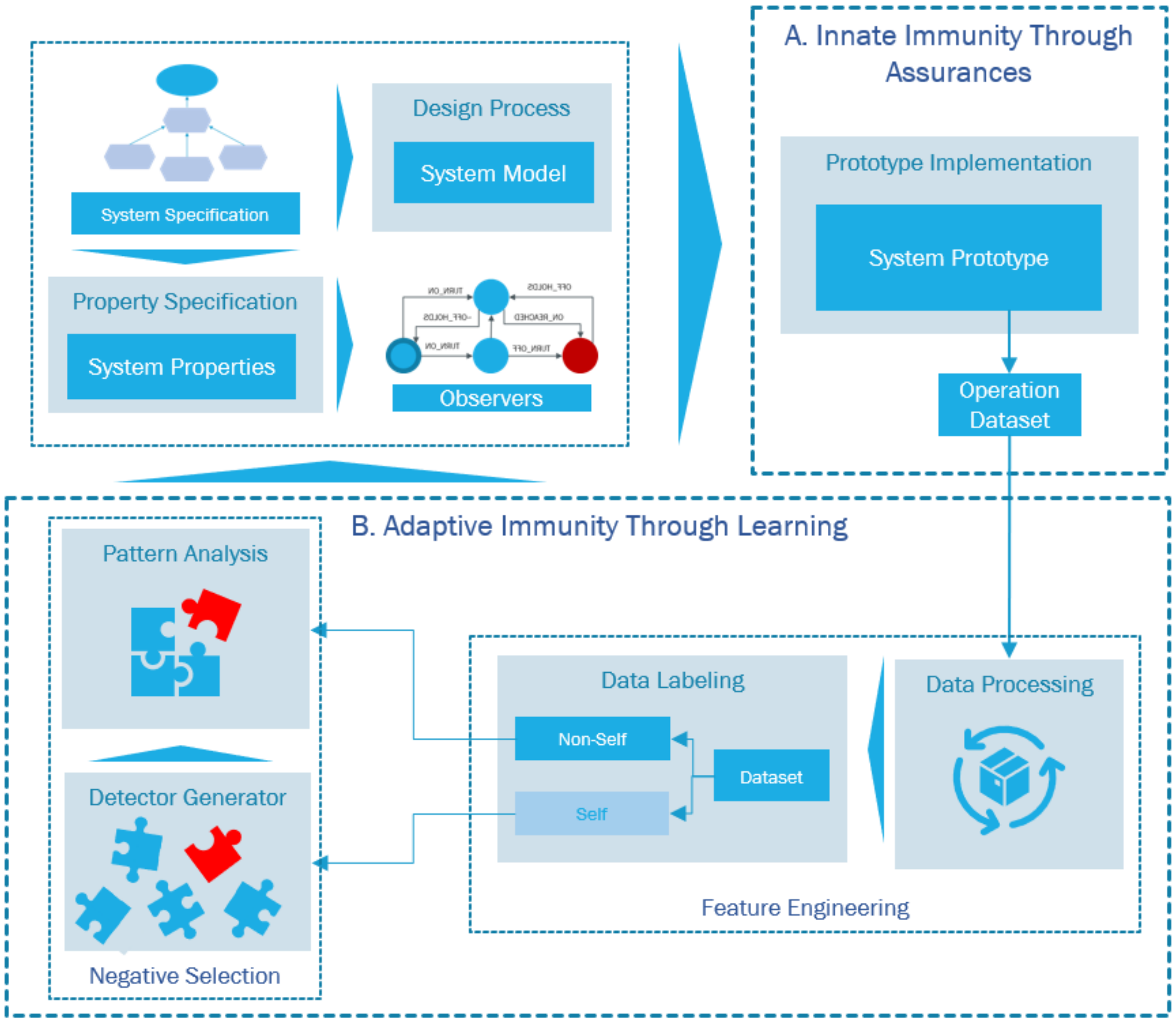


Figure 5.1: Overview of the proposed Methodology

The second step is performed individually for each property in the real-time property set. Initially, the operation data goes through a feature engineering process aiming at the labeling and characterization of the data traces in relation to the particular property being examined. Then, the labeled data is analyzed, and the Negative Selection Algorithm is used with the dataset. A collection of R-chunk detectors are produced as a result of this operation. These detectors, specialized in matching the property violation data, are carefully examined so that the patterns discovered can be comprehended.

The outcome of this process is the identification of relevant environment conditions and system behavior that were not initially considered when designing the property. This information is, then, used to refine the property, and thus, enhance or create new observers.

5.2 Innate Immunity through Assurances

The initial step of our approach addresses the complex relation between the cybernetic and physical natures of the CPS through the implementation of a prototype of the system. The major purpose is to replicate the system's behavior in order to generate reliable data for the analysis phase. Multiple runs of the simulation with varying configurations will be performed as a means to generate an operation dataset that accounts for the variability that will be faced in runtime. This dataset will be passed on to the next stage of our framework to convey the patterns in the anomalous executions by means of artificial intelligence.

The execution of the prototype can be easily related to the innate immunity, during the immunological response. The running simulation will be the first to face the variability in the environment, just as the innate immunity is the first line of defense against the pathogens [2]. In the biological system, the nonspecific immunological mechanism is responsible for triggering the adaptive immunity by phagocyting the antigen and sending it to better enhance the lymphocytes. Similarly, our prototype is assured by "naive" observers, who may not be specific enough to account for all of the scenarios that may happen. The operation dataset derived from the simulation is sent to the next phase to trigger an adaptive response, that learns from the property violation cases and provides the analyst the knowledge needed to enhance the observers for a better response in runtime.

The prototype can be implemented by following a state-of-the-art approach that attempts to assess the complexity of the task. Baras [67], for example, uses a Model-Based System Engineering (MBSE) approach that integrates several tools and languages for this purpose. SysML [68], a system architecture language, is first used for describing the system's structure and behavior. This tool is combined with Modelica [18], a language that allows for the physical processes, i.e. the continuous time modeling of the physical phenomena and the linking of modules that are regulated by mathematical equations. In order to graphically edit and explore a Modelica model, as well as run model simulations and other analyses, an open-source modeling and simulation environment for the Modelica language called OpenModelica [17] can be utilized. Baras also uses MATLAB [69] as a means to describe the software nature of the CPS, meaning the control and signal processing components. The integration of such technologies is done through the Functional Mock-up Interface (FMI) [70], which is a standard for the interchange of dynamic models and co-simulation.

Fei [71], in turn, proposes a less tool-dense modeling guide for CPSs that comprises the 4 layers of abstraction mentioned in Section 2.1, which are modeled in SysML and simulated using only Modelica. Bouskela et al. [72] goes in a similar direction by simulating the system with Modelica and verifying the requirements using the FOrmal Requirements Modeling Language (FORM-L) [73].

To achieve the purpose of this work, a simpler framework for the implementation of the prototype will suffice. Since Modelica is one of the most often used tools in the MBSE field [8], the prototype is written in this language and runs in the OpenModelica simulation environment. Each module is designed to mirror the expected behavior of the system to be, based on the designed software architecture. The naive observers are also implemented so that the execution of the simulation may be monitored for the provision of assurances.

With the prototype in place, many simulations with varying inputs, settings, and setups may be conducted to account for as much context variability as feasible. Each experiment will yield a dataset comprising the execution traces, resource usage, and context conditions at each stage of the execution. In the end, the generated datasets will be stored for the execution of the next step.

5.3 Adaptive Immunity Through Learning

Our framework's next phase tries to diagnose property violations using learning approaches. It is performed for each monitorable property of the system. The data created in the preceding stage, goes through a feature engineering process for reshaping and labeling the rows regarding the property satisfiability. Following that, the set of rows in which the property is met (self-set) is run through the Negative Selection Algorithm to build detectors for the complementary set (nonself-set). These detectors are evaluated in order to uncover patterns in the execution segments that violate the property at hand, which are then utilized to improve the observers.

We relate this step to the Adaptive Immunity mechanism that happens in our Biological Immune System, described in Section 2.3.1, specially with the cellular immunity response. T-lymphocytes are used in this adaptive response, which are created in the bone marrow by a pseudo-random genetic rearrangement process and then matured in the Thymus by being tested against own cells [39]. Similarly, our process randomly generates candidate strings that are tested for match against the operation dataset's self-set, and, just as only T-cells that do not bind strongly with the self-cells are allowed to flow through the blood stream [2], only the strings that do not match the self-set are used in the pattern analysis step, later on.

This process can be further divided into two phases: the feature engineering phase, in which the operation dataset is reshaped and labeled, and the learning phase, where the Negative Selection algorithm generates nonself-detectors that are used to enhance the observers.

5.3.1 Feature Engineering

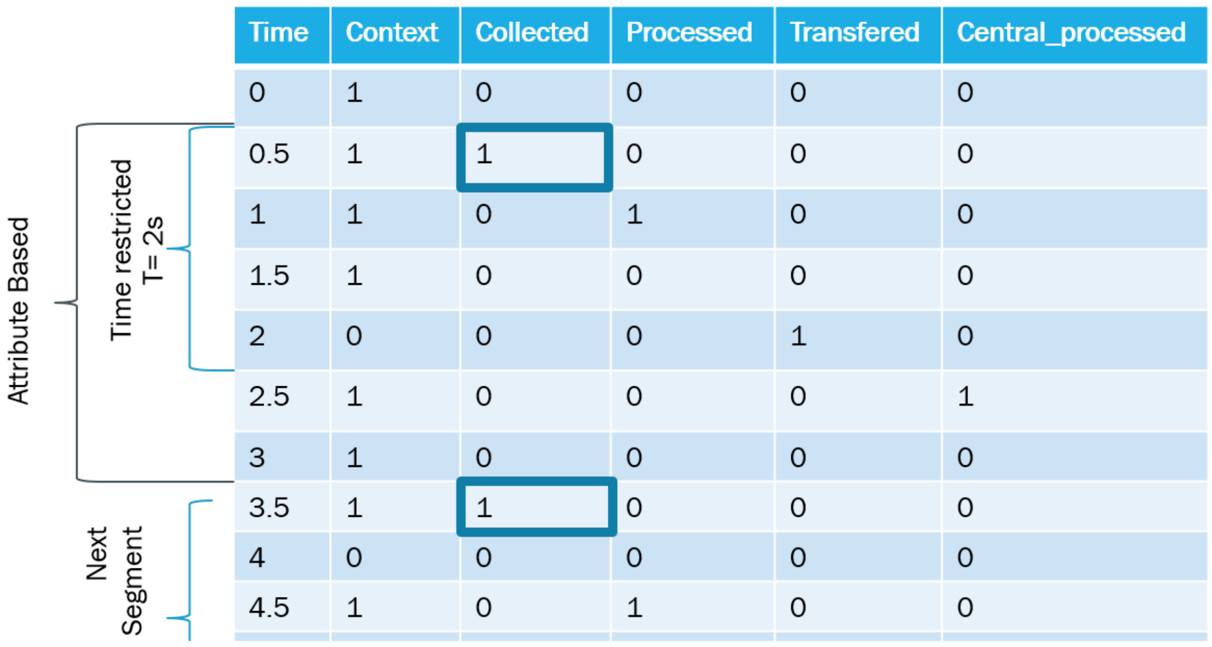
This phase focuses on the shaping of the operation dataset in order to better adjust itself to the learning algorithm. This is accomplished by first segmenting the data, then creating features to define individually a single segment, and lastly classifying the execution fragments as property violations (nonself) or not (self).

Depending on the simulation environment, the operation dataset is retrieved from the simulation as a collection of files, each of which is associated with a single simulation run in the previous step. A simulation file presents the data in a tabular manner, with the columns representing the values of the model's components, resources, signals and so on at every timestamp. The activities of segmentation, feature generation, and labeling are carried out for each individual file, and then concatenated in a resultant dataset.

Initially, the data is split based on the attributes or time constraints of the property being analyzed. For example, suppose the following property from the BSN is being analyzed: "Whenever a sensor node has collected data, the central node will eventually process it". In this scenario, the attribute that indicates when the data has been collected will be used to segment the dataset. An execution segment will start when the collect signal is sent until right before the next signal. However, if the property is time-bounded, the execution segment will begin when the collect signal is transmitted and last for the duration of the constraint. Figure 5.2 illustrates this process. The first column indicates the timestamp, whilst the others show the signals that were sent, with the thick blue box highlighting when the data was collected. On the one hand, the blue bracket alludes to the length of a property time-bounded by 2 seconds, while, on the other hand, the grey bracket indicates the length of the execution segment if only the signal is considered.

Next, new features will be derived to characterize the behavior of the CPS in that particular execution segment, resulting in a single row per segment. For instance, in the abovementioned figure, it would be possible to create features for each signal, indicanting if the data was processed, transfered and if it was processed by the central node, or another that identifies if the context was true at all times. Continuous features could also be created, like one that stores the amount of time between processing and transfering the data, or how long it took for the central node to process the data, and so on.

The quality and relevance of the features produced are essential for the development of the patterns that characterize the property violations, making this activity key to the



Time	Context	Collected	Processed	Transfered	Central_processed
0	1	0	0	0	0
0.5	1	1	0	0	0
1	1	0	1	0	0
1.5	1	0	0	0	0
2	0	0	0	1	0
2.5	1	0	0	0	1
3	1	0	0	0	0
3.5	1	1	0	0	0
4	0	0	0	0	0
4.5	1	0	1	0	0

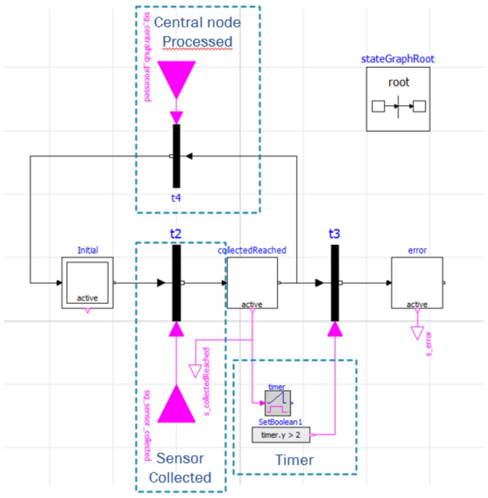
Attribute Based
Time restricted T= 2s
Next Segment

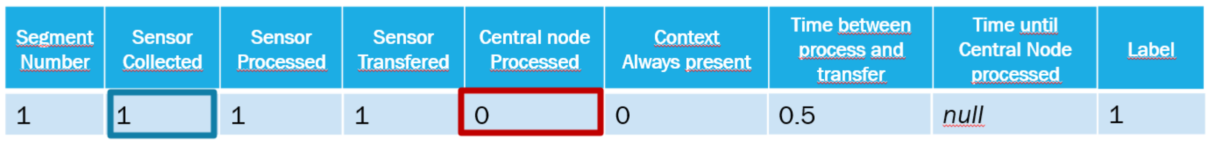
Figure 5.2: Segmentation based on attribute or on time restrictions

effectiveness of our strategy. Additionally, it heavily depends on the domain knowledge of the software developer conducting the activity. The outcome is a new dataset, in which every row uniquely identifies and extracts the main features of an execution segment.

Finally, the classification phase will make use of the recently derived dataset. The rows will be examined in accordance with the observer state transitions and signals, and a label indicating whether the row describes a violation or can be regarded as a regular execution will be issued. Let again the property at hand be: "Whenever a sensor node has collected data, the central node will eventually process it". The execution segment should take into account whether the data was obtained and if the central node processed it by looking at the signals received by the observer. These two behaviors have been mapped during the feature engineering stage and could be utilized in this instance for data labeling. If there is a boolean column with the value *True* for a given segment, indicating that the data has been collected, and another boolean column with the value *False*, indicating that the central node did not process the data, then it can be assumed that the given row, associated with the execution segment, violated the property.

Figure 5.3 displays what would be the row associated with the time-restricted segment from Figure 5.2. The observer starts in the initial state and moves forward if the data is collected. In the next state, it waits for the signal indicating that the Central Node processed the data to return back to the first state. In the example, since 2 seconds were not enough time for the central node to process the data, it was considered not done whithin the time window of the property. Hence, the column "Central Node processed" is





Segment Number	Sensor Collected	Sensor Processed	Sensor Transfered	Central node Processed	Context Always present	Time between process and transfer	Time until Central Node processed	Label
1	1	1	1	0	0	0.5	*null*	1

Figure 5.3: Example of a feature engineered execution segment

zero, and "Time until Central Node processed" is *null*, and therefore the row is labeled 1, meaning that this segment does violate the property.

5.3.2 Negative Selection Algorithm

This step is the heart of the framework. It aims at using the Negative Selection Algorithm to try and find patterns in the nonself-set, in order to find patterns in the execution segments related to property violations that were not considered at first, and thus enhace the system's observers. This will be done in five steps. First, (i) the data obtained in the last step will be analyzed so that the least relevant features may be discarded. Then, (ii) the remaining features will compose a string that will be used in the NSA. Next, (iii) the NSA will be performed and the nonself detectors will be generated and matured. Following that, (iv) the detector set will be thoroughly examined for the discovery of new context variability. Finally, (v) the information obtained will be used to refine both the system's properties and observers.

i. Exploratory Data Analysis

This initial step focuses on checking the integrity of the dataset and exploring the generated features to verify their relation with the property violation label. Another goal of this step is the selection of the features that will be used in the Negative Selection Algorithm.

Here, a series of dataset validations are conducted as part of a sanity check and preliminar analysis. First, the missing data are taken care of, and the required data types and ranges are adhered to the expected. The columns are then examined both individually and collectively to confirm that the simulated system behaves as expected. For instance, if data wasn't collected, it couldn't be processed. Thus, cases where the opposite happens are handled by either removing the faulty occurrences or by refining the simulation model. Besides that, this analysis helps to decrease the number of features in the dataset by eliminating the columns that provide little or no information, based on their variance or the percentage of missing data.

Next, a correlation analysis takes place. Initially, the correlation matrix is computed, pairing the features and scoring their relationship. High correlations between two features indicate that both provide the same information and, thus, one of them can be removed.

Finally, the features that will be used in the NSA are selected. Our work uses the binary version of the Negative Selection Algorithm, as described in Section 2.4, meaning that only binary features can be used, at first. This being said, the boolean features are set apart and have their relation with the label measured. Since it does not matter much if the feature is positively or negatively correlated, only the absolute value is taken into account. Then, the features are sorted based on the strenght of the relationship and the ones that are below a certain threshold are discarded.

It is noteworthy to bring the reason why the columns were sorted. We shall employ the R-chunk as a matching rule in our work, which indicates that the detector candidate is often shorter than the string from the self-set. It is possible that relevant patterns are not seen by a single detector if the columns are arranged randomly. This happens because the important bits might be positioned far apart and the chunk may be too short to consider all of them at once. The likelihood of this happening is reduced when the columns are arranged according to how strongly they relate to the label. Besides that, measuring the correlation allows us to filter out noisy features and, thus, enhance the result of the experiment.

The output of this step are the adjusted execution segment dataset and the ordered list of boolean features that will be used in the Negative Selection Algorithm.

ii. String formatting

As detailed in Section 2.4.2, the process of generation of detectors happens by measuring the similarity between the detector candidates and the dataset. In the binary version of the Negative Selection Algorithm, this assessment is usually performed by the means of a distance function, which compares the characters in two strings and returns a similarity score.

In order to leverage the simplicity and efficiency of distance funcions in our work, this step will focus of formatting the dataset as binary strings. The input of this process will be the adjusted dataset created in the previous step, with only the boolean features selected after the correlation analysis, and ordered by the strength of the relationship with the label. The bits of the binary string for each row are made up of the contents of each column, with the leftmost bit coming from the column that is most correlated to the label, and the least significant bit comming from the column with the weakest relationship with the label.

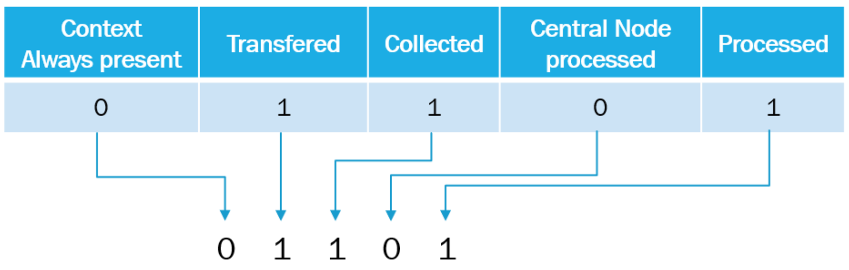


Figure 5.4: Formatting the dataset as binary strings

Figure 5.4 illustrates this process in the execution segment that is being used as an example throughout this section. To present the results of the correlation study from the previous phase, the columns have been sorted, the continuous features have been eliminated, and certain boolean columns have also been deleted.

iii. NSA execution

After gathering the data, preparing the dataset and formatting the strings, the dataset is ready to go through the Negative Selection Algorithm. This subsection will explain the implementation of the training phase of the algorithm, based on the theoretical background in Section 2.4.

Our implementation of the NSA will meet the three requirements listed by Ji and Dasgupta [43] in order to be categorized as a Negative Selection Algorithm. It will make use only of the self-set (i) to randomly generate binary string detectors (ii) in order to identify the self-set counterpart (iii).

The use of only the self-set dataset is the first aspect to be taken into consideration. All rows with the label 1, meaning all rows that indicate a violation of the property of interest will be filtered out of the refined dataset, leaving just the self-set, or the set associated with the typical operation of the system. The idea is that the patterns of the nonself data will be derived from the examination of the self-set alone.

The second aspect considered in the NSA is the usage of detectors. Let X be a string from the self-set with size n, such that $X = x_1x_2x_3...x_n$. A candidate r-sized detector D is another binary string, with $D = d_1d_2d_3...d_r$, with $r \leq n$. The generate-and-test technique, which is further described in Section 2.4.3, will be used to construct the candidates. Simply put, as the name implies, this method generates random binary strings with a size of r to be compared with the strings in the self-set.

The third and final aspect relates to the identification of the self-set counterpart. This will be done by comparing the candidate detectors to the set of self-strings, and discarding the ones that match. In our approach, we will use the R-Chunk distance function for the comparisons (refer to Section 2.4.2 for details). The candidates generated will be tested for each position p, with $0 \leq p \leq n-r$. If there's a match, the candidate is discarded. The remaining ones will be uniquely identified as $t_{p,D}$ and stored for latter analysis. Therefore, the set of detectors will comprise only the candidates that did not match any self-set data.

The algorithm is greatly influenced by two hyperparameters: the size of the detector and the length of the detectors set. As mentioned before, too-short detectors could miss some significant patterns if the relevant features are placed far apart in the string. Long candidates, on the other hand, reduce the algorithm's efficiency since the number of potential strings that may be created at random rises exponentially with the chunk size. Additionally, lengthy detectors suffer from a loss of generalization since they have more fixed bits and, thus, detect less nonself data. The size of the detectors set also have a great impact on the final result, since sets with smaller lenghts may not cover the entire nonself-space and larger sets may increase the time needed to finish the execution.

In this regard, we included a third hyperparameter to workaround the cases when the algorithm takes longer to converge. This parameter indicates the acceptable amount of failed attempts to add new candidates to the detectors set. If this value is reached, the algorithm will exit earlier.

The training of the algorithm will be performed as follows. First, the whole dataset will be split in two: one for the actual training and detector generation, and other for measuring the results and checking how well the detectors perform with unseen data. Second, the training set will have the nonself rows filtered out. Third, random strings of size r will be generated and tested against the self-strings in the training data in all possible positions. If there's a match, the detector candidate is discarded for that particular position. Otherwise, the string is stored along with the position in the detectors set. The third process is repeated until the length of the detectors set reaches a predefined size, or until no new detectors are added to the set after a fixed amount of attempts. Finally, the result of the learning phase is measured by using the detectors to predict property violations in the test set, which contains both self and nonself-data. The actual values are compared with the predicted values and the metrics of precision and recall are evaluated.

iv. Detector Analysis

This step will be responsible for looking at the detectors that were created and understanding the patterns found. The R-Chunk detectors are identified in the set as $t_{p,D}$, with p being the starting position and D the detector string of length r. If step (ii) included mapping boolean features to bits in a string, here the mapping will be done in reverse.

The detectors are sorted based on how many violations are matched. The most relevant ones go through this mapping to perform this analysis.

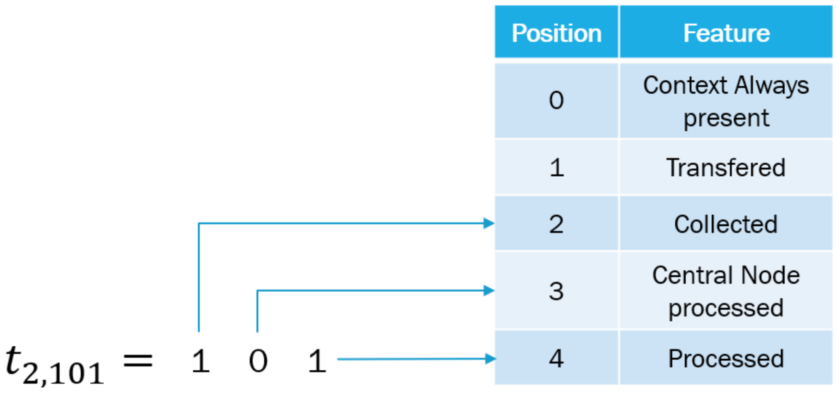


Figure 5.5: Mapping the Detectors to the features

Figure 5.5 shows how the mapping is done. The detector $t_{2,101}$ of size $r = 3$ indicates a property violation pattern in position 2 with the values 101. By looking at the table with the corresponding features and positions, we can see that this pattern identifies the violations that happen when the data is collected, processed, but is not processed by the central node.

One of the three features of the set may already account for a variability that was not yet considered. In this case, the next step could be performed right away. Otherwise, the remaining features that were not used in the generation of the detectors may come into play, like the weakly related booleans and the continuous variables that did not fit the algorithm. These features are analyzed specifically in the rows that were matched by the detector as an attempt to discover new hidden context combinations.

Another possible analysis that can be performed relates to the coverage of the detectors. It is possible that all the nonself-data matched by a detector (i) is also matched by another detector (ii). Therefore, two possible scenarios may arise. First, detectors (i) and (ii) match exactly the same nonself-data. In this case, they are related to the same pattern of anomaly and, thus, can be aglutinated into a single detector. Second, one of the detectors, let's say (i), matches all of the nonself-data matched by (ii) and more. Hence, it is possible to state that (ii) is a specialization of the pattern found in (i), i.e. part of the violations detected by the pattern in (i) can be better detailed by looking at the pattern in (ii). This fact can be useful for the segmentation of the violation patterns found into clusters.

v. Property Refinement

Finally, after the patterns have been unveiled, the system's specification and properties are revisited. Initially, new contexts found are properly elicited and returned to the Contextual Goal Model by a human analyst. Then, we iteratively map them to either existing or newly discovered system properties as well as to system model components. This means that, once we get to the final end of our approach, the updated CGM obtained as a result, can be used as an input for a new execution of the whole methodology.

This stage is comparable to the conclusion of the BIS's adaptive immune response. In the human body, the T-lymphocytes that have undergone differentiation and maturation and have a high affinity for an antigen at hand are pumped via the bloodstream to the site of the inflammation. Whereas in our framework, the collection of NSA detectors triggers the development and improvement of observers, which are then put back into the system model to constantly check out for property violations while monitoring the newly revealed contexts.

Chapter 6

Evaluation

In this chapter, our framework will be evaluated using a variation of the Goal-QuestionMetric (GQM) technique [74]. The GQM plan has been slightly modified by the addition of a new column with the results. The questions pertinent to evaluating our strategy and its outcomes are separated into two primary sections: the first related to the algorithm utilized in the pattern analysis process itself, and the other about the method's overall efficacy. Table 6.1 details the study questions.

Goal 1: Property Violation Identification process		
Question	Metric	Results
How well does our model generalize to unseen data?	Precision and Recall Rates	Precision: 1.0 Recall: 0.9958

Goal 2: Method's contribuition		
Question	Metric	Results
How effective is our approach in the discovery of patterns in property violation scenarios?	Number of patterns found	49 patterns 11 Groups

Table 6.1: GQM Plan

6.1 Experimental Setup

The evaluation of our proposed solution will rely on a implementation of the Body Sensor Network described in Section 4. Our version of this CPS includes a Central Node and five sensors: an oximeter, a thermometer, a heart rate monitor, and an APBD and APBS blood pressure sensors. The patient's readings are assessed within each sensor node, where the health risk percentage of the monitored vital sign is calculated. Subsequently, the risk percentage is relayed to the Central Node, which, in turn, is in charge of collecting and fusing the data from the sensors, assessing the patient's overall risk, and indicating an emergency if one is discovered.

6.1.1 Innate Immunity through Assurances

Prototype Implementation

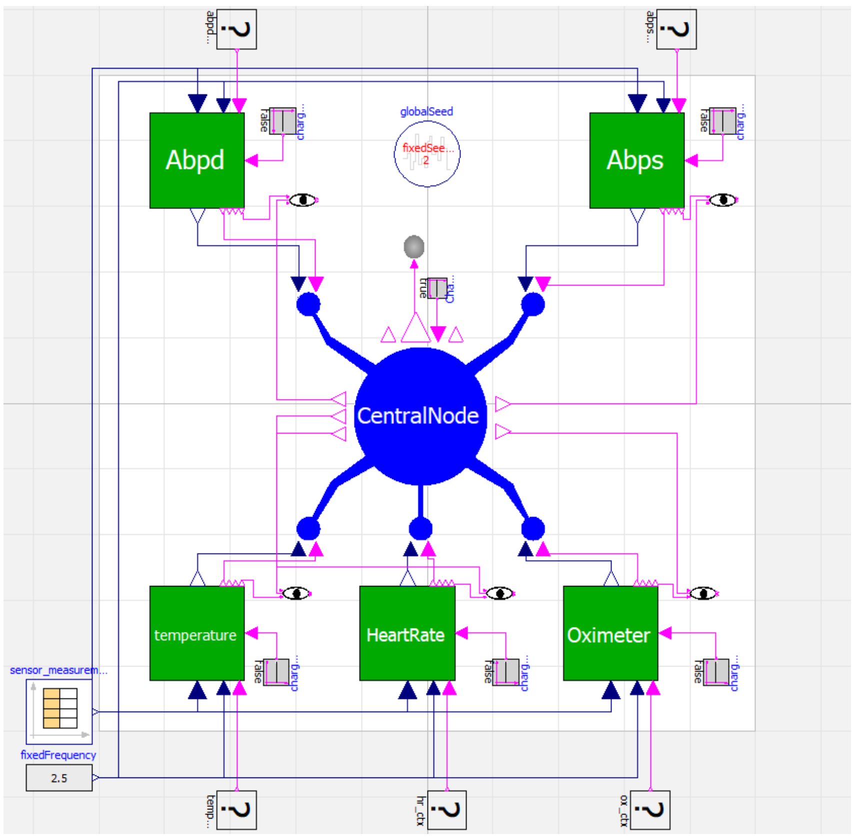


Figure 6.1: BSN Prototype Implemented in OpenModelica

The first part of our technique requires the implementation of a CPS prototype. This is done to anticipate problems that may occur during runtime by modeling and simulating both the system's software components and physical processes. Despite the fact that there are several state-of-the-art approaches for designing such models [67] [71] [72], and since the focus of this work is to elaborate on the property violation patterns using the NSA, a simpler framewrok for the simulation was selected. The Modelica language [18], hence, will be used as the main tool in this process. This language is powerfull enough to represent the acausal continuous-time physical processes of the CPS through equations. It also comprises several built-in libraries for the modeling of circuits, batteries, fluids, noise, equipment deterioration and so on. At the same time, software components can also be described by defining algorithms and functions that account for the behavior of such modules. The graphical aid is achieved by the OpenModelica [17] modeling tool, in which the components and their interactions can be visually modeled as blocks and connectors.

Figure 6.1 depicts the prototype that was implemented in OpenModelica, using the Modelica language. The green blocks account for the sensors, while the blue illustrates the Central Node. In the left lower side, a parameter defines a fixed frequency for data collection that is utilized by every sensor, and above it there is a reference for an external csv file that contains the measurements of the sensors at every second. Around the system, there are some blocks with question marks whose function is to generate random binary numbers to indicate whether the sensor was active. The goal of these blocks is to simulate cases when the sensor stopped responding, or was too far away to transmit reliable data, or had some malfunctioning of sorts that may account for faults in runtime. Next to the sensors there are some gray boxes that are used to indicate when the sensor is plugged into the outlet. Besides that, there are also small rounded blocks that represent the observers, that are modeled to monitor properties of the system associated with each sensor.

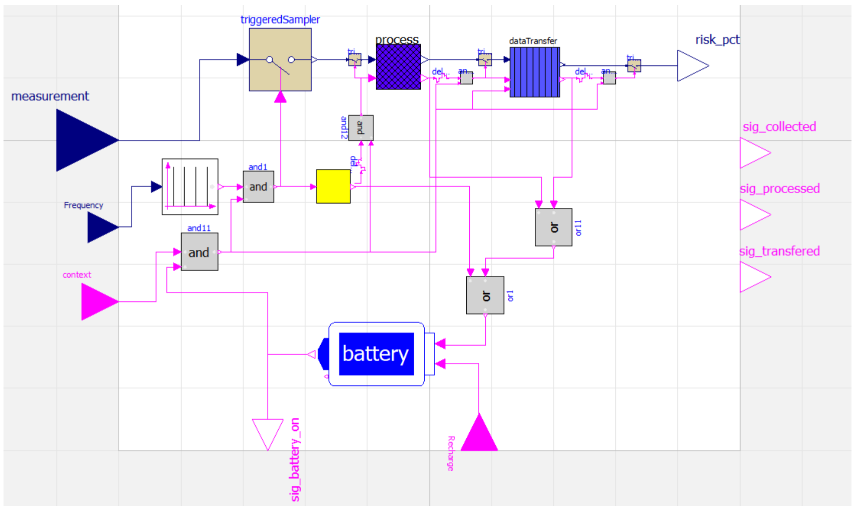


Figure 6.2: The model of a sensor implemented in OpenModelica

By going one step deeper into the model, it is possible to describe how the inner workings of the sensors and the Central Node were implemented. One of the advantages of the Modelica language is its object-oriented character that allows for the reuse of modules throughout the simulation. The sensor described in Figure 6.2 is a good example of this aspect, since the same block is used to model all five sensors. The physical components represented here comprise: a mechanism for sampling a measurement based on a triggered signal, a set of bitwise operators that are often used in microcontroller programming, and a battery. This battery was modeled based on a built-in Modelica library that allows for the modeling of electric circuits, and is composed of a set of switches, a memory cell, and a signal current converter that receives signals from the sensor whenever some process occur to decrease the charge of the memory cell. The magnitude of the charge decrease is multiplied by a randomly generated real number in a way that simulates the decrease that would happen in a real environment. Meanwhile, software components of the sensor are also modeled, like the Process block, which recieves a real valued sensor measurement as an input and relies on an algorithm for determining the health risk percentage of the patient.

The Central Node, illustrated in Figure 6.3, was implemented in a similar fashion. It also makes use of bitwise operations for handling the signals sent, and of an instance of the same battery model as the one found in the sensors. The software components encapsulate functions that fuse the data from the sensors and compute the overall health risk of the patient.

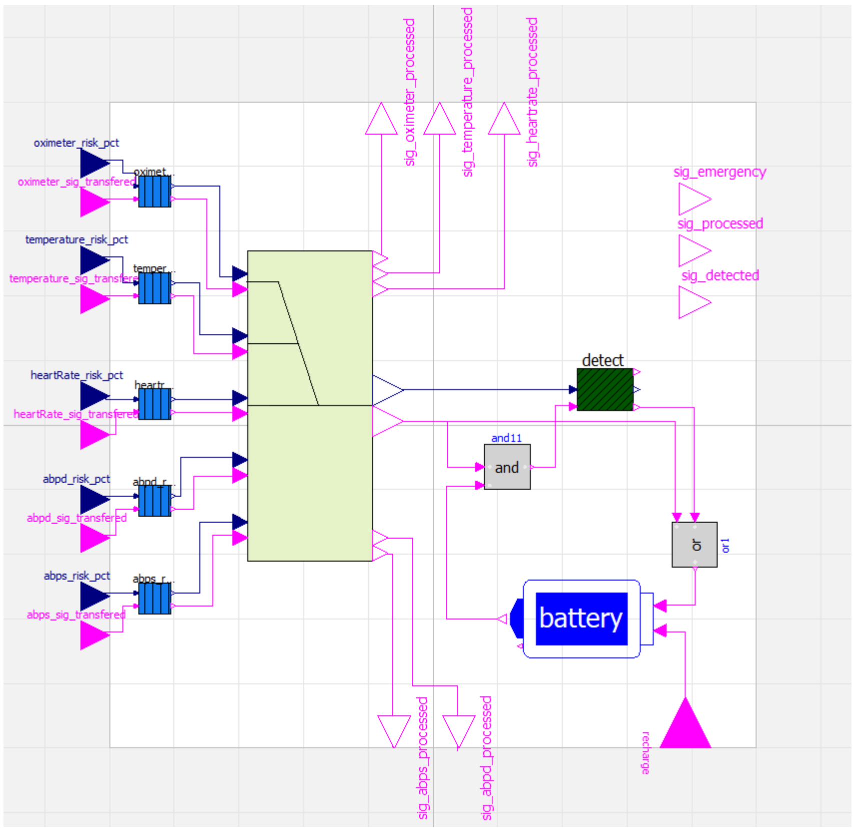


Figure 6.3: The model of the Central Node implemented in OpenModelica

Both the sensors and the Central Node are instrumented through a set of signals that are sent each time a process or a specfic behavior occur, like when the data is collected or transfered for instance. These signals are used by the observers to monitor the properties of interest, and will be used later for the characterization of a execution segment, during the pattern analysis phase. Observers can be easily modeled in Modelica via a built-in library developed for the desing of state graphs. Both the states of the automata and its transitions are described as blocks, with the difference being that the transitions receive signals as input to indicate the passing through the states.

Without loss of generality, to meet the proposed goals of the GQM plan, it will suffice to evaluate the accuracy and efficiency of our technique based on the analysis of a single property. Therefore, Property **P7**, from Section 4 was chosen to guide the next phase of the methodology, since it is related to all the different modules of the CPS. The property states as follows: "Whenever a sensor node has collected data, in within at most 2 seconds, the Central Node will process it." It focuses on the reliability of the system, since its satisfaction guarantees that the collected data will be processed in a reasonable time. Besides that, the Oximeter was the sensor chosen to reify the analysis.

With that being said, the observer related to **P7**, implemented in the work of Carwehl et al. [33], will be deployed in this simulation. The states of the automata were modeled as blocks and the transitions as instances of the *TransitionWithSignal* class of the StateGraph library of Modelica. The signals used in the transitions are provided by the sensor and the Central Node, while a timer is set every time the monitor enters the CollectedReached state and if it does not transition back to the initial state, by the time when the timer runs out, the error state is reached.

Prototype Simulation

With the prototype of the BSN in place, several simulations were executed with varying configurations, patient profiles and environment variables so that the complexities of runtime could be assessed earlier.

As noted in the preceding section, one of the simulation's inputs is a csv file holding the sensors' readings at each second. The patient profile given by these files is constructed using a randomized approach in the Python language in order to be as neutral as feasible. Three functions for generating random points within a range were created: one for indicating a rising tendency, another for indicating a decreasing tendency, and a third for keeping the data trend steady. Initially, a random point is created within the sensor's range of data. Then, one of the three functions and a second value that accounts for the "destination" are randomly picked. The function is then run for the generation of data points starting from the initial value and ending in the destination value, with the selected tendency. For large datasets, the process can be repeated with the destination value becoming the initial value of the new cycle. For example, suppose we are generating the data for the thermometer. The initial value is 36.5C, the destination value generated is 38C and the selected function is the rising tendency. Then a set of datapoints will be randomly genereated in a way that it starts from the initial value and rises until reaching

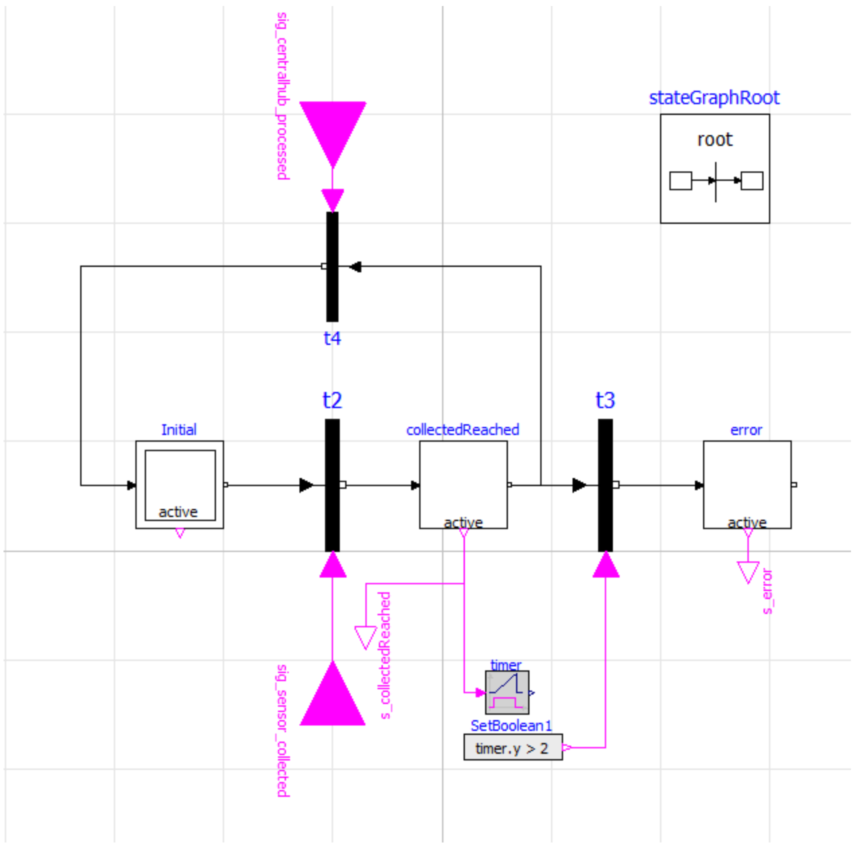


Figure 6.4: The model of the Observer implemented in OpenModelica

38C. This process was performed for each of the five sensors and repeated until the desired data set size.

After the creation of the dataset, the simulation was run. Even though OpenModelica makes it realy easy to perform simulations, it does not scale well, since the csv files and the system configurations must be set manually at each run. To address this problem, a python library named OMPython was used [75]. It is a Python-based interactive session handler for Modelica scripting that is free, open source, and extremely portable. It was utilized to programatically load the modules, alter the parameters and configurations and run the simulation.

To account for as much variability as possible, 1,000 patient profiles were randomly generated and, the same amount of simulations were performed. The operation dataset of each run was extracted and saved as csv files. The simulation was processed in an Intel(R) Core(TM) i5-10210U, 2.10 GHz, 16GB.

6.1.2 Adaptive Immunity Through Learning

After the simulations were run, the extraction of the operation dataset triggered the next phase of the methodology. Tightly related to the adaptive immune response of the BIS, the data was processed and inputed into the Negative Selection algorithm to allow for a more specific response in face of anomalous behavior.

Feature engineering

Initially, a data pipeline was built in the Python language for the processing of the operation dataset. The data was divided into execution segments based on the time constraints of the property **P7** using the procedure outlined in Section 5.3.1. The segments were then summarized into single rows using domain knowledge, and features were extracted from the segment to better characterize it. For example, we created boolean features based on the signals sent between the modules of the system, which may indicate that the data was properly transfered to the Central Node, or that the sensor ceased operating during the computation of the health risk percentage. Other engineered features are related to the description of specific behaviors that might have happened, like the battery running out, or an emergency being detected. Finally, we have also developed real-valued features that reflect, for instance, how long it took to perform some task, or the measurement collected by the sensor.

The labeling task was performed by the development of an algorithm that mimics the behavior of the observer. The observer itself could have been used for this task, by simply looking at its state in the end of the segment. Nevertheless, the error state is a dead end since there are no transitions outside from it. This means that, if this state is reached in the middle of a simulation, all subsequent segments will be also accounted as faulty, even if the system properly handles the situation and goes back to its regular execution. Thus, some adjustments would be required in order to use the observer to pinpoint the traces where property violations happened, which could risk the correctness of the process. This tension was handled by implementing an algorithm that reads the engineered features and replicates the verification that would be performed by the observer. For example, the **P7** observer from Figure 6.4 checks if the sensor collected data and if the Central Node has processed it. Since the execution segments are split based on the time restriction, if the Central Node does not execute, i.e. the boolean feature related to this behavior is set to *False*, then we have found a property violation segment.

Negative Selection Algorithm

For the NSA to be run on the operation dataset, first an Exploratory Data Analysis was performed. For that matter, a sanity check was realized to assess the quality of the data that was generated. Each feature was carefully inspected to see if any faults in the design or in the feature engineering process could be spotted. Examples included columns with an unexpected amount of missing values, wrong data types, and strange behaviors, like data being transfered without the sensor having collected it. After that, the correlation between each pair of features was computed as a means to remove any multicolinearity in the dataset. The idea behind it was that, since highly correlated features bring similar information, one of them can be discarded.

Next, a new correlation analysis took place, but this time only considering the relationship between the boolean features and the property violation label. As explained in Section 5.3.2, only the boolean variables were considered for they would compose the binary string in the Negative Selection Algorithm. The Matthews correlation coefficient (MCC) was utilized for this task [76] since it provides a truthfull and informative description of the relationship between two boolean features. The columns were sorted in a descending way based on the absolute value of the correlation rate, and a threshold of 0.1 was the basis for removing features that were not correlated with the label. The features based on the signals that the observer uses to identify violations, were also removed, since they were used in the making of the label. Table 6.2 shows the 13 features that resulted from this process and their respective position related to the absolute value of the MCC rate. It can be noted that the last 4 features are related to other sensors and have a correlation close to 0.1, which could be easily removed. We, however, have decided to keep them to assess the performance of our model in the presence of noise in the data.

Position	Feature	MCC (Absolute)
0	Oximeter was available during the data transference	0.927324
1	Oximeter transfered data	0.923101
2	Oximeter was available during Central Node data processing	0.849276
3	Oximeter Battery became unavailable at some point	0.508921
4	Oximeter became unavailable during trace	0.462366
5	Oximeter battery became unavailable during data collection	0.300302
6	Oximeter processed data	0.300302
7	Oximeter battery became unavailable during data processing	0.296026
8	Oximeter became unavailable during data transference	0.269047
9	The battery of the ABPD sensor became unavailable	0.136670
10	The battery of the ABPS sensor became unavailable	0.133274
11	The battery of the Heart Rate monitor became unavailable	0.129930
12	The battery of the thermometer became unavailable	0.126224

Table 6.2: Selected Features and their respective positions

Finally, the Negative Selection Algorithm was run, based on the simulation generated dataset and on the selected features. The training phase aims at generating the nonself detectors that will hold the patterns of the anomalous segments. First, the nonself-data, i.e. the rows labeled as property violations, were removed from the set. Then, Afterwards, the execution segments were modeled as binary strings, following the features and positions from Table 6.2. Next, while the desired number of detectors was not reached, a binary string of length $r = 5$ would be generated and tested against the execution segment strings for matching. In the case of a match, the detector candidate would be discarded. Otherwise, it was appended in a list of nonself detectors. The matching function works as follows: for each position p, with $0 \leq p \leq 8$, the algorithm would check if there was a substring of length $l = r = 5$ in the self-set, starting in position p, that was equal to the candidate detector. If the algorithm unsuccessfully attempts to generate a viable detector for a predefined number of times, then the algorithm goes through an early stop.

6.2 Goal 1: Property Violation Identification process

Question: How well does our model generalizes to unseen data? To answer this question propertly, the operation dataset extracted from the simulations was splitted into two: 75% of the set was used for the detector's generation, and the other 25% was set apart to account for the unseen data. The entire dataset had 19868 rows, with 16493 (83%) related to regular operation and 3375 (17%) of the execution segments violated property **P7**. Hence, the split resulted in 14901 rows being utilized in the training phase, but only 12370 rows were used for the generation of the nonself detectors, since the process only makes use of the self-set, i.e. the majority class.

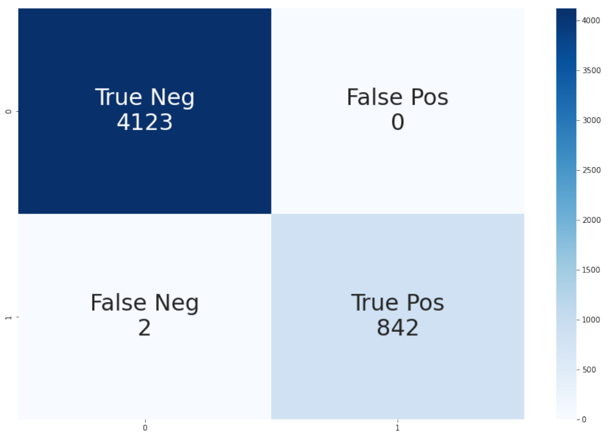


Figure 6.5: NSA's Confusion Matrix

We are using the metrics of precision and recall to assess the performance of the model, since they provide a good summary of the confusion matrix. The recall rate is computed as *True Positives* / (*True Positives* + *False Positives*), i.e. how many of the relevant property violations were found, while the precision rate is calculated as *True Positives* / (*True Positives* + *False Negatives*), and is the same as asking "how many property violations found were relevant." Our main interest was in having a high precision rate, since we wanted to have the lowest value of *False positives* as possible. This happens because the goal was to study the generated detectors to understand the patterns found, and having *False positives* could impact in this task. On the other hand, having *False Negatives* would not have such impact, once they would be related to the patterns that were not discovered by the detectors, thus causing less harm to the overall approach. Figure 6.5 depicts the confusion matrix of the results of the model. No self-data was considered as faulty, meaning that there were no *False Positives*, as desired. Nevertheless, only 2 violations were not assessed by the NSA, which is within an acceptable range.

Table 6.3 shows the precision and recall rates of our approach using the Negative Selection Algorithm over the simulation dataset in relation to the property **P7** for the Oximeter sensor. The accuracy and MCC rates were also computed as a way to have a complementary view on the performance. Besides that, the same labeled dataset was also used as input for some well known Machine Learning algorithms. The One-class SVM and the Isolation Forest were brought to this comparison since they are popular state-of-the-art semi-supervised anomaly detection algorithms [19], and because the One-class SVM is usually compared with the NSA for they both use only one of the classes during the training phase [3]. The Decision tree, the Random Forest and the Gradient Boosting algorithms [20, 21] were also compared for its broad usage in the Machine Learning comunity.

Model	Accuracy	Precision	Recall	MCC
Negative Selection	**0.9995**	**1.0**	**0.9976**	**0.9985**
One-Class SVM	0.7634	0.4180	1.0	0.5467
Isolation Forest	0.7173	0.2928	0.4691	0.2002
Decision Tree	0.9995	1.0	0.9976	0.9985
Random Forest	0.9995	1.0	0.9976	0.9985
Gradient Boosting	0.9995	1.0	0.9976	0.9985

Table 6.3: NSA Performance comparison

From the table, it is clear that the NSA performed way better than its counteparts in the anomaly detection field. Athough it had a very similar performance compared to Random Forest ant Gradient Boosting, the interpretability provided by the algorithm through the observers is paramount to handle not only the detection of property violations, but also to provinding relevant information for the analyst to enhance the verification of the CPS. This will be much more clear in the evaluation of Goal 2.

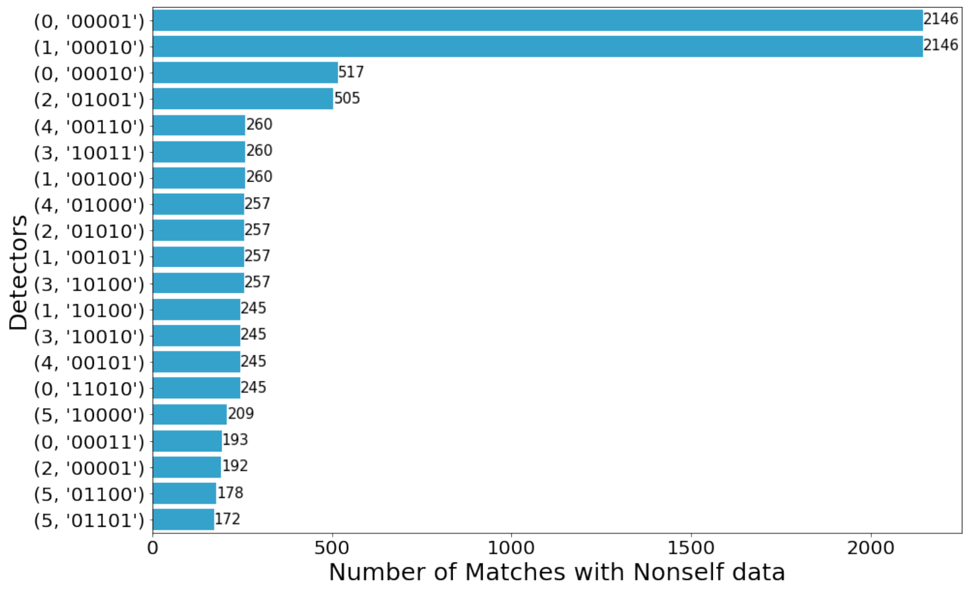


Figure 6.6: Top 20 Detectors with most matches

6.3 Goal 2: Method's contribution

Question: How effective is our approach in the discovery of patterns in property violation scenarios? This question will be assessed by delving deeper on the Detector Analysis step of the proposed methodology. As illustrated in Figure 5.5, the detector's string are reverse mapped to the features by using the values from Table 6.2 and the detector's position.

After running the Negative Selection Algorithm, 229 binary strings were created that did not match with any data from the self-set. Hence, these detectors were designed to cover the self-set complimentary space. However, this does not mean that a detector *must* match with a faulty execution segment, i.e. not all detectors generated are usefull. With that being said, the detectors that did not match nonself data were considered of less importance and, thus, discarded from the analysis. Therefore, from the 229 detectors, we were left with only 68 usefull binary strings.

The bar chart in Figure 6.6 displays the amount of nonself data that were matched by the top 20 detectors. This chart provides the analyst a better understanding of which are the most common and relevant patterns that were found by the algorithm.

By looking at the chart from Figure 6.6, one can easily see that the first two detectors match the same amount of nonself data. Besides that, their patterns seem to overlap, since if they were placed next to each other, considering their position, the difference between the two is the leftmost bit on the first one and the rightmost bit on the second one. This raised the possibility of two or more detectors having detected the same pattern but, because of the fixed size of the R-chunk, they were separated. In order to verify this

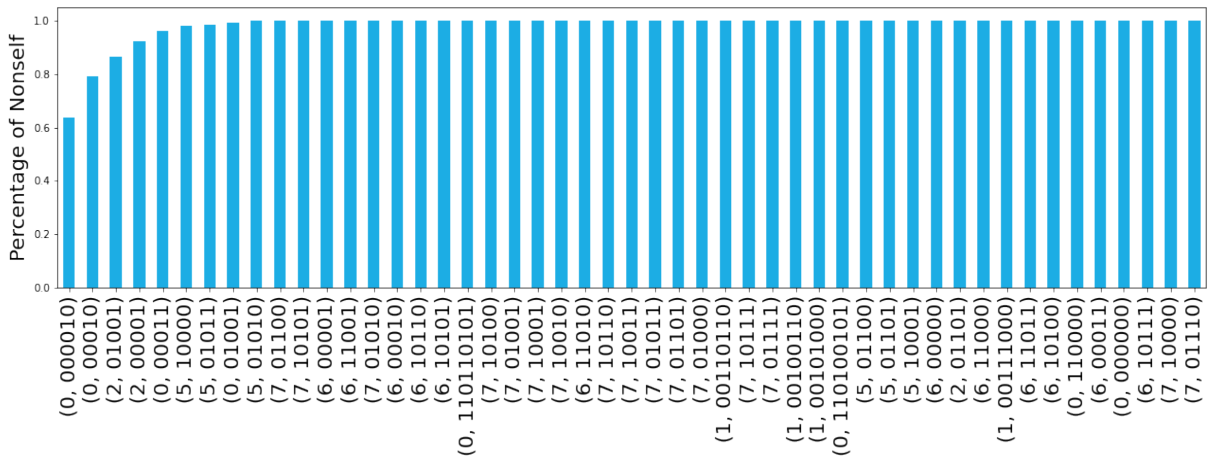


Figure 6.7: Coverage of Nonself data matched by detectors

situation, the MCC rate was computed for each pair of detectors. In the case of a pair being perfectly correlated, we could infer that they had matched the exact same nonself data, thus, they could be seen as the same pattern stored in different detectors. The idea was to agglutinate such pairs in order to have more robust patterns. For example, the first two detectors from Figure 6.6 became a single 6-bit string detector, starting in position 0, with the bits *000010*. After identifying and agglutinating all the identified cases, the total amount of detectors became 49.

Another aspect that need to be considered is the fact that the sum of matches is still greater than the amount of property violation cases in the dataset. Figure 6.7 depicts the percentage of nonself data covered with the addition of the detectors one by one, sorted by relevance. From the chart, we see that only 9 detectors are capable of detecting all the anomalous behavior of the system.

This information raises a relevant fact. The set of nonself data matched by some detectors may comprise the entire set of other detectors with less matches. Another way of looking at it is to think that the patterns found by some detectors are part, or even specializations, of greater patterns of nonself data. Figure 6.8 attempts to illustrate this concept. By analyzing the detectors, we saw that all of the patterns matched by the detector *(1,'00111000')* were also matched by *(0,'00011')*. Moreover, the detector *(0,'00011')* also matched all the nonself data matched by *(1,'001101100')*. Therefore, it is possible to state that the pattern *(0,'00011')* is specialized into the two other patterns, meaning that the bigger pattern can happen under two different circumstances, accounted by the two other patterns.

Figure 6.9 shows the reverse mapping of the highlighted patterns. The detectors were placed in their respective positions, side by side, and the relevat rows from Table 6.2 were brought for a better understanding of the patterns found. The leftmost pattern is the one

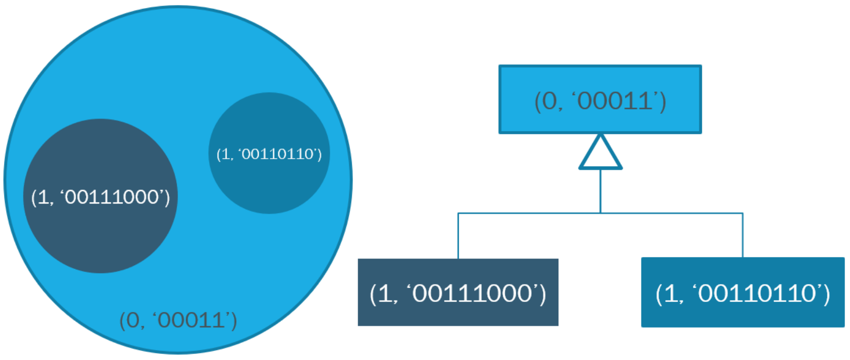
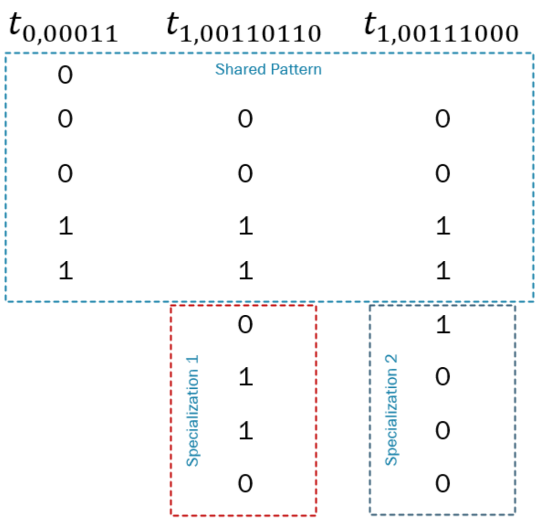
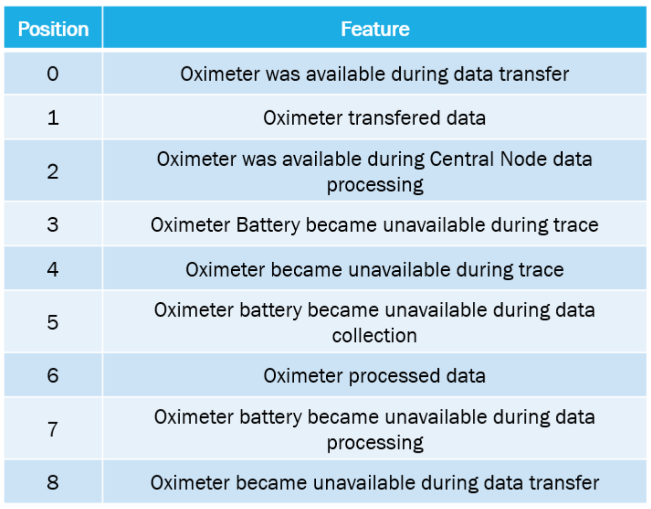


Figure 6.8: Specialization of Patterns

that is specialized in the other two. By the boolean values and the positions, we see that it is related to the violations that happen when the data transference from the sensor to the Central Node did not happen (signal 0 in position 1) and the battery ran out at some point during the trace (signal 1 in position 3). The observer in the middle share this same pattern, but specifies the cases when the battery ran out during the processing of the data (signal 1 in position 7). In the meanwhile, the righmost observer points to the scenarios when the battery ran out while the sensor was acquiring the patient's vital signs. In summary, this group of detectors indentified the cases where the violation happened because the data was not transfered due to the battery running out either during the collection or the processing of the data.





Position	Feature
0	Oximeter was available during data transfer
1	Oximeter transfered data
2	Oximeter was available during Central Node data processing
3	Oximeter Battery became unavailable during trace
4	Oximeter became unavailable during trace
5	Oximeter battery became unavailable during data collection
6	Oximeter processed data
7	Oximeter battery became unavailable during data processing
8	Oximeter became unavailable during data transfer

Figure 6.9: Reverse Mapping of the Specialized Patterns

Finally, the patterns and their specializations were all computed and are laid out in Figure 6.10. The Sankey chart helps us not only to understand the relationship amongst the detectors, but it also provides a visual aid for identifying the strength of the relation. In the figure, the most relevant detectors are placed on the left side of the pair, and

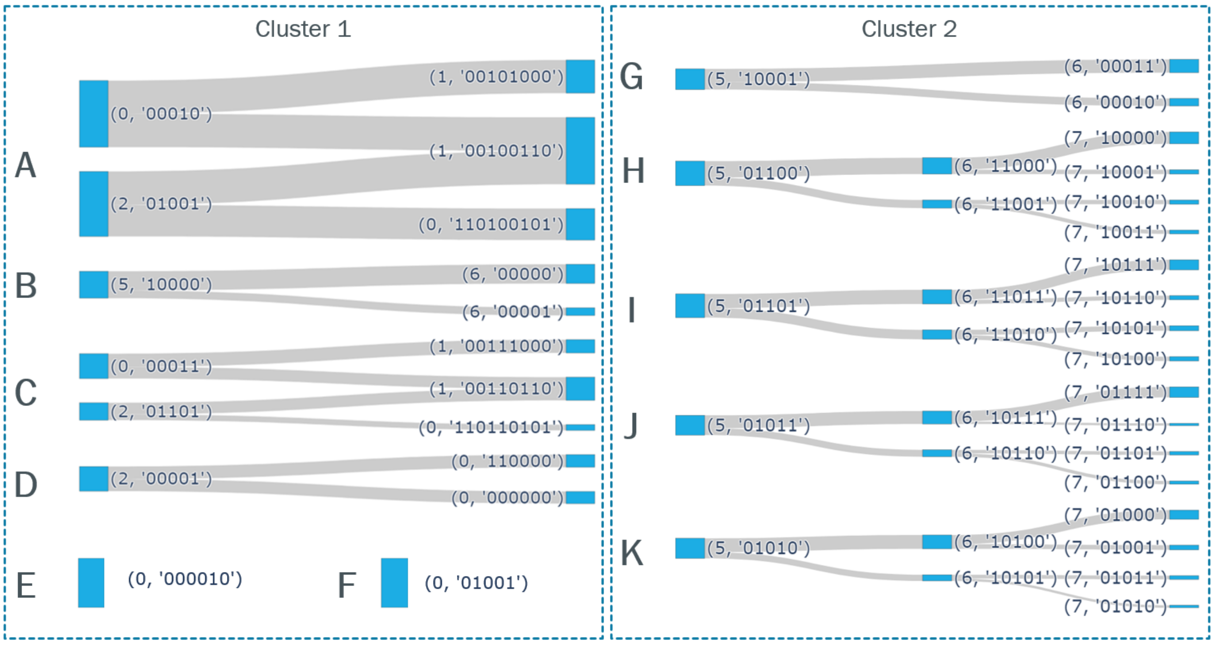


Figure 6.10: Groups of Detectors identified based on the similarity of patterns

their specialization are displayed in the right side. The strength of the relation can be seen by the line that connects them. The larger the line connecting two points is, the more nonself data are detected by the pair. Some relations are simple as the exemple from Figure 6.8, but others can bring more complex structures, like the group formed by detector *(5, ’01011)*. The pattern groups were named and further divided into two clusters, based on their instrinsic characteristics.

Table 6.4 elaborates on the sankey chart from Figure 6.10. It provides more information on the pattern that was identified by each detector group, along with the respective percentage of nonself data matched within each set. Cluster 1 comprises the patterns that matched the most, with the positions closer to 0. This points to the relevance of these patterns, since they include the features that are more related to the violation data. A total of 63% of the nonself set matched with the detector from Group E, the highest value. From the description, we can see that it accounts for the cases when the sensor stopped responding because it was too far away to transmit reliable data, or it had some malfunctioning that may account for faults in runtime. Nevertheless, the absency of the sensor was already addressed in the specification of the BSN [5], and thus is not a novelty that would impact the design of the observer.

Differently, Group A matches the cases in which the data was not transmitted due to the battery running out either during the data collection, processing or transfering. It was the second in percentage of matches, accounting for near 21% of the total of anomalous data. Even though the detectors were positioned on lower bits, Group B goes in a similar

Pattern Group	Detector 1	Detector 2	Detector 3	Pattern Description	Percentage matched
A	(0, '00010')	(1,'00101000')		Lack of battery during data collection	7,61%
	(0, '00010')	(1,'00100110')	(2,'01001')	Lack of battery during data processing	7,70%
	(2,'01001')	(0,'110100101')		Lack of battery during data transfering	7,26%
B	(5,'100000')	(6,'00000')		Lack of battery during data collection	4,41%
	(5,'100000')	(6,'00001')			1,78%
C	(0,'00011')	(1,'00111000')		Lack of battery during data collection	3,05%
	(0,'00011')	(1,00110110')	(2,'01101')	Lack of battery during data processing	2,67%
	(2,'01101')	(0,'110110101')		Sensor absent during transfer	1,33%
D	(2,'00001')	(0,'110000')		Sensor absent during Central Node processing	2,90%
	(2,'00001')	(0,'000000')			2,79%
E	(0,'000010')			Sensor absent at some point	63,59%
F	(0,'01001')			Sensor absent during Central Node processing	0,68%
G	(5,'10001')	(6,'00011')		Lack of battery during data collection	2,84%
	(5,'10001')	(6,'00010')			1,63%
H	(5,'01100')	(6,'11000')	(7,'10000')	Lack of battery during data processing	2,58%
	(5,'01100')	(6,'11000')	(7,'10001')		0,95%
	(5,'01100')	(6,'11001')	(7,'10010')		0,92%
	(5,'01100')	(6,'11001')	(7,'10011')		0,83%
I	(5,'01101')	(6,'11011')	(7,'10111')	Lack of battery during data processing	2,16%
	(5,'01101')	(6,'11011')	(7,'10110')		0,89%
	(5,'01101')	(6,'11010')	(7,'10101')		1,10%
	(5,'01101')	(6,'11010')	(7,'10100')		0,95%
J	(5,'01011')	(6,'10111')	(7,'01111')	Lack of battery during data transfering	2,28%
	(5,'01011')	(6,'10111')	(7,'01110')		0,50%
	(5,'01011')	(6,'10110')	(7,'01101')		0,77%
	(5,'01011')	(6,'10110')	(7,'01100')		0,71%
K	(5,'01010')	(6,'10100')	(7,'01000')	Lack of battery during data transfering	1,99%
	(5,'01010')	(6,'10100')	(7,'01001')		1,01%
	(5,'01010')	(6,'10101')	(7,'01011')		0,80%
	(5,'01010')	(6,'10101')	(7,'01010')		0,53%

Table 6.4: Description of the patterns found in each Detector Group

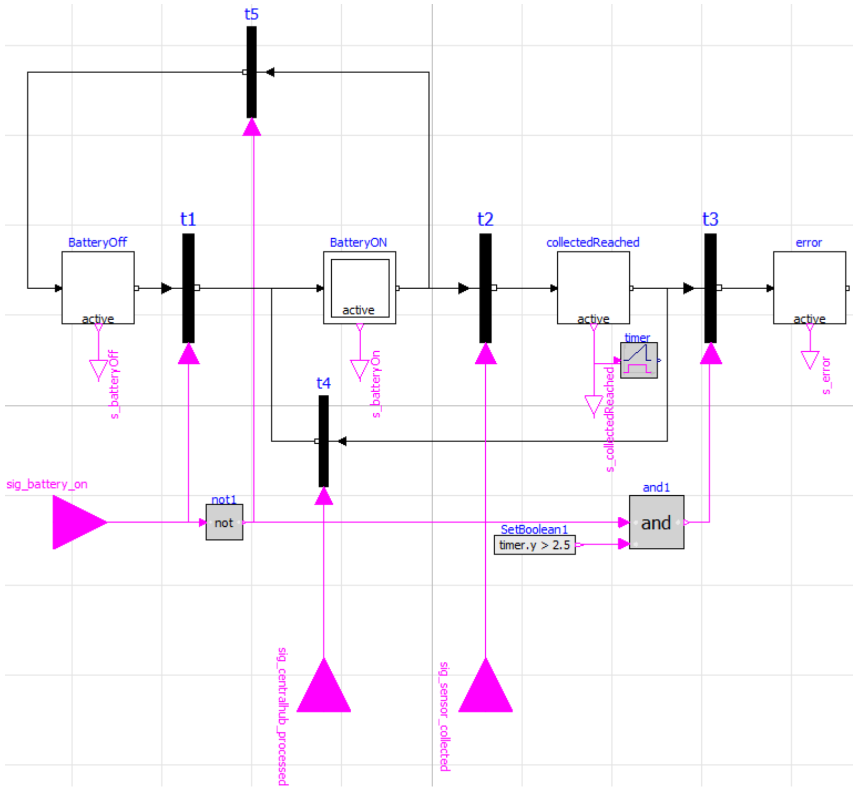


Figure 6.11: P7's Observer updated

direction, by matching violations related to the lack of battery during the acquisition of data, though the bits in the mid-region of the string. Group C, in turn, defined a set of mixed patterns that account for both the battery running out and the malfunctioning of the sensor. Therefore, from these three, it is possible to infer that the charge of the battery is key for the fulfillment of the property **P7**.

Nevertheless, not all the unveiled patterns were informative. Groups D and F, for example, describe scenarios where some further details would be required. For some reason, the absence of the sensor was related to the Central Node not being able to process the data transmitted, which is not very intuitive.

On the other hand, Cluster 2, comprised of groups G through K, represents just a few portion of the nonself dataset, and have a very similar description to groups A, B and C. By looking closely at the detectors, it is possible to verify that the changes happen mainly in the lower bits, which are related to the other sensors' batteries (refer to table 6.2). Based on the domain knowledge, those features are identified as noise since they should not impact directly in the violation of the property for a specific sensor. The threshold defined during the feature selection step brought to the analysis some features that describe the battery of other sensors, which may have caused some unneeded patterns to be formed. Hence, the patterns from this cluster did not add much to the analysis.

All in all, this analysis should provide the CPS analyst with substantial information to address the anomalous situations. By looking at scenarios unveiled by the groups, it is possible to understand that the battery is key for the satisfaction of the property **P7**. This happens since the cell may run out during the collection, the processing, or the transfering of the data, making it impossible for the Central Node to process it within the time restriction. Therefore, it could be the case for the analyst to update this observer, so that it would consider this newly unveiled variability. Figure 6.11 illustrates the changes that could be made to the observer of the property **P7**, based on the Observer catalog from [33]. A new state related to the battery being off would be created so that, if this component runs out after the data was acquired, then the error state would not be reached.

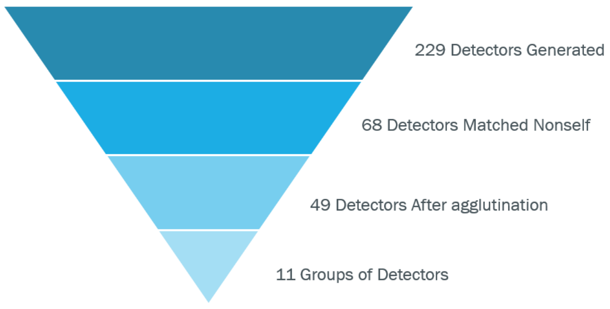


Figure 6.12: Number of Detectors at each step

Therefore, these violation-related detector groups generated from our technique account for the effectiveness of the approach in the discovering of patterns in the simulated data since they were shown to be relevant for the refinement of property **P7** runtime monitor. Figure 6.12 illustrates the overall process of finding relevant patterns. From the 229 detectors generated by the Negative Selection Algorithm, only 68 have matched with nonself data. From those, we concluded that some of them were actually the same pattern, splitted into two or more detectors due to the size limitation of the R-chunk, and thus, could be agglutinated. After merging them, the resultant 49 detectors were grouped into 11 clusters of patters that share a common behavior. These cluster were further analyzed by reverse mapping the strings into features, resulting in the extraction of relevant information about the violation scenarios, and the refinement of the runtime monitor related to the property at hand.

Lastly, the Decision Tree algorithm (DT) was performed in order to compare our approach to another technique. The same data served as input, and the output is a tree depicted in Figure 6.13. Each node of the tree is related to one of the features, with their position is related to its importance. By analyzing it, it is noteworthy that the there is a whole subtree related only to the other sensor's battery, which we already made

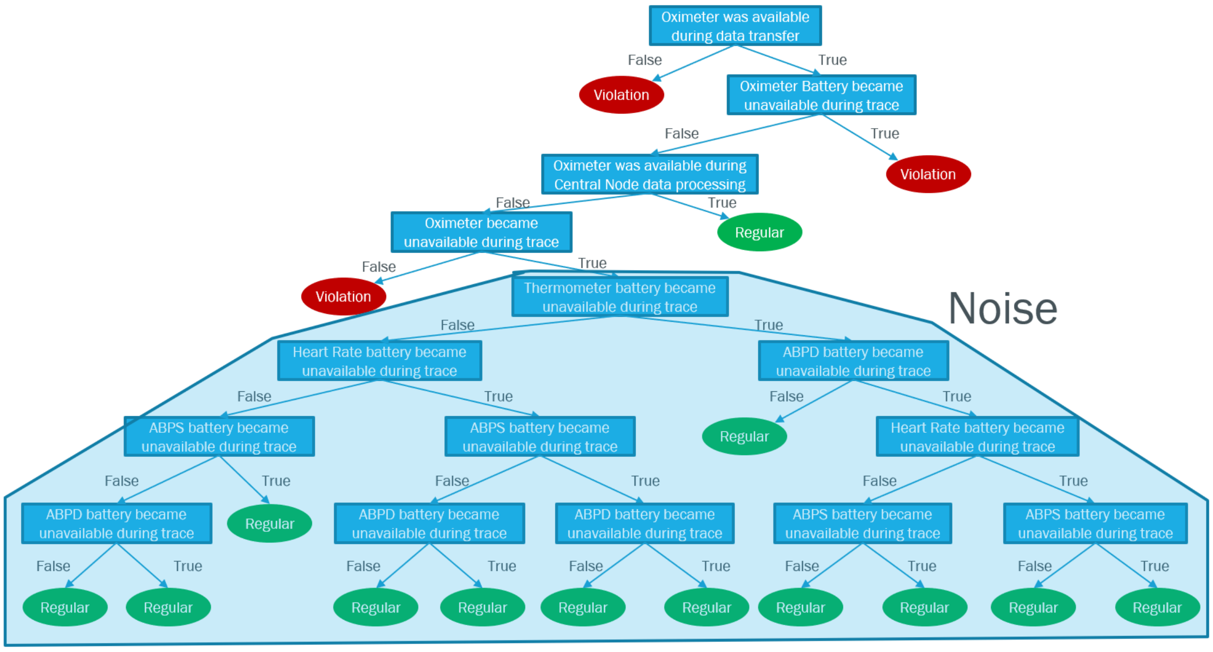

Oximeter was available during data transfer
False
True
Violation
Oximeter Battery became unavailable during trace
False
True
Violation
Oximeter was available during Central Node data processing
False
True
Regular
Oximeter became unavailable during trace
False
True
Violation
Noise
Thermometer battery became unavailable during trace
False
True
Heart Rate battery became unavailable during trace
ABPD battery became unavailable during trace
False
True
Regular
ABPS battery became unavailable during trace
ABPS battery became unavailable during trace
Heart Rate battery became unavailable during trace
Regular
ABPD battery became unavailable during trace
ABPD battery became unavailable during trace
ABPD battery became unavailable during trace
ABPS battery became unavailable during trace
ABPS battery became unavailable during trace
Regular

Figure 6.13: Comparison - Patterns found by using a Decision Tree

the case that it accounts for noise in the data, and thus can be discarded. Therefore, only three remaining patters are left, which can be related to groups E, for the one in the highest level of the three; groups A, B and C for the second highest; and groups D and F for the lowest one. The subtree with the noise found is related to groups G through K. In comparison, our approach was able to identify a greater number of patterns, besides find specializations of each. For example, the DT identified a pattern in which the violation happened because the battery was unavailable at some point during the execution segment. In the meanwhile, our proposed methodology not only found that, but also pointed that this could happen during the data collection, processing and transfering, with the relevance for each scenario. Therefore, even though the precision and recall rates were similar between the two approaches, the explainability provided by ours allows for a better assessment of the reasons behind the property violations.

6.4 Discussion

Even though there are several methods for verifying a system for assurance purposes, such as Model Checking or runtime verification, the structural, dynamic, and organizational complexities of Cyber-Physical systems may become an impediment in this process, both in modeling and in defining the set of properties that will be assessed. The technique proposed in this paper seeks to fill this gap by utilizing well-studied frameworks for modeling the complex link between the discrete-time behavior of software components and the continuous-time acausal equations that govern the physical processes of such systems. Furthermore, the simulated data analysis enables the detection of patterns in data that could not have been accounted for when the system properties were initially elicited. Thus, our technique provides a better understanding of the context in which the CPS will be introduced, allowing the verification process to be enhanced.

Our technique's quantitative evaluation yielded satisfactory results. The use of the Negative Selection Algorithm resulted in performance that was quite close to that of other proeminent Machine Learning algorithms, including those specialized in anomaly detection. The NSA gave a low *False Positive* rate in the presence of unseen data, meaning that no mistakes that could harm the proposed technique occurred. Aside from that, the examination of the patterns discovered by the detectors was also quite good. We were able to select 49 detectors from the 229 produced detectors that were truly important for characterizing property violations that may occur at runtime. Those detectors were also distributed into 11 clusters based on the similarity of the discovered patterns, further assisting the software analyst by displaying the components that are closely connected to the source of the anomalous behavior.

6.5 Threats to validity

- **Construct validity:** We relied on a documented case study (BSN) and its publicly accessible data to ensure that we present a valid and sound input for our evaluation. The specification of the system, in the form of a CGM, as well as the properties derived from it and the observers utilized were all taken from literature. Besides that, the patient profiles were generated based on a randomized process in order to minimize as much as possible bias in the data. However, in spite of our efforts to avoid generating inaccurate data, we were not able to guarantee that the created data represents accurately a real-world circumstance. Further research is required to validate such representation.
- **Internal validity:** Our evaluation relies significantly on the BSN prototype implemented in Modelica, since the patterns discovered are only as good as the dataset used to find them. Therefore, a correct abstraction of the behavior of the BSN into a model for simulation is paramount. The model must account for as much variability as possibile, according to what will be faced during the execution of the CPS. Nevertheless, it is of our understanding that this is not an easy task. Even with the right tools and state-of-the-art framework, the expert's knowledge of the system is highly required, since several aspects of the system are specific for the medical domain. Besides that, differential equations are used to model the behavior

of the physical components, which would also need a more specialized opinion. As stated by Baier and Katoen [27], "Any verification using model-based techniques is only as good as the model of the system."

Moreover, the choice for the binary version of the Negative Selection Algorithm also comes with its disadivantages. Even though modeling the behavior of the CPS as boolean features allows for simplicity and the reverse mapping of binary detectors provide a great level of interpretability, the Binary Negative Selection (BNSA) is well-known for its performance and completeness issues. D'haeseleer et al. [44] pointed out that the generation of binary detectors is a very time-conusming task, that rises exponentially with the size of the self-set. They also state the possible existence of "holes", which are strings for which is impossible to generate valid detectors. This would account for property violation patterns impossible to be unveiled, thus, decreasing the efficiency of our approach. Finally, they also say that an accurate and stable definition of self is key for the success of the BNSA, which is not always the case due to the dynamic complexities of CPS.

- **External validity:** Even though our technique is not domain specific, we understand the evaluation's limitations because it was applied in single example of the medical field. Further study of the technique is required to determine the approach's true usefulness for generalization goals. To assess the technique's real applicability for purposes of generalization, more examination of the technique must be done.

Chapter 7

Conclusion and future work

In this work, we proposed a methodology to systematically enhance the verification task performed by runtime monitors in Cyber-physical systems. The inherent complexities in the relation between the cybernetic and the physical natures may increase the challenge of verifying such systems. The runtime monitors initially designed may suffer from the lack of knowledge of the runtime environment, which might impact in the ability of the system analyst to understand the set of features or behaviors responsible for the anomalous executions.

To address that, we propose the implementation of a prototype of the system, by using a set of tools that are specifically designed to model CPS. Several simulations are performed in the prototype, generating an operation dataset. This set will be reshaped, labeled as property violations or as regular executions, and inputed into the NSA. This immune-based algorithm is capable of generating a set of detectors specialized in identifying property violations. This detectors are then studied as a means to extract the patterns related to the anomalous behavior. Finally, these patterns are used to enhance the runtime monitors, providing the CPS with a more robust verification and, thus, reliable execution.

Our approach was evaluated using the Body Sensor Network. The overall result of the NSA was very satisfactory, with high values of precision and recall, very similar to other machine learning techniques. The advantage of the NSA concerns the generated detectors, which provide the methodology with a high interpretability of the patterns found. In the end, 49 patterns were found, which were distributed in 11 groups based on similarity.

In future works, we plan to expand the analysis of the patterns by using the formalism found in the causality analysis research field to identify the root cause of the property violation. The design of the CPS would greatly benefit from this, since the analyst would be provided with the tools needed to tackle the actual source of the problem, and not only the indicative of where the problem might be.

Even though we tried to be as unbiased as possible, our evaluation relied much on data that was randomly generated. Thus, in a near future, we intend to check how our methodology performs in a real wold scenario, by leveraging data from patients with varying diseases, ages and other important aspects. In this opportunity, we also plan to evaluate our methodology not only for specific properties, but also for each specific problem. This happens because our approach perform well for young patients with heart diseases, but not for older patients with lung cancer, for instance. This may happen also for the variability in the configuration of the simulation. On the one hand, we may find that the NSA performs well for identifying the battery, but not so much for unreliable sensor mesurements, for example. Hence, a thorough evaluation based on predefined scenarios would help us better understand the strenght of the methodology.

Since the evaluation was performed on a single study case, there is a need for broadening the reach of our approach to other domains. Hence, other plans are related to the implementation of study cases with more complex CPS. We see the BSN as a simple, yet powerfull, proof that the concept holds. Nevertheless, modeling a CPS consisting of several different modules, conected through networks, with a set of dynamic complexities and so on would be a interesting way of stressing the proposed solution and seeing if it still holds.

We also see the need for provinding a deeper evaluation of our methodoly by using a more robust and complex example of CPS. Due to time limitations, we have restricted ourselves to a limited number of variabilities that could be found in the BSN environment. Maybe using a real world application, with a prototype designed with industrial standards would help us to assess better the complex relations between the physical and cybernetic aspects and, thus, provide a better understanding of the performance of the approach.

Finally, in this work we made use of the binary version of the NSA in order to leverage its high interpretability. Nevertheless, there can be found in the literature more advanced versions of the algorithm [3], which represent the data as real values and whose detectors are complex strucutres that solve many of the shortcommings found in the binary version. In the future, we intend to look for ways to utilize these state-of-the-art versions of the NSA in our approach, without losing in interpretability.

References

[1] Aleksandrowicz, Gadi, Eli Arbel, Roderick Bloem, Timon D ter Braak, Sergei Devadze, Goerschwin Fey, Maksim Jenihhin, Artur Jutman, Hans G Kerkhoff, Robert Könighofer, *et al.*: *Designing reliable cyber-physical systems.* In *Languages, Design Methods, and Tools for Electronic System Design*, pages 15–38. Springer, 2018. x, 6, 7

[2] Punt, Jenni, Sharon A. Stranford, Patricia P. Jones, and Judith A. and Owen. W. H. Freeman Macmillan Learning, New York, eighth edition edition, 2019. x, 14, 15, 25, 34, 35

[3] Gupta, Kishor Datta and Dipankar Dasgupta: *Negative Selection Algorithm Research and Applications in the Last Decade: A Review.* IEEE Transactions on Artificial Intelligence, PP:1–1, September 2021. x, 16, 17, 18, 19, 25, 27, 53, 65

[4] D'haeseleer, Patrik: *An Immunological Approach to Change Detection: Theoretical Results.* pages 18–26, July 1996, ISBN 0-8186-7522-5. x, 21

[5] Rodrigues, Arthur, Ricardo Caldas, Genaina Rodrigues, Thomas Vogel, and Patrizio Pelliccione: *A Learning Approach to Enhance Assurances for Real-Time Self-Adaptive Systems*, May 2018. x, 29, 30, 57

[6] Lee, Edward A: *Cyber physical systems: Design challenges.* In *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*, pages 363–369. IEEE, 2008. 1, 6

[7] Colombo, Christian, Gordon J Pace, and Gerardo Schneider: *Runtime Verification: Passing on the Baton.* In *Formal Methods in Outer Space*, pages 89–107. Springer, 2021. 1, 10, 24

[8] Mohamed, Mustafa Abshir, Geylani Kardas, and Moharram Challenger: *Model-Driven Engineering Tools and Languages for Cyber-Physical Systems–A Systematic Literature Review.* IEEE Access, 9:48605–48630, 2021. 2, 35

[9] Patankar, M.S.: *Safety Ethics: Cases from Aviation, Healthcare and Occupational and Environmental Health.* CRC Press, 2020, ISBN 9781000083002. `https://books.google.com.br/books?id=FO_eDwAAQBAJ`. 2

[10] Lemos, Rogério, David Garlan, Carlo Ghezzi, Holger Giese, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Danny Weyns, Luciano Baresi, Nelly Bencomo, Yuriy Brun, Javier Cámara, Radu Calinescu, Myra Cohen, Alessandra Gorla, Vincenzo Grassi,

Lars Grunske, Paola Inverardi, Jean Marc Jézéquel, and Franco Zambonelli: *Software Engineering for Self-Adaptive Systems: Research Challenges in the Provision of Assurances*, pages 1–29. September 2017, ISBN 978-3-319-74182-6. 2, 8

[11] Baheti, Radhakisan Sohanlal and Helen Gill: *Cyber-Physical Systems.* 2019 IEEE International Conference on Mechatronics (ICM), 2019. 2

[12] Yegnanarayana, Bayya: *Artificial neural networks.* PHI Learning Pvt. Ltd., 2009. 3

[13] Bäck, Thomas and Hans Paul Schwefel: *An overview of evolutionary algorithms for parameter optimization.* Evolutionary computation, 1(1):1–23, 1993. 3

[14] Dasgupta, Dipankar and Luis Fernando Nino: *Immunological Computation: Theory and Application.* Ingeniería e Investigación, 29:140–140, April 2009. 3, 13, 14, 18, 19, 20, 21, 22

[15] Gao, Zhiwei, Carlo Cecati, and Steven X Ding: *A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches.* IEEE transactions on industrial electronics, 62(6):3757–3767, 2015. 4

[16] Gil, Eric Bernd, Ricardo Caldas, Arthur Rodrigues, Gabriel Levi Gomes da Silva, Genaína Nunes Rodrigues, and Patrizio Pelliccione: *Body Sensor Network: A Self-Adaptive System Exemplar in the Healthcare Domain.* In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 224–230, 2021. 4, 28

[17] *Open Modelica.* `https://openmodelica.org/`. Accessed: 2022-10-27. 4, 34, 46

[18] *Modelica Language.* `https://modelica.org/modelicalanguage.html`. Accessed: 2022-10-27. 5, 24, 34, 45

[19] Villa-Pérez, Miryam Elizabeth, Miguel A Alvarez-Carmona, Octavio Loyola-González, Miguel Angel Medina-Pérez, Juan Carlos Velazco-Rossell, and Kim Kwang Raymond Choo: *Semi-supervised anomaly detection algorithms: A comparative summary and future research directions.* Knowledge-Based Systems, 218:106878, 2021. 5, 53

[20] Prajwala, TR: *A comparative study on decision tree and random forest using R tool.* International journal of advanced research in computer and communication engineering, 4(1):196–199, 2015. 5, 53

[21] Bentéjac, Candice, Anna Csörgő, and Gonzalo Martínez-Muñoz: *A comparative analysis of gradient boosting algorithms.* Artificial Intelligence Review, 54(3):1937–1967, 2021. 5, 53

[22] Banerjee, Ayan, Krishna K Venkatasubramanian, Tridib Mukherjee, and Sandeep Kumar S Gupta: *Ensuring safety, security, and sustainability of mission-critical cyber–physical systems.* Proceedings of the IEEE, 100(1):283–299, 2011. 6, 7

[23] Schätz, Bernhard: *The role of models in engineering of cyber-physical systems–challenges and possibilities.* In *Tagungsband des Dagstuhl-Workshops*, page 91, 2014. 6, 7

[24] Jazdi, Nasser: *Cyber physical systems in the context of Industry 4.0.* In *2014 IEEE international conference on automation, quality and testing, robotics*, pages 1–4. IEEE, 2014. 6

[25] Dowdeswell, Barry, Roopak Sinha, and Stephen G MacDonell: *Finding faults: A scoping study of fault diagnostics for Industrial Cyber–Physical Systems.* Journal of systems and software, 168:110638, 2020. 6

[26] Bolbot, Victor, Gerasimos Theotokatos, Luminita Manuela Bujorianu, Evangelos Boulougouris, and Dracos Vassalos: *Vulnerabilities and safety assurance methods in Cyber-Physical Systems: A comprehensive review.* Reliability Engineering & System Safety, 182:179–193, 2019. 7

[27] Baier, Christel and Joost Pieter Katoen: *Principles of Model Checking*, volume 26202649. January 2008, ISBN 978-0-262-02649-9. 8, 9, 10, 13, 63

[28] Weyns, Danny, Nelly Bencomo, Radu Calinescu, Javier Cámara, Carlo Ghezzi, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean Marc Jézéquel, Sam Malek, Raffaela Mirandola, Marco Mori, and Giordano Tamburrelli: *Perpetual Assurances for Self-Adaptive Systems.* 2019. `https://arxiv.org/abs/1903.04771`. 8

[29] Autili, Marco, Lars Grunske, Markus Lumpe, Patrizio Pelliccione, and Antony Tang: *Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar.* IEEE Transactions on Software Engineering, 41(7):620–638, 2015. 10, 11, 12

[30] Zheng, Xi, Christine Julien, Miryung Kim, and Sarfraz Khurshid: *Perceptions on the State of the Art in Verification and Validation in Cyber-Physical Systems.* IEEE Systems Journal, PP:1–14, November 2015. 10, 23, 27

[31] Leucker, Martin: *Teaching runtime verification.* In *International Conference on Runtime Verification*, pages 34–48. Springer, 2011. 10

[32] Havelund, Klaus, Kim Larsen, and Arne Skou: *Formal Verification of a Power Controller Using the Real-Time Model Checker UPPAAL.* Volume 1601, pages 277–298, May 1999, ISBN 978-3-540-66010-1. 11

[33] Carwehl, Marc, Thomas Vogel, Genaina Rodrigues, and Lars Grunske: *Property Specification Patterns for UPPAAL.* `https://github.com/hub-se/PSP-UPPAAL`, 2022. 11, 13, 24, 31, 48, 60

[34] Bengtsson, Johan, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang yi: *UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems.* Volume 3, pages 232–243, January 1995. 11, 13

[35] Aceto, Luca, Augusto Burgueño, and Kim G. Larsen: *Model checking via reachability testing for timed automata.* In Steffen, Bernhard (editor): *Tools and Algorithms for the Construction and Analysis of Systems*, pages 263–280, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg, ISBN 978-3-540-69753-4. 11

[36] Henzinger, Thomas, Xavier Nicollin, Joseph Sifakis, Sergio Yovine, and Miniparc zirst Lavoisier: *Symbolic Model Checking for Real-Time Systems.* Information and Computation, 111, August 1996. 13

[37] KamalMishra, Prashant and Mamta Bhusry: *Artificial Immune System: State of the Art Approach.* International Journal of Computer Applications, 120:25–32, June 2015. 14, 16

[38] Farmer, J.Doyne, Norman Packard, and Alan Perelson: *The Immune System, Adaptation, and Machine Learning.* Volume 22, October 1986. 14, 16

[39] Aickelin, Uwe, Dipankar Dasgupta, and Feng Gu: *Artificial immune systems*, pages 187–212. January 2014. 15, 35

[40] Garrett, Simon: *How Do We Evaluate Artificial Immune Systems?* Evolutionary computation, 13:145–77, February 2005. 16

[41] Naqvi, Syed Moeen, Merve Astekin, Sehrish Malik, and Leon Moonen: *Adaptive Immunity for Software: Towards Autonomous Self-healing Systems*, January 2021. 16

[42] Forrest, Stephanie, Alan Perelson, Lawrence Allen, and Rajesh Cherukuri: *Self-Nonself Discrimination in a Computer.* Proceedings of the International Symposium on Security and Privacy, November 1995. 17, 21

[43] Ji, Zhou and Dipankar Dasgupta: *Revisiting Negative Selection Algorithms.* Evolutionary computation, 15:223–51, February 2007. 18, 40

[44] D'haeseleer, Patrik, Stephanie Forrest, and Paul Helman: *An immunological approach to change detection: algorithms, analysis and implications.* pages 110–119, May 1996. 18, 22, 63

[45] González, Fabio, Dipankar Dasgupta, and Jonatan Gomez: *The Effect of Binary Matching Rules in Negative Selection.* Volume 2723, pages 195–206, July 2003. 19, 20

[46] Chmielewski, Andrzej and Slawomir Wierzchon: *Hybrid Negative Selection Approach for Anomaly Detection.* September 2012, ISBN 978-3-642-33259-3. 21

[47] Ayara, M., Jon Timmis, R. Lemos, Leandro De Castro, and R. Duncan: *Negative selection: How to generate detectors.* Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS), January 2002. 21, 22

[48] Luckcuck, Matt, Marie Farrell, Louise A Dennis, Clare Dixon, and Michael Fisher: *Formal specification and verification of autonomous robotic systems: A survey.* ACM Computing Surveys (CSUR), 52(5):1–41, 2019. 23

[49] Akella, Ravi and Bruce M McMillin: *Model-checking BNDC properties in cyber-physical systems.* In *2009 33rd annual ieee international computer software and applications conference*, volume 1, pages 660–663. IEEE, 2009. 23

[50] Botaschanjan, Jewgenij, Manfred Broy, Alexander Gruler, Alexander Harhurin, Steffen Knapp, Leonid Kof, Wolfgang Paul, and Maria Spichkova: *On the correctness of upper layers of automotive systems.* Formal aspects of computing, 20(6):637–662, 2008. 23

[51] Jones, Austin, Zhaodan Kong, and Calin Belta: *Anomaly detection in cyber-physical systems: A formal methods approach.* In *53rd IEEE Conference on Decision and Control*, pages 848–853. IEEE, 2014. 23

[52] Webster, Matt, Neil Cameron, Michael Fisher, and Mike Jump: *Generating certification evidence for autonomous unmanned aircraft using model checking and simulation.* Journal of Aerospace Information Systems, 11(5):258–279, 2014. 24

[53] Zhang, Xinan, Gilbert Foo, Mahinda Don Vilathgamuwa, King Jet Tseng, Bikramjit Singh Bhangu, and Chandana Gajanayake: *Sensor fault detection, isolation and system reconfiguration based on extended Kalman filter for induction motor drives.* IET Electric Power Applications, 7(7):607–617, 2013. 25

[54] Poon, Jason, Palak Jain, Ioannis C Konstantakopoulos, Costas Spanos, Sanjib Kumar Panda, and Seth R Sanders: *Model-based fault detection and identification for switching power converters.* IEEE Transactions on Power Electronics, 32(2):1419–1430, 2016. 25

[55] García, Emilio, Neisser Ponluisa, Eduardo Quiles, Ranko Zotovic-Stanisic, and Santiago C Gutiérrez: *Solar panels string predictive and parametric fault diagnosis using low-cost sensors.* Sensors, 22(1):332, 2022. 25

[56] Abid, A, MT Khan, IU Haq, S Anwar, and J Iqbal: *An improved negative selection algorithm-based fault detection method.* IETE Journal of Research, pages 1–12, 2020. 25

[57] Chopdar, Sumitsingh Mohansingh and A.K. Koshti: *Fault Detection and Classification in Power System Using Artificial Neural Network.* In *2022 2nd International Conference on Intelligent Technologies (CONIT)*, pages 1–6, 2022. 25

[58] Govender, P and DA Kyereahene Mensah: *Fault diagnosis based on the artificial immune algorithm and negative selection.* In *2010 IEEE 17Th International Conference on Industrial Engineering and Engineering Management*, pages 418–423. IEEE, 2010. 25

[59] Alizadeh, Esmaeil, Nader Meskin, and Khashayar Khorasani: *A negative selection immune system inspired methodology for fault diagnosis of wind turbines.* IEEE transactions on cybernetics, 47(11):3799–3813, 2016. 26

[60] Ren, Yanheng, Xianghua Wang, and Chunming Zhang: *A novel fault diagnosis method based on improved negative selection algorithm.* IEEE Transactions on Instrumentation and Measurement, 70:1–8, 2020. 26

[61] Fritzson, Peter: *Introduction to modeling and simulation of technical and physical systems with Modelica.* John Wiley & Sons, 2011. 27

[62] Pessoa, Leonardo, Paula Fernandes, Thiago Castro, Vander Alves, Genaína N Rodrigues, and Hervaldo Carvalho: *Building reliable and maintainable dynamic software product lines: An investigation in the body sensor network domain.* Information and Software Technology, 86:54–70, 2017. 28, 29

[63] Lo, Benny PL, Surapa Thiemjarus, Rachel King, and Guang Zhong Yang: *Body sensor network–a wireless sensor platform for pervasive healthcare monitoring*, 2005. 28

[64] Ali, Raian, Fabiano Dalpiaz, and Paolo Giorgini: *A goal-based framework for contextual requirements modeling and analysis.* Requirements Engineering, 15(4):439–458, 2010. 29

[65] Henzinger, Thomas A, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine: *Symbolic model checking for real-time systems.* Information and computation, 111(2):193–244, 1994. 30

[66] Vogel, Thomas, Marc Carwehl, Genaína Nunes Rodrigues, and Lars Grunske: *A property specification pattern catalog for real-time system verification with UPPAAL.* Information and Software Technology, page 107100, 2022. 30

[67] Baras, John S: *Formal Methods and Tool-Suites for CPS Security, Safety and Verification.* Engineering Secure and Dependable Software Systems, 53:1, 2019. 34, 45

[68] Balestrini-Robinson, Santiago, Dane F Freeman, and Daniel C Browne: *An object-oriented and executable SysML framework for rapid model development.* Procedia Computer Science, 44:423–432, 2015. 34

[69] Higham, Desmond J and Nicholas J Higham: *MATLAB guide.* SIAM, 2016. 34

[70] Blochwitz, Torsten, Martin Otter, Martin Arnold, Constanze Bausch, Christoph Clauß, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Manuel Monteiro, Thomas Neidhold, *et al.*: *The functional mockup interface for tool independent exchange of simulation models.* In *Proceedings of the 8th international Modelica conference*, pages 105–114. Linköping University Press, 2011. 34

[71] Deng, Fei: *Modeling and Simulation of CPS based on SysML and Modelica (KG).* 2019. 35, 45

[72] Bouskela, Daniel, Alberto Falcone, Alfredo Garro, Audrey Jardin, Martin Otter, Nguyen Thuy, and Andrea Tundis: *Formal requirements modeling for cyber-physical systems engineering: An integrated solution based on FORM-L and Modelica.* Requirements Engineering, 27(1):1–30, 2022. 35, 45

[73] Nguyen, Thuy: *FORM-L: A Modelica Extension for properties modelling illustrated on a practical example.* In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, number 96, pages 1227–1236. Linköping University Electronic Press, 2014. 35

[74] Van Solingen, Rini, Vic Basili, Gianluigi Caldiera, and H Dieter Rombach: *Goal question metric (gqm) approach.* Encyclopedia of software engineering, 2002. 44

[75] Consortium, Open Source Modelica: *OpenModelica Python Interface and PySimulator.* `https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/ompython.html`, 2022. 49

[76] Chicco, Davide and Giuseppe Jurman: *The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation.* BMC genomics, 21(1):1–13, 2020. 51