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

Enhancing Property Specification of Cyber-Physical Systems using Negative Selection

João Paulo C. de Araujo

Dissertação apresentada como requisito parcial para qualificação 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 Property Specification of Cyber-Physical Systems using Negative Selection

João Paulo C. de Araujo

Dissertação apresentada como requisito parcial para
qualificação do Mestrado em Informática

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

Prof. Dr. Patrizio Pelliccione
Gran Sasso Science Institute

Prof. Dr. Lars Grunske
Humboldt-Universität zu Berlin

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Programa de Pós-graduação em Informática

Brasília, 11 de Novembro de 2022

Resumo

Os sistemas ciberfísicos são uma realidade definitiva na nossa vida quotidiana, especialmente nos últimos anos. No entanto, a complexidade inerente a estes domínios suscita alguns desafios como por exemplo acontecimentos imprevistos, dificuldades em representar os processos cibernéticos ou físicos, e o conhecimento incompleto dos contextos do ambiente. Tudo isso pode tornar os CPS pouco fiáveis em tempo de execução, causando efeitos desastrosos.

Procurar inspiração em processos de outros campos é uma atividade muito comum na Informática. A natureza, especialmente a biologia, há muito que serve como uma fonte frutuosa de metodologias como as abordagens de inteligência artificial. O Algoritmo de Selecção 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 especificação do sistema.

Neste trabalho, é proposta uma metodologia que visa aumentar a fiabilidade de CPS. Isto é feito através de um diagnóstico sistemático das violações das propriedades do sistema baseado em dados gerados por um protótipo, realizado nas fases iniciais de desenvolvimento. Um algoritmo de inspiração imunológica chamado Seleção Negativa (NSA) serve como método de redundância analítica para isolar e identificar a causa da violação de propriedade no sistema. É possível que, através do arrazoamento sobre as razões pelas quais as violações de propriedade acontecem, a especificação do sistema e a própria propriedade possam ser refinadas, 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

Cyber-physical systems are a definite reality in our day-to-day lives, specially in the recent years. Nevertheless, the complexity inherent of these domains raises some challanges like unforeseen events, difficulties in depicting either the cyber or the physical processes, and the incomplete knowledge of the environment contexts, for example, might make the CPS unreliable at runtime, which could have disastrous effects.

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. 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 specification.

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. 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 system specification and the property themselves may be refined, fault-tolerant mechanisms may be added, and, thus, safer and better applications might be written.

Keywords: Cyber-physical Sistems, 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 [4]. They are a definite reality in our day-to-day lives, specially in the recent years. It is fair to say that the development of autonomous systems and the Internet of Things has taken us quite near to the future visions we had a few decades ago. Flying cars 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 are increasingly 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.

Modern techniques view formal methods as the most promising means of delivering such evidences [5]. Model checking, for instance, is a model-based method run tipically offline to check the fulfillment of goals and properties for all attainable states in the model. However, this is a difficult undertaking yet to be perfectly achieved [6]. 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 [5]. These uncertainties 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 complexities of representing both the cyber and physical aspects of any CPS challenges the task of modeling such systems. Oversimplified models may be invalidated for not anticipating failures dependant on the two layers [7]. Often times, the verification mechanism well depicts either the cyber or the physical process, but not both. Physical processes, for instance, are modeled by differential equations while the discrete behavior can expressed by state machines [8]. Even though this two-fold segmented modeling may suffice for the development of CPSs in some cases, it creates a significant challenge for verifying the safety and correctness of the overall system and the physical-behavioral interactions at a component level.

In an ideal scenario, developers should be granted complete knowledge of the system before entering the implementation phase. This would allow the verification at design time through the examination or the anticipation of the potential context and system configurations . In reality, both scenarios are impractical since there are too many potential outcomes to predict. Additionally, because the information needed is only available at runtime, it becomes challenging to forecast from the earliest phases of development. All of this can lead to inaccurate estimations of a system's reliability [7].

Hence, there is a need of new ways to address these issues, that account for the complexity of performing verification in these CPSs in a way that increases the reliabitiy of the system while still in the early phases of development.

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, Evolutionary Algorithms (EA) were based on the Darwinian theory of evolution, and the biologia immune system led to the creation of Artificial Immune Systems (AIS). These new developing soft computing approaches have already been employed in a variety of disciplines since the early 1990s.

The Negative Selection Algorithm [9], 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, isolating the fault and enabling the development of fault-tolerant mechanisms that increase the reliability 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.

Therefore, after taking into account the existing state of the field, and all the restrictions mentioned in the preceding sentences, we condense the goal of this study into the following research question:

RQ: how to accurately enhance property specifications of CPS despite the complex interactions between its physical and cyber nature?

1.2 Proposed Solution

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. 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 system specification and the property themselves 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, the system specification in the form of a Contextual Goal Model (CGM) is used as input to model the behavior of the system as timed automata. Then, by using pattern catalogs, real-time system properties are derived from the CGM and expressed as observer automata. The system model along with the observers, then, undergo a model checking process to assert the system's correctness and the satisfiability of the properties. Finally, a prototype of the system is implemented and several simulations are performed with varying configurations, input data and parameters in order to account for the context variability that may arise in runtime.

The second phase focus on the causality analysis of property violations in the data extracted from the prototype. The data, initially, is feature engineered and labeled as self or nonself, meaning regular data traces or faulty traces that incurr in violation. Then, the NSA is executed by generating random detectors based on the features of the dataset. These detectors go through a censoring process, in which they are tested against the self data 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 [10], 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 context variability 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 follows to the second phase of our approach, which can be compared to the Adaptive Immunity response of the BIS. The biologica 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 produce useful observer automata similar to T cells.

1.3 Evaluation

We experimentally assess our strategy by designing a version of the Body Sensor Network (BSN) [11]. 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. The CPS, along with its observers, are going to be modeled as timed automata in the UPPAAL tool [12], where the Model Checking process will also take place. Then, a system prototype will be implemented in a simulation environment called OpenModelica, which is a well known graphic modeling tool for the development of CPS prototypes in the Modelica language. 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 efficiency of the framework will also be accounted for by two different assessments. First, in the usage of observers as a means for verification. This will be done by performing A/B tests on the simulated prototype with and without the mechanism in order to evaluate the consumption of battery in the two scenarios. Second, the efficiency of our implementation of the Negative Selection Algorithm will be assessed. In this case, we will evaluate how well it performs in comparison to other well known change detection statistical methods like Random Forest , in relation to metrics like precision and recall.

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. Chapter 4 presents some works that are strongly related. Chapter 3 futher details the proposed approach. Finally, in Chapter 5, the planning of execution of the research is presented.

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 [4]. 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 [13].

It's not new for physical and digital processes to be integrated into a system. The notion of engineered systems that integrate computer softwares with physical processes have been refered to as "embedded systems" for some time [4]. Car electronics, games, weapons systems, household appliances, and toys can be named as a few successful examples. In order to delineate the distinction, Schatz [14] states that, even though CPSs encompasses embedded systems, the cyber-physical integration in CPSs happens both on a local and a global scale. Lee [4] and Jadzi [15] goes in a similar direction by distinguishing CPSs and embedded systems in terms of networking and outsourcing of computational power. Jadzi [15] 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 [16] adds that, in this sense, CPSs can be seen as seamless entities that form entire systems.

A CPS comprises three main modules [15]: 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, which interfaces the two. Besides that, a communication interface is also required for the data exchange with other CPSs, a central unit or a cloud.

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 [17]. Banerjee et al. [13] 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 [14] 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 [17], 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 [5]. 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 [18]. 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 [5]. 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. [19] 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 [5]. 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.
- **Runtime Verification:** A system-driven lightweight procedure that relies on gathering data from an active system to determine whether specific properties are being violated.
- **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.

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 [18]. 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 model checking process in a correct manner, the properties must be specified in a precise and unambiguous way [18]. 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. [20] 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.

2.3 Goal Oriented Requirements Engineering

The degree to which a software system satisfies its requirements has a significant impact on its quality [21]. One of the lenses through which these requirements can be seen is the stakeholder goals. Goal-Oriented Requirements Engineering (GORE) approach that follow a goal-oriented perspective and alludes to the usage of goals for every activity of the Requirements Engineering area and aims to acquire and represent systems and software requirements at the intentional level [22]. Goal modeling, in particular, reflects what stakeholders hope to accomplish in terms of goals and potential alternate solutions.

Lamsweerde, in [23], sheds light on the reasons why the Requirement Engineering (RE) field can benefit from a goal-driven perspective. He says that goals offer a precise criterion for requirements completeness and pertinence, which are amongst the main RE concerns. Goals also helps stakeholders understand the reason behind the requirements, since they are easy to grasp by nontechnical decision-makers, they naturally offer the rationale behind them, and they allow for the traceability between the strategic goals and the technical details. Moreover, goals guide the derivation of the requirements that supports them.

The next sections will focus on explaining what is meant by goals, and on detailing a well known framework for goal-oriented contextual modeling and analysis.

2.3.1 Goals

According to the systematic literature review done by ElSayed et al. [24], two major definitions of the term *goal* can be found. The first one, introduced by Lamsweerde, says that "goals are objectives achieved by the system through cooperation of agents in the software-to-be and in the environment". While the second one, made by Anton, states that goals are high-level aims that aid in understanding why the system is necessary, and thus, allowing for decisions to be made in accordance to them.

The formulation of the system's goals specify the properties that need to be assured. In this sense, goals can be crafted depending on the desired level of abstraction. They can reflect high level strategic concerns or low level technicalities [23]. Goals also address many sorts of concerns, including both functional and nonfunctional. Functional goals describe goals that are associated with the services to be delivered, whereas nonfunctional goals are related to the system quality.

Another criterion used to characterize goals divides them based on whether they are quantifiable [24]. On the one hand, hard goals are the ones that can be formally verified. On the other hand, soft goals cannot have their fulfillment easily determined. The latter allows the comparison of different alternatives of refinement. Lamsweerde also mentions way of verifying the completeness of the elicited requirements. Let R be the set of requirements, As the environment assumptions, D the domain properties and G the set of goals. For each goal g, such that $g \in G$, it must be true that:

$$R, As, D \models g \text{ with } R, As, D \not\models \text{False}$$

2.3.2 Contextual Goal Model (CGM)

Goal modeling is the technique responsible for representing systems in GORE. As mentioned previously, it provides a powerful way of understanding the needs of the stakeholders besides figuring out the motives behind the development of a piece of software. In the end, a goal model revolves around laying out user goals and ways to meet them [25].

TROPOS [26] is a software development methodology used to model early requirements. In its heart, there is his modeling language, which is the base for constructing conceptual goal models. It rests in the concepts of actor, goal, plan, resource, dependency, capability and belief. An Actor is an entity that possess intentionality and represents agents, roles or positions. An actor can have none to multiple goals. A Goal is related to the interests of a particular actor, and is divided in soft goals and hard goals, meaning goals whose definition and satisfiability criteria are unclear, for the former, or trenchant, for the latter. A Plan can be defined as "a way of doing something", and as a declaration of which refinement paths should be taken in the directed tree in order to achieve the main goal. A Resource corresponds to an entity with physical or informational attributes. Dependencies are subordination relationships between two actors which point out that one depends on the other, the depender and the dependee, severally. The Capability express an actor's competency to define, choose and execute a plan in order to satisfy a goal. Finally, a Belief declares the world's knowledge of an actor.

A goal model usually utilizes a directed graph tree to delineate the goals in a top-down manner so that goals can be successively refined via AND/OR decomposition and ultimately satisfied by the leaf nodes, which represent tasks that must be performed by actors. On the one hand, AND-refinements tie in a goal with a group of subgoals, and the fulfillment all the subgoals is the only sufficient condition for its fulfillment. On the other hand, the OR-refinement relates a goal to a set of alternative subgoals, meaning that meeting one of the subgoals is enough for satisfying it. The refinement comes to an end when each subgoal is utterly achievable by some particular agent capable of monitoring and controlling it [22]. Figure 2.1 depicts both AND- and OR-Refinements.

Since variants were only allowed by regular Goal Models, but not clear-cut defined, an extension of TROPOS, named Contextual Goal Model (CGM), was created to meet the notions of context and goals. It specifies explicitly which changes in the environment

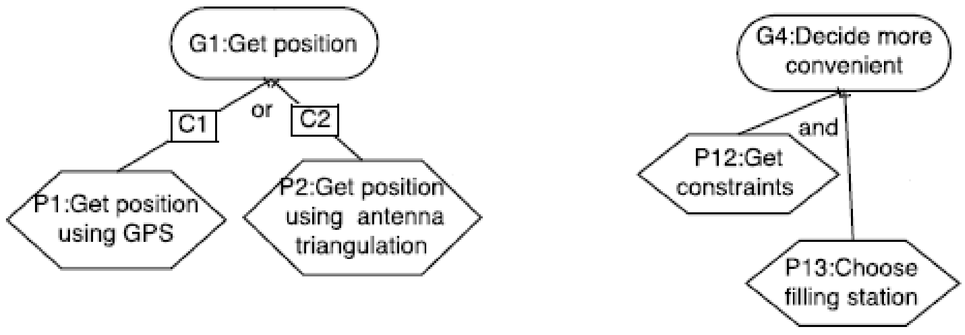

G1:Get position
or
C1
C2
P1:Get position using GPS
P2:Get position using antenna triangulation
G4:Decide more convenient
and
P12:Get constraints
P13:Choose filling station

Figure 2.1: AND/OR-Refinements

can be adopted and when. This allows for systematic derivation of variants based on the several different contexts that may appear [25]. Figure 2.1 describes *C1* and *C2* as contexts of a OR-refinement, as an example of how contexts are depicted in Contextual Goal Models.

2.4 Biological Immune Systems (BIS)

The primary goal of the Biological Immune System (BIS) is to protect the body from potentially harmful material [27]. 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.

In 1986, Farmer et al. [28] 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 be computationally useful [29]. Nevertheless, by that time, the resources available were not powerful enough to allow its usage in real systems [30].

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 [27]. The Negative Selection, along with the

Clonal Selection, Immune Network Theory and the Danger Theory are amongst the most researched approaches in the literature [2].

This section is structured as follows: initially, the main concepts of the Biological Immune System will be laid out. Then, the Negative Selection algorithm will be explained in depth, for it will be exercised throughout this work.

2.4.1 Overview of the Biological Immune System

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.

Main Concepts

Immunity can be defined as the ability to respond to unknown substances [9]. 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 [1].

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 [9]. 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" [9]. 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 [28] 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).

Overview of the Immune Response

Figure 2.2 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 [31]. 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

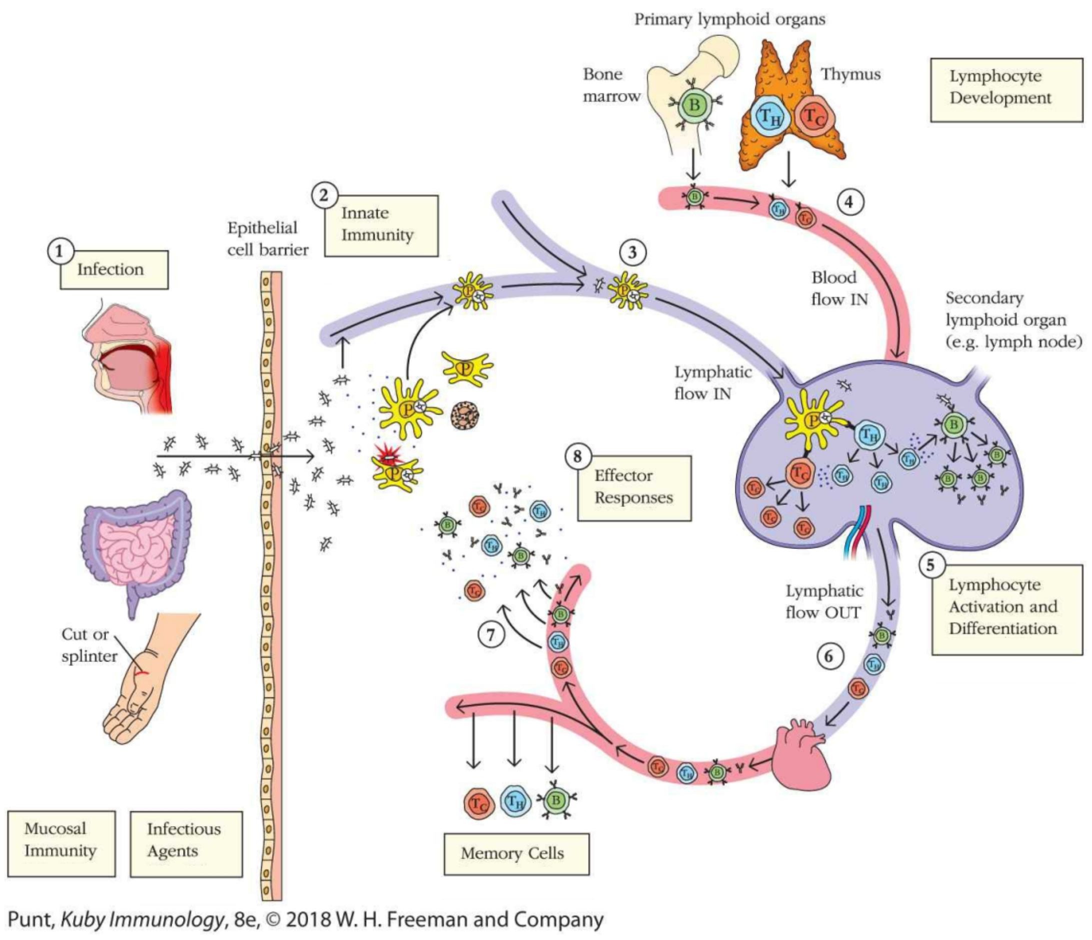


Figure 2.2: Main Processes of an Immune Response [1]

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.

2.5 Negative Selection Algorithm

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 [2] 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 [32]. 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 [2], 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 value.
- **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.

Some discussion has been made over the Data Type and Representation characteristics since they limit all the others [33] [2] [9]. 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. Nevertheless, 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 and the string representation will suffice.

Even though different implementations may fall in different buckets, Ji and Dasgupta [33] 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.

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.

2.5.1 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 [2].

Figure 2.3 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 candiDateTypes is generated by some predefined process and undergo a censoring process based on the self samples. The candiDateTypes 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. The same distance function is utilized here to check whether the new

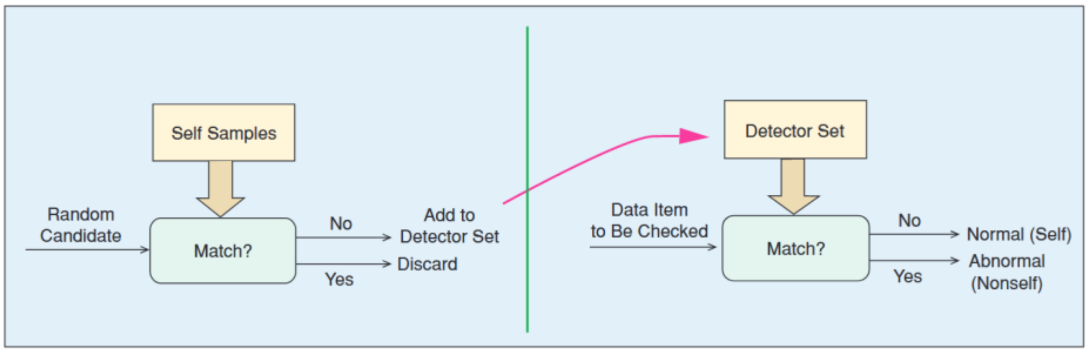


Figure 2.3: The two steps of the NSA [2]

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 candiDateTypes are generated and the definition of the matching function that will account for the affinity measurement.

2.5.2 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 [34]: the Hamming Distance, the R-Contiguous and the R-Chunk matching rules.

- **Hamming Distance:** This function measures the number of characters that differ between two strings [9]. 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 2.4 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

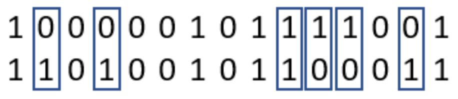


Figure 2.4: Illustration of the Hamming Distance Matching Rule

an IntegerType, 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 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 [34].

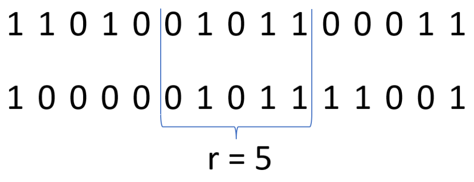


Figure 2.5: Illustration of the R-Contiguous Matching Rule

Figure 2.5 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 [9].

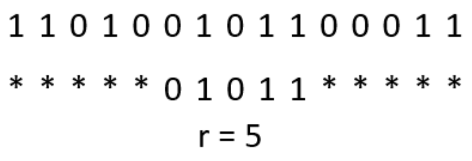


Figure 2.6: Illustration of the R-Chunk Matching Rule

Figure 2.6 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 [34] 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 [35], 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.

2.5.3 Detector Generation

Both Ayara et al. [36] and Dasgupta and Niño [9] 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 [32]. 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 [3].

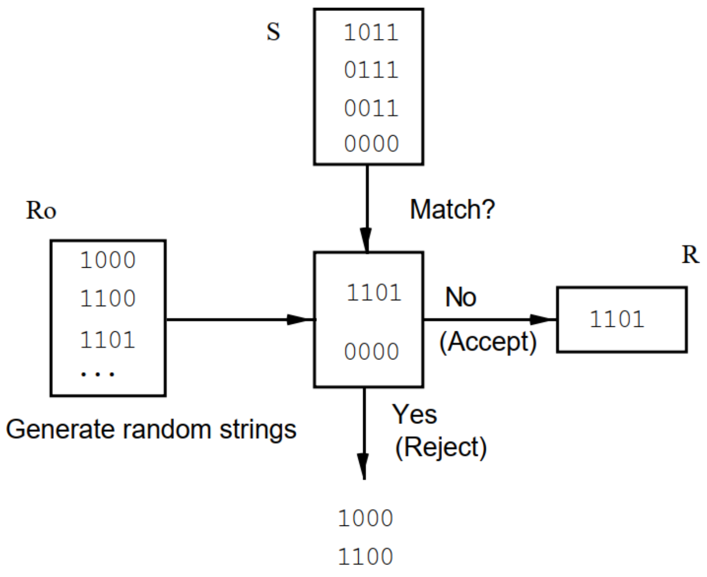


Figure 2.7: Example of the Exhaustive Detector Generation Process [3]

This generate-and-test method can be further explained with Figure 2.7 in which random binary strings are generated and compose the R_o 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 be inneficient [37], besides being costly in terms of computational use of resources [36]. 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. [37]. 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 another algorithm introduced by D'haeseleer et al. [37], 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 candiDateTypes 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 [9]. 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.

Chapter 3

An AIS-based approach to reduce uncertainty in self-adaptive systems

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. The first and foremost is to assist the early stages of the system conception by reducing the uncertainties in context variability. The secondary goal, a byproduct of the first, focuses on increasing the ability to confidently and systematically monitor and analyze CPSs. The methodology combines learning techniques with different assurance provision methods in order to account for contextual uncertainties that could impair the dependability of the CPS.

This chapter is structured as follows: First, we provide an overview of the framework, explaining in high level each step. Next, the Immune System metaphor is described, showing the relations thought to better understand the approach. Finally, each step is further detailed in the subsequent sections.

3.1 Overview

The framework comprises two main steps, as depicted in Figure 3.1. The fist step uses as input the Contextual Goal Model (CGM) specification of the CPS to be developed. Then, a model-checking process takes place to provide formal evidences that the requirements of the system are satisfied. This is done by first modeling the main components of the SAS in a verification tool, like UPPAAL [12]. Then, a set of real-time properties are derived from the specification, so that the correctness and completeness of system may be assured.

After the assurances have been provided through model-checking, we implement a prototype of the system, based on the expected behavior and on the context conditions. Besides that, naive observers are also implemented, derived from a property specification pattern catalog, and serve the purpose of monitoring the system properties during the execution. The prototype is run, and a dataset is extracted, containing the traces of execution, the consumption of resources and the context conditions at each moment of the execution.

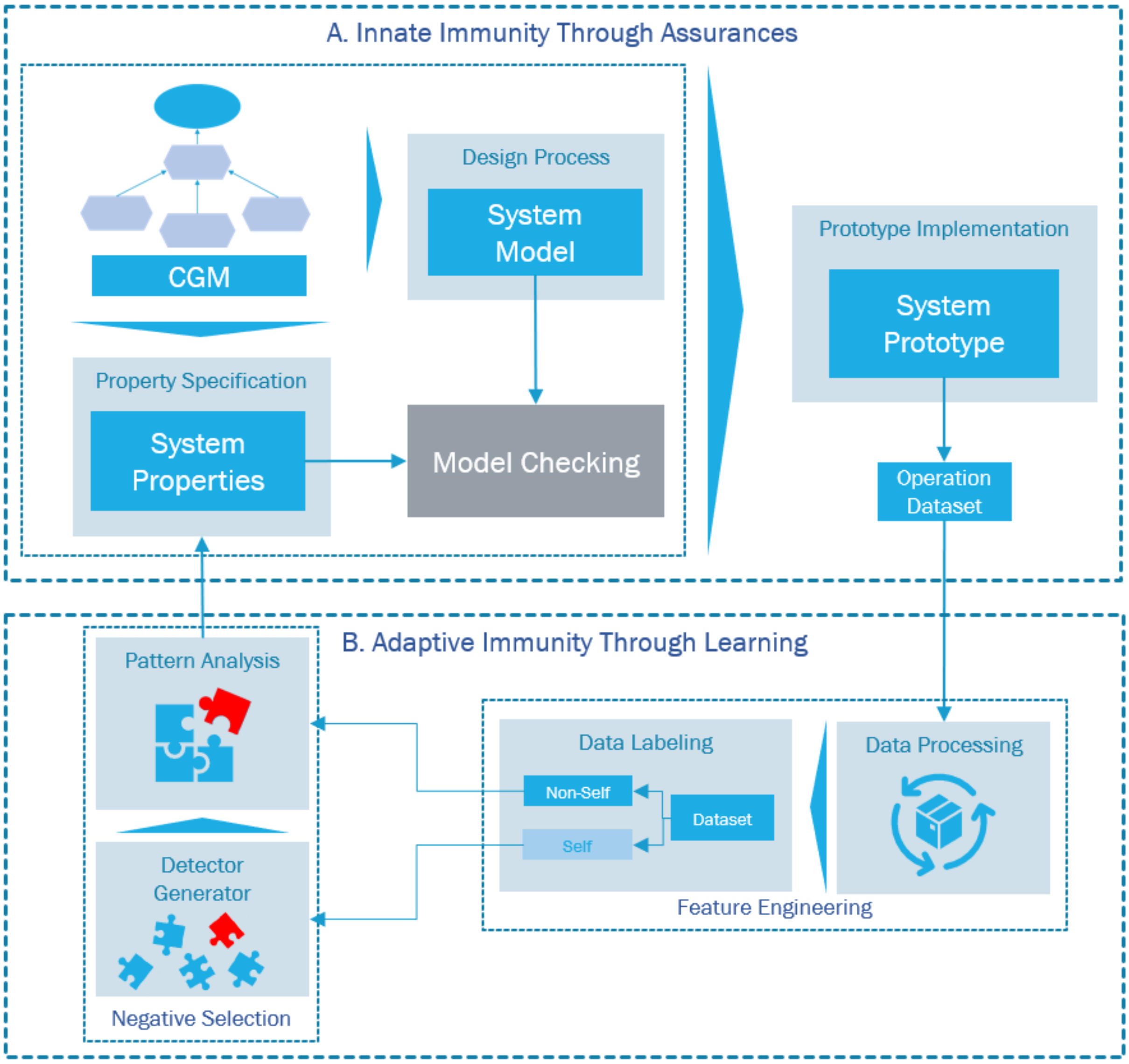


Figure 3.1: Overview of the proposed Methodology

The second step happens right after the first one, and 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 that did not satisfy the particular property being examined. Next, 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 context conditions 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. The enhanced observers trigger a new execution of the first step. The described steps go back and forth until no new relevant information is found.

3.2 Innate Immunity through Assurances

The initial step of our approach has two primary objectives: the provision of assurances and the implementation of a prototype of the system being developed. An earlier project from our research lab, where the goal specification is mapped into models for real-time analysis, served as inspiration for both functions [38].

3.2.1 Model Checking

This activity is related to the first goal. Providing assurances for the SAS happens by initially modeling the core architectural elements, along with their behavior, then by specifying a set of properties of interest, and finally, by conducting a model checking process in a verification tool like UPPAAL [12] to assert the system's correctness and the satisfiability of the properties.

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 system architecture. The automatons 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, we might use the Autili et al. [20] property specification pattern catalog and framework (PSPFramework) 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. 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. In our work, the resultant property set will be written in Timed Computational Tree Logic (TCTL) [39], since that is the language used by UPPAAL when verifying the formalized model's real-time features.

Our work also leverages the use of the Observer Technique as an additional step towards providing evidence that the system behaves as expected. This approach is utilized in the checker to allow the verification of properties that go beyond simple reachability [40]. It 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 [41]. The translation is performed by using an Observer Template catalog [42] 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.

Finally, the generated formulas will be verified for reachability, safety or liveness in the modeled system, by a process called Model Checking. The TCTL model-checking problem is to determine for a particular timed automaton TA and TCTL formula ϕ whether TA $\models \phi$ [18]. In this work, a tool named UPPAAL [12] will 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.

3.2.2 Prototype Implementation

After assuring the correctness of the system through model checking, the second activity of this step is the implementation of a prototype for the system being developed. The major purpose is to replicate the system's behavior in order to generate reliable data for the context analysis process. The prototype operation dataset will be passed on to the next stage of our framework to convey the context discovery potential by means of artificial intelligence while still in the design phase.

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 contextual variability in the environment, just as the innate immunity is the first line of defense against the pathogens [1]. Inherited from the individual's progenitors, the non-specific 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 uncertainty in its surroundings. The observers are enhanced in the next step, by learning from the operation dataset, genereated by the simulation.

Having said that, the prototype is implemented by a simulation tool, based on the designed software architecture. Each module is build in a way to reflect the intended behavior of the system to be. The naive observers are also implemented so that the execution of the simulation may be monitored for the provision of assurances. In our work, we will use the Modelica language and the Open Modelica tool for this purpose.

The Modelica Language is a language used to design cyber-physical systems. It allows for the causal linking of modules that are regulated by mathematical equations [43]. It is a text-based language used to specify all components of a model, and to organize them into packages. In order to graphically edit and explore a Modelica model, as well as run model simulations and other analyses, a suitable Modelica simulation environment is required [44]. The Open Modelica is a tool that fills in this gap by providing an open-source modeling and simulation environment for the Modelica language [45]. It is considered to be the most complete open-source modelica simulation tool [46].

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.

3.3 Adaptive Immunity Through Learning

Our framework's next phase tries to improve context specification using learning approaches. It is performed for each monitorable TCTL property specified in the previous phases. 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 novel context variability, which is 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 2.4.1, specially with the cellular immunity. T lymphocytes are used in this adaptive response, which are created in the bone marrow by a pseudo-random genetic rearrangement process and then developed in the Thymus by being tested against own cells [31]. Similarly, our process generates randomly 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 [1], only the strings that do not match the self-set are used to enhance the observers later on.

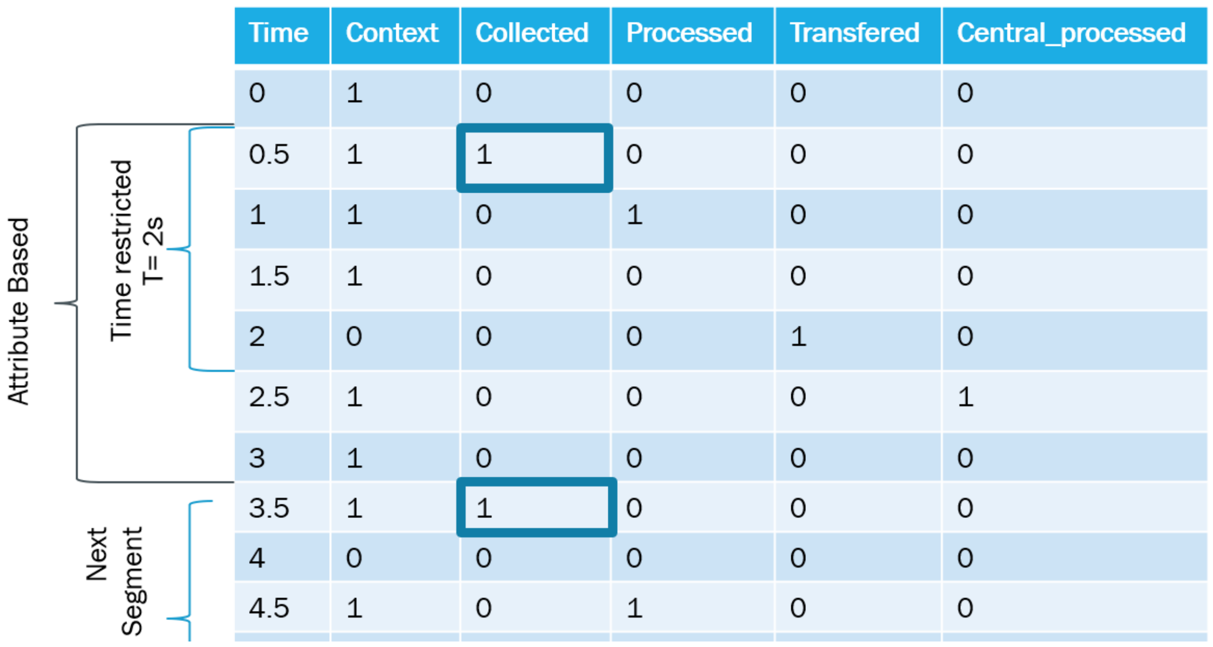
This process can be further divided into two phases: the feature engineering phase, in which the operation dataset is reshaped and labeled, and the learning phase, where the Negative Selection algorithm generates nonself-detectors that are used to enhance the observers.

3.3.1 Feature Engineering

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

The operation dataset is retrieved from Open Modelica 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 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-bound, the execution segment will begin when the collect signal is transmitted and last for the duration of the constraint. Figure 3.2 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.



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

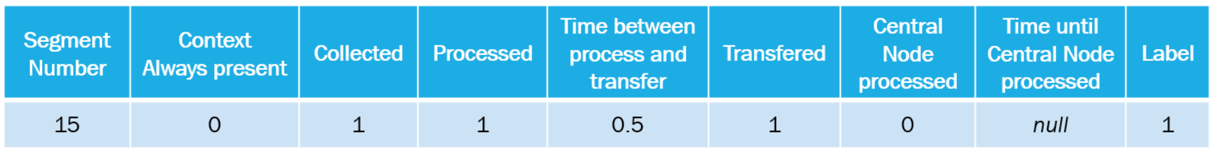
Attribute Based
Time restricted T= 2s
Next Segment

Figure 3.2: Segmentation based on attribute or on time restrictions

Next, new features will be derived to characterize the execution segment, resulting in a single row. 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 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 TCTL property, 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. Both of these details may have been produced 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 TCTL property.



Segment Number	Context Always present	Collected	Processed	Time between process and transfer	Transfered	Central Node processed	Time until Central Node processed	Label
15	0	1	1	0.5	1	0	*null*	1

Figure 3.3: Example of a feature engineered execution segment

Figure 3.3 displays what would be the row associated with the time-restricted segment from Figure 3.2. Since 2 seconds was not enough time for the central node to process the data, it was considered not done whithin the time window of th property. Hence, the column "Central Node processed" is zero, and "Time until Central Node processed" is *null*, and therefore the row is labeled 1, meaning that this segment does violate the property.

3.3.2 Negative Selection Algorithm

This final phase 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 new context variability 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 phase 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 TCTL properties and observers.

i. Exploratory Data Analysis

This initial step focuses on exploring the generated features to verify their relation with the property violation label, besides checking the integrity of the dataset. 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 and verifications were 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 singly and collectively to confirm their behavior. For instance, if data wasn't collected, it couldn't be processed. Thus, cases where the opposite happens are handled. This analysis also 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. The features that are highly correlated, positively or negatively, indicate that the same information is provided and, thus, one of them is 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 2.5, 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 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 2.5.2, 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.

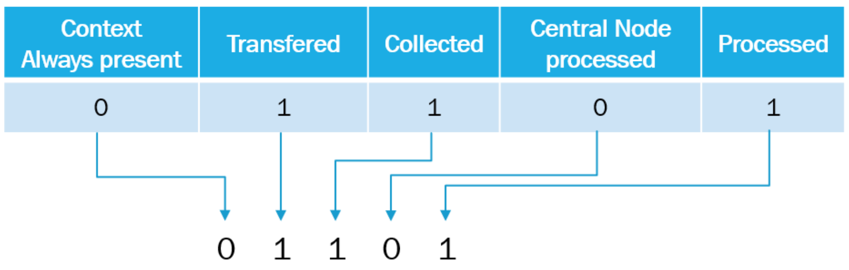


Figure 3.4: Formatting the dataset as binary strings

Figure 3.4 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 2.5, that will be used in this work.

Our implementation of the NSA will meet the three requirements listed by Ji and Dasgupta [33] 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 2.5.3, 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 2.5.2 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.

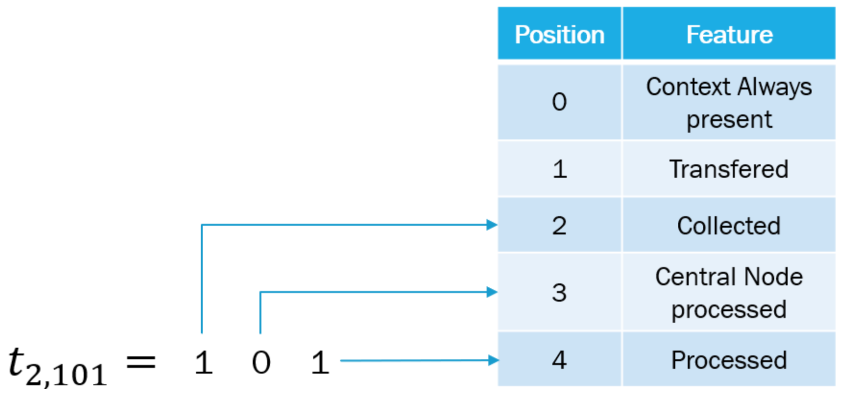
In this regard, we included a third hyperparameter to workaround the cases when the algorithm takes longer to converge. This parameter indicates the acceptable amount of failed attempts to add new candidates to the detectors set. If this value is reached, the algorithm will exit earlier.

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.



$t_{2,101} = 1 \quad 0 \quad 1$

Position	Feature
0	Context Always present
1	Transfered
2	Collected
3	Central Node processed
4	Processed

Figure 3.5: Mapping the Detectors to the features

Figure 3.5 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 context 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.

v. Property Refinement

Finally, after the new context variabiliy have been unveiled, the system's specification and properties are revisited. Initially, these contexts 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, is 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 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 4

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.

4.1 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 [7].

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 [47]. Hence, oversimplified models may be invalidated for not anticipating failures dependant on the two layers [7].

Akella et al. [48], 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 representint 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. [49] 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. [50] 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 [51] 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 [52]. 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 [42].

In the proposed framework, the model checking process will be integrated with simulation. This will be done by implementing a prototype in Modelica [43], 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.

4.2 Fault Analysis and the Negative Selection

Fault detection and diagnosis is yet another way to increase the reliability of CPSs. Zhang et al. [53] suggests a model-based detection, isolation, and system reconfiguration technique 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 [1], has emerged as a viable alternative [56].

Gupta and Dasgupta [2] 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.

Despite the fact that our technique, on the one hand, displays detectors as binary strings, which have well-known shortcomings in the literature, on the other hand, the explainability given by them eliminates the need for prior knowledge of the system's probable faults.

4.3 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 [7], 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 [2] 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 5

Schedule

This research is intended to last for twenty four months, starting in the beginning of 2021 and fininshing in the end of the following year. All the work was divided in twelve tasks that ranges from the formalization of the research approach until the submission of an article and the defense of the thesis. For a better planning, the tasks were clustered into five major blocks to be performed sequentially.

Table 5.1 shows when each task is intended to be performed throughout the available time. The thesis will be written accross the entire period, whilst the literature will be reviewed at least at every three months, so that the work might be always up to date with the state-of-the-art. The last few months will be devoted to writting a paper for the validadion of the community.

Initial effort (months 1-6)

- **T1:** Thesis writing;
- **T2:** Literature review;
- **T3:** Approach formalization (terms, tools, process, methods, quality/assurance)

Development (months 7-12)

- **T1:** Thesis writing;
- **T2:** Literature review (update);
- **T4:** Method implementation;

BSN case study (months 13-15)

- **T1:** Thesis writing;

- **T2:** Literature review (update);
- **T5:** Experimental setup and implementation of the BSN prototype;

Evaluation (months 16-19)

- **T1:** Thesis writing;
- **T2:** Literature review (update);
- **T6:** Negative Selection Algorithm Implementation;
- **T7:** Writing review (feedback from experiments);
- **T8:** Analyze and report results

Publications (months 20-24)

- **T1:** Thesis writing;
- **T8:** Analyze and report results
- **T9:** Article writing;
- **T10:** Article submission (possible venues: TAAS, SEAMS, IST, IEEE Transactions, JSS);
- **T11:** Article revision;
- **T12:** Thesis defense

task / month	**1-6**	**7-8**	**9-12**	**13**	**14-15**	**16**	**17-19**	**20-21**	**22**	**23**	**24**
T1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
T2	✓	✓		✓		✓					
T3	✓										
T4			✓								
T5					✓						
T6							✓				
T7							✓	✓			
T8							✓	✓			
T9									✓		
T10									✓		
T11										✓	
T12											✓

Table 5.1: Work plan for 2021-2022

References

[1] Punt, Jenni, Sharon A. Stranford, Patricia P. Jones, and Judith A. and Owen. W. H. Freeman Macmillan Learning, New York, eighth edition edition, 2019. vii, 13, 15, 26, 27, 37

[2] 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. vii, 13, 16, 17, 18, 37, 39

[3] D'haeseleer, Patrik: *An Immunological Approach to Change Detection: Theoretical Results.* pages 18–26, July 1996, ISBN 0-8186-7522-5. vii, 20

[4] 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

[5] 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. 1, 7, 8

[6] 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`. 1

[7] 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. 2, 35, 38

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

[9] Dasgupta, Dipankar and Luis Fernando Nino: *Immunological Computation: Theory and Application.* Ingeniería e Investigación, 29:140–140, April 2009. 2, 13, 17, 18, 19, 20, 21

[10] 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. 3

[11] 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

[12] 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. 4, 22, 24, 25

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] Baier, Christel and Joost Pieter Katoen: *Principles of Model Checking*, volume 26202649. January 2008, ISBN 978-0-262-02649-9. 7, 8, 9, 25

[19] 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

[20] 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. 9, 24

[21] Horkoff, Jennifer, Fatma Aydemir, Evellin Cardoso, Tong Li, Alejandro Maté, Elda Paja, Mattia Salnitri, Luca Piras, John Mylopoulos, and Paolo Giorgini: *Goal-oriented requirements engineering: an extended systematic mapping study.* Requirements Engineering, 24, June 2019. 10

[22] Van Lamsweerde, Axel: *From system goals to software architecture.* Formal Methods for Software Architectures, pages 25–43, 2003. 10, 11

[23] Lamsweerde, Axel: *Goal-Oriented Requirements Engineering: A Guided Tour.* pages 249–262, February 2001, ISBN 0-7695-1125-2. 10

[24] Elsayed, Iman, Zeinab Ezz, and Eman Nasr: *Goal Modeling Techniques in Requirements Engineering: A Systematic Literature Review.* Journal of Computer Science, 13:430–439, September 2017. 10

[25] Ali, Raian, Fabiano Dalpiaz, and Paolo Giorgini: *A goal-based framework for contextual requirements modeling and analysis.* Requirements Engineering, 15(4):439–458, 2010. 11, 12

[26] Bresciani, Paolo, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos: *Tropos: An agent-oriented software development methodology.* Autonomous Agents and Multi-Agent Systems, 8(3):203–236, 2004. 11

[27] KamalMishra, Prashant and Mamta Bhusry: *Artificial Immune System: State of the Art Approach.* International Journal of Computer Applications, 120:25–32, June 2015. 12

[28] Farmer, J.Doyne, Norman Packard, and Alan Perelson: *The Immune System, Adaptation, and Machine Learning.* Volume 22, October 1986. 12, 14

[29] Garrett, Simon: *How Do We Evaluate Artificial Immune Systems?* Evolutionary computation, 13:145–77, February 2005. 12

[30] Naqvi, Syed Moeen, Merve Astekin, Sehrish Malik, and Leon Moonen: *Adaptive Immunity for Software: Towards Autonomous Self-healing Systems*, January 2021. 12

[31] Aickelin, Uwe, Dipankar Dasgupta, and Feng Gu: *Artificial immune systems*, pages 187–212. January 2014. 14, 27

[32] 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. 16, 20

[33] Ji, Zhou and Dipankar Dasgupta: *Revisiting Negative Selection Algorithms.* Evolutionary computation, 15:223–51, February 2007. 17, 31

[34] González, Fabio, Dipankar Dasgupta, and Jonatan Gomez: *The Effect of Binary Matching Rules in Negative Selection.* Volume 2723, pages 195–206, July 2003. 18, 19, 20

[35] Chmielewski, Andrzej and Slawomir Wierzchon: *Hybrid Negative Selection Approach for Anomaly Detection.* September 2012, ISBN 978-3-642-33259-3. 20

[36] 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. 20, 21

[37] D'haeseleer, Patrik, Stephanie Forrest, and Paul Helman: *An immunological approach to change detection: algorithms, analysis and implications.* pages 110–119, May 1996. 21

[38] 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. 24

[39] 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. 25

[40] 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. 25

[41] 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. 25

[42] Carwehl, Marc, Thomas Vogel, Genaina Rodrigues, and Lars Grunske: *Property Specification Patterns for UPPAAL.* `https://github.com/hub-se/PSP-UPPAAL`, 2022. 25, 36

[43] *Modelica Language.* `https://modelica.org/modelicalanguage.html`. Accessed: 2022-10-27. 26, 36

[44] Fritzson, Peter, Hans Olsson, and Martin Otter: *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification*, September 2007. 26

[45] *Open Modelica.* `https://openmodelica.org/`. Accessed: 2022-10-27. 26

[46] Fritzson, Peter, Adrian Pop, Adeel Asghar, Bernhard Bachmann, Willi Braun, Robert Braun, Lena Rogovchenko-Buffoni, Francesco Casella, Rodrigo Castro, Alejandro Danós, Rüdiger Franke, Mahder Gebremedhin, Bernt Lie, Alachew Mengist, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Wladimir Schamai, Martin Sjölund, and Per Ostlund: *The OpenModelica Integrated Modeling, Simulation and Optimization Environment.* October 2018. 26

[47] 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. 35

[48] 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. 35

[49] 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. 35

[50] 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. 35

[51] 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. 36

[52] 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. 36

[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. 36

[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. 36

[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. 37

[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. 37

[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. 37

[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. 37

[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. 38

[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. 38

[61] Fritzson, Peter: *Introduction to modeling and simulation of technical and physical systems with Modelica.* John Wiley & Sons, 2011. 38