Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

---

Tesi di Laurea Magistrale in Ingegneria Informatica

Analysing a safe communication protocol in the railway signaling domain with Timed Automata and Statistical Model Checking

Author
Irene Rosadi

Supervisors
Prof. Alessandro Fantechi
Prof. Enrico Vicario

Co-supervisor
Ph.D. Davide Basile

Degree Year 2019/2020

Contents

Introduction

This thesis focuses on the modeling and the safety requirements verification of a communication system in the railway signaling domain, where the use of standard interfaces and formal methods is increasing and is also expanding at the industrial level. The Shift2Rail Joint Undertaking initiative, established under the Horizon 2020 R&I European program, identifies both formal methods and standard interfaces as two key concepts, promoting their adoption in the Research and Innovation activities aimed at improving the competitiveness of the European rail system.

The adoption of formal methods and standard interfaces can favor the interoperability between systems provided by different suppliers, thus increasing the market competition while ensuring higher safety standards. To promote the use of these two key aspects in the European signaling area, the Eulynx and the 4SECURail initiatives were conceived with the specific aim of standardizing the interfaces and the elements of the signaling systems, and contributing to the widespread diffusion of the formal methods in the industrial applications, respectively.

In particular, the Eulynx project focuses on the adoption of a model-based approach for the specification of requirements to enrich the traditional documentations expressed in natural language and significantly reduce the risk of ambiguities. At the same time, one of the 4SECURail objectives is the

development of a demonstrator to show how formal methods can contribute in the validation and the verification of standard interfaces.

This thesis takes into account the same railway signaling subsystem identified by the 4SECURail project to exercise the formal methods demonstrator, providing a formal analysis of the system starting from its safety requirements. The selected case study concerns the RBC/RBC handover protocol specified in natural language through a public standardized interface.

One of the aims of this work is to contribute to the 4SECURail evaluation of the cost-benefit analysis for the adoption of formal methods by providing some useful data on the efforts made to reach the results we are about to describe in this thesis. For the formal modeling and verification of the system, we chose to use Stochastic Timed Automata and Statistical Model Checking, and in particular the UPPAAL SMC toolbox has been used.

The resulting artifacts can provide a further contribution for the definition of unambiguous system requirements, enriching the documentation with consistent and formally verified model-based specifications as promoted by the Eulynx project.

A brief summary of the content of each chapter is shown below:

- In the first chapter, we present more in detail the Shift2Rail, the Eulynx and the 4SECURail initiatives, describing their main objectives related to the adoption of formal methods and standard interfaces;
- In the second chapter, an overview on both the Model Checking and the Statistical Model Checking is given. In addition, we show the main aspects of the UPPAAL SMC toolbox, focusing on its modeling and query languages and thus introducing the Timed Automata formalism;

- The first part of the third chapter is dedicated to the description of the case study and in particular the Safe Application Intermediate (SAI) sub-layer, that is the portion of the RBC/RBC handover interface we implemented. In the second part of the chapter, we show in detail the formal model we developed through the UPPAAL SMC toolbox;
- In the fourth chapter, we present the verification and analysis results of the system. The safety properties we verified through the statistical model checking are described together with their evaluation results. Moreover, the contribution of this thesis to the 4SECURail cost-benefit analysis on the use of formal methods is provided.

Chapter 1

Standard interfaces and formal methods in railway domain

Standard interfaces and formal methods have been identified as a promising match in the railway area with the precise aim of enhancing the overall railway market system. Widespread use of these two aspects can lead to significant innovations, which are essential to achieve the increasing safety and efficiency requirements of railway networks, as many real projects have already shown. Two relevant examples are the model-based development and testing of the on-board equipment of the Metrô Rio Automatic Train Protection system [1] and the automatic train operating system for METEOR, driverless metro in Paris, whose safety-critical software was developed using B formal method [2], but plenty of other projects in railway signaling area have successfully included applications of formal methods to standard interface validation, in last decades [3], [4].

Significant evidence of the importance of adopting these solutions in the rail sector is provided by the European rail network, which has particularly

suffered from fragmentation and inefficiencies, due to the patchwork of disparate regional and national systems, networks and technical operating standards that characterized it. Major investments in Research and Innovation (R&I) at EU level have been made and are still aimed at achieving a more competitive and resource-efficient European transport system, rationalizing research programs and ensuring the interoperability of the developed systems to promote competition [5].

1.1 Shift2Rail initiative

To help to address such challenges and to strengthen Europe's competitive position, an important strategic initiative, Shift2Rail Joint Undertaking (S2R JU), was established in 2014 under Horizon 2020 R&I program, to promote an EU coordinated approach to research and innovation in the rail sector.

S2R JU is a European public-private partnership in the rail sector, whose main target is pursuing research and innovation activities and market-driven solutions to accelerate the integration of new and advanced technologies into innovative rail product solutions, in support of the achievement of SERA (Single European Railway Area) to improve the attractiveness and competitiveness of the European rail system. The stated goal of this initiative is to double the capacity of the European rail system and to increase reliability and service quality by 50% while halving life-cycle costs [6].

Focusing on railway signaling systems, two key concepts S2R JU has identified are formal methods and standard interfaces. These concepts are useful to reduce both time and high costs for procurement, development, delivery and maintenance of signaling equipment. While the purpose of the former is to ensure correct behavior, interoperability and safety, the latter

is needed to increase market competition and standardization, thus aiding the process of reduction of long-term life cycle costs and vendors locked-in solutions.

Furthermore, S2R JU aims to demonstrate all the benefits that the adoption of formal methods and standard interfaces can bring in both technical and commercial directions, applied to selected applications. The expected result concerns the expansion of the industrial diffusion of these two key aspects, which can consequently favor interoperability, introducing new market perspectives and export opportunities [7].

1.2 Eulynx project and interface standardization

Standards can be described as the fundamental building blocks for both product development and manufacturing. They establish uniform specifications, procedures and requirements, related to a specific system, to maximize the reliability of methods, processes and final products. When instead there is a lack of standardization, interoperability, reliability and quality are inevitably compromised.

In European railway signaling area, the absence of standard interfaces results in huge costs to make the different equipment from different suppliers compatible with each other. Furthermore, old assets and architectures need to be renewed, taking advantage of the new technologies made available by Information Technology (IT) industry.

For these reasons, a proposal for joint efforts to overcome the inefficiency and the problems of EU rail sector fragmentation was made in 2014, when a

European cooperative initiative, Eulynx, grouped together different railway Infrastructure Managers (IMs) from 6 European countries. Eulynx project now includes up to 13 IMs, and its main purpose is the standardization of interfaces and elements of the signaling systems, to significantly reduce their lifecycle costs and installation time [8].

The importance of Eulynx corporation is based on sharing not only methods, processes or requirements but also technological innovations and advances, with the aim of making standard freely available to third parties to stimulate competition among suppliers.

Eulynx's process of standardization focuses on the creation of interfaces between field equipment and interlocking, the core of signaling system architecture. Interlocking is a set of signal apparatus placed on the track in order to prevent conflicting movements among trains or train derailment through an arrangement of track devices, such as points, signals and level crossing, ensuring train safety.

Thanks to Eulynx standard interfaces, it is possible to reorganize the interlocking system as a modular system, which implies being able to have different parts supplied separately and independently. This approach is particularly effective due to the different life expectations of IT-based control logic with respect to field elements: the electronic interlocking equipment has a shorter lifetime and needs more frequent replacements than long-lasting investments in field mechanical technology. Standardized interfaces enable disconnecting the lifecycles of interlocking components from those of field elements, offering the possibility of replacing single components, instead of the full assets.

This scenario offers market access also for specialized manufacturers to produce single, compatible modules, overcoming the concept of monolithic

systems based on proprietary interfaces, and significantly reducing costs for production and maintainability, while maintaining the required high levels of safety and reliability. As a consequence, the risk of dependency on a limited number of suppliers, when upgrading or renewing products, disappears, while the competition for the same product among various producers increases, lowering production costs [9], [10].

Furthermore, Eulynx provides stable basis for the digitalization of the railway signaling system, promoting herself to achieve the complete definition and standardization of interfaces for the new digital interlocking architecture, supporting in the meantime migrations from different existing configurations. Indeed, the digitization of Command, Control and Signalling (CCS) can only be successful if it is carried out hand in hand with the process of standardization, and a key element to reach this goal is represented by the use of Model-Based System Engineering (MBSE) [8].

MBSE is the practice of defining the functional behavior of the interfaces through unambiguously semi-formal, executable models, which provide an efficient way to explore, update, and communicate system aspects to stakeholders, while significantly reducing dependences on traditional documents. The use of models to specify software requirements and system design results in quality improvements and system modeling artifacts reuse, reducing the risks of ambiguity and the possibility of misinterpretation, with respect to document-based approaches. This is particularly important because nowadays classical systems are evolving to complex Systems-of-Systems (SoS), thus involving a significant component of interoperability, which derive in further challenging requirements, like flexibility, adaptability or expandability. Moreover, MBSE introduces model-based testing: by early testing and

validating system characteristics, the use of models facilitates timely learning of properties and behaviors, enabling fast feedback on requirements and design decisions. Simulation of complex system and SoS interactions are another opportunity to accelerate learning, together with an appropriate fidelity [11], [12].

Eulynx uses a MBSE approach with the OMG (Object Management Group) standard modeling language UML/SysML (Unified Modeling Language/System Modeling Language) for the requirement definition, thus improving interface usability and significantly reducing development problems related to uncertainties, like missing requirements and not yet detected ambiguities or contradictions. UML/SysML is a standardized notation to support the modeling of complex systems, allowing a standardized description of their specification through different kinds of diagrams, like state machines, activity and sequence diagrams, each representing a different view of the system. Thanks to UML/SysML adoption as semi-formal model-driven design methodology, Eulynx can build more robust specifications, easier to read and understand by system stakeholders.

However, despite having a fully defined syntax, semi-formal methods suffer from the lack of precise semantics, whose definition can be incomplete, thus maintaining a certain degree of generality in the specification of the execution model. UML/SysML supports this variety of different execution paradigms by leaving some key semantic elements unconstrained and by defining explicit semantic variation points. This defers the completion of a specific definition of semantics to either human interpretation or interpretation through software tools. Both of these possibilities are affected by ambiguity: while human interpretation is linked to the drafting of natural

language documentation for the description of the semantic behavior, software execution tools realize specific semantics "by suitably constraining the unconstrained semantic aspect and providing specifications for any desired variation at semantic variation points", and "different execution tools may semantically vary in these areas (not explicitly constrained) in executing the same model" [13].

This characteristic of semi-formal methods makes them easier to use and to understand, but at the same time makes it difficult to consistently verify and test system requirements. To limit ambiguity as much as possible and to achieve consistency and better confidence in the correctness of the system, a mathematical formalism that offers fully defined syntax and semantics is needed, and formal methods can do their part in this direction.

It is interesting to notice that also OMG's identified requirements for SysML v2 next-generation modeling language focus on including precise and well-specified semantics, to avoid ambiguity and to permit the consistency validation of the model from a logical perspective [14].

1.3 Formal methods and 4SECURail project

Today, formal methods are widely applied to the design and the development process of critical components for safety-critical systems, becoming one of the main technologies used also within railway industries.

> *"Formal Methods refer to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems."* [15]

In this NASA definition of formal methods, *"mathematically rigorous techniques"* refers to the fact that formal specifications are well-formed state-

ments in mathematical logic, and formal verifications are rigorous deductions in that logic, meaning that various steps follow from rules of inference and can be checked by a mechanical process. Formal specification and formal verification can be described as the two domains of formal methods.

Formal specification concerns the definition of the system requirements, that is both the functionalities required for the system and the specific behaviors that the system must have, using notations derived from formal logic. The resulting specifications may be processed automatically by software tools, allowing the debugging of the requirements during the development phase and can be used as detailed system documentation, too. The purpose of formal specification is to prevent ambiguity and to avoid different interpretations that can bring to costly and time-wasting incompatibility problematics during the system integration phase. This clearly gives a better understanding of the system, compared to requirements specified through natural language, which is affected by numerous and well-known issues, as described in [16].

Formal verification concerns the analysis of a specification for consistency and completeness feedback, that is requirements satisfaction must be proven from assumptions, using proof methods from formal logic. It provides means to symbolically examine the entire state space of the system design and establish a correct property that is true for all possible inputs. However, such a complete verification is rarely done in practice, due to the high execution times, so more often formal verification is applied to safety-critical parts of a system, a strategy that can bring an important increase of confidence in the system.

Using formal methods in the design process of a system allows description and reasoning about the behavior of that system in a formal manner, but

there are some limitations not to be underestimated. Formal methods cannot provide an absolute guarantee of correctness, because they can only be used to prove system correctness with respect to its specification. There is no guarantee that the specification itself is correct and fault free, and also that the produced model is accurate enough. However, the above limitations can turn into a strength, providing a significant possibility of discovering requirement inaccuracies and modeling errors, and thus permits their correction and removal [17].

Focusing on the railway domain, despite the large number of successful applications of formal methods, as previously described, no universally accepted technology has emerged. Even if applicable standards (e.g. CENELEC EN 50128 for the development of software for railway control and protection systems) mention formal methods as highly recommended practices [18], they do not provide clear guidelines on how to use them in a cost-effective way. The absence of a clear idea of which benefits can result from the adoption of formal methods, both from a technical and economical point of view, and the variety of tools, which makes difficult to select the most appropriate formal methods to use in the development of a system, are some of the aspects that act as an obstacle to the widespread use of formal methods [19].

Nevertheless, signaling area is the one, within railway domain, where formal methods have been mostly used. This success in applications related to railway signaling is explained by the absence of complex computations and hard real-time constraints, characteristic that makes it easier to generate a formal specification and then verify it. Moreover, railway signaling sector is a safety-critical area, thus implying the necessity of formal proof and verification of dependability properties [20].

Furthermore, these aspects determine a major diffusion of formal methods compared to semi-formal methods in the area of railway signaling, regardless of the increase in complexity due to the fully defined mathematical formalism of the former. This complexity, combined with a lower understandability of the model, is another obstacle to the widespread diffusion in industrial applications of formal methods, which require a preliminary training period to achieve specialized mathematical knowledge, thus increasing development costs. As a consequence, formal methods are mostly associated with academic applications rather than industrial ones.

In this scenario, 4SECURail (FORmal Methods and CSIRT for the RAILway sector), a Shift2Rail initiative launched in 2020 to address the Open Call S2R-OC-IP2-2019, takes a two-fold perspective: a Demonstrator for the use of Formal Methods (FM) and the implementation of Computer Security Incident Response Team (CSIRT) for Railways [21].

The overall objectives of the Workstream1 “Demonstrator Development for the use of Formal Methods in Railway Environment" are:

- The development of the demonstrator consisting of the process to be followed to provide a formal validated model of a smart signaling system, and of a list of the most suitable tools to support such process;
- The identification of a railway signaling subsystem, described by means of standard interfaces, to be used as test case to exercise the formal methods demonstrator;
- The specification and evaluation of the cost/benefit ratio and learning curves for adopting the demonstrator in the railway environment.

In particular, 4SECURail focuses on clarifying the utility of the objectives from the point of view of the Infrastructure Managers because they are

expected to benefit most from the adoption of formal methods and standard interfaces. Indeed, one of IMs responsibilities is the definition of a rigorous and validated specification of the system, with the purpose of providing it to multiple different developers that should produce equivalent products.

Focusing on particular domains like railways, due to the complexity of railway infrastructure, the above scenario results in a multitude of subsystems that must correctly interact among themselves. These subsystems can also be developed by different suppliers, thus adding to IMs the responsibility of building rigorous specifications that extend to the verification of the interactions between various components. Moreover, while some properties can be verified by reasoning at the level of single subsystems, specific properties can be related to the composite behavior of several subsystems, whose verification is responsibility of the IMs. For this reason, IMs should provide high-quality specifications for suppliers to work with, allowing a deep understanding of the model, and this is exactly the goal of the aforementioned IM initiative Eulynx [22].

The demonstrator developed by 4SECURail aims to show how formal methods can help to solve this model validation and verification problem, enforcing the desired system behaviors at specification level to support IMs work, while providing the basis for the study of the cost-benefit analysis of the approach and the evaluation of the learning curve for the use of selected methodologies and tools.

This thesis work comes within the framework of 4SECURail project, taking into account the same case study proposed by the initiative for the evaluation of the formal methods demonstrator. The selected case study, UNISIG SUBSET 098 - RBC/RBC Safe Communication Interface, concerns

the RBC/RBC handover protocol, which is needed to manage the interchange of train control supervision between two neighbouring RBCs.

To address the main objectives of Workstream1 inside the 4SECURail project, this thesis work provides a formal analysis of the safety requirements allocated in the Safe Application Intermediate (SAI) sub-layer, concerning the protection against specific communication errors during RBCs message exchange. Despite the adoption of UML/SysML methodology, in continuity with the EULYNX project, 4SECURail admits the possibility of translation of the UML state machine structure into other formal notations supported by tools with stronger verification capabilities, such as Event-B, LNT or UPPAAL, to overcome the problems of uncertainties of the semantics [7].

For this reason, this thesis work exploits directly formal methods to build a fully defined mathematical model and verify it with respect to its safety requirements. Finally, this thesis has the opportunity to contribute to the ongoing 4SECURail studies about learning curve and cost-benefit analysis of formal methods adoption, by providing data about training and formal modeling efforts.

Chapter 2

Statistical Model Checking and UPPAAL SMC

In this chapter, Statistical Model Checking (SMC) is discussed together with the UPPAAL SMC toolbox and its main features, with the purpose of giving a general overview of the techniques and tools used for the development of this thesis work.

The first section briefly analyses what Model Checking is, describing the characteristics, strengths and drawbacks of the well-known method for automatic model verification. In the second section, Statistical Model Checking is presented, focusing in particular on the UPPAAL SMC toolbox, which uses SMC approach to evaluate probabilistic properties of interest on real-timed stochastic systems.

2.1 Model checking: an overview

Model checking (MC) is a technique for performing a fully automatic formal verification on systems, with the purpose of validating the expected behavior and ensuring its correctness.

Given an interpretation structure M, the Model Checking procedure decides whether M is a model of a logical formula ϕ, i.e. whether M satisfies ϕ, which can be expressed as $M \models \phi$. This rough definition suggests the three main steps of MC workflow, which are: modeling, specification and verification [23], [24].

In the modeling step, the behavior of the system is abstracted through an interpretation structure M, which is a finite-state model with nodes and arcs, where the former represent the different states of the system and the latter all the possible transitions between them.

It is important to understand at which level of abstraction the truth or the falsity of the properties can be assessed, to build a model that adequately represents the behavior of the system, capturing those aspects that affect the correctness of the properties, while abstracting away those details that are irrelevant for the verification of the desired properties.

Two main structures used for modeling are Kripke Structures (KS) and Labeled Transition Systems (LTS), and both of them extend classical graphs with specific labels: in KS, nodes are annotated with atomic propositions to describe how states are modified by the transitions, while in LTS arcs are annotated with actions to describe the transitions which cause a state change. These annotations allow to add more information to the model.

The specification step regards system requirements and their translation in a logical formalism: a logical formula ϕ is a property derived from requirements that the system has to satisfy. A common logical formalism used to express system properties is temporal logic, which enables reasoning about system behavior over time, permitting to focus on sequences of events. The properties that are usually expressed on a model concern safety properties, which state that "nothing bad ever happens", and liveness properties, which instead state that "something good eventually happens".

Temporal logic is characterized by different interpretations depending on how the system is considered to change with time, but the main classification is between linear-time and branching-time logic.

Linear Temporal Logic (LTL) allows to describe execution properties of the system as sequences of states: at each moment of time, there is only one possible successor state and thus only one possible future.

Branching Temporal Logics, like CTL (Computation Tree Logic), describe properties that depend on the branching structure of the model: at each moment, there may be several different possible futures, because each state can have different possible successor states. Thus, the semantics of a branching temporal logic can be associated with a tree of states rather than a sequence, and each path in the tree represents a single possible computation. The tree itself represents all possible computations, starting from a root state.

The choice of using a linear-time or a branching-time logic depends on the properties of the system to be analysed: branching-time logics are often used to analyse reactive systems, instead linear-time logics are preferred when only path properties are considered.

Deciding whether M satisfies ϕ means verifying the correctness of the model relative to a specification. This verification step is performed through

Model Checking algorithms, thus it is completely automatic, and the result can be positive (M satisfies ϕ, i.e. $M \models \phi$) or negative (M does not satisfy ϕ, i.e. $M \not\models \phi$) and in this last case, a counterexample is provided. This error trace is particularly important because it details the reasons why the model does not satisfy the specification, leading to the failure of the checked property. Studying the counterexample, it is possible to go back to the point where the model behavior resulted in property failure, fix the problem and then verify the specification again. Moreover, this counterexample analysis allows to verify whether the error concerned incorrect modeling of the system or an incorrect specification: in the first case, the model is modified to meet the desired property, while in the other case the revision addresses the formalization of the requirements.

Regardless of the error type, the unsatisfiability of a property forces error removal, therefore bringing an increase of confidence in the correctness of the model. When the model satisfies all the properties and no errors are found, the model can eventually be refined to include additional aspects and new functionalities that were not considered in the first abstraction of the system, and then restart the verification step.

In Figure 2.1, a schematic summary of this methodology of Model Checking is shown. This methodology can be applied a posteriori when analysing the behavior of a system with respect to its requirements: MC checks whether a fine-state model, derived from the existing system, meets the given specification, verifying its correctness.

Another Model Checking methodology, instead, concerns the formalization of both the model and the properties it must verify, starting from a common specification of the system requirement. In this approach, the purpose of MC is verifying the correctness of the specification, checking through

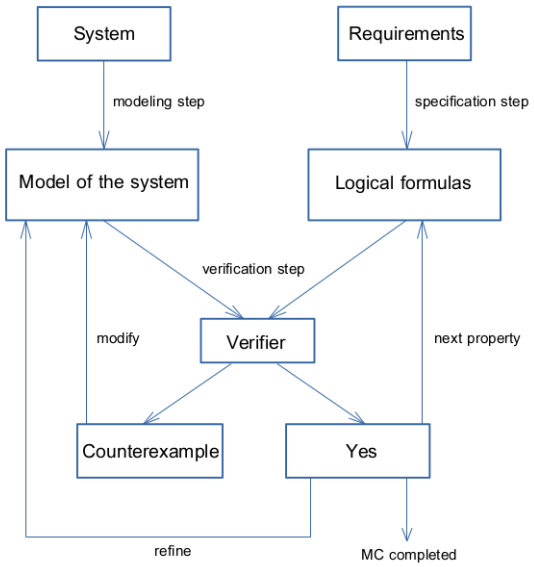

Figure 2.1: A possible workflow of Model Checking

modeling and verification steps the presence of ambiguities or inaccuracies as soon as possible, thus avoiding the scenario where the error occurrences are revealed after the system completion.

The resulting workflow starts from the system requirement specification from which both the model and the properties are derived, and through model verification, the behavior is checked with respect to the requirements, long before system development. This leads to possible revisions of the specification both during modeling and verification steps, contributing to the improvement of the completeness and clarity, as well as the correctness, of the specification itself.

Hence, the Model Checking methodology can vary depending on the proposed scenario, and a common or a different documentation can alter the workflow of modeling and specification steps; in any case, the final result is a definition of the model of the system and some logical formulas of properties

that the system must satisfy.

As shown in the next chapter, this thesis work follows the specification level approach, verifying the system requirements with respect to the model built starting from its specification.

Focusing on Model Checking algorithms, there exist different approaches depending on the formalism used to model the system and the temporal logics adopted to specify the system requirements. They are typically based on an exhaustive state-space search of the model of the system: for each state of the model, it is checked if the state satisfies the desired property or not.

However, this verification task can fail to terminate because the number of possible states of the model may exceed the amount of computer memory available. This trend for the number of states in a model to grow exponentially, as the size of the system being represented is increased linearly, is known as the state-space explosion problem.

To cope with this state-space explosion problem, some approaches have been proposed as an alternative to Model Checking algorithms and their explicit representation of the system. Some examples are symbolic Model Checking algorithms, Bounded Model Checking and On the fly Model Checking.

In symbolic Model Checking algorithms, each state of the system is encoded as an assignment of Boolean values, i.e. 0 or 1 values, to the set of state variables associated with the system. This means that each state can implicitly be represented as a Boolean formula over the set of state variables. Also transition relations are represented as Boolean formulas over two sets of variables, where the first set encodes pre-states and the second set encodes post-states of the transitions, and finally, the properties of the system

(expressed in logical formulas) can be represented as Boolean formulas too, associating each formula with the set of states that validate it. Through particular data structures called Ordered Binary Decision Diagrams (OBDDs), Boolean formulas can be easily represented, thus permitting an extremely compact representation of the system even for large models. Model Checking is then performed manipulating the OBDD representation of the system.

Bounded Model Checking (BMC) is a technique that allows the reduction of the state-space in depth, performing the search for a counterexample, given a boolean formula to evaluate, while the length of the execution is bounded by some integer k. If no error is found then the value of k is incremented and the search is repeated again, until either a counterexample is found, thus proving the unsatisfiability of the property, or a threshold for k is reached, thus stopping the execution and leaving the verification of the property incomplete.
Clearly, BMC aims to find a counterexample rather than verify the total correctness of the logic formula, to considerably reduce the complexity of Model Checking. For this reason, BMC problems are usually reduced to propositional satisfiability problems concerning the negation of the formula to be verified on the system, and are solved by SAT methods.

On the fly Model Checking algorithms are characterized by a top-down approach that does not need global knowledge of the complete state-space of the model, unlike traditional Model Checking algorithms which require the complete generation of the state-space to realize the exhaustive search needed to verify the correctness of a logical formula, following a bottom-up approach. Instead, On the fly algorithms generate the state-space following a stepwise procedure, starting from a given state and following the transitions

to adjacent states, while keeping track of all the paths that are being generated. For each of them, the information about the satisfaction of the formula that is checked is updated. This approach permits to generate only those parts of the state-space that may provide information on the satisfaction of the formula and irrelevant parts are not taken into consideration, mitigating the problem of state-space explosion.

2.2 Statistical Model Checking

As previously described, MC algorithms require a lot of memory space and time, when dealing with particularly complex models, and both characteristics make those algorithms scale poorly to large systems. Furthermore, the aforementioned approaches rely on deterministic state-transition graphs and temporal logics that only formalize the relative ordering of states and events.

These formalisms fail to satisfactorily model and verify systems that require stochastic behaviors and assertions about quantitative aspects of time, because they disregard the continuous dynamics of the physical environment. Indeed, real-time aspects are increasingly taken into consideration due to the development of systems where time correctness plays a central role, and also stochastic behaviors, such as time-delays or random error occurrences, are essential for modeling and verifying complex systems.

Real-timed stochastic systems include both characteristics, achieving randomness with state-transitions that can be equipped with probability distributions.

A numerical approach to verify systems with stochastic behaviors is probabilistic Model Checking: algorithms based on this technique compute the

exact probability for the system to satisfy the property of interest and then compare it with the value of a probability threshold, resulting in high accuracy. The drawback of these algorithms is that they compute the probability for all the executions of the system, resulting in intensive memory usage, which makes them not scalable to large-size models.

Alternatively, Statistical Model Checking is a simulation-based approach to probabilistic Model Checking: it combines simulation and statistical methods to analyse stochastic systems and estimate satisfaction probabilities of their properties.

In SMC algorithms, the verification of quantitative properties of stochastic systems over time is achieved by simulating the system for finitely many runs, and using hypothesis testing to infer whether the samples provide statistical evidence for the satisfaction or violation of the specification, with a predefined level of statistical confidence. Since sample runs of a stochastic system are drawn according to the distribution defined by the system, they can be used to get estimates of the probability measure on executions [25].

Given a stochastic system S and a property ϕ, SMC can be used to answer three types of question:

- **Quantitative check**

  Which is the probability that S satisfies ϕ?

- **Qualitative check**

  Is the probability that S satisfies ϕ greater or equal to a certain threshold $p \in [0, 1]$?

- **Comparison**

  Is the probability that S satisfies ϕ_1 greater or equal to the probability that S satisfies ϕ_2, where ϕ_1 and ϕ_2 are two different properties?

SMC can also support the analysis of rare events, i.e. events having very low probability of occurrence, and hence that are very unlikely to be observed with pure simulation. Indeed, running pure simulations may involve little or no runs at all showing rare events. Thanks to probability distribution handling, it is possible to artificially inflate the probability of occurrence of rare events, making them more likely to happen and thus to be observed in some runs. This can be crucial because rare events can still reveal system errors that can cause the occurrences of hazards, and that have therefore to be fixed.

The main advantage that SMC offers with respect to an exhaustive probabilistic Model Checking approach concerns the applicability of the method to larger and more complex systems. SMC can scale better because it does not need to generate and then explore the full state-space of the model, thus avoiding the aforementioned state-space explosion problem, and the required simulations can be distributed over parallel runs, being independent of each other.

However, SMC has also some disadvantages: contrary to traditional and to probabilistic Model Checking, it only provides probabilistic guarantees about the correctness of its answer, which is the drawback of avoiding the exhaustive state-space exploration. Moreover, depending on the setting of accuracy and precision parameters of the simulations, sample size can grow very large when the model checker's answer is required to be highly accurate, resulting in more time-consuming simulations.

Therefore, to achieve the desired results in the verification step using SMC, the right trade-off between analysis speed and precision, which depend on statistical parameter settings, must be found.

2.3 UPPAAL SMC toolbox

UPPAAL SMC is a tool that implements SMC techniques to perform automatic verification of real-time stochastic systems. It is the stochastic extension of UPPAAL, a toolbox for real-time system verification jointly developed by Uppsala University and Aalborg University and first released in 1995. UPPAAL toolbox is designed to verify real-time systems that can be modeled as networks of timed automata extended with integer variables, structured data types, user-defined functions and channel synchronization. It has been applied successfully in several case studies, and this large application determines its constant development.

UPPAAL performs symbolic Model Checking on real-time systems, where clock values are abstracted into regions and the decidability is based on the notion of region equivalence over clock assignments: states whose clock assignments are within the same region satisfy the same logical properties. Each region (also called zone) is defined through clock constraints and, in UPPAAL, zones are represented symbolically as Difference Bound Matrices (DBMs), which are matrix-form representations of clock constraints [26].

UPPAAL SMC extends the UPPAAL toolbox allowing the representation of systems via networks of timed automata, where stochastic behavior can be modeled with probabilistic choices and time delays with probability distributions. Furthermore, Model Checking is enhanced with new queries related to the stochastic interpretation of timed automata.

2.3.1 Modeling language

UPPAAL modeling language implements Timed Automata theory to perform Model Checking verification [27].

A Timed Automaton is a finite-state machine extended with clock variables, and a system in UPPAAL is modeled as a network of Timed Automata in parallel. A *state* (also called location) may include bounded discrete variables that can be read and written, and common arithmetic operations can easily be applied. When referring to a *state of the system*, the locations of all automata, the clock values and the values of the discrete variables take part in its definition. A clock value is obtained thanks to a clock valuation function that returns a non-negative real value for a given input clock.

Timed Automata are often composed into a network of automata over a common set of clocks and actions. Timed Automata of a network may communicate between each other via broadcast channels and shared variables, and clock values of all Timed Automata increase with the same speed. Clock values can be compared to integers, allowing to form guards that can enable or disable state transitions, affecting the behaviors of the Timed Automaton.

UPPAAL modeling language extends Timed Automata with a set of parameters of different types (e.g. int, double, bool), constants, bounded integer variables, arrays, record and custom types.

Moreover, it allows synchronization between two or more Timed Automata through the use of channels. A binary synchronization channel is declared as `chan c` and the firing of the edge labeled with `c!` synchronizes with another labeled `c?` (if several edges are labeled `c?`, the choice for synchronization occurs non deterministically). Broadcast channels are declared as `broadcast chan c` and the sender `c!` synchronizes with all enabled re-

ceivers `c?`, or executes `c!` action without synchronization if no receivers are enabled (broadcast synchronization is a non-blocking action).

States in UPPAAL can be *urgent* or *committed*. *Urgent* states are states where time is not allowed to pass: if any of the Timed Automata is in an urgent state, time cannot progress and delay transitions cannot be taken unless all urgent states are left. *Committed* states are urgent states where the next transition must involve an outgoing edge of at least one of the committed locations of the network: if a committed state is entered and it is the only committed state in the network, an outgoing edge from that state must be taken in the next transition.

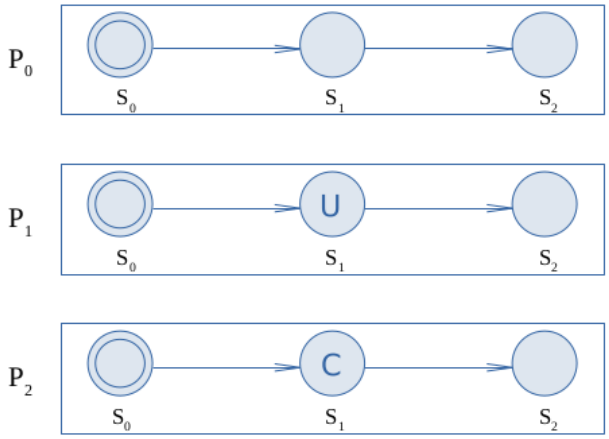

Figure 2.2: Example of three automata in parallel

Referring to Figure 2.2, it is possible to see three different automata in parallel: `P0` with normal states, `P1` with an urgent state (marked with `U`) and `P2` with a committed state (marked with `C`). When the network of `P0`, `P1` and `P2` automata is in initial state ⟨ `P0.S0, P1.S0, P2.S0` ⟩, time can progress and any edge can be fired. Instead, when considering state ⟨ `P0.S0, P1.S1, P2.S0` ⟩, `P1` is in a urgent location, so time cannot progress, but any edge can

be fired. Finally, in ⟨ `P0.S0, P1.S0, P2.S1` ⟩ state, `P2` is in a committed location, so time cannot progress and the only edge that can be taken is the edge from `P2.S1` to `P2.S2` , that is the only outgoing edge from a committed location in the current state of the network.

Finally, UPPAAL adds some expressions to Timed Automata states and edges, like:

- **Select**

  The select label allows the definition of a variable whose value is selected non-deterministically inside a specified range, when firing an edge

- **Guard**

  The guard label refers to a side-effect free expression that must evaluate to a boolean value, and can be associated with an edge to control its execution

- **Update**

  The update label is a list of expressions with a side-effect that is executed when firing an edge

- **Invariant**

  The invariant label refers to a side-effect free expression that must be verified in its referring state

UPPAAL SMC modeling formalism is based on a stochastic extension of the Timed Automata formalism described above. This extension consists of probabilistic choices (with user-defined weights) instead of non-deterministic choices where multiple enabled transitions are available, and instead of non-deterministic choices for time delays, there are probability distributions which

are resolved by either uniform distributions in cases of time-bounded delays (in agreement with the specified delay interval) or exponential distributions (with user-defined rates) in cases of unbounded delays [28].

We will also refer to probabilistic choices as *probabilistic branching* and to the branches of a probabilistic choice as *probabilistic branches*.

A model in UPPAAL SMC consists of a network of interacting Stochastic Timed Automata, where the communication is restricted to broadcast synchronizations, thus having only non-blocking synchronization channels. During simulations, Stochastic Timed Automata components independently and stochastically decide how much to delay before firing a transition, but only the component that chooses the minimum delay will actually execute the edge.

2.3.2 Query language

The UPPAAL query language consists in a simplified version of TCTL (Timed Computation Tree Logic), which in turn derives from traditional CTL branching temporal logic, with the addition of clocks inside formulas [27]. The TCTL query language expresses state formulas, which describe individual states of the model, and path formulas, which instead quantify over traces of the model. The former are expressions that do not need a complete observation of the behavior of the model, because they are evalueted on single states; the latter, instead, are further classified into reachability, safety and liveness formulas, based on relative properties, and UPPAAL simplified version of TCTL does not allow nesting of the latter classes of formulas.

Reachability properties ask whether a given state formula ϕ possibly can be satisfied by any reachable state, meaning that there exists a path, starting

from the initial state, such that ϕ is eventually satisfied along that path. In UPPAAL, these properties are written as $E <> \phi$.

Safety properties state that *"Something bad will never happen"*, but the corresponding formula is formulated positively, resulting in *"Something good is invariantly true"*. Thus, $A[\,]\,\phi$ express that state formula ϕ should be true in all reachable state, while $E[\,]\,\phi$ express that ϕ should be true in all reachable state of at least one path.

Liveness properties are of the form "something good will eventually happen", and $A <> \phi$ means that state formula ϕ is eventually satisfied along all paths.

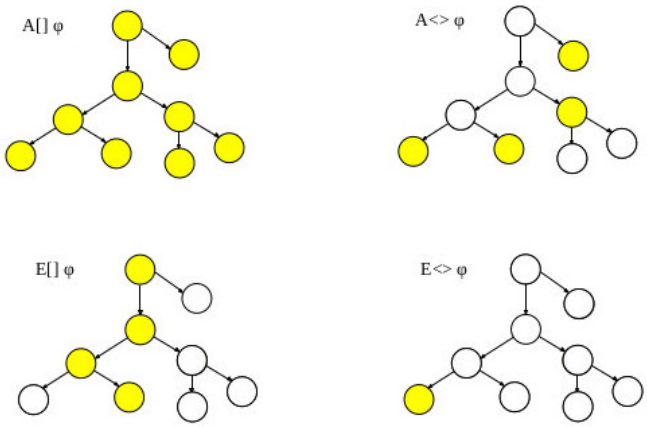

Figure 2.3: Examples of formulas in UPPAAL

The UPPAAL SMC query language extends these standard MC queries, allowing to check properties related to the stochastic interpretation of timed automata, and to express such properties the Weighted Metric Temporal Logic (WMTL) is used [28]. In addition to traditional operators from temporal logic, WMTL provides a time-bounded *Until* operator: the formula

$\phi_1 \, U_{x \leq b} \, \phi_2$ is satisfied by a run if ϕ_1 holds on the run until ϕ_2 is satisfied, and this must happen before the value of the clock x exceeds bound $b \in \mathbb{N}$.

From this time-bounded Until operator, it is possible to derive time-bounded Eventually and Always operators: hence, $P_M \, (\Diamond_{x \leq b} \, \phi)$ denote the probability that a random simulation run of the model M eventually satisfies ϕ before clock x exceeds bound b.

UPPAAL SMC uses statistical Model Checking algorithms, which permit to answer three types of questions, as seen above:

- Probability estimations $P_M \, (\Diamond_{x \leq b} \, \phi)$
  In Uppaal SMC, the probability confidence interval can be estimated by the query $Pr \, [\, bound \,] \, (\, \phi \,)$;

- Hypothesis testings $P_M \, (\Diamond_{x \leq b} \, \phi) \geq p$, for $p \in [0, 1]$
  In UPPAAL SMC, the query used to perform the hypothesis testing is $Pr \, [\, bound \,] \, (\, \phi \,) \geq p$;

- Probability comparisons $P_M \, (\Diamond_{x \leq b} \, \phi_1) \geq P_M \, (\Diamond_{x \leq b} \, \phi_2)$
  In UPPAAL SMC, a probability comparison is expressed by the query $Pr \, [\, bound_1 \,] \, (\, \phi_1 \,) \geq Pr \, [\, bound_2 \,] \, (\, \phi_2 \,)$;

The *bound* variable inside the above queries defines how simulation runs have to be bounded: for example, they can be bounded by implicit time, by explicit cost or by a number of discrete steps of the simulation.

UPPAAL SMC uses a different algorithm to solve probability estimation queries with respect to the algorithm used for hypothesis testing and probability comparison queries. The probability estimation algorithm executes a number of simulation runs sufficient to guarantee that the probability to

be estimated lies within an approximation interval $[p - \epsilon, p + \epsilon]$ with a confidence $1 - \alpha$, where p is the resulting estimated probability, α is the probability of false negatives and ϵ is the probability uncertainty. Hence, the values of both α and ϵ, which consitute the configuration parameters for the application of UPPAAL SMC, affect the number of simulation runs needed to guarantee that property.

Only probability estimation queries have been used in the verification step of this thesis work to find out the probabilty of undesirable and unsafe configurations of the model.

Furthermore, Uppaal SMC supports the evaluation of expected values of *min* or *max* of an expression that evaluates to a clock or an integer value. This evaluation is expressed as $E[bound; N](min : expr)$ for the minimum average evaluation and $E[bound; N](max : expr)$ for the maximum average evaluation, where N is the number of runs to be executed and *expr* is the expression to evaluate.

Chapter 3

Case study and modeling

This thesis takes into account a portion of the same railway signaling subsystem identified by the 4SECURail project to exercise the formal methods demonstrator. The 4SECURail selected case study concerns the RBC/RBC handover interface as specified by UNISIG into Subset 039 – FIS for the RBC/RBC Handover and by Subset 098 – RBC/RBC Safe Communication Interface. These two subsets provide a public standardized interface that specifies the requirements for the handover protocol between neighboring RBCs in natural language.

The public availability of these requirements played a relevant role in the adoption of the RBC/RBC handover interface as the case study for the validation of the 4SECURail formal methods demonstrator. Moreover, RBCs are a typical example of products that can be developed and provided by different suppliers, and their interoperability is essential to guarantee efficient communications within the infrastructure railway system. Therefore, the RBC/RBC handover interface results extremely appropriate to investigate how natural language specifications may create the possibility of diverging

interpretations [29], providing a significant estimation of formal methods advantages.

The objective of carrying out some formal analysis on existing standard specifications concerns the possibility of providing a more accurate definition of the requirements. Indeed, this procedure allows finding possible residual uncertainties, like ambiguities or contradictions, which could lead to interoperability issues. An improvement of the specification can instead significantly reduce this risk, permitting the production of different implementations based on non-diverging architectures.

As a result, the formal model developed during this thesis work can be used as a further artifact to complement the analysed case study. It can enrich the existing documentation by providing unambiguous modeling of the system behavior starting from its specification. Moreover, the verification of the model is based on the translation of the safety requirements of the system into formally verifiable properties, which can also complement the documentation.

It is important to specify that the practices of requirements elicitation or requirements re-writing are out of scope for this thesis, needing some specialized knowledge and technical competence from Infrastructure Managers to be correctly implemented. Hence, when facing with ambiguous requirements, the need for further details is pointed out and possible personal implementation choices are reported.

By exercising the demonstrator on this railway case study, the main 4SECURail objective is the evaluation of the cost-benefit analysis for the adoption of formal methods, compared to similar processes based on the traditional methodologies [30]. This thesis work aims also to provide some

significant data on the use of formal methods by performing formal modeling and verification on part of the same specifications of the 4SECURail case study, measuring the effort needed for the different steps of these processes, thus contributing to the cost-benefit analysis of the railway initiative.

Focusing on the selected case study, the handover procedure is used to manage the interchange of train control supervision between two neighboring RBCs. A railway line is divided into several RBC (Radio Block Center) supervision areas and each RBC is responsible for providing movement authorities to permit the safe movement of trains. These movement authorities are generated based on information from both external track-side and on-board equipment.

Referring to Figure 3.1, when a train is approaching the end of the area supervised by one handing over RBC, an exchange of information with the accepting RBC takes place to manage the transaction of responsibilities [31]. The communication between RBCs consists of an exchange of NRBC messages, i.e. messages sent to or received from a neighbor RBC.

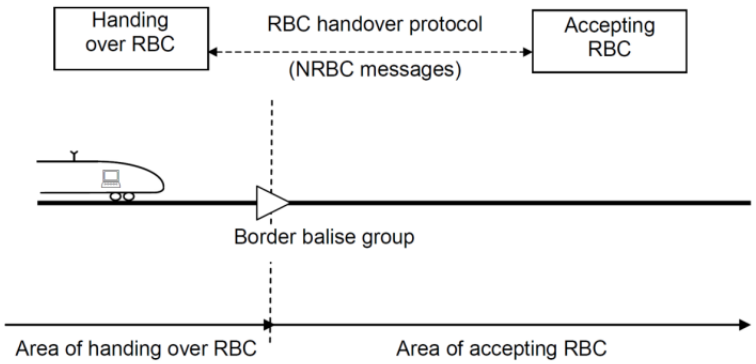

Figure 3.1: The RBC/RBC handover protocol (from Subset 039 [31])

This protocol is based on a layered structure. The higher layer corresponds to an application process that addresses high-level functionalities, such as the generation and the reception of information to communicate with peer RBC entities, or the re-establishment of the safe connection when it is lost due to errors in lower layers. This layer communicates with an underlying layer, the Safety Functional Module (SFM) which specifies the requirements related to the safety of the communications. The SFM layer provides adequate protection against the threats identified by CENELEC and specified in the EN 50159 European Standard [32].

According to this standard, a generic transmission system can be affected by the following list of communication errors:

- **Repetition**

  A message already sent is sent again in the message stream;

- **Deletion**

  A message is removed from the message stream;

- **Insertion**

  An additional message is implanted in the message stream;

- **Re-sequencing**

  The ordering of messages in a stream is changed;

- **Corruption**

  The content of a message is corrupted;

- **Delay**

  The reception of a message happens later than planned;

- **Masquerade**

  A message not intended for the receiver is intentionally implanted in the message stream.

The SFM implements the safety services identified by the standard to reduce the risk associated with the occurrences of those threats. Indeed, the SFM layer consists of two distinct sub-layers, the SAI (Safe Application Intermediate) sub-layer and the Euroradio SL (Euroradio Safety Layer), and their combination provides a safe protection strategy for the open transmission system.

The Euroradio SL implements the Message Authenticity and the Message Integrity safety services, which protect the system against corruption, masquerade and insertion threats by providing a safety code (Message Authentication Code) and a connection identifier (source and destination ids).

The SAI sub-layer instead implements the Message Sequence and the Message Timeliness safety services. The protection is achieved with a sequence number for deletion, re-sequencing and repetition threats, and with a delay defense technique, achieved with either the EC (Execution Cycle) technique or the TTS (Triple Time Stamp) procedure, for delay threat.

The Euroradio SL communicates with the Communication Functional Module, which implements Transport, Network and Data Link layers, but provides also an Adaptation layer between the Euroradio SL and the Transport layer.

3.1 Safe Application Intermediate sub-layer

This thesis work focuses on the SAI sub-layer of the RBC/RBC handover protocol and consists of the formal modeling of a system starting from the

specifications of the Subset 098 [33].

As stated in the subset, the SAI sub-layer protects against possible threats during the communication between RBCs, the interface to Euroradio SL and the interface to the application process, also referred to as SAI User. The protection against deletion, repetition, re-sequencing and delay threats that can occur in a transmission system is achieved by adding a sequence number and either an EC counter or a triple time stamp (TTS).

The sequence number permits the identification of the correct message stream and the detection of possible message deletion, repetition or re-sequencing errors during the communication, while both EC counter and TTS protect against the obsolescence of messages due to transmission delays.

In this thesis work, we chose to model the TTS delay defense technique, so from now on only this technique will be taken into account as transmission delay protection.

It is important to notice that the protection provided by the sequence number and the TTS concerns the detection of the occurrences of those errors. The SAI sub-layer neither checks the correctness of the content when a message is received, nor provides recovery actions in case errors are detected. The SAI protocol is only responsible to send and receive messages generated by the RBC application process, ensuring that the exchanged message streams are not affected by any of the aforementioned errors. This prevents the delivery to the SAI User of erroneous messages that could generate potentially unsafe situations. The actions needed to protect against the consequences of a communication error are responsibility of the application process, and are out of scope for this thesis work.

The SAI sub-layer is responsible to add both the sequence number and the TTS to the messages coming from the SAI User and directed to the other RBC device. These messages contain the information that the RBC devices must exchange to permit the handover of the train. We will refer as application messages to the messages coming from the SAI User. The SAI sub-layer adds in a structured header the sequence number and the TTS values, together with an additional message type field value, before forwarding the message to the Euroradio SL. The structured header allows the peer SAI entity to extrapolate the information needed to detect if a communication error has occurred.

In Figure 3.2, the structure of the SAI header is shown.

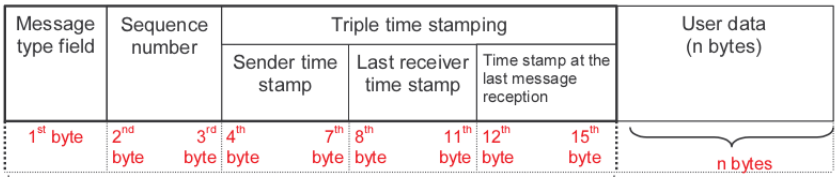

Figure 3.2: Structure of the SAI header, if TTS is used (from Subset 098 [33])

The first byte of the header is dedicated to the *message type field*; its value identifies the type of the message that is exchanged. When TTS defense technique is used, the message type field can assume six different values: from value 01 to value 05, the message type field see a particular message inside the clock offset update procedure (refers to sub-section 3.1.2), and value 06 refers to an application message protected by TTS (user data contains SAI User information to be exchanged).

The second and the third bytes are left for the *sequence number* field, hence they define the sequence number of the message inside the message stream.

The following twelve bytes, from fourth to fifteenth, are occupied by TTS values. The *sender time stamp* field defines the time-stamping of the sender when the message is delivered to the local Euroradio SL entity, i.e. the Euroradio SL of the same RBC device of the SAI sending the message. The *last receiver time stamp* field contains the last timestamp received from the sender peer entity (the "receiver") and contained in the *sender time stamp* field of the last message received. The *time stamp at the last message reception* field indicates the SAI local timestamp when the last received message has been delivered by the local Euroradio SL entity.

The SAI sub-layer provides the interface to local SAI User and local Euroradio SL through the definition of a SAI protocol for connection, disconnection and application data exchange procedures. This protocol relies on safe service primitives of which only the functional specification is provided (the particular implementation does not impact on RBC interoperability).

The interface between the SAI User and the SAI sub-layer is composed of the following safe service primitives:

- SAI-CONNECT.request is used by the SAI User to initiate the establishment of a connection at the SAI level;
- SAI-CONNECT.indication is used by the called SAI entity to notify the SAI User of the request for a connection establishment;
- SAI-CONNECT.response is used by the called SAI User to accept the connection request to the SAI entity;
- SAI-CONNECT.confirm is used by the initiating SAI entity to inform the calling SAI User about the successful establishment of the safe connection;

- SAI-DATA.request is used by the SAI User to transmit application data to the peer entity;
- SAI-DATA.indication is used by the SAI entity to indicate to the SAI User that some data have been received successfully from the peer entity;
- SAI-DISCONNECT.request is used by the SAI User to enforce the safe connection release;
- SAI-DISCONNECT.indication is used by the SAI entity to inform the SAI User about the safe connection release.

The interface between the SAI entity and the Euroradio (ER) SL is composed of the following safe service primitives:

- Sa-CONNECT.request is used by the SAI entity to initiate the establishment of a safe connection;
- Sa-CONNECT.indication is used by the called ER entity to notify the SAI entity of the request for a connection establishment;
- Sa-CONNECT.response is used by the called SAI entity to accept the connection request to ER entity, without any specific authorization from upper layers;
- Sa-CONNECT.confirm is used by the initiating ER entity to inform the calling SAI entity about the successful establishment of the safe connection;
- Sa-DATA.request is used by the SAI entity to transmit application data to the peer entity;

- Sa-DATA.indication is used by the ER entity to indicate to the SAI entity that some data have been received successfully from the peer entity;
- Sa-DISCONNECT.request is used by the SAI entity to enforce the safe connection release;
- Sa-DISCONNECT.indication is used by the ER entity to inform the SAI entity about the safe connection release.

The following figures illustrate the SAI protocol for the connection procedure (Figure 3.3), for the disconnection procedure (Figure 3.4) and the application data exchange procedure (Figure 3.5).

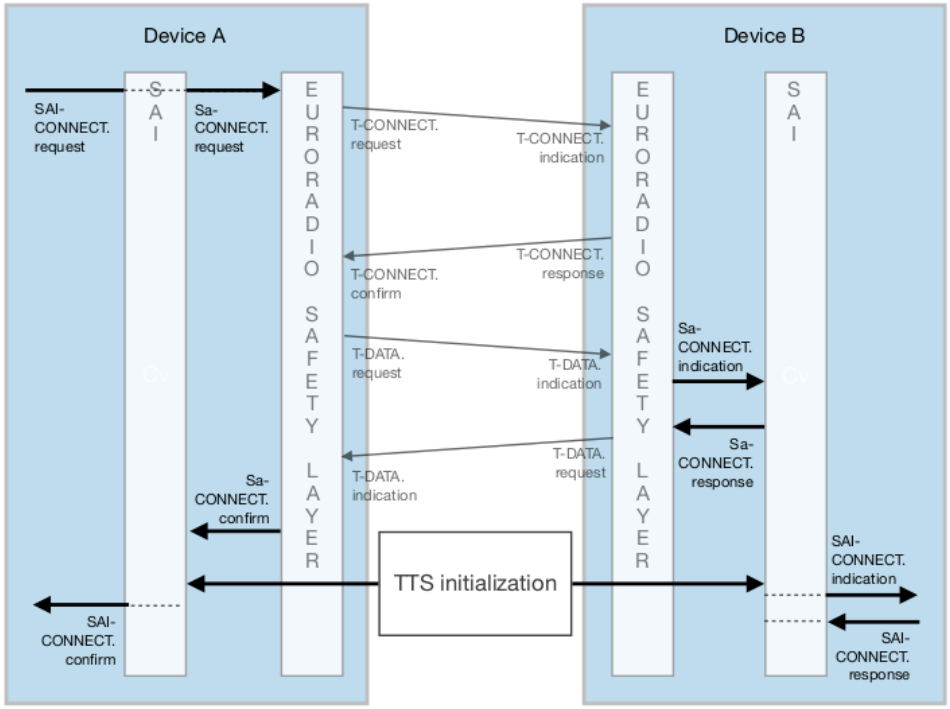

Figure 3.3: Connection procedure (adapted from [33], Figure 7)

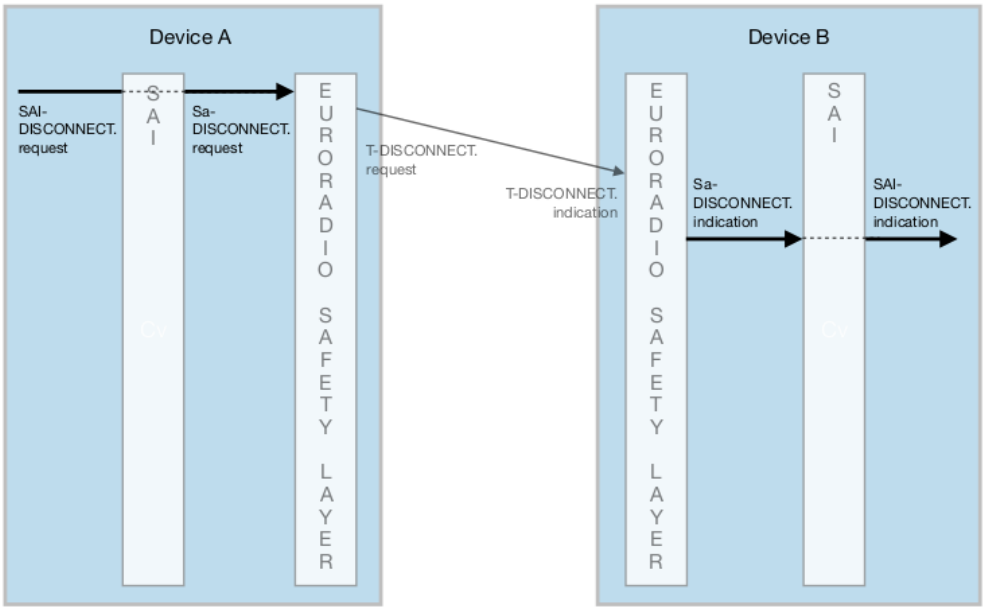

Figure 3.4: Disconnection procedure (adapted from [33], Figure 8)

The procedure for the exchange of application data messages starts when the SAI User uses the SAI-DATA.request primitive. The local SAI entity receives the message and adds the SAI header, specifying the adequate message type field (value 06 for the Application Message protected by TTS), the sequence number and the TTS fields. Then, it uses the Sa-DATA.request primitive to deliver the message composed of the SAI header and the user data to the Euroradio SL. When the Euroradio SL receives a message from the underlying layers, it uses Sa-DATA.indication primitive to pass the message to the SAI entity, which checks that the message type field is one of the values expected, then checks the sequence number and finally checks the TTS fields.

If all the checks are completed successfully, the SAI entity uses the SAI-DATA.indication primitive to pass the accepted message to the SAI User.

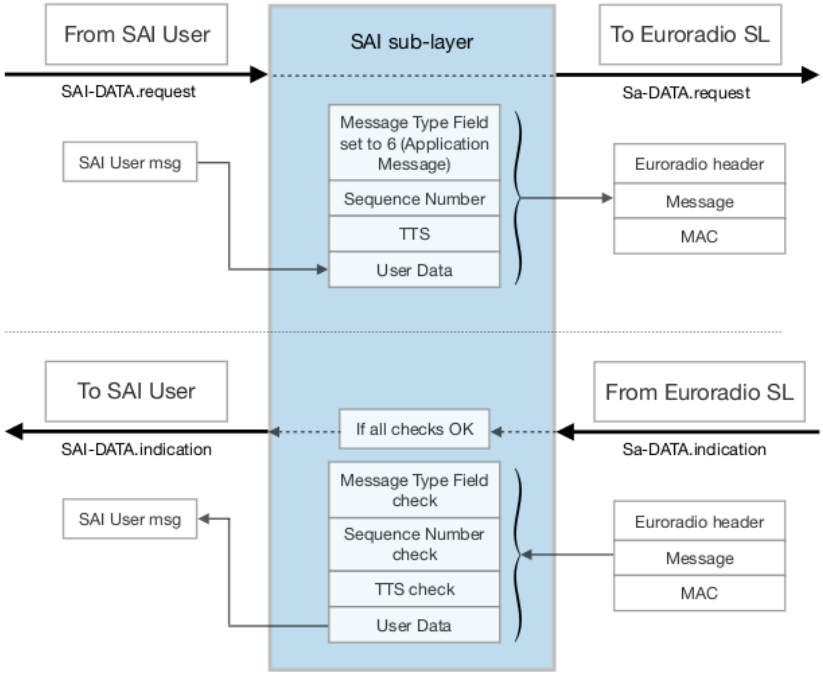

Figure 3.5: Application data exchange procedure (adapted from [33], Figure 9)

If instead one of the checks fails, the SAI sub-layer shall release the connection when a configurable number of successive errors is reached. Indeed, each SAI sub-layer defines a T_{succ_err} parameter whose value can be set to 1 for highly-available systems, or 2 for systems where an error occurrence can be tolerated.

3.1.1 Sequence number

The sequence number is a value that is added in the sequence number field of the SAI header for each message that the SAI User exchanges with the peer entity. The role of the sequence number is to make the receiving SAI entity aware of any occurrence of an error in the message stream exchanged with the sender SAI entity. Each communicating direction uses an indepen-

dent sequence of message numbering, and there is no requirement for the first sequence number received from the peer entity after the connection set-up. Hence, the first sequence number is always accepted, and the sequence number check is performed from the second message of the stream.

The value of the sequence number is incremented by one at each message transmission in the same direction, and when its value reaches the maximum allowed value, at the next message transmission the sequence number is set to 0.

To verify the correctness of the message stream, the sequence number difference with the previously accepted message is checked: if the computed difference is 1, the received message can be accepted and passed to the SAI User because no communication errors are detected. Instead, if the difference is different from 1, the message is treated as a sequence error and the SAI User is notified.

In each SAI sub-layer, a parameter N has to be configured. $N-1$ defines the maximum number of missing messages allowed, so N can be equal or greater than 1. When there are N missing messages, the SAI sub-layer commands the safe connection release and the message is discarded. Instead, if the number of missing messages is between 1 and $N-1$, the SAI User is notified but the message is not discarded and the user data is passed to the upper layer. Hence, N shall be set to 1 if no missing messages are allowed. Finally, the received message is discarded if its sequence number is less than the sequence number of the last accepted message and for equal sequence numbers, the SAI User is also notified.

In Figure 3.6, the behavior of the sequence number defense technique is shown in case of repetition, deletion and re-sequencing communication errors

for $N = 3$, meaning that up to two missing messages are allowed.

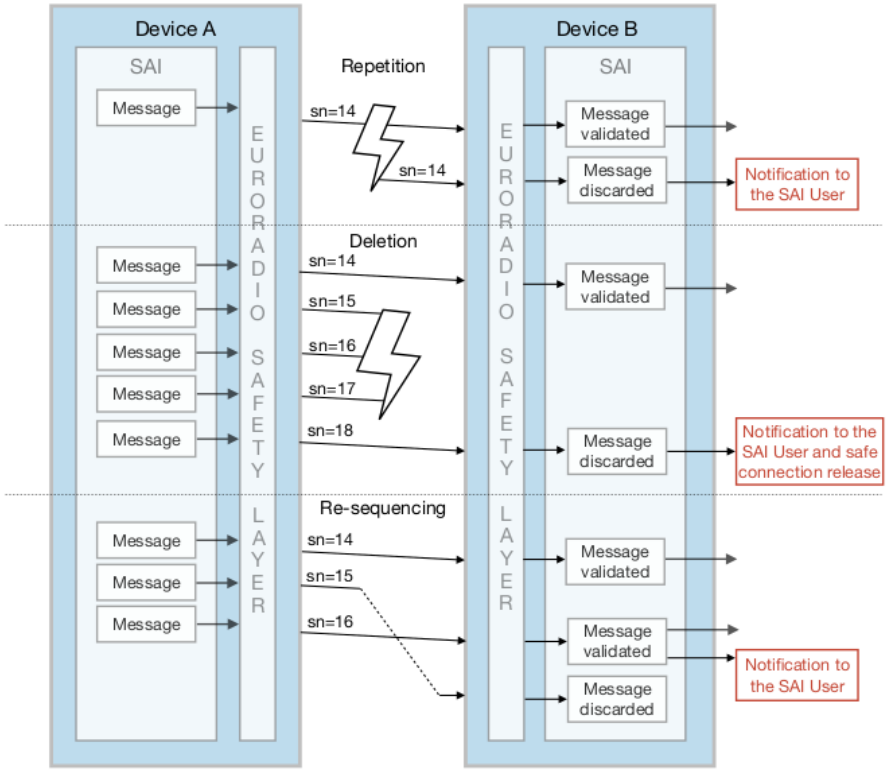

Figure 3.6: Sequence number defense technique for possible sequencing errors (adapted from [33], Figure 11)

3.1.2 Triple Time Stamp

The Triple Time Stamp technique is used to detect communication errors due to transmission delay. It is based on the clock-offset estimation between the two communicating devices, which permits the evaluation of the time validity of the incoming messages. This estimation is performed after the safe connection establishment and before the exchange of the first application message, allowing each device to estimate the clock offset between its internal clock and the one of the peer devices. The device that initiates the safe

connection is also responsible to start the clock offset update procedure, and it is referred to as the *Initiator*, while the device that receives the request for the safe connection establishment is referred to as the *Responder*.

The procedure can include the optional cycle time information in case of systems using a cyclic transmission. In this thesis work, we decided not to use this cycle time at the SAI level and to refer to a more general scenario where the responsibility of checking time-to-reply is assigned to the SAI User. Therefore, this cycle time at the SAI level will no longer be taken into consideration.

The procedure for TTS initialisation (also referred to as the clock offset update procedure) consists of the exchange of five messages between the two devices. This message exchange is handled by the SAI sub-layer, so the User Data field of the messages remains unused, unless otherwise specified. The structure of the five messages is described below:

1. **OffsetStart message**

   The first message of the procedure, sent by the Initiator to the Responder. Its header follows the structure of the messages shown in Figure 3.2: the message type field is set to 01; the sequence number field is set to the first of the message stream for the Initiator; the sender time stamp field defines the timestamp of the Initiator at the delivery of the OffsetStart message; both the last receiver time stamp and the time stamp at the last message reception fields are set to 0, as there is no previous message from the Responder.

2. **OffsetAnsw1 message**

   The second message of the procedure. It is sent by the Responder after the reception of the OffsetStart message by the Initiator. In its

header, the message type field is set to 02 and the sequence number is set to the first of the message stream for the Responder. Concerning the TTS fields, the sender time stamp defines the timestamp of the Responder at the delivery of the OffsetAnsw1 message, the last receiver time stamp contains the Initiator timestamp transmitted in the last message received by the Responder and the time stamp at the last message reception is the time value of the Responder at the reception of the last message.

3. **OffsetAnsw2 message**

   The third message of the procedure. It is sent by the Initiator after the reception of the OffsetAnsw1 message. Its message type field is set to 03 and the sequence number is set according to the message stream of the Initiator. The sender time stamp is the timestamp of the Initiator, the last receiver time stamp contains the last timestamp transmitted from the Responder and the time stamp at the last message reception contains the timestamp of the Initiator at the last message reception from the Responder.

4. **OffsetEst message**

   The fourth message of the procedure. It is sent by the Responder after the reception of the OffsetAnsw2 message. The message type field is set to 04 and the sequence number is set according to the message stream of the Responder. The TTS fields contain the Responder timestamp at the delivery of the OffsetEst message (sender time stamp), the last timestamp received from the Initiator (last receiver time stamp) and the timestamp of the Responder at the last message reception from the Initiator (time stamp at the last message reception). Inside the user

data field, four additional data are specified. The first additional field is *offset sign*, which gives the algebraic sign of the Responder minimum offset estimation (0 for a positive or null value, 1 for a negative value). The second field is the *minimum offset estimation* computed by the Responder. The third and the fourth fields are respectively the *offset sign* field for the maximum offset estimation and the *maximum offset estimation* field computed by the Responder itself.

5. **OffsetEnd message**

   The fifth and last message of the procedure. It is sent by the Initiator after the reception of the OffsetEst message from the Responder. Its message type field is set to 05 and the sequence number is set according to the Initiator message stream. The TTS fields contain the Initiator timestamp at the delivery of the OffsetEnd message (sender time stamp), the last timestamp received from the Responder (last receiver time stamp) and the timestamp of the Initiator at the last message reception from the Responder (time stamp at the last message reception). Inside the user data field, a *check field* information is added to provide the result of the clock offset estimation checks to the Responder. The check field value is set to 1 if the check is validated, 0 otherwise.

The procedure for the TTS initialisation between the two devices can be outlined as follows:

- After receiving the Sa-CONNECTION.confirm primitive from the Euroradio SL, the SAI sub-layer of the Initiator device starts the clock offset update procedure by sending the OffsetStart message. At the transmission of the message, the SAI Initiator starts a timer T_{init_start},

and if the OffsetAnsw1 message is not received at the T_{init_start} timer expiration, the safe connection is released.

- At the OffsetStart message reception, the Responder replies with the OffsetAnsw1 message, and it starts a timer T_{res_start} for the reception of the OffsetAnsw2 message: if it is not received at the T_{res_start} timer expiration, the SAI Responder releases the safe connection.

- If the Initiator receives the OffsetAnsw1 message from the Responder before the T_{init_start} timer expiration, it can estimate the maximum and the minimum offsets between its clock and the clock of the Responder. The maximum offset estimation is computed as the difference between the Initiator timestamp at the last message reception (OffsetAnsw1) and the sender timestamp of the Responder contained within the OffsetAnsw1 message. The minimum offset estimation is instead computed as the difference between the last receiver time stamp and the time stamp at the last message reception both contained within the OffsetAnsw1 message. The last receiver time stamp field inside the OffsetAnsw1 message contains the time stamp sent by the Initiator in the OffsetStart message, while the time stamp at the last message reception field contains the timestamp of the Responder at the reception of the OffsetStart message. The Initiator sends the OffsetAnsw2 message and starts again T_{init_start} timer, this time to check the reception of the OffsetEst message within its expiration.

- If the Responder receives the OffsetAnsw2 message before the T_{res_start} expiration, in turn, it computes the maximum and the minimum offsets between its clock and the clock of the Initiator. The maximum offset estimation is computed as the difference between the Responder

timestamp at the last message reception (OffsetAnsw2) and the sender timestamp of the Initiator contained within the OffsetAnsw2 message. The minimum offset estimation is computed as the difference between the last receiver time stamp and the time stamp at the last message reception, both contained within the OffsetAnsw2 message. The last receiver time stamp field inside the OffsetAnsw2 message contains the timestamp sent by the Responder in the OffsetAnsw1 message, while the time stamp at the last message reception field contains the time stamp of the Initiator at the reception of the OffsetAnsw1 message. Note that these clock offset estimations for Initiator and Responder are computed through the same formulas, but with different input values, which depend on the transmission times of the system. At this point, the Responder sends the OffsetEst message containing its offset estimations to the Initiator and starts the T_{res_start} timer for the OffsetEnd message reception.

- If the OffsetEst message arrives before the T_{init_start} timer expiration, the Initiator checks whether the clock offset estimation was successful or not. This check is made comparing the Initiator and the Responder offsets estimations: if the sum of the Initiator maximum offset estimation and the Responder minimum offset estimation is equal to 0 and the modulus of the sum between the Initiator minimum offset estimation and the Responder maximum offset estimation is less than a T_{off_max} fixed parameter, then the offset check is validated, otherwise it fails. The Initiator sends the OffsetEnd message to inform the Responder about the result of the offset check. Then, in case of failure, it releases the connection, otherwise, it confirms the safe connection establishment to the SAI User through the SAI-CONNECT.confirm primitive.

- If the OffsetEnd message is received before the T_{res_start} expiration, the Responder takes into account the clock offset estimation result. In case of failure, it releases the connection, otherwise, it communicates the establishment of a safe connection to the SAI User through the SAI-CONNECT.indication primitive.

The minimum clock offsets estimation is used at the reception of each application data message to check the time validity of its transmission. As previously described, the SAI sub-layer is responsible for setting the three TTS fields inside the exchanged message. The procedure for TTS consists of the sender SAI which puts its timestamp in the sender time stamp field, the last timestamp received from the peer entity in the last receiver time stamp field and the timestamp at the reception of the last message from the peer entity in the time stamp at the last message reception field.

Figure 3.7 shows an example of message exchange between two devices where the behavior of the TTS fields is highlighted.

At the reception of an application data message, the receiver SAI estimates the sending time of the message in terms of its clock:

$$T_{receiver} = T_{time\_stamp\_sender} - \Delta T_{extra\_delay} + T_{rec\_offset\_min}$$

where $T_{time\_stamp\_sender}$ is the sender timestamp received in the message, ΔT_{extra_delay} is the sum of the extra delays due to processing times and $T_{rec\_offset\_min}$ is the minimum clock offset estimation of the receiver SAI. $T_{receiver}$ is the estimation of the application data time transmission in terms of the receiver clock. Hence, the freshness of the received message is computed as the difference between the timestamp of the receiver SAI at the message reception and $T_{receiver}$, the sending time estimated in terms of the

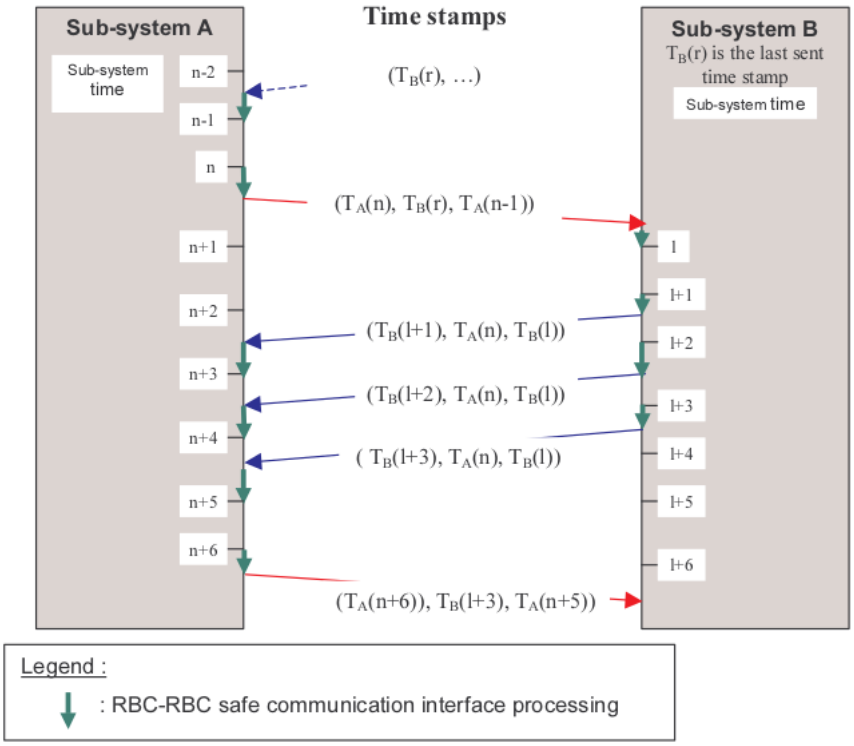

Figure 3.7: Exchanged messages using Triple Time Stamp principle (from Subset 098 [33])

receiver clock. If this difference does not exceed the T_{max} value, the delay of the message is acceptable, where the T_{max} value is a parameter of the SAI sub-layer that corresponds to the maximum validity time of the incoming messages from the peer entity. Instead, if the difference exceeds T_{max}, the delay is not tolerated and the message is rejected.

The SAI sub-layer is also responsible for the clock offset estimation update procedure, which is performed periodically according to a fixed value. When a clock offset update request is sent, the SAI initiator of the update procedure starts a T_{start} timer to supervise the time needed to perform the update. The structure of the messages used in the update procedure is compliant with the structure of the application data message with user data field empty. When

receiving a clock offset update answer, the SAI initiator verifies that the received answer is related to the last clock offset update request by checking that the last receiver time stamp field of the message is equal to the time stamp at the sending of its last request. If the check is successful, the SAI initiator updates both maximum and minimum offsets estimations. If the T_{start} timer expires before receiving the answer, the update procedure fails and the SAI initiator shall repeat the procedure until it succeeds.

3.2 Formal model description

In this thesis, we decided to model the SAI sub-layer of the RBC/RBC handover protocol starting from the specifications of the Subset 098 [33], and perform formal verification on the safety properties that the model should meet. The tool used to model and verify the case study is UPPAAL SMC; this choice reflects the real-time characteristics of the system emerging from the aforementioned requirements.

In particular, the TTS technique requires the modeling of a real-time system where the transmission times of the messages exchanged can be measured to protect the system from possible transmission delays. The UPPAAL SMC toolbox not only allows to work with real-time systems and stochastic behaviors, but also allows to formally prove the correctness of the model with respect to its requirements.

Moreover, this choice is in line with the 4SECURail objectives; indeed, the goal of the 4SECURail demonstrator is to experiment and demonstrate the use of formal methods for the definition of robust and reliable system requirements, and analysing the experience gained from it. Although the initiative suggests a preferred working tool, its goal is not to identify the

best choice for formal methods and tools to be used in the railway context, and similar experiences involving other tools to perform formal modeling and verification can provide interesting results.

The modeling of the system is focused on the SAI sub-layer as we are interested in analysing the protection mechanism against the possible threats identified by the CENELEC standard for a communication system. In particular, the SAI sub-layer protects the system against the threats of message repetition, deletion, re-sequencing and delay. The model includes also the SAI interfaces to the Euroradio SL and the SAI User adjacent layers, to allow accurate modeling of the connection, the disconnection and the data exchange procedures at the SAI level. Hence, both the Euroradio SL and the SAI User behaviors have been abstracted away, modeling only those aspects, such as the safe service primitives, that constitute the interface to the SAI sub-layer and allow to model its working environment.

3.2.1 Modeling choices

The models are defined through the specification of template automata. Each template automaton may have a set of parameters of any type (e.g. int, bool, etc), and local declarations of constants, variables, user functions and clocks. Global declarations are instead accessible from all the templates and can include clocks, constants, variables, functions and channels.

The system is defined as a network of processes that interact with each other in parallel; a process is instantiated from a template where all its parameters, if any, are set. The synchronizations between different processes of the system are obtained through broadcast channels, which are required in order to perform statistical model checking. We choose to declare as broad-

cast channels all the safety service primitives specified in the requirements to make the communications between the SAI and its adjacent layers as close as possible to the subset specification. Each primitive is configured as an array of channels where the indexing allows to identify the synchronized process.

For example, given an array of $N = 2$ channels defined as `broadcast chan c`$[N]$, the send-action `c`$[0]$`!` synchronizes with all the corresponding receive-actions `c`$[0]$`?`, while the send-action `c`$[1]$`!` synchronizes with all the corresponding receive-actions `c`$[1]$`?`.

It is important to notice that the UPPAAL channel synchronization does not support value passing, which however can be achieved in conjunction with the use of global shared variables where the sender writes the data to be passed and the receiver reads and empties them. This is possible because, referring to a global `channel c`, the update on the edge of a send-action `c!` is evaluated before the update on the edge of the receive-action `c?`, thus allowing the receiver to access the shared variables written by the sender in the same transition and then empty it.

Anyway, due to the non-blocking characteristics of the broadcast channels, even if no process can receive, the sender can still execute the `c!` action, and if this happens in conjunction with the value-passing, the shared variable could not be accessed and emptied. This implies that the synchronization between the send and the receive actions has not occurred because the signal of a sender has been "lost" by the receiver, thus possibly leading to undesired configurations. Therefore, to detect such events, the state invariant can be used to ensure that the shared variables used for the value-passing are always empty, meaning that they are always read and then emptied whenever they are written. This implies that signals are always received and never lost, overcoming the possible issues that the undetected message loss can cause.

In Figure 3.8, the overall architecture of the system is shown. The system we are about to present is composed of two communicating devices, an Initiator device that sends the request to establish a connection and a Responder device that receives the connection request. When referring to the partner device, for example when describing the sending of a message, we consider the Responder device as the partner of the Initiator device, and vice versa.

Each device is modeled using three modules: the SAI User, the SAI and the Euroradio SL modules. The SAI User and SAI modules are adjacent and can communicate with each other. The same applies also to the SAI and Euroradio SL modules. Both the Initiator and the Responder devices are composed of all these three modules, and when referring to the peer entity of one of the modules for a particular device, we mean the corresponding module of the partner device.

Finally, the Euroradio SL modules of both devices can receive failure notifications from the Communication System module, a component of the system that abstracts both the Euroradio SL lower layers and the transmission system. In particular, this component can communicate the occurrence of a disruptive connection release to both the Initiator and the Responder devices.

The communications between adjacent modules of the same device are modeled as instantaneous transmissions, which means that the processing times are abstracted away. Instead, we chose to focus the real-time aspects of the system on the communication between the two partner devices. Indeed, this communication consists of the exchange of signals which are affected by stochastic delays, simulating the transmission delays that characterize the radio communications.

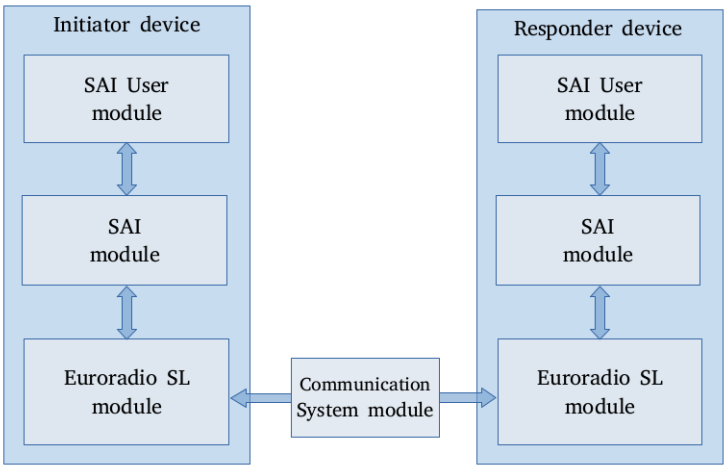

Figure 3.8: The overall system architecture

Referring to the templates that constitute the SAI module, they are characterized by an `id` parameter. This parameter allows identifying which device each template belongs to, i.e. the Initiator or the Responder device. Indeed, the id value is used for the array indexing, permitting each device to have separate synchronization channels for the communication exchange between its adjacent modules, and distinct access to all the shared variables used to model the system. Moreover, also the SAI User and the Euroradio SL modules are characterized by a `SAI_id` parameter, which is used to identify the device they belong to and the adjacent SAI module to communicate with.

We decided to assign id 0 to identify the Initiator device and the Initiator SAI templates and id 1 to identify the Responder device and Responder SAI templates. Anyway, some of the templates that we are about to describe have a Boolean `initiator` parameter that specifies the role of the belonging device in the RBC/RBC communication, i.e. the Initiator or the Responder. As the configured id parameter already identifies the role of the device, the initiator parameter is redundant, but we decided to include it to give better readability to the model where this information is used, such as when entering

the connected state.

While both the SAI User module and the Euroradio module functionalities are implemented through single templates, the SAI module is split into multiple sub-modules to reduce the complexity of each of them. In fact, the responsibilities of both the SAI User and the Euroradio modules were abstracted away and only the interface with the SAI is implemented, while the SAI module functionalities are completely modeled.

In particular, the SAI module is divided into three sub-modules: the connection establishment, the TTS technique and the defense techniques. The TTS technique sub-module is in turn divided into the TTS initialisation and the TTS update procedure sections.

The following list shows the templates that make up a single device:

- The SAI_User template that abstracts the SAI User module behavior. It is instantiated specifying the `SAI_id` parameter and the `initiator` parameter;
- The SAI_Conn_Ini/SAI_Conn_Res templates model the SAI connection establishment sub-module. They implement the corresponding connection procedure depending on the role of the device. The templates are instantiated specifying the `id` parameter corresponding to the device they belong to;
- The SAI_TTS_Init_Ini/SAI_TTS_Init_Res templates model the TTS initialisation section of the SAI TTS technique sub-module. They implement the corresponding TTS initialisation procedure depending on the role of the device. Its parameter is the same as the SAI_Conn_Ini/SAI_Conn_Res templates;

- The SAI_Sender template models the SAI defense techniques sub-module. It implements the message sending procedure. It is instantiated specifying the `id` and the `initiator` parameter that identifies the device it belongs to and its role;
- The SAI_Receiver template models the SAI defense techniques sub-module. It implements the check procedure for all the incoming messages. Its parameters are the same as the SAI_Sender template;
- The SAI_Update_Req template models the TTS update procedure section of the SAI TTS technique sub-module. It sends the offset estimations update requests. Its parameters are the same as the SAI_-Sender template;
- The SAI_Update_Answ template models the TTS update procedure section of the SAI TTS technique sub-module. It sends the offset estimations update answers. Its parameters are the same as the SAI_-Sender template;
- The Euroradio_SL_Env template that abstracts the Euroradio SL module behavior implementing the message exchange with the partner device. It is instantiated by specifying the `id` parameter and the `receiver` parameter.

In Figure 3.9, a graphic representation of the SAI module is shown. Notice that for the connection establishment sub-module and the TTS initialisation section, both the Initiator and the Responder templates are represented, but only one of them is instantiated according to the role of the device itself.

In the next sub-sections, we give a detailed description of all the templates we developed for the modeling of the system.

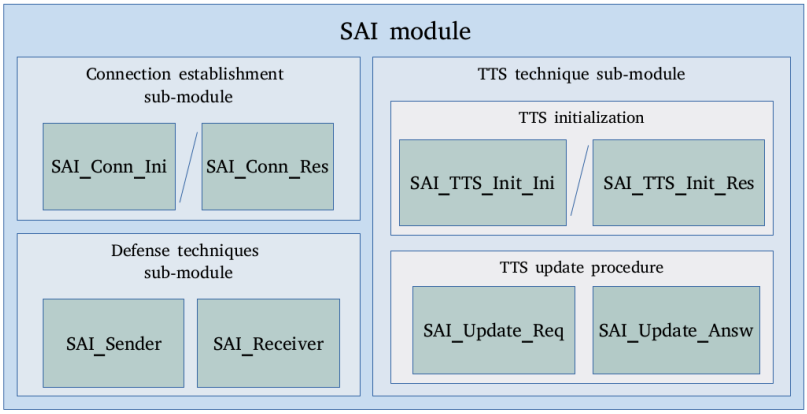

Figure 3.9: The SAI module architecture

3.2.2 SAI User module

The SAI User module is implemented through the SAI_User template shown in Figure 13. This template models the behavior of the SAI User layer and has to be instantiated for both the communicating devices. It is defined by a `SAI_id` constant parameter, which identifies its belonging device and permits the communication with the adjacent SAI module, and an `initiator` parameter, which is a constant Boolean specifying the role of the SAI User in the RBC/RBC communication.

As mentioned above, when a pair of RBC devices need to establish a safe connection, one of the two devices (the Initiator) starts the communication sending the connection request to the other device (the Responder). Hence, the Initiator SAI_User instantiation is characterized with the `initiator` parameter set to `True`, while the Responder SAI_User instantiation has got its `initiator` parameter set to `False`.

As described below, the behavior of the two SAI Users differs only for the connection procedure, but once the safe connection is established, they both refer to a common behavior. For this reason, we decided to model

only one parametric template, instead of two distinct templates, to define the Initiator and the Responder SAI User behaviors. This choice avoids the duplication of a consistent portion of the SAI User model by defining a common section after the safe connection establishment and specifying two distinct procedures for the connection, one for the Initiator and the other for the Responder. These two procedures are differentiated through the use of the `initiator` parameter.

From the `Disconnected` location, the Initiator SAI_User has the responsibility to send the safe connection request through the channel `SAI_CONNECT_request` to its adjacent SAI module. The process then waits in the `TryConnection` location for incoming SAI communications about the result of the connection procedure: a failure is advertised by a `SAI_DISCONNECT_indication` receive-action, while a `SAI_CONNECT_confirm` receive-action determines the successful safe connection establishment. The Initiator SAI_User device has also the responsibility to send again the connection request if no reply arrives within `T_conn_max` time units.

The Responder SAI_User waits in the `Disconnected` location until its adjacent SAI module synchronizes through the `SAI_CONNECT_indication` channel; if this happens, then the SAI_User answers with the `SAI_CONNECT_response` send-action to confirm the connection establishment.

Once the safe connection is established, through two local clocks, `t_msg` and `t_reply`, the SAI_User manages both the sending of a new message to its peer entity according to a `msg_freq` fixed interval of time, and the forced safe connection release, if `T_conn_max` time units have passed without replies.

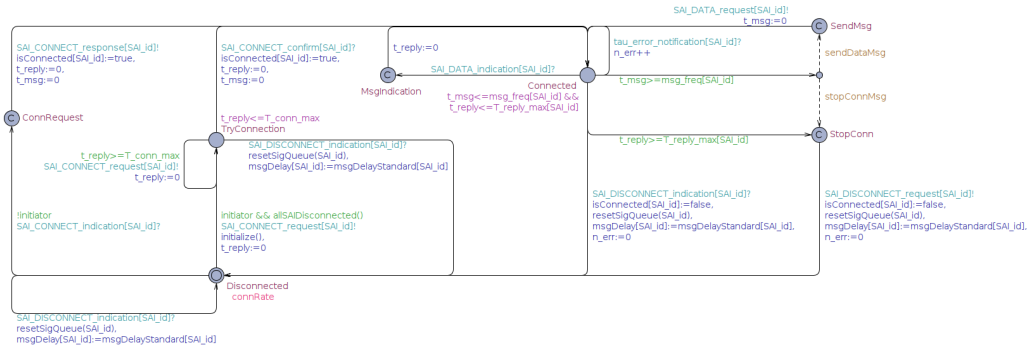

Figure 3.10: The SAI_User template

When sending a new message, through a probabilistic choice (represented in the model by dotted lines edges) the SAI_User can command to its adjacent SAI module either the request for a new application message sending (`SAI_DATA_request` with `sendDataMsg` weight) or the request for the safe connection release (`SAI_DISCONNECT_request` with `stopConnMsg` weight). This disconnection mode models the possibility that the SAI User decides to release the connection when the communication with the other device has reached the end. In the probabilistic branching, the probability of the branch with `sendDataMsg` weight is determined as

$$\frac{\texttt{sendDataMsg}}{\texttt{sendDataMsg} + \texttt{stopConnMsg}}$$

i.e. as the ratio between the weight of a particular branch over the sum of the weights of all the probabilistic branches of the same node.

When the SAI_User is in the `Connected` location, it can receive from its adjacent SAI module the communications about incoming accepted application messages from its peer entity (`SAI_DATA_indication`) and about notifications of erroneous message reception (`tau_error_notification`).

Moreover, through the receive-action `SAI_DISCONNECT_indication`, the adjacent SAI module refers that the safe connection has been released. This communication can originate from the SAI module itself when the conditions leading to the release of the connection are detected. It can also result from a disconnection request sent by the peer entity or it can depend on failures occurring at the Euroradio SL or lower layers.

3.2.3 Communication System and Euroradio SL modules

The communication system module is implemented through the Communication_System template, which abstracts the Communication Functional Module, composed of all the layers below the Euroradio SL, and the transmission system. In particular, this template models the disruptive connection release scenario, which is caused by some failures of the communication system. This template is instantiated once and has no parameters that characterize it. When a disruptive connection release happens, both devices are notified of the error through a broadcast synchronization channel.

The Communication_System template is shown in Figure 3.11. To model the low probability of a disruptive connection release occurrence, we adopted a minimum waiting time `T_commSystemCheck_min` to perform a check of the communication system, together with an exponential rate `commSystemCheck-Rate` to generate the exact delay of the transition.

Furthermore, the check edge has two probabilistic branches: one branch with `safeConn` weight determines that the connection is safe, while the other branch with `disruptiveConnRelease` weight determines that a failure has occurred. In this last case, both devices are notified through a

`tau_CommSystem_fault` send-action that synchronizes with the Euroradio SL modules (one-to-many communication), which in turn forward a disconnect indication to their adjacent SAI modules.

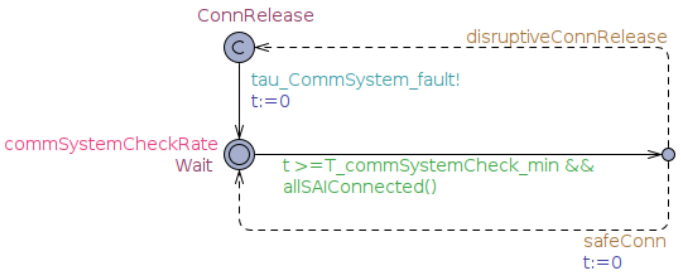

Figure 3.11: The Communication_System template

The Euroradio SL module is implemented through the Euroradio_SL_Env template, which represents the Euroradio Safety Layer environment. This template focuses on the modeling of the interface to the SAI services, while abstracting away the Euroradio SL main functionalities, like the protection against corruption, masquerade and insertion threats, which are out of scope for this thesis.

The Euroradio_SL_Env template shown in Figure 3.12 models the message exchange procedure between the two communicating devices. As seen for the SAI_User template, also the Euroradio_SL_Env template is instantiated for both the Initiator and the Responder devices and it is defined by the `SAI_id` constant parameter that identifies its belonging device and permits the signal forwarding with the adjacent SAI module. The Euroradio_SL_Env template is characterized also by a `receiver` constant parameter that identifies the id of the partner device to allow the sending of the message to the peer entity. Hence, the Initiator Euroradio_SL_Env instantiation has the `receiver` parameter set to the Responder id, and vice versa.

It is important to notice that both the Initiator and the Responder devices can send and receive messages from the partner device. From now on, we refer to the device that sends a message as the Sender device and its partner device that receives the message as the Receiver device.

To implement the message exchange procedure, we decided to define a shared queue as a two-dimensional array of incoming signals with fixed dimension $N = 2$ (the number of communicating devices) and handle it as a FIFO queue. Hence, both the Initiator and the Responder devices have their own signal queue (also referred to as buffer) and they can access it with their ids. This decision of using signal queues to simulate the exchange of messages between the two communicating devices is closely related to the implementation choice of using these buffers to model the occurrence of the previously described CENELEC threats.

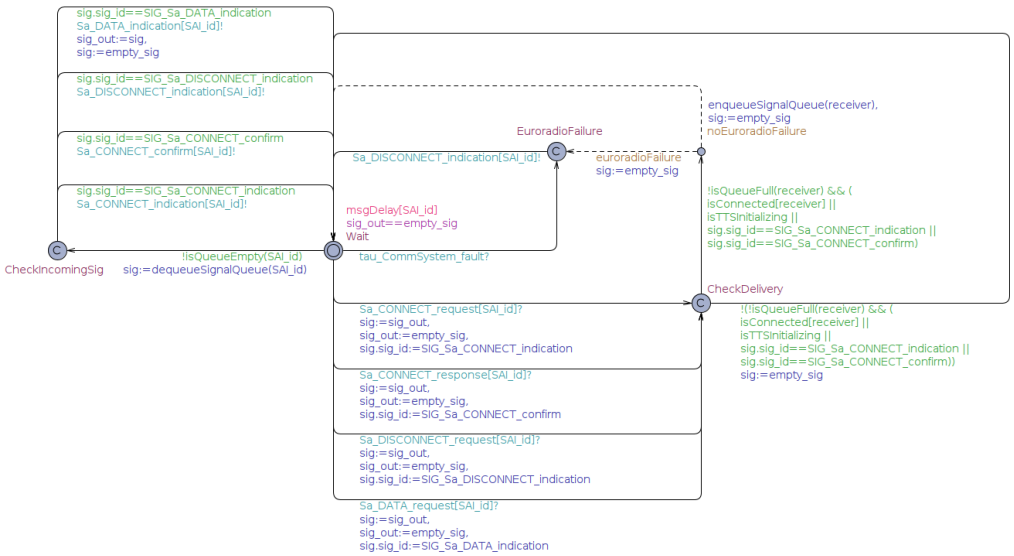

Figure 3.12: The Euroradio_SL_Env template

Back to the Euroradio SL module, each Euroradio_SL_Env template can both enqueue the outgoing signals in the queue of the Receiver device

(the enqueuing is instantaneous) and dequeue the incoming signals from its own signal queue (the dequeuing is instead affected by stochastic delay). Indeed, when the signals are removed from the queue, the transmission delay is modeled through a delayed transition where the exact delay is generated by the exponential distribution with rate `msgDelay[SAI_id]`.

In the `Wait` location, the Euroradio_SL_Env waits for receive-actions from the adjacent SAI module to send signals to the partner device. The signal is passed from the SAI module to the Euroradio_SL_Env through the `sig_out` shared variable (implementing value-passing) combined with the channel synchronization.

A *signal* is defined following the structure `sig_t`:

typedef struct {

int sig_id;

msg_t msg;

} *sig_t*;

where `sig_t` field encodes the signal related to the message (the corresponding values are shown in Table 3.1), and msg field contains the exchanged message (refer to sub-section 3.2.4 for `msg_t` structure).

Outgoing signal	Code	Incoming signal	Code
SIG_Sa_CONNECT_request	1	SIG_Sa_CONNECT_indication	3
SIG_Sa_CONNECT_response	4	SIG_Sa_CONNECT_confirm	2
SIG_Sa_DISCONNECT_request	5	SIG_Sa_DISCONNECT_indication	6
SIG_Sa_DATA_request	7	SIG_Sa_DATA_indication	8

Table 3.1: The signals encoding and the Euroradio_SL_Env outgoing/incoming signals transformation

When the signal is read from the `sig_out` shared variable, its signal identifier is changed according to the outgoing/incoming signals transformation shown in Table 3.1. This step abstracts away the actions of the Euroradio SL and its lower layers, allowing to transform an outgoing signal from the Sender Euroradio_SL_Env in the expected incoming signal for the Receiver Euroradio_SL_Env.

A signal is delivered only if the signal queue of the Receiver device is not full (the number of possible queued signals is bounded by the size of the queue) and at least one of the following conditions is true:

- The Receiver device is connected;
- The TTS initialisation is running;
- The incoming signal is a connect signal.

If the signal meets these conditions, a probabilistic branching decides if the signal is encoded in the Receiver's signal queue or a failure of the Euroradio SL blocks its delivery, and in this last case, a `Sa_DISCONNECT_indication` is issued to the adjacent SAI module.

Hence, we modeled a transmission system where we assume that failures can occur at the Euroradio SL message sending and that the conditions of the Receiver signal queue are known by the Sender device. This implies that the transmission of a signal is not performed when the Receiver signal queue is full, thus having no failures. Anyway, the configuration of the system is such that the probability of having a full signal queue is very low (refer to the next chapter for the relative query and its probability result).

If the signal queue is not empty, the Euroradio_SL_Env can dequeue the first signal (we assume that no failures can occur in the message receiving)

and sends to its adjacent SAI module the corresponding incoming signal according to the encoded `sig_id`. In the case of application messages, the signal is passed to the SAI module through the shared variable `sig_out`.

In addition to the failure in conjunction with a signal delivery, the Euroradio_SL_Env can stop to provide its services also when the Communication_System fails. Hence, at any time a `tau_CommSystem_fault` receive-action arrives, the Euroradio_SL_Env communicates the release of the connection to its adjacent SAI module.

In Figure 3.13, a successful message exchange procedure for an Application Message is shown. The Initiator Euroradio SL module takes the outgoing signal forwarded by the SAI module through the `Sa_DATA_request` channel, changes the signal identifier according to the outgoing/incoming signals transformation and then enqueues the signal in the Responder signal queue. The Responder Euroradio SL module dequeues the signal from its signal queue, reads the incoming signal id and forwards the signal to its adjacent SAI module through the `Sa_DATA_indication` channel as suggested by the `sig_id` field.

We have inserted the `sig_out==empty_sig` invariant in the `Wait` location of the Euroradio_SL_Env template, and also in other standard (i.e. neither committed nor urgent) locations of the model, to verify that each broadcast channel including a value passing is actually synchronized, i.e. for each send-action that writes in the `sig_out` variable, there is a receive-action that reads and empties it.

During a simulation run, if a state of the system is reached where the `sig_out==empty_sig` invariant does not hold, the simulation stops and an error message for the invariant violation is provided. This permits to detect if send-actions do not synchronize with any receive-actions when the value

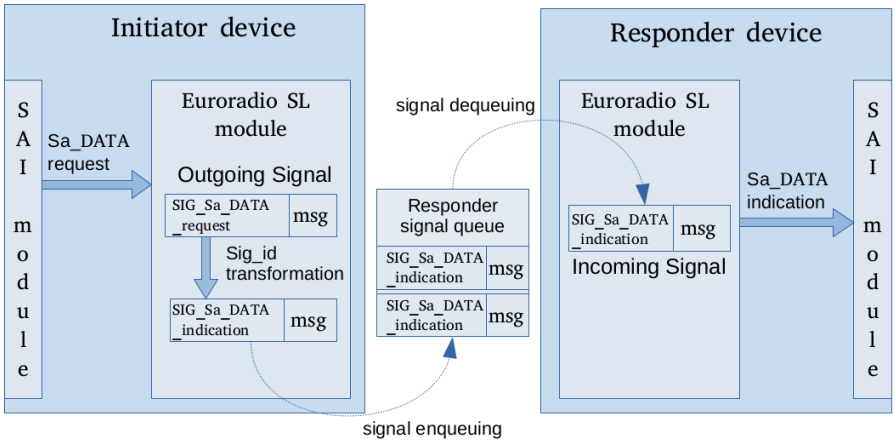

Figure 3.13: The message exchange procedure at Euroradio SL level

passing through the `sig_out` variable is involved, a condition that is possible due to the use of non-blocking broadcast channels, as mentioned above. Hence, the invariant guarantees that this kind of signal forwarding is always received and not lost, solving the problem of using broadcast channels.

3.2.4 SAI module

As mentioned above, the modeling of the SAI module has been split into multiple sub-modules due to the complexity that a single template would have in including all the responsibilities of the SAI sub-layer described in the specification. Moreover, this decision allows improving the modularity of the system, with distinct units that interact with each other and promote the separation of concerns.

Indeed, each of the sub-modules is implemented as separate templates where each of them models a portion of the required behaviors, for a total of eight distinct templates:

- **Connection establishment sub-module**

  This sub-module concerns the establishment of a connection between the two communicating devices. It is composed of two distinct templates, one instantiated for the Initiator device and the other instantiated for the Responder device, because of the different behaviors of the two devices at the SAI level. In particular, the SAI_Conn_Ini template implements the procedure for the connection establishment performed by the Initiator SAI, while the SAI_Conn_Res template implements the connection procedure counterpart performed by the Responder SAI.

- **TTS technique sub-module**

  The implementation of the TTS technique is composed of two distinct sections that reflect the two different applications related to the use of the Triple Time Stamp:

  - **TTS initialisation**

    When the Initiator SAI sends the connection request to its peer entity (the Responder SAI), it waits until the connection confirm arrives. At this point, the TTS initialisation procedure needs to be executed before the Application Message exchange can start. As for the connection establishment sub-module, also the TTS initialisation procedure is implemented through two distinct templates, one for the Initiator device and the other for the Responder device, having behaviors and responsibilities completely different from each other. Hence, the SAI_TTS_Init_Ini template is instantiated only for the Initiator device, while the SAI_TTS_-Init_Res template is instantiated only for the Responder device.

These two templates implement the exchange of the five messages of the TTS initialisation at the SAI level to compute the minimum and the maximum offsets estimations.

- **TTS update procedure**

  When the Initiator SAI sends the connection request to its peer entity (the Responder SAI), it waits until the connection confirm arrives. At this point, the TTS initialisation procedure needs to be executed before the Application Message exchange can start. As for the connection establishment sub-module, also the TTS initialisation procedure is implemented through two distinct templates, one for the Initiator device and the other for the Responder device, having behaviors and responsibilities completely different from each other. Hence, the SAI_TTS_Init_Ini template is instantiated only for the Initiator device, while the SAI_TTS_Init_Res template is instantiated only for the Responder device. These two templates implement the exchange of the five messages of the TTS initialisation at the SAI level to compute the minimum and the maximum offsets estimations.

- **Defense techniques sub-module**

  The defense techniques are implemented through two distinct templates that apply the principles of both the sequencing number and the TTS techniques. The SAI_Sender template models the component of the SAI module responsible for setting the sequence number and the TTS fields to each Application Message that the SAI User commands to exchange with the peer entity. The number sequencing allows the partner device to know the message stream being exchanged, while the TTS fields permit to verify the freshness of the incoming messages, to detect

eventual communication errors. Indeed, the other template that constitutes this sub-module is the SAI_Receiver template, which models the component of the SAI module that protects against the errors that can occur in a communication system, commanding itself the connection release if certain unsafe conditions meet. These two templates need to be instantiated for both the Initiator and the Responder devices because they model common behaviors and responsibilities of the devices, but as for the TTS update procedure templates, they have the initiator parameter to improve the readability of the model.

All these templates include a `Disconnected` and a `Connected` location and we choose to use the broadcast synchronization channels with the adjacent SAI User also for these SAI templates, exploiting one-to-many communications in order to switch between the two locations. Therefore, when the `SAI_CONNECT_confirm` send-action, forwarded by the `SAI_TTS_Init_Ini` template (responsible for carrying out the TTS initialisation procedure for the Initiator device), synchronizes with the Initiator SAI_User, it synchronizes also with all the other instantiated Initiator SAI templates to communicate that the connection has been established.

The same can be applied for the templates relative to the Responder device, where the `SAI_CONNECT_response` send-action, forwarded by the Responder SAI_User, synchronizes with all the instantiated Responder SAI templates, bringing them into the `Connected` location.

Similarly, the disconnection of all the instantiated SAI modules of a device occurs in conjunction with either the sending of a `SAI_DISCONNECT_request` from the SAI User module of the device or the reception of a `SAI_DISCONNECT_indication` to the SAI User module of the device from the peer entity.

Starting from the connection establishment procedure, as mentioned above, we decided to distinguish between the Initiator device and the Responder device because of the different behaviors and responsibilities that the SAI module express in this first part of the modeling: the Initiator SAI performs the connection request, while the Responder SAI waits for an incoming request to be answered.

In Figure 3.14 and Figure 3.15, the SAI_Conn_Ini and the SAI_Conn_Res templates are shown.

From the `Disconnected` location, the SAI_Conn_Ini template synchronizes with the `SAI_CONNECT_request` receive-action from the SAI_User. Through the `Sa_CONNECT_request` send-action, the signal is passed to the adjacent Euroradio_SL_Env, which sends it to the Responder device, as previously described. The SAI_Conn_Ini template waits in the `WaitConnConfirmSig` location the confirm signal from the Responder device, but it can also receive another `SAI_CONNECT_request` receive-action from the adjacent SAI_User, or a `Sa_DISCONNECT_indication` from the Euroradio_SL_Env due to a failure of the lower layers.

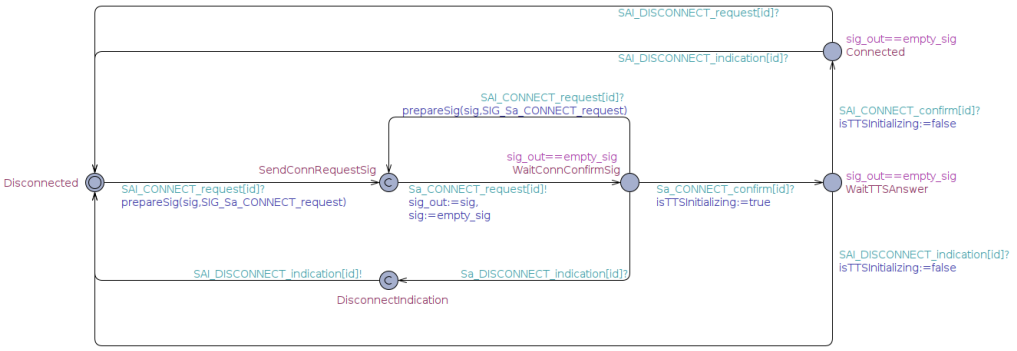

Figure 3.14: The SAI_Conn_Ini template

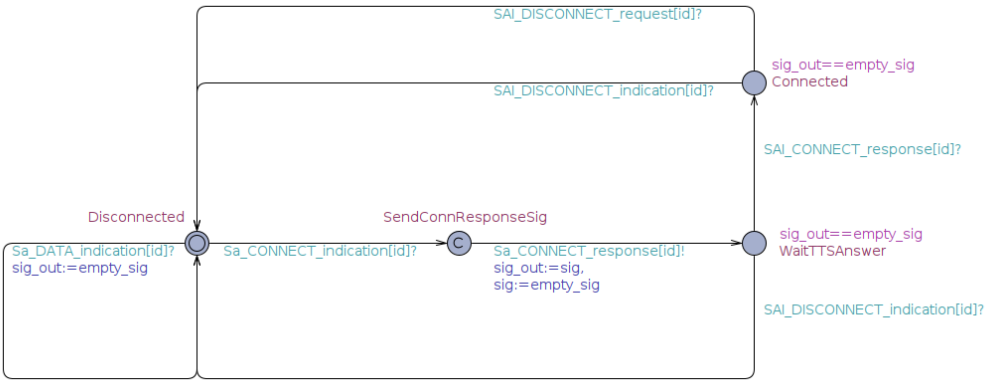

Figure 3.15: The SAI_Conn_Res template

If the `Sa_CONNECT_confirm` arrives, the TTS initialisation procedure starts and the SAI_Conn_Ini template waits for the result communication from the SAI_TTS_Init_Ini template. A positive result (the safe connection has been established) leads to the `Connected` location, while a negative result (the connection cannot be established) brings back to the `Disconnected` location, where the SAI_Conn_Ini template waits for a new connection request from its adjacent SAI_User.

The SAI_Conn_Res template waits for the `Sa_CONNECT_indication` receive-action coming from its adjacent Euroradio_SL_Env. When a connect indication arrives, it means that the Initiator device has sent a request to establish a safe connection and the SAI_Conn_Res answers with a `Sa_CONNECT_response` send-action that its adjacent Euroradio_SL_Env enqueues in the Initiator signal queue as shown before. As the SAI_Conn_Ini, also the SAI_Conn_Res waits for the TTS initialisation result to arrive from the SAI_TTS_Init_Res template and according to the incoming receive-action, it moves to the `Connected` or the `Disconnected` location.

These two templates model the first part of the connection procedure, i.e the connection establishment. Anyway, before the two devices can exchange some Application Messages, the TTS initialisation procedure to compute the minimum and maximum offsets estimations must be performed. The SAI_-TTS_Init_Ini and the SAI_TTS_Init_Res templates model the exchange of the five messages needed to perform the TTS initialisation and compute the offsets estimations. Both templates are parameterized with the `id` value to identify the device they belong to, during the channel synchronizations. Moreover, this allows to easily coupling the two distinct templates in the case in future only one parametric template to perform the TTS initialisation is required.

In Figure 3.16, the SAI_TTS_Init_Ini template shows the TTS initialisation procedure performed by the Initiator SAI sub-layer.

The SAI_TTS_Init_Ini starts the clock offset update procedure when receiving the `Sa_CONNECT_confirm` action from the adjacent Euroradio_-SL_Env and the first message exchanged for the TTS initialisation procedure is the OffsetStart message.

The structure of a message is defined by the following `msg_t` struct:

typedef struct {

mtf_t mtf;

sn_t sn;

int s_ts;

int last_r_ts;

int s_last_r_ts;

data_t user_data;

} *msg_t*;

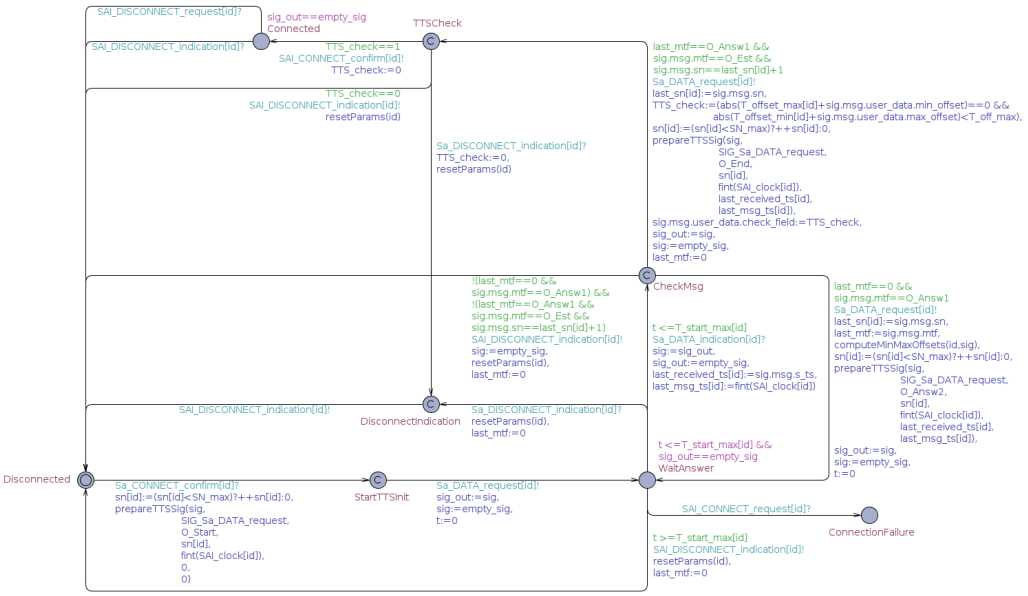

Figure 3.16: The SAI_TTS_Init_Ini template

The `mtf` and `sn` are the message type field and the sequence number associated with the message, respectively, where the `mtf_t` and `sn_t` define the allowable range of values for those fields. The `s_ts` field represents the sender timestamp, i.e. the timestamp at the sending of the message. The `last_r_ts` field contains the timestamp received in the last message by the peer entity. The `s_last_r_ts` field is the timestamp at the last message reception. Finally, the `user_data` field represents the application message content to exchange with the partner device.

These fields are set with the appropriate values at each message sending. The timestamp sampling is modeled through the use of a global array of $N = 2$ clocks named `SAI_clock`. Each device can access its own clock and when a SAI template needs to timestamp a message, thanks to the `fint` UPPAAL function, it can convert the double value sampled from its clock to an integer value. The two clocks can be initialised to a configurable initial

value (instead of starting from `0.0`) and a positive temporal drift can be applied to one of the clocks to simulate devices with non-aligned clocks.

Also to keep track of the sequence number stream, the last timestamp received from the peer entity and the timestamp at the last message reception, three global arrays named `sn`, `last_received_ts` and `last_msg_ts` respectively are used. Through the `id` parameter, each device (that is, all the templates that model the behavior of a device) can access its own position inside the arrays, and read or write its content. To make this possible, global variables are mandatory; indeed, local variables can only be read or written from that template where they are locally defined and cannot be used when different templates need to access a shared resource.

After that the SAI_TTS_Init_Ini sends the OffsetStart message, it waits in the `WaitAnswer` location for the OffsetAnsw1 message from its peer entity to arrive. In this location, the invariant $T \leq T\_start\_max$ acts as a time-bound on the waiting for the answer and forces a disconnection in case this bound is exceeded. Once the OffsetAnsw1 message arrives within the time-bound, the SAI_TTS_Init_Ini saves the last timestamp received from the peer entity and the timestamp at the reception of the message. Furthermore, it updates the last sequence number received from the peer entity writing on the `last_sn` variable, to keep track of the message stream of the communicating device and allow the detection of possible sequencing errors.

Finally, it saves in a local variable named `last_mtf` the type of the message received inside the procedure's message exchange. Indeed, these five messages must arrive in the sequential order we previously described, and if an error is found, a disconnect indication is forwarded to the SAI_User. Then, the SAI_TTS_Init_Ini computes the minimum and maximum offset

estimations, sends the OffsetAnsw2 message to the Responder device and again starts the timer to check the answer delivery time.

If the OffsetEst message arrives before the expiration of the timer, thanks to the offsets information that the peer entity has sent in the `user_data` field of the message, the SAI_TTS_Init_Ini checks if the safe connection with the Responder device can be established.

The structure of the `user_data` field of the message is as follows:

typedef struct {
    int min_offset;
    int max_offset;
    int check_field;
} *data_t*;

where the `min_offset` and the `max_offset` fields are used by the SAI_TTS_Init_Res template to communicate to the SAI_TTS_Init_Ini template its offsets estimations, and the `check_field` is used by the SAI_TTS_Init_Ini to communicate the offset check result to the SAI_TTS_Init_Res. The `user_data` field of the messages exchanged between the two devices is used only for these two communications, because the content of the exchanged Application Messages is abstracted away, and therefore the `user_data` field remains unused.

The SAI_TTS_Init_Ini sends the OffsetEnd message with the result of the check written in the `check_field` of the `user_data` to inform the Responder device, and according to the obtained result, it enters either the `Connected` location if the offset check was successful or the `Disconnected` location in case of failure.

If the Initiator SAI_User sends another `SAI_CONNECT_request` while the Initiator SAI module is performing the TTS initialization procedure, the ConnectionFailure location of the SAI_TTS_Init_Ini template is reached.

The SAI_TTS_Init_Ini counterpart of the Responder device is implemented in the SAI_TTS_Init_Res template, shown in Figure 3.17. Like the SAI_TTS_Init_Ini, also the SAI_TTS_Init_Res template defines the the `last_mtf` local variable to keep track of the received message ordering and starts the `T_start_max` timer at a message transmission to check the response times.

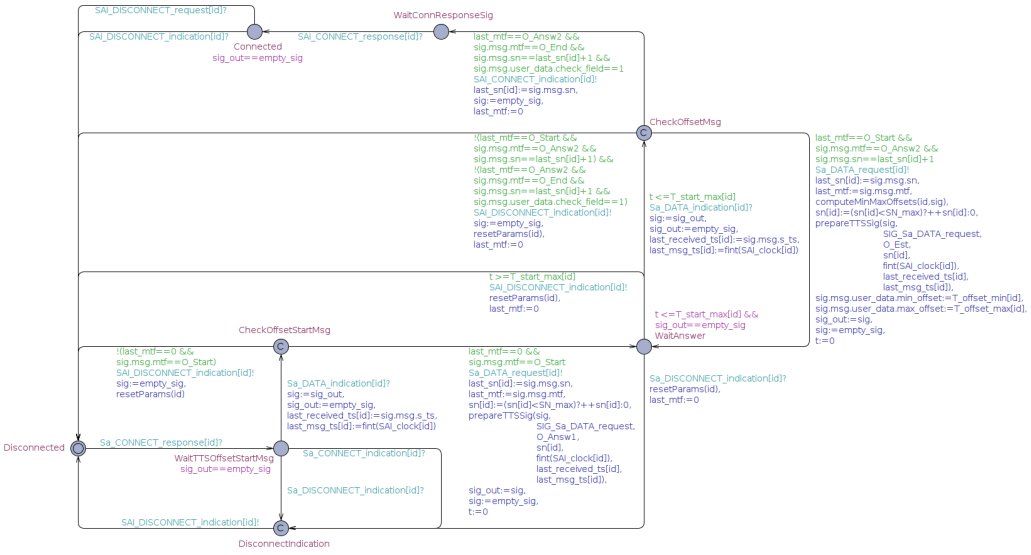

Figure 3.17: The SAI_TTS_Init_Res template

After the `Sa_CONNECT_response` communication, the SAI_TTS_Init_Res waits for the OffsetStart message from the Initiator SAI module to arrive. If instead it receives a new connect indication, the procedure is stopped and a disconnect indication is sent to its adjacent SAI User module.

At the OffsetStart message reception, the SAI_TTS_Init_Res updates the `last_received_ts` and the `last_msg_ts` variables and responds to the peer entity with the OffsetAnsw1 message. If the OffsetAnsw2 message arrives before the expiration of the timer, the SAI_TTS_Init_Res computes the minimum and the maximum offsets estimations and sends them to the Initiator SAI module within the OffsetEst message. With the reception of the OffsetEnd message, the SAI_TTS_Init_Res can send a `SAI_CONNECT_indication` to the SAI_User, if the offset check was successful, and then enter the `Connected` location after the `SAI_CONNECT_response` receive-action. Instead, in case of a failure, it sends the `SAI_DISCONNECT_indication` to the SAI_User and enters the `Disconnected` location.

Notice that in both the SAI_TTS_Init_Ini and the SAI_TTS_Init_Res templates, a disconnect indication to the SAI_User is sent as soon as an error in the message exchange procedure occurs. The connection procedure and the five message exchange for the TTS initialization must be executed without mistakes. Moreover, both these templates handle the `Sa_DISCONNECT_indication` receive-action coming from the Euroradio_SL_Env when a failure of the Euroradio SL or the lower layers occurs during the TTS initialization. If this happens, the two templates that implement the TTS initialization module forward the communication to their respective SAI_User.

Once that the two SAI modules are connected, they behave the same way: both the Initiator and the Responder SAI must send and receive the Application Messages, check if communication errors have occurred and need to execute the offsets update according to a certain time interval. For this reason, the next templates that model the TTS update procedure, the send-

ing and the receiving of Application Messages are implemented in a single parametric version that both the communicating devices need to instantiate in the system definition.

As mentioned above, the following templates use the `id` parameter to identify the device they belong to between the Initiator and the Responder devices and the `initiator` parameter as a Boolean flag to improve the readability of the model when entering the `Connected` location. As seen before, the templates relative to the Initiator SAI module move to the `Connected` location according to the `SAI_CONNECT_indication` receive-action, while the Responder SAI module templates become connected according to the `SAI_CONNECT_response` receive-action.

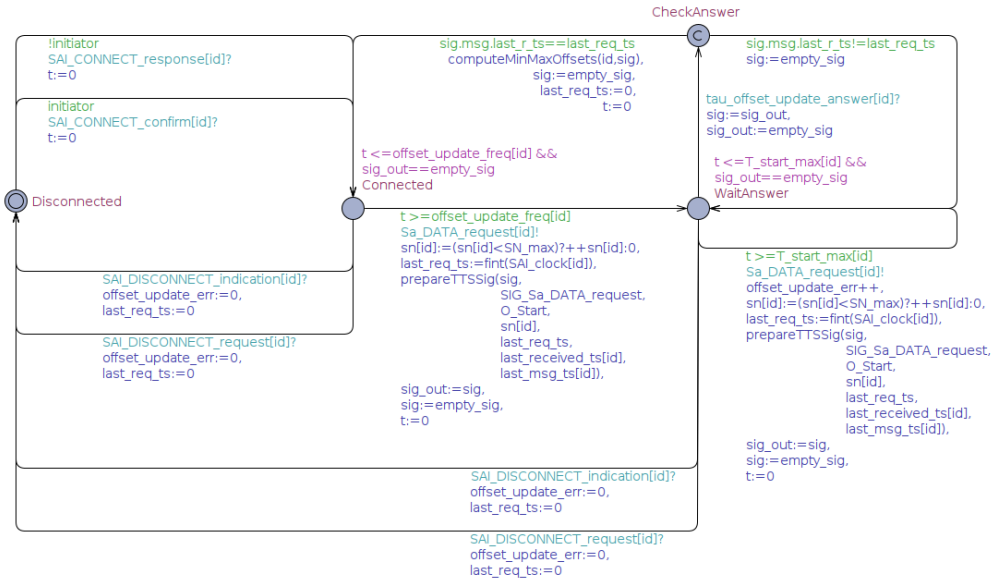

Figure 3.18: The SAI_Update_Req

Starting from the TTS offset update procedure, the SAI_Update_Req template shown in Figure 3.18 is responsible to regularly send an offset update request to the partner device. Both the Initiator and the Responder

devices can send an update request message to their partner device, so from now on the distinction between the Initiator device and the Responder device is no longer needed and no longer considered. Instead, the distinction that we take into account is related to the role of the device in the message exchange: the device that sends the message is referred to as the *Sender* device, while the device that receives the message is referred to as the *Receiver* device.

The time interval that governs the sending of the request is the `offset_update_freq` global variable. When the SAI_Update_Req sends an offset request, it writes the timestamp of the message transmission in the local variable `last_req_ts` to recognize if an incoming offset update answer is related to its last sent request.

At the request sending, the SAI_Update_Req starts a timer to check if the time needed to receive the answer is acceptable. If the timer exceeds the `T_start_max` value, then a new request is sent to the partner device, and an error counter is incremented. Instead, if an answer arrives before the timer expiration, if the content of the `last_receiver_timestamp` field of the message is the same as the `last_req_ts`, then the minimum and the maximum offsets estimation are updated and the timer for the next update request is reset. Anyway, if the `last_req_ts` check fails, the received answer is ignored and the SAI_Update_Req keeps waiting for the right answer to arrive before the timer expiration.

When an offset update request arrives at the Receiver device, it has to answer with an offset update answer message to allow the Sender device of the update request message to complete the offset update procedure. The Sai_Update_Answ template shown in Figure 3.19 is the SAI module responsible to send the TTS offset update answer message when a device receives a TTS

offset update request message from the partner device. When a tau_offset_-update_request arrives, an offset update answer is immediately sent back.

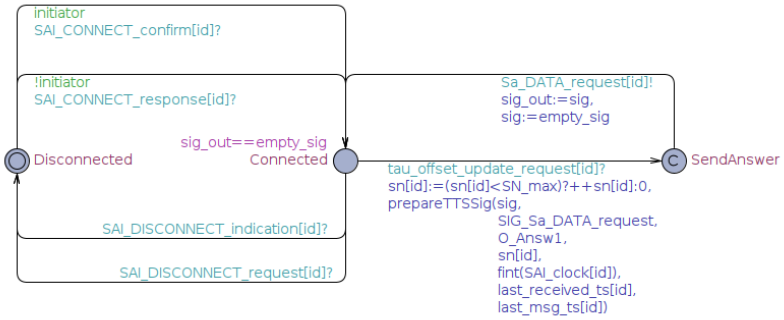

Figure 3.19: The SAI_Update_Answ

To distinguish the offset update request and the offset update answer messages, we chose to use the same message type fields of the OffsetStart and OffsetAnsw1 messages of the TTS initialization procedure. Both the `tau_offset_update_answer` and the `tau_offset_update_request` actions that communicate the receiving of those offset update messages are forwarded by the SAI_Receiver template, which is responsible for the receiving and the verification of the incoming messages and will be presented shortly below.

The last two templates that compose the SAI defense techniques module are the SAI_Sender and the SAI_Receiver templates. As we noticed for the TTS offset update procedure templates, also these two templates are instantiated for both the Initiator and the Responder devices, so this distinction is replaced by another differentiation that concerns the role of the device during the message exchange, such as the Sender device and the Receiver device.

The SAI_Sender template shown in Figure 3.20 is the SAI module responsible for sending the messages to the peer entity. Once that the safe

connection is established, the SAI_Sender synchronizes with both the SAI_-DATA_request and the SAI_DISCONNECT_request channels used by its adjacent SAI_User to command a particular message sending.

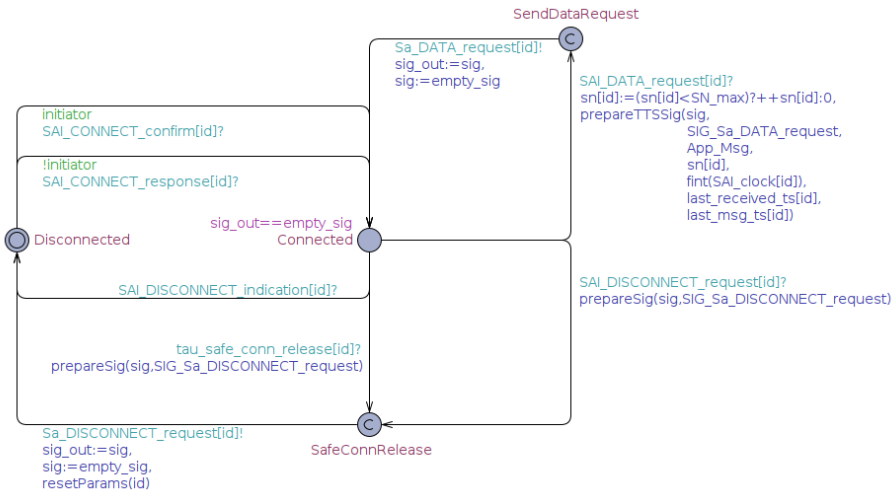

Figure 3.20: The SAI_Sender template

When a new Application Message is requested, the SAI_Sender of the Sender device prepares the signal to be forwarded to its adjacent Euroradio_-SL_Env, which in turn enqueues the signal inside the Receiver's signal queue. The signal is composed of the signal id corresponding to the `Sa_DATA_-request`, the message type field corresponding to the Application Message, the sequence number of the Sender message stream and the TTS fields. The sender timestamp is sampled from the `SAI_clock` of the Sender device, while the last timestamp received from the other SAI entity and the timestamp at the last message reception are taken from their respective variables. The sequence number is incremented by one at each new message transmission until the maximum allowable value is reached. When this happens, the next transmission is set to 0.

The `SAI_DISCONNECT_request` communication results in a `Sa_DISCONNECT_request` to be sent to the peer entity; the same disconnect request is also forwarded in case of the reception of a `tau_safe_conn_release` by the SAI_Receiver template, which commands the release of the connection due to the detection of unacceptable communication errors.

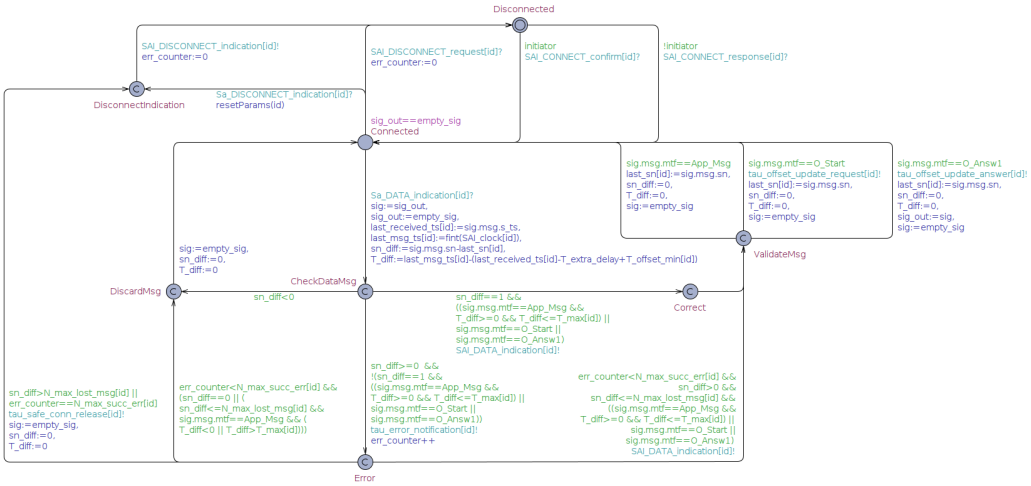

Figure 3.21: The SAI_Receiver template

In Figure 3.21, the SAI_Receiver template is shown. This template implements the protection against the repetition, deletion, re-sequencing and delay threats that can occur in a transmission system. When a `Sa_DATA_indication` is received, the `last_received_ts` and `last_msg_ts` variables are updated. Then, the sequence number difference (referred to as `sn_diff`) between the sequence number received in the message and the last sequence number stored in the `last_sn` variable is computed. Also the freshness of the received message (referred to as `T_diff`) is computed as the difference between the timestamp at the message reception (the `last_msg_ts` value just updated) and the estimation of the message time transmission in term of the SAI_Receiver clock, i.e. the sum of the last received timestamp (the

`last_received_ts` value just updated) and the minimum offset estimation computed by the Receiver device during the TTS initialization, with the extra delay subtracted.

The `sn_diff` and `T_diff` computed values determine the SAI_Receiver behavior:

- If $sn\_diff == 1$ and $0 \leq T\_diff \leq T\_max$ for an Application Message, the received data message is not affected by sequencing errors and the transmission delay is acceptable. The `T_max` configurable global variable defines the maximum validity time of the incoming data message. For the offset update request and answer messages, only the `sn_diff` is considered, as these messages are needed to update the offset estimations with respect to the current conditions of the communication system, and a distinct timer checks the time validity of the update procedure when it is running.
  When these conditions apply, the SAI_Receiver notifies its adjacent SAI_User of the correct message reception through the `SAI_DATA_indication`, moves first to the `Correct` location and then to the `ValidateMsg` location. Here, according to the message type field of the received message, the SAI_Receiver can forward a `tau_offset_update_request` or a `tau_offset_update_answer` send-action to the SAI_Update_Answ or the SAI_Update_Req templates respectively when concerning offset update request or offset update answer messages. Otherwise, in case of an Application Message, the `Connected` location is entered without further actions.

- If $sn\_diff < 0$, the message is discarded and the SAI_User is not notified;

- If $sn\_diff \geq 0$ and the previous conditions are not verified (either the $sn\_diff \neq 1$ or the transmission delay is not acceptable, i.e. $T\_diff > T\_max$), the SAI_Receiver notifies the SAI_User through the `tau_error_notification`, updates its error counter and then enters the `Error` location. Here, if the maximum number of either successive errors or lost messages is reached, the SAI_Receiver immediately sends a `tau_safe_conn_release` to the SAI_Sender, which in turn sends a disconnect request to the peer entity, and a `SAI_DISCONNECT_indication` to the SAI_User to command the release of the connection. Instead, if the maximum number of successive errors is not reached and the received message is either a repetition of the last accepted message ($sn\_diff == 0$) or its transmission delay is not acceptable, the message is discarded. Anyway, the message can be validated if both its transmission delay and the number of lost messages are acceptable ($0 \leq T\_diff \leq T\_max$ and $1 < sn\_diff \leq N\_max\_lost\_msg$).

Moreover, the SAI_Receiver synchronizes with the `Sa_DISCONNECT_indication` channel from the Euroradio_SL_Env. When a disconnect indication is received, the SAI_Receiver forwards the communication to the SAI_User before entering the `Disconnected` location and also all the other SAI templates of the same device that synchronize with the `SAI_DISCONNECT_indication` channel move to the `Disconnected` locations.

Before continuing, we refer that one aspect of UPPAAL SMC toolbox modeling language limited our implementation of the system. Indeed, when defining the structure of a message, we declared as integer values the three TTS fields defining the sender timestamp, the last received timestamp from

the partner device and the timestamp at last message reception. This was necessary because the record types specified through the `struct` keyword in UPPAAL cannot contain members of type `double`. Therefore, to set the values of the TTS fields, we need to convert the sampling of the clock variables to integers.

3.2.5 Fault_Injector

To model all the possible threats that can affect the communication system, we decided to introduce the Fault_Injector template shown in Figure 3.22. This template acts as a fault injector in the signal queue of the two communicating devices, determining the occurrence of communication errors or simulating transmission problems that could lead to unacceptable delays.

In order to be able to handle the probability of fault injection occurrence, the Fault_Injector template provides a minimum waiting time to perform an attempt. In addition, two probabilistic branches decide whether the attempt to perform a fault injection is successful or not. By fine-tuning the `fault` and `noFault` weights associated with the probabilistic branches, the probability of running the fault injector can be adjusted according to the desired fault occurrences.

Only if the signal queue of the non-deterministically chosen device to perform the fault injection is not empty, all the devices are connected and the attempt is successful, a probabilistic branching decides which threat to perform:

- **Deletion threat**

  The first signal of the queue is removed;

- **Repetition threat**

  The first signal is repeated if the queue is not full, otherwise no repetition is performed;

- **Re-sequencing threat**

  The first signal is shifted of one or two positions inside the queue, if at least another signal is present, otherwise no re-sequencing is performed;

- **Transmission delay threat**

  The `msgDelay` variable of the selected device is updated with the `msgDelayInjected` value.

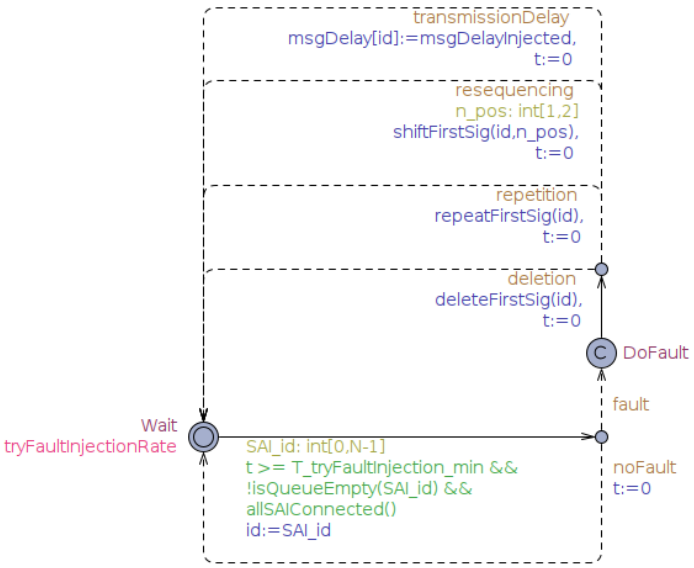

Figure 3.22: The Fault_Injector template

The transmission delay is based on the update of the `msgDelay` variable, which defines the rate for the exponential distribution of the edge performing the signal dequeue. The injected rate is lower than the standard rate

assigned to `msgDelay`, and during the generation of the exact delay with the exponential distribution, the smaller the rate is specified, the longer the delay is preferred. Hence, with the injected rate it is possible to simulate a longer transmission delay, increasing the probability to exceed the validity time for the incoming messages.

This longer transmission delay may contribute to accumulating messages within the signal queue, which is desirable for achieving the re-sequencing threat performed by the Fault Injector, but if excessive it can fill the buffers before the connection is released. Anyway, when the transmission delay threat is performed, with a high probability the connection between the two devices is soon released due to the detection of one of the communication errors seen before. And for this reason, we decided to restore the device `msgDelay` rate at the same time as the safe connection release, thus having the standard value again when starting a new connection.

Chapter 4

Verification and analysis results

In this fourth chapter of the thesis, we describe how the templates are instantiated to compose the UPPAAL system definition, and the results obtained through the formal verification of the properties derived from the requirements that the system should meet.

In particular, we investigate the safety properties related to the protection against the CENELEC threats, but also some modeling aspects of the system are analyzed. Moreover, because most of the requirements of the Subset 098 [33] are traceable to the templates of the model, we provide a table for the mapping, together with some considerations about the modeling choices we made with respect to the specification. Finally, the data about the effort needed to model and verify the system are provided.

4.1 System definition and configuration

According to the specification, each RBC shall be able to handle more than one handover transaction simultaneously, but only one RBC/RBC com-

munication between a pair of RBCs must be active at one time (ref. 4.2.1.2 of Subset 039 [31]). For this reason, we chose to consider the communication between a pair of RBCs as a separate process that each of the RBCs involved handles as an independent thread. So, assuming there is no interaction between independent threads, there are also no concurrency issues between them. This choice allows us to model a single pair of communicating devices, focusing on the verification of the protection techniques in case of communication error occurrences.

The system we model and verify through statistical model checking is defined as two communicating devices, an Initiator and a Responder RBCs, that establish a connection and exchange Application Messages using the interface defined in Subset 098 until a connection release occurs. The connection release can be requested when the communication has reached the end, or can result from a failure of either the transmission system or the Euroradio SL, or can result from the detection of communication errors. After the disconnection, the two communicating devices can establish a new safe connection.

Each communicating device is modeled with multiple UPPAAL processes that interact in parallel to simulate the desired behavior.

We defined two constant integer values as the fixed ids for the devices and a Boolean constant that identifies the Initiator device:

```
const int ID_SAI_INITIATOR = 0;
const int ID_SAI_RESPONDER = 1;
const bool Initiator = true;
```

The following processes are the template instantiations for the Initiator device:

```
user_ini = SAI_User(ID_SAI_INITIATOR,Initiator);
sai_conn_ini = SAI_Conn_Ini(ID_SAI_INITIATOR);
sai_tts_init_ini = SAI_TTS_Init_Ini(ID_SAI_INITIATOR);
sai_sender_ini = SAI_Sender(ID_SAI_INITIATOR,Initiator);
sai_receiver_ini = SAI_Receiver(ID_SAI_INITIATOR,Initiator);
sai_update_req_ini = SAI_Update_Req(ID_SAI_INITIATOR,Initiator);
sai_update_answ_ini = SAI_Update_Answ(ID_SAI_INITIATOR,Initiator);
euroradio_env_ini = Euroradio_SL_Env(ID_SAI_INITIATOR,ID_SAI_RESPONDER);
```

The following processes are the template instantiations for the Responder device:

```
user_res = SAI_User(ID_SAI_RESPONDER,!Initiator);
sai_conn_res = SAI_Conn_Res(ID_SAI_RESPONDER);
sai_tts_init_res = SAI_TTS_Init_Res(ID_SAI_RESPONDER);
sai_sender_res = SAI_Sender(ID_SAI_RESPONDER,!Initiator);
sai_receiver_res = SAI_Receiver(ID_SAI_RESPONDER,!Initiator);
sai_update_req_res = SAI_Update_Req(ID_SAI_RESPONDER,!Initiator);
sai_update_answ_res = SAI_Update_Answ(ID_SAI_RESPONDER,!Initiator);
euroradio_env_res = Euroradio_SL_Env(ID_SAI_RESPONDER,ID_SAI_INITIATOR);
```

Moreover, the system is composed of one instantiation of the Communication_System template, which models the occurrence of transmission failures that lead to the disruptive connection release.

```
communication_system = Communication_System();
```

Finally, the Fault_Injector template is instantiated to introduce within the system the possibility of stochastically modeling all the CENELEC threats that affect a communication system.

```
fault_injector = Fault_Injector();
```

The UPPAAL system definition is the synchronous product of all the listed processes that are instantiated from the templates whose parameters, if any, are configured as described.

In Figure 4.1, a detailed graphical representation of the instantiated model architecture and the parameter configuration is shown. This figure shows which modules access the main global variables defined to model the message exchange procedure and to detect possible communication errors.

The channel synchronizations between the templates are represented at higher level to give a general overview of the interactions between the components of the system. Actually, all the templates inside the SAI module synchronize with each other and with the SAI User and Euroradio SL modules through broadcast channels and one-to-many communications.

In order to produce a model that is highly reusable and easily evaluable on different operational scenarios, we defined a series of constant parameters that need to be configured before the system can be simulated. For some of these parameters, the specification suggests which values to use with particular systems (for example highly-available systems), whereas the definition of other parameters is left to the specific application settings.

In Table 4.1, we listed all the constant parameters that define the system together with the configuration used to run our verification. These parameters are defined as Integer unless otherwise specified. Note that the `SN_max` parameter is set to 32767, which is the maximum integer value that can be assigned to an integer variable in UPPAAL (in the Subset 098 [33], the maximum value for the sequence number is 65535).

In Table 4.2, we show the weights of the probabilistic choices that determine the stochastic behavior of the system.

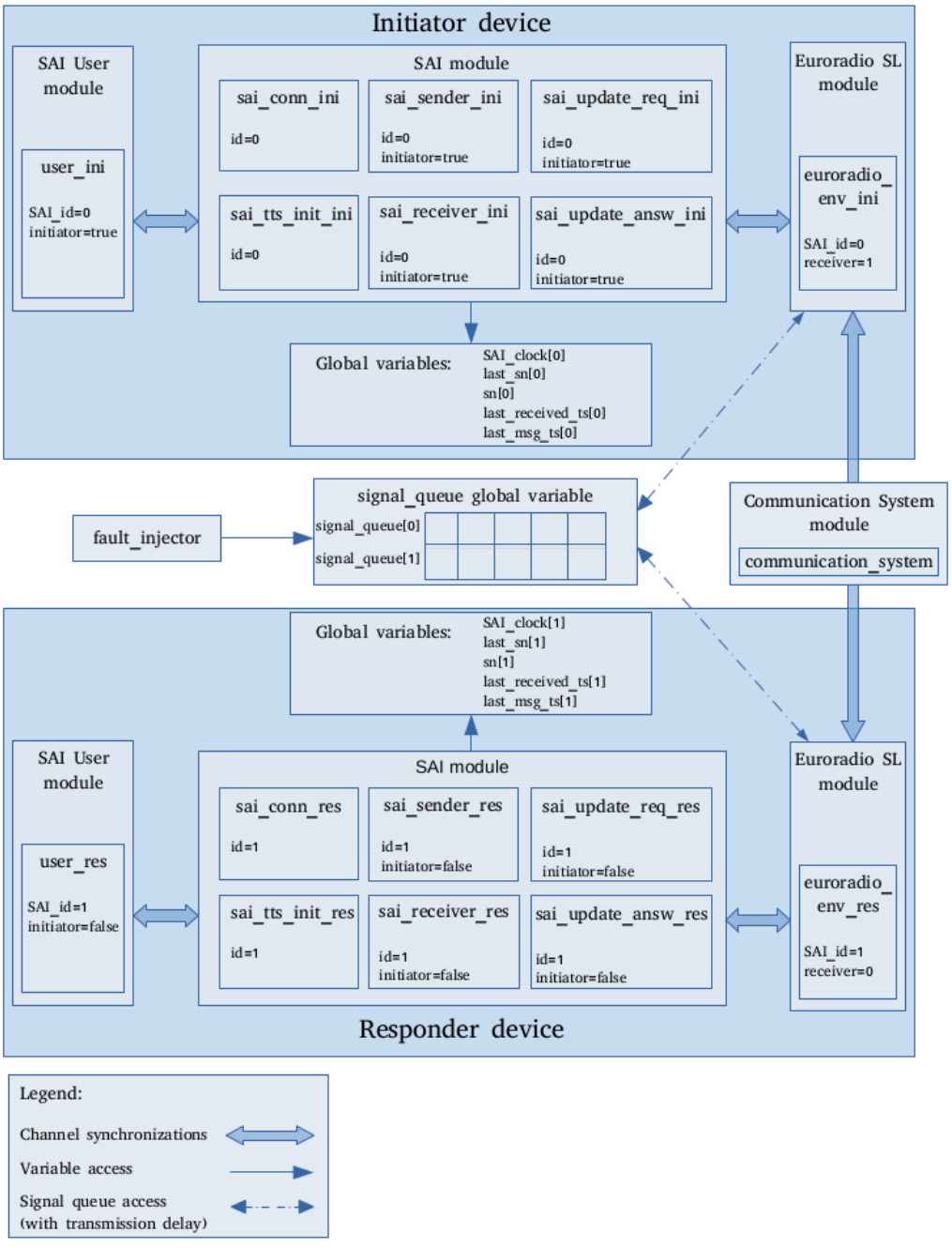

Figure 4.1: The model architecture

In Table 4.3, the exponential rates we specified to model the real-time aspects of the system are shown. Here, it is interesting to notice how the

injected rate for the message delay to model the transmission delay threat is defined as $\frac{2}{5}$ of the standard message delay rate, thus augmenting the probability to have a longer delay at the signal dequeuing from the signal queue.

Constant	**Description**	**Value**
SN_max	sequence number max value	32767
queue_size	size of signal queue	15
N_max_lost_msg	number of max lost messages + 1	3
N_max_succ_err	number of max successive errors + 1	2
msg_freq	SAI_User frequency of message exchange	8
T_reply_max	SAI_User max time value without reply	3*msg_freq
T_start_max	max answering time value in TTS init	2
T_conn_max	max time value to get the connect confirm	4*T_start_max
T_off_max	max difference between offset estimations	3
T_max	max validity time of incoming message	5
offset_update_freq	frequency for offset update procedure	250
T_extra_delay	signal processing time	0
T_commSystemCheck_min	CommunicationSystem min time value to check connection	250
T_tryFaultInjection_min	FaultInjector min time value to try fault injection	10
s_time (double)	SAI_clock starting time	0.0
temporal_drift (double)	temporal drift between SAI_clock	0.0

Table 4.1: System constant parameters set-up

As it is possible to notice from the Table 4.2, the possibility of injecting a fault as a communication error in the message stream (see the Fault Injector in Chapter 3, Section 3.2.5) is determined stochastically with a 50% probability of injection success by means of equal weights for the two cases `fault` and `noFault`.

This choice of inflating the probability of occurrence of communication errors allows an efficient verification of the system when facing the threats identified by the CENELEC standard. Indeed, communication systems are

built to minimize the occurrence of communication errors, and for a more realistic modeling of our system, such probability should be low. But in this case the occurrence of a communication error would be a rare event and a very large number of simulation runs would be needed to spot it. Instead, by increasing its probability, we are able to inject these errors inside the system and thus we can measure the effectiveness of the implemented defense techniques.

Weight	**Description**	**Value**
sendDataMsg	SAI_User weight to send Application Messages	99
stopConnMsg	SAI_User weight to send disconnect requests	1
safeConn	CommunicationSystem weight for safe connection check	99
disruptiveConnRelease	CommunicationSystem weight for disruptive connection release	1
noEuroradioFailure	Euroradio_SL_Env weight for no failure occurrence	99
euroradioFailure	Euroradio_SL_Env weight for failure occurrence	1
noFault	FaultInjector weight for no fault injection	1
fault	FaultInjector weight for fault injection	1
transmissionDelay	FaultInjector weight for transmission delay threat	1
deletion	FaultInjector weight for deletion threat	1
resequencing	FaultInjector weight for resequencing threat	1
repetition	FaultInjector weight for repetition threat	1

Table 4.2: System constant weights set-up

Note that the purpose of our analysis does not concern the accurate quantification of measures such as performances or delays. Instead, we focus on the qualitative verification of the effectiveness of the defense techniques described in the specification. For this reason, rather than assigning realistic values to the parameters of our model as a quantitative analysis requires, we

Rate (double)	Description	Value
connRate	Initiator SAI_User connection rate	1.0
msgDelayStandard	standard message delay	2.0
msgDelayInjected	injected msgDelay for transmission delay threat	0.4*msgDelayStandard
commSystemCheckRate	CommunicationSystem check connection rate	1.0
tryFaultInjectionRate	FaultInjector try fault injection rate	1.0

Table 4.3: System constant rates set-up

opted to work with values that are functional for a fast and effective qualitative analysis. Hence, by increasing or decreasing certain parameters, we are able to handle the simulation scenarios according to the particular needs of the analysis.

Moreover, in a fully-specified model with respect to the SAI requirements can result difficult to perform qualitative verification through traditional methods, due to the required complete state space generation. This state space explosion problem makes the time needed to complete a verification impracticable. Therefore, a statistical approach can be useful to perform faster evaluations using a stochastic model of the system where the verification of different operating scenarios is allowed simply by fine-tuning the values of the parameter configuration.

In our case, we are interested in the evaluation of the defense technique protection against threats identified by the CENELEC standards. Hence, the configuration of certain parameters is done according to the need for a stable connection where the probability of the SAI_User to send a disconnection request, or the Communication_System to perform a disruptive connection release, or the Euroradio_SL_Env to fail, are very low. This is why we assign high values to the `sendDataMsg`, the `safeConn` and the `noEuroradioFailure` weights, as to favor the release of the connection from

the SAI module through the protection mechanisms.

Table 4.1, Table 4.2 and Table 4.3 collect the default values for the parameter configuration of the system. In almost all the simulations performed to verify the correctness of the model we refer to these default values, unless otherwise mentioned.

We remark that almost all of these parameters are defined as two-dimensional arrays to allow the differentiation of the values associated to each of the two devices. However, during our tests, the same values were used for both the devices, so only one configuration value is listed.

4.2 Model verification

In this section, we present the formal analysis of the system described in the previous section, showing each of the properties of interest aimed at verifying the absence of errors in the proposed defense techniques. All the properties are related to the probability estimation of the occurrence of a specific hazard. Each query derived from a property is evaluated through the UPPAAL SMC statistical model checker and its verification results are reported. If the probability of occurrence of the hazards is close to zero, hence we can say that, according to this qualitative verification, the model satisfies its safety requirements with a certain degree of confidence dictated by the parameters of the statistical model checker (see Table 4.4).

Notice that, when evaluating a property, the tool will not return the exact value of that property, but rather an interval obtained through simulations. The number of simulations is decided based on the statistical parameters of the model checker.

Statistical parameter	**Value**
Probability of false negatives (α)	**0.0005**
Probability uncertainty (ϵ)	**0.005**

Table 4.4: UPPAAL SMC configuration of statistical parameters

In Table 4.4 are reported the probability of false negatives α and the probability uncertainty ϵ (also referred to as u). Let p' be the probability that has been estimated through simulations and p be the unknown "real" probability whose value is approximated by p'. The tool executes a number of simulations sufficient to guarantee that $P_r(|p' - p| \leq \epsilon) \geq 1 - \alpha$, that is the interval $[p' - \epsilon, p' + \epsilon]$ contains p with probability greater than or equal to the confidence interval $1 - \alpha$.

Basically, reducing α or ϵ requires a greater number of simulations to guarantee the evaluated property.

All the experiments whose results we will show below are performed on a machine with Intel® Core(TM) $i7 - 8565U$ CPU at 1.80GHz x8 and 8 GB of RAM, and UPPAAL 4.1.24 academic version from 2019 is used.

The UPPAAL SMC simulation runs are configured with the statistical parameters shown in Table 4.4, unless otherwise mentioned. This statistical parameter set-up allows to reach confidence 0.9995 for the estimated probability of being in the probability interval provided by the UPPAAL verifier.

Moreover, each simulation has a bound on the time units to verify the system behavior. We set the `msg_freq` parameter to 8 time units, so we decided to bound all our simulations to 1000 time units, thus allowing to perform faster simulations but still inclusive of a non-negligible message ex-

change.

The formulas shown in the next sub-sections follow the notation of the UPPAAL SMC queries, but to improve their readability, we decided to report the names of the templates instead of the names associated with the instantiation of each process. For example, if a formula is verified for both the sai_sander_ini and the sai_sender_res, we simply write SAI_Sender.

4.2.1 Modeling aspects and sensitivity analysis

In this first sub-section, we see some properties that focus on some modeling aspects that the system should meet. The verification of these properties permits to have a feedback on the modeling of the system, showing if the chosen configuration is consistent and appropriate to verify the main aspects of the behavior of our system.

For these simulations, the UPPAAL SMC statistical parameters differ from those in Table 4.4 and both the probability of false negatives α and probability uncertainty ϵ have been set to 0.05 in order to perform faster analysis and give rough qualitative evaluations.

As previously underlined during the description of the model, the signal queue used to exchange messages between the two communicating devices has a fixed size. When the maximum number of elements is reached in the Receiver device queue, the Euroradio SL module of the Sender device does not enqueue the outgoing signal, assuming a global knowledge of the signal queue conditions, and the message is lost without failures. We also noticed that in conjunction with some of the threats we inject to verify the system, the two signal queues are more likely to fill up.

However, through this first formula, we show how the configuration of the system manages to keep the probability of signal queue filling low enough.

$$\varphi_1 = Pr\,[\leq 1000]\,(<>\,(exists(id : id\_t)\; isQueueFull(id)))$$

This formula measures the probability that at least one of the two signal queues is full within 1000 time units. `IsQueueFull()` is a global user function that returns `True` if the signal queue of the specified device is full, otherwise it returns `False`.

The evaluation of φ_1 takes 29 runs and the probability of such event to occur is in [0, 0.0981446] probability interval with 95% of confidence.

Notice that all the formulas we evaluate follow this precise form, that is the probability of reaching a specific configuration within 1000 time units. This aspect of having a sort of "template" for the definition of our formulas where only the logical conditions change makes the formalisation of the properties less complicated than it is commonly believed when thinking about the use of formal techniques. However, this type of evaluation only allows to check if a state of the system where a particular configuration (the one specified in the logical condition) holds is ever reached, and to be able to identify certain configurations we needed to add some extra locations to our model.

The next formula concerns the connection procedure and in particular the setting of the `T_conn_max` parameter we defined for the Initiator SAI_User template. This parameter is used for timing the new connection request to send to the Responder device if no answer to the previous request arrives.

In the subset specification, during the TTS initialization procedure, a `T_start_max` maximum waiting time for the incoming offset messages from the Responder device is provided. Hence, we opted to take the `T_conn_max` value in function of the `T_start_max` parameter, i.e. we chose `T_conn_max` = `c`∗`T_start_max`, where `c` is an integer constant.

Recall that in the SAI_TTS_Init_Ini template is present the ConnectionFailure location. The reason for this "extra" location is to assign a quantitative value to the `T_conn_max` parameter such that it is ensured that no other requests are sent by the SAI_User while the SAI module is still performing the TTS initialization procedure. This location is reached when the SAI_TTS_Init_Ini template receives a `SAI_CONNECT_request` from its adjacent SAI_User and the five offset message exchange is still in progress. Through the formal verification, it is possible to find the threshold value for the constant `c` beyond which the model degrades, i.e. there is a non-negligible probability to enter the `ConnectionFailure` location. In particular, a sensitivity analysis is performed to empirically compute the value for the constant `c`.

The φ_2 formula computes the probability of the SAI_Conn_Init_Ini template to reach the ConnectionFailure location within the time units bound.

$$\varphi_2 = Pr\,[\leq 1000]\,(<>\ \ (SAI\_TTS\_Init\_Ini.ConnectionFailure))$$

We evaluated φ_2 performing several experiments at the varying of the constant `c` and Figure 4.2 shows the maximum values of the probability intervals derived from the evaluation of φ_2. The degradation of the model occurs for `c` = 3. For this reason, we decided to set `T_conn_max` = 4∗`T_start_max` as the value for the default configuration.

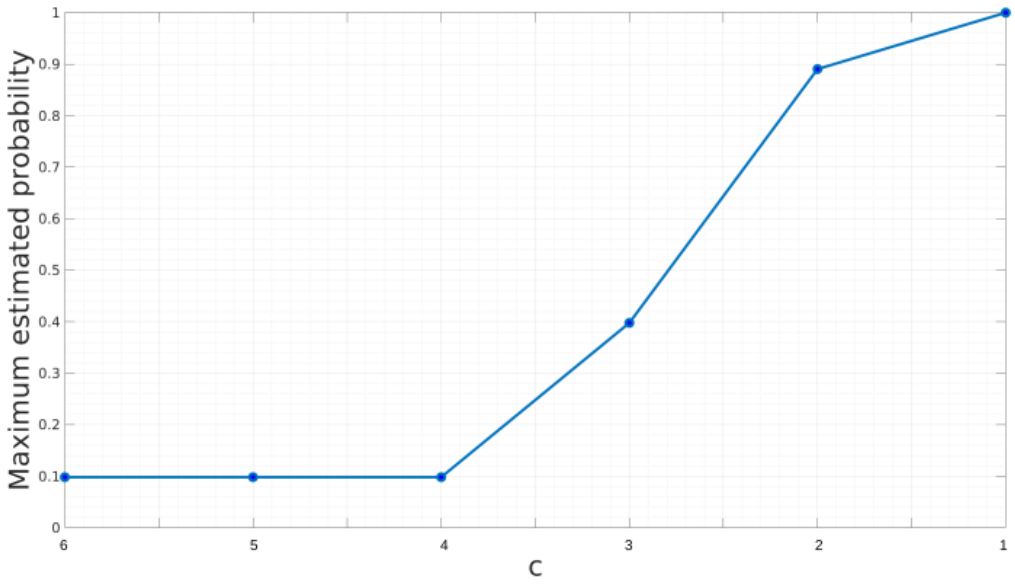

Figure 4.2: The trend for the maximum probability estimation of φ_2 as the constant `c` decreases

Finally, we verify that the Fault Injector we modeled to inject into the system the threats identified by the CENELEC standard actually performs the fault injections in the signal queue of the two communication devices. This happens when the Fault Injector template enters the `DoFault` location, so we use the statistical model checker to evaluate the following formula:

$$\varphi_3 = Pr\,[\leq 1000]\,(<>\ \ (Fault\_Injector.DoFault))$$

The evaluation of φ_3 takes 29 runs to bound with 95% of confidence the probability of a fault injection to occur in $[\,0.901855,\ 1\,]$ probability interval, thus meaning that at least one fault is always injected in each simulation of the system.

4.2.2 Safety properties

In this sub-section, we verify the safety properties related to the effectiveness of the defense techniques described in the specification and implemented in our model. We decided to focus on the probability estimation of some particular hazards to occur, showing that the probability of anything bad to happen is extremely low.

The hazards we focus on concern "bad" configurations of the system and have been identified by a manual review of the model based on the requirements. In this sub-section, we follow a schematic methodology where first the probability of occurrence for a particular threat, among those specified in the requirements, is evaluated, and then we evaluate the probability that the system does not behave as expected when that particular threat occurs. This second probability evaluation makes sense only if the first probability evaluation is non-negligible, thus meaning that the system is actually verified when the threat is "real", i.e. the system has a non-negligible probability to face with it.

Note that an analysis of the severity of the identified hazards, i.e. checking if the occurrence of a particular hazard decreases the performance of the system, is out of scope for this thesis.

The first two hazards we describe concern the probability of having false positives (φ_4) or false negatives (φ_5) because the system does not correctly handle the incoming messages. A false positive occurs when the system receives a correct message that it is considered as an erroneous message. Instead, a false negative occurs when a message affected by some communication errors is treated as a correct message by the system.

To measure the probability of, respectively, false positives and false negatives, the following φ_4 and φ_5 formulas are analysed:

$$\varphi_4 = Pr\,[\leq 1000]\,(<>\ \ (SAI\_Receiver.sn\_diff == 1\ \&\&$$

$$SAI\_Receiver.T\_diff \geq 0\ \&\&$$

$$SAI\_Receiver.T\_diff \leq T\_max\ \&\&$$

$$(SAI\_Receiver.Error\ ||\ SAI\_Receiver.DiscardMsg)))$$

$$\varphi_5 = Pr\,[\leq 1000]\,(<>\ \ (SAI\_Receiver.sig.msg.mtf == App\_Msg\ \&\&$$

$$(SAI\_Receiver.sn\_diff! = 1\ ||\ SAI\_Receiver.T\_diff < 0\ ||$$

$$SAI\_Receiver.T\_diff > T\_max)\ \&\&\ SAI\_Receiver.Correct))$$

The φ_4 formula checks if the SAI_Receiver of both the Initiator and the Responder devices receives a correct message, i.e. the sequence number difference with the previous message is 1 ($sn\_diff == 1$) and its delay is acceptable ($T\_diff \geq 0\ \&\&\ T\_diff \leq T\_max$), and the system treats it as an error, i.e. the SAI_Receiver template enters under this condition the `Error` location or the `DiscardMsg` location.

The φ_5 formula checks if the SAI_Receiver of both the Initiator and the Responder devices receives an erroneous Application Message, i.e. the sequence number difference with the previous message is not 1 ($sn\_diff! = 1$) or its delay is unacceptable ($T\_diff < 0\ ||\ T\_diff > T\_max$), and the system considers it as a correct message, i.e. the SAI_Receiver template enters the `Correct` location.

Now we are able to describe the importance of having modeled the location `Correct` in the SAI_Receiver template (see Chapter 3 Section 3.2.4). Indeed, this location is necessary to distinguish the correct messages from the messages that are validated despite the sequence number difference with the previous accepted message is different from 1, as specified by the Subset 098 [33]. The φ_5 formula could not have been written using the `ValidateMsg` location instead of the `Correct` location because an incoming erroneous message with $sn\_diff! = 1$ could still be accepted (if the sequence number gap is tolerable) and the SAI_Receiver template would enter the `ValidateMsg` location anyway. Hence, a distinction was necessary to perform this particular query.

Concerning the threats that can occur in a communication system, we identified six possible hazards:

1. A message earlier than the last accepted one is not discarded.

   This hazard is caused by the injection of the re-sequencing threat. To verify that some re-sequencing injections actually occur in the system, we check what is the probability of receiving a message whose sequence number difference with the last accepted message is negative ($sn\_diff < 0$). If the re-sequencing threat is injected in the system, the probability computed with the following formula should be non-negligible:

   $$\varphi_6 = Pr\,[\leq 1000]\,(<>\ \ (SAI\_Receiver.sn\_diff < 0\ ))$$

   Instead, using the formula φ_7, we check the probability for the SAI_-Receiver template to enter either the Error, the Correct or the Vali-

dateMsg locations, if $sn\_diff < 0$ holds. This probability must be close to zero, thus meaning that it is unlikely for this unsafe scenario to occur.

$$\varphi_7 = Pr\,[\leq 1000]\,(<>\ \ (SAI\_Receiver.sn\_diff < 0\ \&\&$$
$$(SAI\_Receiver.Error\ ||\ SAI\_Receiver.Correct\ ||$$
$$SAI\_Receiver.ValidateMsg)))$$

2. A message with the same sequence number of the last accepted one is validated.

   This hazard is caused by the injection of the repetition threat in the system. With the φ_8 formula, we check that this kind of threat is actually injected in the system, and then we use the φ_9 formula to verify that all possible unsafe configurations have a low probability to occur:

$$\varphi_8 = Pr\,[\leq 1000]\,(<>\ \ (SAI\_Receiver.sig\,!= empty\_sig\ \&\&$$
$$SAI\_Receiver.sn\_diff == 0\ ))$$

$$\varphi_9 = Pr\,[\leq 1000]\,(<>\ \ (SAI\_Receiver.sig\,!= empty\_sig\ \&\&$$
$$SAI\_Receiver.sn\_diff == 0\ \&\&$$
$$(SAI\_Receiver.Correct\ ||\ SAI\_Receiver.ValidateMsg)))$$

In particular, φ_8 computes the probability for the SAI_Receiver of receiving an incoming message ($sig! = empty\_sig$) and the sequence number difference with the last accepted message is null ($sn\_diff == 0$). In φ_9 we verify that the unsafe scenario where the SAI_Receiver enters the `ValidateMsg` or the `Correct` locations never occurs.

3. A message arrived with both an acceptable delay and with a sequence number difference between 2 and `N_max_lost_msg` with respect to the last accepted message. The message is discarded or accepted without being notified as an erroneous message.

   This hazard is due to the deletion threat and can also be caused indirectly by the re-sequencing threat. We check that the occurrences of incoming messages are actually consistent (they occur with non-negligible probability), where the computed sequence number difference with the last accepted message is between 2 and `N_max_lost_msg` and their delay is acceptable. This check is done with the following formula:

$$\varphi_{10} = Pr\,[\leq 1000]\,(<>\ \ (SAI\_Receiver.sn\_diff > 1\ \&\&$$

$$SAI\_Receiver.sn\_diff \leq N\_max\_lost\_msg\ \&\&$$

$$SAI\_Receiver.T\_diff \geq 0\ \&\&\ SAI\_Receiver.T\_diff \leq T\_max))$$

   Then we verify that under this condition the system never enters the unsafe scenarios where then message is considered correct or the message is discarded (respectively, the SAI_Receiver enters the `Correct` or the `DiscardMsg` locations) through the following formula:

$$\varphi_{11} = Pr\,[\leq 1000]\,(<>\ \ (SAI\_Receiver.sn\_diff > 1\ \&\&$$

$$SAI\_Receiver.sn\_diff \leq N\_max\_lost\_msg\ \&\&$$

$$SAI\_Receiver.T\_diff \geq 0\ \&\&$$

$$SAI\_Receiver.T\_diff \leq T\_max\ \&\&$$

$$(SAI\_Receiver.Correct\ ||\ SAI\_Receiver.DiscardMsg)))$$

4. A message with both an unacceptable transmission delay and a sequence number difference between 2 and `N_max_lost_msg` included with respect to the last accepted message is validated.

   This hazard is similar to the previous one, but in this case the transmission delay is not tolerable, and hence it can also be caused in combination with the transmission delay threat. In particular, we report in order the two formulas used to verify the occurrence of unacceptable delayed messages with a tolerable sequence number difference and to check the probability that the system enters unsafe scenarios by considering these messages correct and validate them:

$$\varphi_{12} = Pr\,[\leq 1000]\,(<>$$

$$(SAI\_Receiver.sig.msg.mtf == App\_Msg\ \&\&$$

$$SAI\_Receiver.sn\_diff > 1\ \&\&$$

$$SAI\_Receiver.sn\_diff \leq N\_max\_lost\_msg\ \&\&$$

$$(SAI\_Receiver.T\_diff < 0\ ||$$

$$SAI\_Receiver.T\_diff > T\_max)))$$

$$\varphi_{13} = Pr\,[\leq 1000]\,(<>$$

$$(SAI\_Receiver.sig.msg.mtf == App\_Msg\ \&\&$$

$$SAI\_Receiver.sn\_diff > 1\ \&\&$$

$$SAI\_Receiver.sn\_diff \leq N\_max\_lost\_msg\ \&\&$$

$$(SAI\_Receiver.T\_diff < 0\ ||$$

$$SAI\_Receiver.T\_diff > T\_max)\ \&\&$$

$$(SAI\_Receiver.Correct\ ||\ SAI\_Receiver.ValidateMsg)))$$

Notice that both φ_{12} and φ_{13} focus on the Application Messages; indeed, for the offset update request and answer messages only the `sn_diff` is considered whilst their transmission delay is checked with a separate timer, and the SAI_Receiver template validates them without checking their `T_diff`.

5. A message with a sequence number difference greater than `N_max_lost_msg` with respect to the last accepted message is discarded or validated.

   This hazard is caused by the deletion threat. To verify the behavior of the system in conjunction with this type of hazard, we needed to modify some of the default parameters in order to have a non-negligible probability to observe the occurrence of such hazard. For this reason, the next two formulas we present have been simulated with `N_max_lost_msg` = 2 (i.e. the system can tolerate only 1 message lost and when 2 or more messages are lost, the connection release is requested)

and `deletion` weight = 10, as to increase the probability of seeing this risk.

$$\varphi_{14} = Pr\,[\leq 1000]\,(<>$$
$$(SAI\_Receiver.sn\_diff > N\_max\_lost\_msg))$$

$$\varphi_{15} = Pr\,[\leq 1000]\,(<>$$
$$(SAI\_Receiver.sn\_diff > N\_max\_lost\_msg\ \&\&$$
$$(SAI\_Receiver.Correct\ ||\ SAI\_Receiver.ValidatedMsg\ ||$$
$$SAI\_Receiver.DiscardMsg)))$$

Through φ_{15}, we can check that the SAI_Receiver does not enter neither the `Correct`, nor the `ValidateMsg`, nor the `DiscardMsg` locations, behaviors that could lead the system to unsafe scenarios.

6. A message with the sequence number difference with respect to the last accepted message equal to 1 but with an unacceptable delay is treated as a correct message and validated.

   This hazard can be caused by the transmission delay threat. First we check if messages within the correct message stream of the partner device (no sequencing errors are detected) but with non tolerable transmission delays have a consistent probability of occurring through the following formula:

$$\varphi_{16} = Pr\,[\leq 1000]$$

$$(<>\ \ (SAI\_Receiver.sig.msg.mtf == App\_Msg\ \&\&$$

$$SAI\_Receiver.sn\_diff == 1\ \&\&$$

$$(SAI\_Receiver.T\_diff < 0\ ||$$

$$SAI\_Receiver.T\_diff > T\_max)))$$

We think it is also interesting to verify if the resulting probability of unacceptable transmission delay occurrences is mainly caused by the transmission delay threat injection, and that consequently it is unlikely to detect a non-tolerable transmission delay with the standard configuration for the msgDelay rate. For this reason, we perform another simulation of φ_{16}, where the `transmissionDelay` weight is set to 0, so cancelling the possibility for the Fault Injector to perform a transmission delay threat, thus verifying if there are non-negligible occurrences of messages with a non-tolerable transmission delay even with the standard `msgDelay` rate. We refer to this evaluation of φ_{16} formula where the `transmissionDelay` weight is set to 0 as φ'_{16}.

Then, getting back to the default parameter configuration, we verify that the system's behavior when facing incoming messages with unacceptable delays is not wrong, leading the system to an unsafe scenario. This could happen if the SAI_Receiver enters the `Correct` or the `ValidateMsg` locations, as stated by the next formula:

$$\varphi_{17} = Pr\,[\leq 1000]\ (<>$$

$$(SAI\_Receiver.sig.msg.mtf == App\_Msg\ \&\&$$

$$SAI\_Receiver.sn\_diff == 1 \;\&\&$$

$$(SAI\_Receiver.T\_diff < 0 \;||$$

$$SAI\_Receiver.T\_diff > T\_max) \;\&\&$$

$$(SAI\_Receiver.Correct \;||\; SAI\_Receiver.ValidateMsg)))$$

Both φ_{16} and φ_{17} focus on the Application Messages because the offset update request and answer messages must be validated even if their transmission delay is unacceptable.

The last three safety properties address the sending of Application Messages and the release of the connection when the maximum number of successive errors is reached.

First, we verify that the SAI module receives any Application Message from neither the SAI User nor the Euroradio SL modules if the clock offset estimation (i.e. the TTS initialization) has not been completed yet. The corresponding formula φ_{18} checks if either the SAI User enters the `SendMsg` location or the SAI module (in particular the SAI_TTS_Init and the SAI_-Receiver templates) receives an Application Message from the Euroradio SL module (we check the message type fields of their incoming messages) when the TTS initialization is running:

$$\varphi_{18} = Pr\,[\leq 1000]\,(<>\;\; (isTTSInitializing \;\&\&$$

$$(SAI\_User.SendMsg \;||$$

$$SAI\_TTS\_Init\_Ini.sig.msg.mtf == App\_Msg \;||$$

$$SAI\_TTS\_Init\_Res.sig.msg.mtf == App\_Msg \;||$$

$$SAI\_Receiver.sig.msg.mtf == App\_Msg)))$$

Next, we check that the SAI module forwards to the SAI User module only those messages that have been accepted, thus not affected by communication errors or with tolerable sequencing errors.

Recall the SAI_User template in Chapter 3, Section 3.2.2 and its `MsgIndication` location. This location is used to detect the action `SAI_DATA_indication` from the SAI_Receiver template. Indeed, the UPPAAL logic does not allow to predicate on actions but only on locations, so to detect if the SAI_User template receives the `SAI_DATA_indication`, we need to add the intermediate committed `MsgIndication` location. Without this location, we could not be able to verify the correct reception of the signal by the SAI_User template.

The following formula checks if the SAI_User enters the `MsgIndication` location even if the SAI_Receiver has not validated the incoming message:

$$\varphi_{19} = Pr\,[\leq 1000]\,(<>\ \ (SAI\_User.MsgIndication\ \&\&$$
$$SAI\_Receiver.sig\,!= empty\_sig\ \&\&$$
$$!(SAI\_Receiver.Correct\ ||$$
$$SAI\_Receiver.ValidateMsg)))$$

Finally, the last property we verify checks if the SAI module correctly commands the release of the connection when reaching the maximum number of successive errors during the message exchange with the partner device. The following formula verifies whether the SAI_Receiver is located neither in the `DisconnectIndication` nor in the `Error` locations when the number

of successive errors has reached the maximum number allowed, thus meaning that the safe connection release has not been requested (i.e. an unsafe scenario):

$$\varphi_{20} = Pr\,[\leq 1000]\,(<>$$

$$(SAI\_Receiver.err\_counter == N\_max\_succ\_err\ \&\&$$

$$!\,(SAI\_Receiver.DisconnectIndication\ ||\ SAI\_Receiver.Error)))$$

These last three formulas are evaluated in scenarios where the system can reach unsafe configurations due to the fault injection of communication errors.

In Table 4.5, we present the summary of the results and the performances obtained by verifying through the statistical model checker all the safety properties described in this sub-section. The statistical parameters used for these simulations are those specified in 4.4, apart from the rows marked with the $*$ symbol, which have been evaluated with both the probability of false negatives α and the probability uncertainty ϵ set to 0.05. Indeed, each $*$ row concerns a formula where the probability evaluation of specific communication threats to be injected into the system is checked, and then a faster and rough evaluation is enough to show that this probability of occurrence is non-negligible.

Instead, regarding the probability interval evaluated for each safety property (the rows without $*$), a more precise evaluation allows to show how the probabilities of unsafe scenarios are low (according to the statistical parameters), even if the injection of communication errors happens with high probability.

This result meets our expectations, augmenting our confidence that both the model and the adopted defense technique are correct.

Formula	Scenario	Runs	Evaluation	Verification/kernel/ elapsed time used	Resident/virtual memory usage peaks
φ_4	False positives detection with fault injection	757	[0,0.00999058]	28.2s / 0s / 28.2s	27.720KB / 51.152KB
φ_5	False negatives detection with fault injection	757	[0,0.00999058]	32.51s / 0s / 32.592s	27.720KB / 51.152KB
φ_6	Re-sequencing fault injection	124*	[0.0337242,0.133311]	2.35s / 0s / 2.356s	24.744KB / 48.168KB
φ_7	Re-sequencing fault injection	757	[0,0.00999058]	23.22s / 0s / 23.225s	27.464KB / 51.152KB
φ_8	Repetition fault injection	327*	[0.669745,0.769612]	8.26s / 0s / 8.263s	28.724KB / 52.156KB
φ_9	Repetition fault injection	757	[0,0.00999058]	33.67s / 0,01s / 33.712s	29.464KB / 53.152KB
φ_{10}	Deletion and re-sequencing fault injection	307*	[0.696799,0.796691]	3.46s / 0s / 3.464s	30.708KB / 54.140KB
φ_{11}	Deletion and re-sequencing fault injection	757	[0,0.00999058]	14.83s / 0s / 14.823s	31.448KB / 55.136KB
φ_{12}	Deletion, re-sequencing and transmission delay fault injection	68*	[0.00358196,0.102241]	1.36s / 0s / 1.354s	32.708KB / 56.140KB
φ_{13}	Deletion, re-sequencing and transmission delay fault injection	757	[0,0.00999058]	14.65s / 0s / 14.652s	31.704KB / 55.136KB
φ_{14}	Deletion fault injection	79*	[0.00790082,0.106991]	1.45s / 0s / 1.443s	34.584KB / 58.164KB
φ_{15}	Deletion fault injection	757	[0,0.00999058]	14.11s / 0,01s / 14.111s	35.320KB / 59.156KB
φ_{16}	Transmission delay fault injection	402*	[0.464871,0.564758]	5.85s / 0s / 5.852s	11.300KB / 33.904KB
$\varphi_{16'}$	No transmission delay fault injection	29*	[0,0.0981446]	0,57s / 0s / 0.569s	17.088KB / 40.052KB
φ_{17}	Transmission delay fault injection	757	[0,0.00999058]	14.69s / 0s / 14.697s	10.296KB / 32.908KB
φ_{18}	Fault injection	757	[0,0.00999058]	18.29s / 0s / 18.291s	11.072KB / 33.904KB
φ_{19}	Fault injection	757	[0,0.00999058]	28.91s / 0,01s / 28.946s	11.324KB / 33.904KB
φ_{20}	Fault injection	757	[0,0.00999058]	28.62s / 0s / 14.621s	11.324KB / 33.904KB

Table 4.5: Evaluation results of safety properties

Note that the two evaluations of the φ_{16} formula (referred to as φ_{16} for the evaluation of the transmission delay threat with the transmission delay fault injection enabled, and as φ'_{16} for the evaluation of the transmission delay threat with the transmission delay fault injection disabled) confirm that

the injection of the transmission delay threat determines a high probability for the SAI_Receiver template of receiving a message with an unacceptable delay, which is instead very unlikely if no transmission delay threat is injected in the transmission system (i.e. the injected fault is the only possible source of excessive transmission delay).

4.3 Lesson learned

In this section, we report issues we met during the modeling and analysis, which are related to either undefined aspects or ambiguities due to the usage of natural language for specifying the requirements.

Note that when referring to the model implemented as described in the Subset 098, we write "the specified model"; instead, when referring to the developed model where we resolved the issues independently from the specification, we write "our model".

The outcome of our modeling activity looks quite important: indeed, since we needed to give a personal, although reasonable, interpretation to certain ambiguous aspects, different interpretations of this aspects could be given as well by different suppliers. Consequently, an undesired but possible scenario can be the non-interoperability between RBCs developed and provided by different suppliers. Actually, in [34] it is reported that interoperability issues were encountered during the handover procedure between RBCs produced by different suppliers.

The first problem we met concerns the implementation of the sequence number defense technique. Indeed, in the Subset 098 specification, it is stated that "Once the sequence number value reaches the maximum value,

the value of the sequence number at the next message transmission shall be set to 0." (ref. 5.4.7.1.7). Anyway, when describing how the sequence number technique acts to detect sequencing errors, it is suggested that only the sequence number difference with the previous message shall be checked (ref. 5.4.7.1.5), leading to an unsafe situation when the sequence number reaches the maximum value, as described below. Moreover, when listing all the possible scenarios concerning the values of the received sequence number and the sequence number of the last accepted message (ref. 5.4.7.2.4 – 5.4.7.2.7), the check procedures refer only to a sequential ordering of the sequence numbers and no zero-crossing is ever mentioned, unlike it does for the timestamp information contained in the TTS fields (ref. 5.4.8.6.4).

In our model, we decided to implement a simple difference between the received sequence number and the sequence number of the last accepted message, implementation that respects and does not contradict the requirements. Anyway, this implementation choice does not handle the zero-crossing of the sequence number value. Indeed, when the maximum value for the sequence number (`SN_max`) is reached, the next message sequence number shall be 0, and assuming that the message stream is correctly received, the computed difference would be negative ($sn\_diff < 0$) and the message erroneously discarded.

This problem has not emerged in all the previous simulations because the maximum allowable value for the sequence number field was never reached. However, by lowering the maximum sequence number value, it is possible to verify that this unsafe scenario indeed exists due to the sequence number zero-crossing problem, i.e. a correct message with sequence number field set to 0 is discarded or considered as a communication error when the last accepted sequence number value is `SN_max`.

Through the formula φ_{21}, we verify if the above unsafe scenario is reachable for the `SN_max` parameter set to a lower value, for example 100.

$$\varphi_{21} = Pr\,[\leq 1000]\,(<>\ \ SAI\_Receiver.sig\,!= empty\_sig\ \&\&$$

$$last\_sn == SN\_max\ \&\&\ SAI\_Receiver.sig.msg.sn == 0\ \&\&$$

$$(SAI\_Receiver.DiscardMsg\ ||\ SAI\_Receiver.Error)))$$

The φ_{21} formula checks if the SAI_Receiver enters either the `Error` location or the `DiscardMsg` location when receiving an incoming message ($sig\,!= empty\_sig$) with sequence number 0 ($sig.msg.sn == 0$) and its last accepted message had sequence number `SN_max` ($last\_sn[id] == SN\_max$). This formula refers to the scenario in which the Sender device performs the zero-crossing of the sequence number. Even if the message stream is correct, the Receiver device behaves as if a communication error occurred.

The result of the evaluation of φ_{21} (where both the probability of false negatives α and the probability uncertainty ϵ are set to 0.05) says that the probability of occurrence of this unsafe scenario is in [0.797987 , 0.897941] probability interval with 95% of confidence, computed in 211 runs.

This confirms that our implementation choice does not satisfy the safety requirements of the system when facing the sequence number zero-crossing problem, while still respecting the sequence number specification described in the requirements. Basically, this proves that the current version of Subset 98 is unsafe against the CENELEC threats of re-sequencing, repetition and duplication.

With φ_{21} we focus on the particular scenario of the SAI_Receiver dealing with the reception of a correct message but considering it as a previous

message, thus discarding it. Nevertheless, plenty of similar scenarios can occur when the communication error injection and the sequence number zero-crossing mix together. For example, we could ask the following question: "what happens if the last accepted sequence number is 1 and the received sequence number is `SN_max`?". In our model, if $sn\_diff = SN\_max - 1 > N\_max\_lost\_msg$, the safe connection release is forced by the SAI_-Receiver, because of the non-tolerable message loss.

Anyway, in the specified model the same example could also be associated with the reception of a message older than the last accepted one and it is solved by simply discarding the received message.

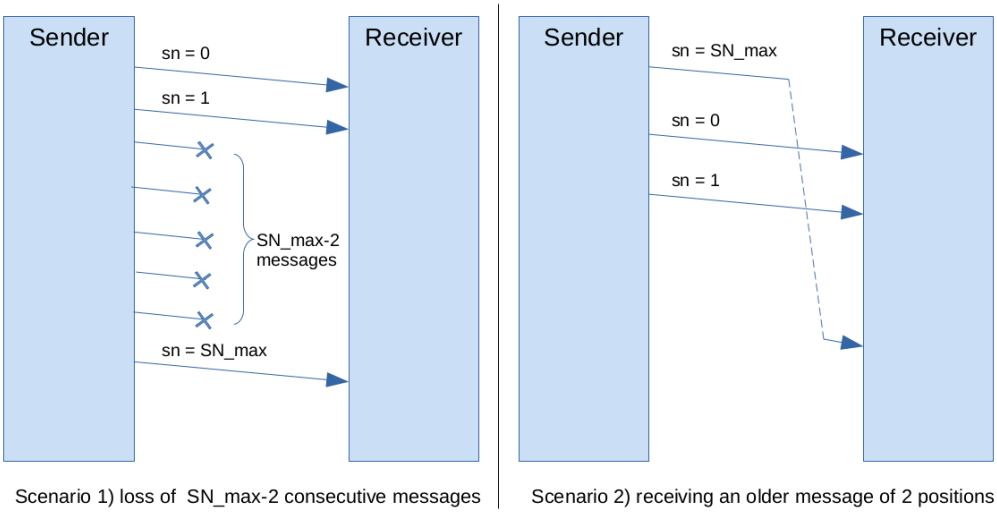

Figure 4.3: Two possible scenarios where the incoming message has sequence number equals to `SN_Max` and the last accepted message had sequence number equals to 1

Referring to this example, we identified two possible scenarios (see Figure 4.3) where the incoming message has sequence number equals to `SN_max` and the last accepted message had sequence number equals to 1:

1. The incoming message is the first message arrived since the last correct message reception (message with sequence number 1), after the loss of $SN\_max - 2$ consecutive messages. Then, the safe connection is released if $SN\_max - 2$ is greater then `N_max_lost_msg`;

2. The incoming message with sequence number equal to `SN_max` is the last message sent prior to the sequence number zero-crossing of the Sender device, but due to a particular transmission delay, it is delivered to the Receiver device after that the message with sequence number 1 has been received. Then, the message is discarded, being an older message of 2 positions.

Due to the sequence number zero-crossing, it is impossible to distinguish between the two scenarios without making an assumption on the maximum number of consecutive lost messages, which for example could be $\frac{SN\_max}{2}$. In this case, still referring to Figure 4.3, the first scenario of losing SN_max-2 consecutive messages should be discarded in favor of the second scenario because it is violating our assumption, provided that SN_max-2 is greater than $\frac{SN\_max}{2}$.

Notice that this insight is only meant to show how the zero-crossing of the sequence number can lead to possible unsafe scenarios: indeed, finding the right implementation to correctly deal with this problem requires domain experts and thus is out of scope for this thesis. Instead, finding a possible source of errors in the specification that could lead the system to unsafe scenarios is one of the formal analysis carried out in this thesis.

To mitigate this sequence number zero-crossing problem, we propose a simple but effective "solution" that allows our model to meet the safety

requirements although at the cost of its availability. Our proposal is to force the release of the safe connection from the SAI module when the maximum value for the sequence number is reached. This mechanism prevents the SAI_Receiver from handling the sequence number zero-crossing because no sequence number zero-crossing occurs at all. Note that a new connection must be re-established between the two devices if the communication was not over yet.

In our model, the same problem can occur to the timestamp information contained in the TTS fields. Indeed, the zero-crossing of the timestamps is not considered safe with respect to the transmission delay and a connection release is forced by the Sender device when the maximum allowable value for the TTS fields is reached. Hence, we do not provide any solution for the timestamp zero-crossing problem.

In Figure 4.4 and in Figure 4.5, the templates to check the sequence number and the SAI clock respectively are shown. They have been added to our model in a second modeling phase to mitigate the sequence number and the TTS zero-crossing problems. Each of these two templates is instantiated twice, one for the Initiator device and one for the Responder device and through the `id` parameter they can refer to the belonging device. To command the safe connection release, the `Sa_DISCONNECT_indication` channel is used.

De facto, this solution is a mitigation of the problem that allows our model to satisfy the safety requirements. Anyway, the Subset 098 [33] should give more details regarding the sequence number protection mechanism to avoid possible unsafe scenarios due to the zero-crossing problems.

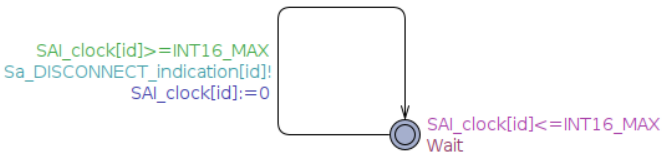

Figure 4.4: SAI_SN_Check template

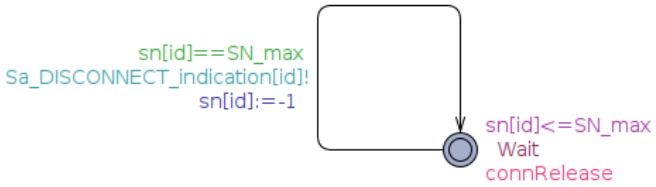

Figure 4.5: SAI_Clock_Check template

In the Subset 098 specification, we also found a scenario where no information is given to understand the behavior of the system. Recall the SAI_TTS_Init_Res template (Chapter 3, Section 3.2.4). After the Responder SAI module answers to the incoming `Sa_CONNECT_indication` of the peer entity with the `Sa_CONNECT_response`, the SAI_TTS_Init_Res template waits for the TTS initialization procedure to start. In case the OffsetStart message sent by the Initiator fails to arrive, the SAI_TTS_Init_Res template keeps waiting for the OffsetStart message in the `WaitTTSOffsetStartMsg` location. If a new connection request is sent by the Initiator, the Responder receives another `Sa_CONNECT_indication`.

In the Subset 098, there is no mention for this scenario: it is not included in the SAI initial procedures at the error handling section (ref. 5.4.10.1.3) as no TTS initialization for the Responder is started yet, and no communication with the SAI User can be assumed, as the interactions between the SAI and

the SAI User modules of the Responder device start after the successful TTS initialization.

We identify two possible actions for the Responder SAI module:

- The new `Sa_CONNECT_indication` request from the Initiator device is not considered a communication error, and the SAI module of the Responder device answers again with the `Sa_CONNECT_response`, and keeps waiting for the TTS initialization procedure to start;
- The new `Sa_CONNECT_indication` request from the Initiator device is considered a communication error as it is not the expected message. Hence, the SAI module of the Responder starts again the connection establishment procedure, and waits for another `Sa_CONNECT_indication` from the Initiator device to arrive.

In our model, we chose to implement the second scenario where the failed connection procedure is interrupted to restart a new one from the beginning. Obviously, this implies that the Initiator must send again a connection request, meaning that the two devices need at least three connection procedures to establish the connection, while in the first scenario two connection procedures could be enough.

Regarding the safety aspects of the two scenarios, in the Subset 098 specification no significant information is provided. Anyway, we found a non-specified behavior of the system that could lead to possible unsafe configurations if no specific actions are implemented to manage the scenario described above.

Another aspect in the specification that we found ambiguous and poorly detailed concerns the procedure of the clock offset update. In the require-

ments it is stated that "The structure of the message used for the clock offset update shall be compliant to the structure of application data message with no user data field." (ref. 5.4.8.7.3) and a "clock offset update request" and a "clock offset update answer" are mentioned (refer to Figure 21 of [33]) as the two messages exchanged for the update procedure.

Anyway, assuming that both the two communicating devices can perform the update procedure and that the two distinct update procedures can overlap (i.e. while handling its update procedure, a device could receive the update request of the partner device), the exchange of a simple Application Message with no user data field is not enough to discern a request message from an answer message. Indeed, considering the scenario in which the Initiator device sends an offset update request to the Responder device. When the Initiator device receives an Application Message with no user data field from the Responder device, it cannot know if the received offset update message refers to either an answer to its request message, or a request sent by the Responder (the two distinct offset update procedure has overlapped).

In our model, we assumed that the Application Messages used for the update procedure are exchanged through the `Sa_DATA_request/Sa_DATA_-indication` primitives and we re-used the TTS initialization message type fields to distinguish the two types of update messages, and in particular the OffsetStart for the update request message and the OffsetAnsw1 for the update answer message. These implementation choices were made necessary to model a working offset update procedure despite of the lack of details in the specification. Again, the fact that different suppliers could implement these aspects independently could significantly compromise the interoperability of RBCs provided by different suppliers.

Still focusing on the procedure of the clock offset update, we found another non-specified behavior of the system: when a SAI module receives an offset update answer message, it has to check if the answer message is related to the last clock offset update request (ref. 5.4.8.7.7). The check is performed by comparing the last receiver timestamp field of the message with the timestamp of the last offset update request (i.e. the timestamp of the SAI module at the sending of the last offset update request message). To update the offset estimations, these two values must be equal. Anyway, the Subset 098 [33] does not specify the actions to perform in case this check is not successful.

We identified two possible behavior for the SAI module: it can ignore the "wrong" answer message and keep waiting for the "right" answer message to arrive within the allowed time interval, or it can consider the unsuccessful check as a failure of the clock offset update procedure, thus implying a new offset update request to be sent to the partner device.

Notice that the "wrong" answer message simply refers to an offset update answer message related to a previous update request, while the "right" answer message refers to an offset update answer message related to the last update request. A SAI module can receive a "wrong" answer message if the answer message to a request `r` arrives after the expiration of the timer related to the request `r` (for example due to an excessive transmission delay), and when a new request `r'` has already been sent.

In our model, we decided to ignore the reception of a "wrong" offset update answer. We made this choice because the other behavior can result in a loop of requests and "wrong" answers to be entered. Indeed, the answering device keeps sending answer messages but they arrive when another request has already been sent, so the timestamp checks result unsuccessful. In this

case, the update of the offset estimations is not performed, resulting in a potentially unsafe scenario since the minimum offset estimation is involved in the TTS defense technique.

Moreover, the offset update procedure updates both the minimum and the maximum offset estimations. Anyway, the maximum offset estimation is used in the TTS initialization procedure to check if the connection establishment is successful, but once the safe connection is established, only the minimum offset estimation is used when checking the transmission delay at the reception of incoming messages. Hence, the maximum offset estimation keeps to be updated even if its value is never used.

Furthermore, we noticed that in the specification (ref. 5.4.10.1.5) it is stated that an independent error counter shall be used to monitor the failure of the offset update procedure, and the update execution shall be repeated without delay. Actually, this error counter remains unused, and we think that further details should be given for a better comprehension of its purpose, otherwise it seems useless.

Focusing on the error handling specifications (ref. 5.4.10), we noticed that in the list of possible errors, both a "transmission delay" and a "delayed messages" are referred, but the difference between these two types of errors is never mentioned. We assumed that these two errors actually refer to the same transmission delay threat, but their distinction could be misleading. Indeed, it is stated that "an independent error counter shall be used and the procedure shall be repeated without delay."(ref. 5.4.10.1.5) also for the "procedure for detection of transmission delay". If the "procedure for detection of transmission delay" refers to the calculation of the transmission delay at the reception of a new incoming message, the repetition of the procedure

without delay could not be performed because it is not responsibility of the SAI module to request a new Application Message to the partner device (this is responsibility of the SAI User). We decided to ignore this part of the specification due to the many ambiguities encountered and we suggest that this section on error handling should be more and better detailed.

Finally, we discuss the configuration choice of the maximum number of successive errors (`N_max_succ_err`) parameter. In the specification it is stated that `N_max_succ_err` must be set to 1 when a release of the connection at the first error is recommended (ref. 5.5.5.2.1), and should be set to 2 in those configurations where the release of the connection shall occur after two successive errors (ref. 5.5.5.2.2). We chose to test the release of the connection after two successive errors in conjunction with the possibility of admitting a small loss of messages ($N\_max\_lost\_msg > 1$). Indeed, if `N_max_succ_err` is equal to 1, the `N_max_lost_msg` parameter greater than 1 would be useless: even if one message loss is allowed, this event is handled as a communication error, thus implying the release of the connection after `N_max_succ_err` successive errors. Basically, no more than one message can be loss even if $N\_max\_lost\_msg > 1$.

During the message exchange between the two communicating devices, if the `N_max_succ_err` is set to 2, one sequencing error is tolerable, for example the loss of at most $N\_max\_lost\_msg - 1$ messages, but when another communication error occurs, the connection must be released. Anyway, we noticed that when a message arrives and a transmission delay is detected, the message is rejected due to its non-tolerable delay and the error counter is incremented, according to the error handling procedure. This means that at the next message reception, another communication error is immediately detected, as the sequencing of the message stream is for sure not respected.

Indeed, the sequence number of the incoming message cannot be equal to the last sequence number accepted +1 because the previous message of the stream has been rejected due to its non-tolerable delay. Hence, the error counter is increased and its value reaches the `N_max_succ_err` threshold, resulting in the release of the connection. This means that the ability of the system to tolerate the occurrence of one communication error is tied to the type of the error being detected: if the error is a transmission delay, then the system is no longer able to accept the next incoming message as correct.

Note that there is no mention of this behavior of the system in the Subset 098 [33], hence we are not sure whether this is exactly how the system should behave.

Concluding this section, we confirm that the modeling of a system starting from its natural language specification is extremely useful in detecting possible issues in requirements. The presence of poorly detailed aspects can lead to safety problems, but also to have behaviors that compromise the interoperability between devices provided by different suppliers.

A schematic summary of the problems described in this section is provided in Table 4.6.

The formal verification step helped in identifying the modeling problems reported in this section, thus resulting extremely useful during the modeling step of our model. It has been used to check the correctness of the model by debugging modeling errors during its development, as well as to formally validate the requirements of the system defined in the specification.

Problem	**Description**	**Problem's identification**
SAI sequence number	In the Subset-098, no reference to the zero-crossing problem is made. Hence, threats due to re-sequencing, repetition and deletion are not mitigated in this corner case.	This problem has been identified at the modeling step of this defense technique.
SAI connection procedure	If the StartOffset message sent by the Initiator fails to arrive at the Responder device, no specification for the Responder SAI behavior is given.	This problem has been identified during the verification step of the model.
SAI clock offset update messages	The offset update procedure request and answer messages are Application Messages with no user data field, but there is no way to distinguish a request from an answer in the case of overlapping update procedures.	This problem has been identified during the verification step of the model.
SAI clock offset update messages error counter	During the clock offset update procedure, when an error occurs an independent error counter shall be increased but the procedure is repeated until successful. The error counter is unused.	This problem has been identified by analyzing the subset specification.
SAI clock offset update procedure	During the clock offset update procedure, when the timestamp check performed to an incoming answer is not successful, no specification for the SAI module behavior is given.	This problem has been identified at the modeling step of the clock offset update procedure.
SAI maximum offset estimation	After the connection establishment, the maximum offset estimation is unused, hence its update is useless.	This problem has been identified by analyzing the subset specification.
SAI error handling	Both the "transmission delay" and the "delayed messages" are referred, but the difference between these two types of error is never mentioned.	This problem has been identified at the modeling step of the error handling.
SAI maximum number of successive errors	The parameter related to the maximum number of successive errors can be set to 2, but when a transmission delay threat occurs and increases the error counter, the next incoming message in turn increases the error counter that reaches its maximum allowable value and the connection is immediately released. Hence, having this parameter set to 2 can results useless and misleading.	This problem has been identified during the verification step of the model.

Table 4.6: Summary of the problems found in the Subset 098 specification [33]

4.4 Considerations on modeling effort

As already mentioned in the previous chapters, this thesis has the opportunity to contribute to the ongoing 4SECURail project providing further

data for the cost-benefit analysis for the adoption of formal methods based on the efforts for training and modeling steps.

Starting from the training, the necessary knowledge to work with formal methods has been provided during the *Software Dependability* university course, where the basis for the modeling aspects and the logic required to understand the model checking algorithm have been studied. The effort for this training part can be measured in terms of CFU (Crediti Formativi Universitari, corresponding to ECTS − European Credit Transfer and Accumulation System − credits) associated to the discussion about formal methods in this course, which are almost 7 CFU.

Concerning the development of this thesis, the work efforts can be divided in a first period of more or less two weeks dedicated to the UPPAAL SMC toolbox, and a second period of approximately four months focused on the modeling and the verification of the system. In particular, in the first period the syntax and the semantics of UPPAAL SMC was studied to become familiar with the tool, while the second period was first dedicated to the comprehension of the case study specification and then to the formal modeling and the verification of the system.

Actually, the last period is not characterized by such a clear division between the modeling and the verification phases because they have been carried out hand in hand and a clear distinction of the effort made is not possible. Indeed, since the beginning of the modeling phase, a constant activity of verification of the model was made.

The modeling of the system started from the implementation of the connection and the disconnection procedures and the sequence number defense technique. Through the verification of this first prototype of the model, use-

ful counterexamples provided by the UPPAAL SMC verifier allowed to debug modeling errors. Once completed this first prototype of the model, a first evaluation on simplified safety requirements was performed to check the sequence number defense technique functioning. Then, the integration with the TTS defense technique was made. The counterexamples provided by the simulation runs helped also in this second modeling phase to debug the model. After completing the modeling of both the TTS initialization procedure and the implementation of the protection system against the transmission delay threat, the safety verification on the fully defined model was performed to check its behavior.

Summarizing, by summing up the 7 CFU associated with the training step and the 24 CFU associated with the thesis work composed of the modeling and the verification steps, we obtain a total of 31 CFU. Considering that each CFU conventionally corresponds to 25 working hours, we can estimate an effort of 775 working hours per person to reach the result we described in this thesis, starting from no knowledge of formal methods.

The actual effort concerning the working hours for the completion of this thesis work is in line with this estimation made in terms of CFU. Note that this estimation can be validated by consulting the online regulations of the Degree Course [35].

Activity	**Hours**
Learning	169
Modeling + Verification	606

Table 4.7: Working hours division per activity

In the Table 4.7, an indicative division for the working hours required for

each activity is shown.

This thesis gives also a contribution in terms of the analysis of the case study, reporting the ambiguous aspects that emerged during the modeling of the system and providing useful and rigorously formalised artifacts like the model and the safety properties that can enrich the existing documentation. Moreover, considering that the formal methods are still a subject of study in the field of railway industrial applications, this thesis represents a further contribution to evaluate the usefulness not only in terms of costs but also in terms of the benefits deriving from the adoption of the formal method in this domain. In [36], the developed model is publicly available.

Template	**Ref**
SAI_User	5.4.2, 5.4.5
SAI_Conn_Ini/ SAI_Conn_Res	5.4.5.1, 5.4.5.2
SAI_TTS_Init_Ini/ SAI_TTS_Init_Res	5.4.8.3 - 5.4.8.5, 5.4.10.1.3
SAI_Sender	5.4.5.3, 5.4.6, 5.4.7.1.6, 5.4.7.1.7, 5.4.8.6.1 – 5.4.8.6.3, 5.4.8.6.8
SAI_Receiver	5.4.5.3, 5.4.6, 5.4.7.1.4, 5.4.7.1.5, 5.4.7.2, 5.4.8.6.9 – 5.4.8.6.14, 5.4.10.1.4
SAI_Update_Req/ SAI_Update_Answ	5.4.8.7, 5.4.10.1.5
Euroradio_SL_Env	5.4.5
Communication_System	6.5.4.3

Table 4.8: Requirements mapping between the developed model and the specification

We conclude this chapter showing in Table 4.8 the mapping between the specification and the templates of our model and in Table 4.9 some grammatical mistakes found in the Subset 098 [33].

Error	**Correction**	**Ref**	**Page**
EURODIO	EURORADIO	Figure 8	24
Last received time stamp	Last receiver time stamp	5.4.8.4.2, 5.4.8.4.4, 5.4.8.4.6, 5.4.8.4.8, 5.4.8.4.10	32-36
Local time when the last message was received:	Time stamp at the last message reception:	5.4.8.4.6	34
Sender time stamp: This field defines the time stamp of the responder.	Sender time stamp: This field defines the time stamp of the initiator.	5.4.8.4.10	35
OffsetEnd(T_{init5},T_{res4}, OK/notOK)	OffsetEnd(T_{init5},T_{res4},T_{init4}, OK/notOK)	Figure 17	37
At the OffsetAnsw message transmission	At the OffsetAnsw1 message transmission	5.4.8.5.5	38
If the initiator receives the OffsetAnsw message	If the initiator receives the OffsetAnsw1 message	5.4.8.5.6	38

Table 4.9: List of grammatical mistakes found in the Subset 098 [33]

Chapter 5

Conclusions and future work

The formal analysis of the case study described in this thesis shows how the adoption of formal methods can help to detect ambiguities in the specifications of the requirements of a system, in particular when only natural language documents are provided.

This thesis focused on the formal modeling and verification of a communication system in the railway signaling domain. The analyzed case study is a portion of the RBC/RBC handover protocol specified by UNISIG into the SUBSET 098 – RBC/RBC Safe Communication Interface.

Starting from this specification, we formalized the functioning of the Safe Application Intermediate sub-layer through a network of Stochastic Timed Automata using the UPPAAL SMC toolbox. Then we verified the safety requirements of the model to enlighten possible errors derived from the specification of the system. By increasing the probability of occurrence of rare events, such as the occurrence of communication errors in a transmission system, it is possible to observe if the system itself is able to correctly handle potentially unsafe scenarios.

The use of Stochastic Timed Automata permits to model stochastic behaviors of real-time systems, while the Statistical Model Checking allows to perform fast qualitative analysis to evaluate the probability of particular scenarios. In our case, the analysis of the system focused on the probability estimation of the occurrence of unsafe scenarios. Indeed, ambiguous and poorly detailed specifications may compromise the safety of the system.

This thesis provides interesting results on the analysis of the specified case study, including some ambiguities detected in the specification that could lead to the non-interoperability of RBC devices developed by different suppliers. Moreover, a contribution to the 4SECURail evaluation of the cost-benefit analysis for the adoption of formal methods is given, showing how this thesis work led to the identification of possible critical aspects in a reasonable working time both for learning and for developing the analysis.

The developed model can be further analysed with a different parameter configurations, for example using specific values suggested from the Infrastructure Managers. Indeed, our parameter configuration is functional to carrying out a fast and effective qualitative analysis of the model, but a quantitative analysis can still be performed with realistic parameters suggested by domain experts.

Moreover, we chose to implement the TTS defense technique against the transmission delay threat, but in the Subset 098 specification also the alternative EC defense technique is presented. Hence, a further interesting future work can be the development of another model where the EC defense technique is implemented and an analysis similar to the one we have discussed is performed.

List of Figures

Bibliography

[1] Alessio Ferrari, Alessandro Fantechi, Gianluca Magnani, Daniele Grasso, and Matteo Tempestini. The metrô rio case study. *Science of Computer Programming*, 78(7):828–842, 2013.

[2] Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. METEOR: A successful application of B in a large project. In *International Symposium on Formal Methods*, pages 369–387. Springer, 1999.

[3] Alessandro Fantechi. Twenty-five years of formal methods and railways: what next? In *International Conference on Software Engineering and Formal Methods*, pages 167–183. Springer, 2013.

[4] Jean-Raymond Abrial. Formal methods: Theory becoming practice. *J. UCS*, 13(5):619–628, 2007.

[5] European Commission. COMMISSION STAFF WORKING DOCUMENT EXECUTIVE SUMMARY OF THE IMPACT ASSESSMENT accompanying the document proposal for a council regulation establishing the shift2rail joint undertaking., 2013.

[6] Shift2Rail. Shift2rail initiative.

[7] 4SECURail. Specification of formal development demonstrator-CNR-1.0.

[8] Eulynx. Eulynx project.

[9] NL Bui. An analysis of the benefits of EULYNX-style requirements modeling for prorail, 2017.

[10] Maarten van der Werff, Bernd Elsweiler, Bas Luttik, and Paul Hendriks. The use of formal methods in standardisation of interfaces of signalling systems. *IRSE News*, (256):15–17, 2019.

[11] Ana Luísa Ramos, José Vasconcelos Ferreira, and Jaume Barceló. Model-based systems engineering: An emerging approach for modern systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(1):101–111, 2011.

[12] scaledagileframework. Model based system engineering, 2021.

[13] omg. Precise semantics of UML state machines, 2019.

[14] Sanford Friedenthal. Requirements for the next generation systems modeling language (sysml® v2). *INSIGHT*, 21(1):21–25, 2018.

[15] NASA Langley Formal Methods. What is formal methods?

[16] Daniel M Berry. Ambiguity in natural language requirements documents. In *Monterey Workshop*, pages 1–7. Springer, 2007.

[17] R Cary and R Spitzer. The avionics handbook. In *4S Symposium, Small Satellites Systems And Services Proceedings:(2004)*. New York/CRC press/2001, 2001.

[18] EN CENELEC. 50128-railway applications-communication, signalling and processing systems-software for railway control and protection systems. *Book EN*, 50128, 2020.

[19] Maurice H ter Beek, Arne Borälv, Alessandro Fantechi, Alessio Ferrari, Stefania Gnesi, Christer Löfving, and Franco Mazzanti. Adopting formal methods in an industrial setting: the railways case. In *International Symposium on Formal Methods*, pages 762–772. Springer, 2019.

[20] Alessandro Fantechi, Wan Fokkink, and Angelo Morzenti. Some trends in formal methods applications to railway signaling. *Formal Methods for Industrial Critical Systems*, pages 61–84, 2013.

[21] 4SECURail. 4SECURail project.

[22] Davide Basile, Maurice H ter Beek, Alessandro Fantechi, Alessio Ferrari, Stefania Gnesi, Laura Masullo, Franco Mazzanti, Andrea Piattino, and Daniele Trentini. Designing a demonstrator of formal methods for railways infrastructure managers. In *International Symposium on Leveraging Applications of Formal Methods*, pages 467–485. Springer, 2020.

[23] M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking: A tutorial introduction. In *SAS*, 1999.

[24] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[25] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International conference on runtime verification*, pages 122–135. Springer, 2010.

[26] Kim G Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In *International Symposium on Fundamentals of Computation Theory*, pages 62–88. Springer, 1995.

[27] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal. *Formal methods for the design of real-time systems*, pages 200–236, 2004.

[28] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[29] Alessio Ferrari, Gloria Gori, Benedetta Rosadini, Iacopo Trotta, Stefano Bacherini, Alessandro Fantechi, and Stefania Gnesi. Detecting requirements defects with nlp patterns: an industrial experience in the railway domain. *Empirical Software Engineering*, 23(6):3684–3733, 2018.

[30] 4SECURail. Case study requirements and specification SIRTI 1.0.

[31] subset 039 - FIS for the RBC/RBC Handover - (issue 3.2.0), 2015.

[32] EN CENELEC. 50159, Railway applications Communication, signalling and processing systems safety-related communication in transmission systems.

[33] subset 098 - RBC/RBC Safe Communication Interface - (issue 1.0.0), 2007.

[34] Sara Morselli. *Il nuovo servizio ferroviario ad Alta Velocità "Frecciarossa": analisi delle performance.* PhD thesis.

[35] Università degli studi di Firenze. Regolamento.

[36] Irene Rosadi. https://github.com/irenerosadi - github repository.