University of Cincinnati

Date: 4/20/2021

I, Nawar Obeidat Ph.D., hereby submit this original work as part of the requirements for the degree of Doctor of Philosophy in Computer Science & Engineering.

It is entitled:

Using Formal Methods to Build and Validate Reliable and Secure Smart Systems via TLA+

Student's name: **Nawar Obeidat Ph.D.**

This work and its defense approved by:

Committee chair: Carla Purdy, Ph.D.

Committee member: Shaaban Abdallah, Ph.D.

Committee member: Nan Niu, Ph.D.

Committee member: Massoud (Max) Rabiee, Ph.D.

Committee member: Boyang Wang

38273

Using Formal Methods to Build and Validate Reliable and Secure Smart Systems via TLA+

Doctor of Philosophy

Department of Electrical Engineering and Computer Science

Collage of Engineering and Applied Science

by

Nawar Obeidat

Advisor

Dr. Carla Purdy

Committee Members:

Dr. Nan Niu

Dr. Massoud Maxwell Rabiee

Dr. Boyang Wang

Dr. Shaaban A Abdallah

April, 2021

Abstract

Verification and validation are very important in the design of any system. For safety-critical systems, especially, the validation level is crucial. In this work, developing a robust procedure for designing smart, safe, secure systems is our eventual goal. There are different methods for verification and validation, but formal methods are used widely to validate these systems due to their high level of confidence for guaranteeing correctness, security, and safety properties. TLA+ is a formal specification language and its TLC model checker will be used in this work to formally verify and validate several example systems, and to find security hacks. Designing smart systems has recently received much attention because of the ability to connect heterogeneous sub-systems together to build a fully controlled smart system. Here we propose models for three systems: a smart school building system, an ADS-B system for avionics, and a SCADA system. We model the systems using UML diagrams to describe objects, states, and sequences of actions. Formal methods techniques are used to verify the properties of the systems. TLA+ is used for formal modeling. For the smart school building system, the ADS-B system, and the SCADA system, we verify their properties and correctness using the TLA+ toolbox and the TLC model checker. Also, we use the TLC model checker to find example security bugs we introduce into the designs.

Acknowledgments

First and foremost, all praise and thanks are due to “Allah” for allowing me to be able to accomplish this work. I would like to express my acknowledgments to my advisor, Professor Carla Purdy, for her mentorship, help and support during my stay at the University of Cincinnati. Words cannot describe how appreciative I am to her. Also, I would like to thank the members of my dissertation committee Dr. Massoud Maxwell Rabiee, Dr. Nan Niu, Dr. Boyang Wang, and Dr. Shaaban A Abdallah for their effort, time, help and feedback regarding this work. I would like to thank University of Cincinnati for granting me a scholarship and assistantship support. I would like to acknowledge and thank my Dad and Mom and all my family, for always supporting and encouraging me over the years. Finally, I would like to acknowledge my lovely husband, Dr. Mohammed Ababneh, for his encouragement, support, and help in providing valuable comments and technical feedbacks. Thanks for my beloved kids, Taha, Jad, and Naya. Their smiles get rid of my exhaustion and fatigue.

Contents

List of Figures

List of Tables

Chapter 1: Introduction and Overview

1.1 Introduction

Formal methods have been used in the design cycle of different kinds of systems to guarantee correctness and safety for different applications. Many of these systems worked as specified because of formal methods help. Formal methods have the ability to catch errors and bugs that other methods can't find. Many prominent companies have been using formal methods in the verification and validation of their complex systems [2,4,8]. Formal methods can interact with design systems like UML and SysML [48] and, in addition, they can provide a formal verification of the system properties.

Researchers have modeled many smart systems such as smart office, home automation systems, smart library, smart campus, airport systems, and various vehicles. The way that most of these systems were modeled and implemented focused on correct operation and did not ensure security for the system. We have designed a smart school building system that achieves smart system features and safety properties [35, 45], a SCADA system with security protections, and an improved ADS-B system model for avionics [46].

In this work, chapter 1 will present an introduction and overview of verification and validation, formal methods, specification languages, TLA+ and why we have chosen to use it, as well as the related work in using formal methods, especially TLA+, in different kinds of applications. Chapter 2 will provide an example of using TLA+ and its model checker TLC to solve a simple well-known problem, along with our work in modeling a smart school building model using TLA+ and TLC. This chapter shows how the TLC model checker can find a security bug in the model. Chapter 3 will define the ADS-B system and explain why it is important, and will show

how we modeled the ADS-B system using UML, TLA+ and the TLC model checker for validation. We model in detail one of the proposed security solutions and show its validity. Chapter 4 will illustrate what a SCADA system is and why it is vulnerable, and will show we modeled an example SCADA system using UML and TLA+. For this example, we used the TLC model checker to validate our model first, then we added a security problem and the showed how the TLC model checker found this security bug. Chapter 5 will include the conclusions of all our work and discuss future work.

1.2 Verification and Validation

Verification and validation are independent terms that are used together to check the quality of a system and to check that it works as it supposed to work with no bugs or failures. Basically, these processes check whether or not the system meets its requirements and specifications. The difference between the two terms is as follows: in validation we're checking to see if we are building the right system, and in verification we're checking to see if we're building it in the right way. In other words, in validation we're checking the user's "requirements", and in verification we're checking if the "specifications" derived from those requirements are implemented correctly [10].

There can be a lot of uncertainty and vagueness in the process of determining if a model is "good". In the modeling process, the most controversial step is evaluating a model with no uncertainty. Philosophical viewpoints can include differences that can lead to confusion. The central concern is usually validation, which is linked with terms like "verification", "qualification", and "confirmation" that are trying to ensure that "validation" is achieved in a

proper context. Some modelers use the term “validation” which means the model meets its performance requirements and is acceptable, while other modelers use the term “evaluation” to avoid the use of the term “validation” [11].

The term “verification” is a wide-ranging term that includes all approaches needed to show the system possesses specific properties. That may include a simple test case that proves a limited fact that the system will accomplish a certain result. The term “verification” has been used to increase the systematic and elaborated mathematical techniques to establish that the system possesses some specific properties. These properties may be a set of abstract specifications which include general terms like safety, liveness, and/or termination and which cover the implementation of the system’s specification or the correctness of realizations. There are different forms of verification. One of them is “formal verification” which uses formal mathematical languages [5].

If we take cyber-physical systems (CPSs) as an example, these systems have complex models because of the integration of software, hardware, and physical components which may cause a state explosion when modeling. In these models, formal verification is sometimes prevented by state explosion, because formal verification must include all of the system’s concerns. It’s important that any abstract model include all the important details to verify the system (or part of the system) [13].

From the software engineering point of view, a CPS has specific functional requirements that need to be validated to have a system accepted. But the non-functional requirements in our case of building and testing the CPS are not less important than functional requirements because we’re looking not only at functionality requirements for an accepted CPS, but we also want to be able to trust the CPS.

Software and hardware components, along with operating systems, need better development processes to improve the existing technologies. The software and hardware should be highly reconfigurable, dependable, and sometimes certifiable as components and as fully integrated systems, with respect to certain requirements. These complex systems should have trustworthiness which may not exist in current cyber infrastructures. Testing "till all money runs out" is not a smart strategy. Scientific and evidence-based methods are necessary to improve reliability. So, new algorithms, models, tools, and methods are needed to incorporate validation and verification of systems and software at the control design level [14].

Using formal methods for verification and validation in different kinds of systems has increased and the power of using them has been demonstrated [2, 4]. In this work, we will work on using formal methods for verification. Section 1.3 will illustrate more details.

1.3 Formal Methods

In control theory, to achieve stability, systems of nonlinear differential equations (complicated dynamics) are used and must be shown to work not just theoretically, but also in practice. Sometimes optimality, which might require meeting certain cost restrictions, for example, must be shown, along with determining the paths of a stable system. In formal synthesis, a system's specifications, such as safety and liveness, as well as the system's extensive requirements can be specified using temporal logic for controlling simple systems like digital circuits or through modeling with finite state graphs. The integral development of safety critical systems and cyber physical systems increased the need for computational tools to validate more complex systems derived from temporal logic specifications [12].

Formal methods can be used as development, specification, and verification tools in software and hardware systems. Formal methods are mathematically based techniques. They use software tools to implement systems that will meet their specifications accurately. Formal methods provide clarity and simplicity and remove complexity, which is one of the main goals in the system development process. Formal methods use formal verification schemes that ensure the system must be correct before it is accepted, and this makes formal methods different from other methods. All these reasons make formal methods highly trusted, and an excellent alternative to replace or enhance existing verification tools [43].

As in [15], formal methods can be used for verification in different hardware/software applications. Some examples include Ethernet switches, routers, security applications, and seL4 (OS microkernels) [15]. Many reputable high-tech companies such as NASA, Amazon, Intel, AMD, and IBM, have used formal methods. NASA has applied formal methods techniques in big projects like Unmanned Aircraft System (National Airspace System) and Air Transportation System (Next Generation). Intel used formal methods for verification in projects such as cache coherent protocol verification, Intel IA-64 architecture optimization, and iCore i7 processor validation. IBM has used formal methods as well for verification of registers and power gates, and for its IBM Power7 microprocessor [15].

There are several advantages to using formal methods, especially in industry. First, formal methods are highly trusted; they demonstrate system correctness better than other methods. Second, they have the ability to handle complex, very large problems and systems. Third, they decrease the cognitive burden, i.e., they support tools that allow engineers to use methods that are relatively easy to learn and apply. Finally, they can provide a big return on investment, which is interesting to industrial companies who can see them as a method that can work for a wide

range of applications and problems with minimum effort and time [4]. On the other hand, using formal methods has some drawbacks since it requires time and training to learn how to use them.

In order to apply formal methods, we need to model the system by translating it into an abstract mathematical model and then developing a specification. This means describing a system's duties and properties using a formal specification language. We'll illustrate specification languages in more detail in the following section.

1.4 Specification Languages

Formal methods use formal languages, which are called specification languages. Specification languages are used during requirements and system analysis, as well as during system design. Specification languages provide a much better way to describe a system than programming languages that are used to generate executable code for a system. They are not used to describe the "how?", they used to describe the "what?". The importance of specification languages in verifying program correctness follows from their ability to support proofs [16].

Understanding formal system syntax, semantics, and proof rules is required to apply specification languages. A language is determined by the syntax and the semantics, while the proof system is supported by the proof rules. Specifications are represented in the language as expressions, and reasoning with respect to properties of these specifications is represented by the proof system. We can stipulate designs of computing systems (hardware and/or software) using specification languages or recommend requirements to the system, or formally describe a domain [5].

A common essential assumption of many specification methods is that programs are modelled as model-theoretic or algebraic structures which contain a combination of sets of data values along with functions applied to those sets. This abstraction level agrees with the view that the correctness of the input/output behavior of a program takes priority over all its other properties. Specification should be able to allow a process of modification before implementing the specification. This modification process results in an executable algorithm which can then be represented by a programming language [16].

Specification languages have many useful features. They have a mathematically precise base that gives them an accurate syntax and semantics. Also, specification languages support abstraction, which simplifies the main properties of the system and helps developers create better, clearer code.

Many specification languages such as Raise [17], TLA+ [18], Z [19], VDM [20], and B [21] are used for modeling real world scenarios. In this work, we will focus on using the TLA+ specification language for verification.

Section 1.5 illustrates more details about TLA+. How is it defined? And why did we choose it?

1.5 TLA+

1.5.1 Introduction

TLA+ is high level modeling language that is used in modeling distributed and concurrent systems. It is mathematically based and used to describe systems precisely. TLA+ can be used to remove essential design errors that are hard and expensive to find and correct at the code level.

This section describes TLA+ based on work by Merz [3] and by Lamport [7]. It provides details about TLA+ and its TLC model checker.

TLA stands for the Temporal Logic of Actions formulas. TLA+ was designed by Lamport for describing and reasoning about distributed algorithms formally. He published his *Specifying Systems* book [7] which defines TLA+ and shows how to use it along with all its supporting tools. Lamport also introduced linear-time temporal logic, and designed TLA+ specifications to be arranged in modules which can be independently reused.

In a quest for minimality of concepts, TLA+ doesn't formally differentiate between specifications and properties; both of them are written as logical formulas, while concepts such as hiding of the internal state, refinement, and composition of systems are represented using logical connectives of quantification, implication, and conjunction. In addition to its expressiveness, TLA+ is also supported by tools like theorem provers and model checkers to help a designer in developing formal specifications.

The following section will introduce the TLC model checker which we will use in our work.

1.5.2 TLC Model Checker

TLC is the TLA+'s model checker that checks the specification. It is where programs can be executed and compiled and the TLA+ module can be verified and validated. This process provides confidence that a model successfully imitates the intended system and thus that it is a basis for more comprehensive designs, and, eventually, for implementations.

Designers need supportive tools while doing analysis. Simulation helps the designer complete execution traces and helps in finding deadlocks or other unexpected behavior. Model checkers and theorem provers are deductive tools that help in the formal verification of properties. TLC model checker is extremely useful and powerful for verification and validation. For TLA+ models, TLC model checker can explore the state space of finite-state instances and compute results. Besides the model, TLC needs a configuration file as a second input file for defining the finite-state instance of the model to be analyzed, which states and which properties need to be verified over that finite-state instance, and which of the model's formulas represent the system specification.

In the following chapters, we will work on many systems' applications using TLA+ and its TLC model checker.

1.5.3 Why TLA+

We decided to use TLA+ for many reasons. First, it possesses strong validation features which are useful for many different kinds of applications/systems which we will illustrate in section 1.6 with more details. Second, it supports abstraction and works on the design level, which gives it the ability to build a more precise system with more confidence. Third, it's relatively easy to learn. Forth, it has a powerful model checker (TLC). Fifth, it's mathematically based and allows you to specify the system and its properties. Fifth, we can use its PlusCal language mode that looks very much like C. Sixth, we can check for safety and liveness properties, which are critical properties for many systems. Finally, TLA+ has a powerful TLC model checker that can check for specifications and supports overall system analysis.

In recent years, system designers have extended UML, originally designed to model software systems, to mixed hardware-software systems where each component is a black box which ultimately will be instantiated as hardware, software, or a combination. TLA+ can also been extended to model systems of mixed hardware-software components, as explained, for example, in [83].

1.5.4 TLA+ vs. Other Specification Languages

There are several different specification languages that had been used in industry. These include Alloy [38], Microsoft VCC [39], B [21], Z [19] and TLA+ [18].

In comparison to Alloy [38], TLA+ is direct, simple, and does not require so many layers of identifications in modeling nested structures. TLA+ is a more expressive language compared to Alloy, whose expressiveness is limited. TLA+ supports high-level functions, and it's a flexible language when there is a need to edit the specification details, compared to Alloy, which is less flexible and does not support some high-level functions like recursion. On the other hand, Alloy Analyzer model checker is very efficient compared to TLC model checker and it has the ability to handle important large analyses which TLC is unable to handle. Although Alloy Analyzer is faster than TLC, it crashes, or hangs in some cases when analyzing larger systems.

In comparison to TLA+, VCC allows writing "ghost code" which is a superset of the C programming language, but the downside of it is that it's much more verbose than TLA+. TLA+ has its efficient TLC model checker that is able to handle huge state-spaces at a very good throughput rate [4]. In addition it can efficiently use multiple cores instead of using single

memory like B, VCC, and Event-B model checker. TLA+ has an excellent feature called Trace Explore that allows tracing every single state and finding the results for each state, which makes tracing, finding, and fixing bugs much easier. In comparison to Alloy which has this same tracing feature, it does work for systems with only a few steps or variables (small systems). In comparison, TLA+ supports the liveness property (which means that something good will eventually occur) better than any of the other specification languages. TLA+ is so powerful because it is a very expressive language since it uses a mathematical formula [4].

In the following section we describe examples from the literature which show the use of formal methods (especially TLA+) for verification of different system applications. This section provides a literature survey of the use of formal methods to verify and validate different systems applications. The sub section 1.6.2 contains a literature survey for modeling different systems applications using the TLA+ specification language.

1.6 Related Work

1.6.1 Related Work Using Formal Methods

Using formal methods for verification has become more frequent recently because of the efficiency, reliability, and quality of testing for finding errors and bugs that are hard to detect easily. This section will illustrate work using formal methods for verification and validation with the focus on the TLA+ specification language.

In 2006, Zafer et al. [23] applied the specification language Z to an automated train control system for the specifications of critical components of the system. First, they modeled the static

components of the system using graph theory, then they described the entire state space by integrating those components using Z notation. To model topology in graph theory, real topology was transferred and then crossings, switches, and level crossings were formalized. Finally, they interatred these components to define the entire interlocking system. They used Z notation to describe formal specification of the system, and the Z/EVES tool to analyze the model.

One of the objectives of this research was to prove the power of formal methods and show how to apply it to complex systems instead of only to simple systems. Another objective of this work was to use Z notation at an abstract level to model the railway interlocking system. Even though this work does not represent a real-world problem, it shows the power of applying formal methods (Z notation in this case) to complex safety critical systems successfully and is a good example for all researchers who are interested in applying formal methods. This research is beneficial for the railway industry because it does not focus on a particular system, it focuses on general concepts and principles of an interlocking system. They used Z notation for integrating the system, formalizing graph theory, which was not an easy mission, and finally they analyzed the specification using the Z/EVES tool.

In 2015, Afzaal et al. [24] worked on improving the wireless sensor and actor network (usually "activator" term used instead of "actor" network). This field has been attractive to researchers and has seen a lot of improvement in modeling recently. But still there is a need to work on big challenges in this field because of the critical large-scale applications and the safety and security critical aspects. In their work, Afzaal et al. designed a model for a Subnet Based Backup Assigning (SBBA) algorithm that partitions the wireless sensor and actor networks (WSAN) into subnets.

This work classifies critical and non-critical nodes in each subnet. Each critical node assigns a suitable backup to observe the primary critical node and inter-actor connectivity is preserved within the subnet. Also, in each subnet, a gateway node is selected to communicate with other subnets and this gateway node assigns a suitable backup to observe the primary gateway node and inter-gateway connectivity that is defined among subnets. In order to verify and validate the proposed algorithm, they used the VDM-SL formal approach to analyze and do formal specification of the SBBA algorithm in WSANs. First, they modeled a subnet of WSAN as a dynamic graph and used VDM-SL to implement SBBA in a formal specification. They successfully analyzed, verified, and validated the SBBA algorithm specification using the VDM-SL toolbox.

In 2017 Kamali et al. [25] focused on self-directed vehicles in their search. They stated that multiple autonomous vehicles will most likely be coordinated into platoons or convoys on our highways and that this is likely to occur in the near future. So the behaviors of the autonomous vehicles in platoons should be certified before deploying these platoons. This is not an easy mission, and it needs more than current certification requirements.

This work showed how formal methods can be useful to analyze increasingly autonomous, new systems. They represent the vehicle platooning as a multi-agent system where each vehicle carries out “autonomous decisions” that will be captured by the other agents. They used formal verification to guarantee that the safety requirements will never be violated by these self-directed decision-making agents in vehicle platoons. They verified the individual agent's code using formal methods. However, they didn’t scale it to the full system. Thus, they combined two approaches because the primary verification of autonomous behavior was not captured by the global system verification technique. This allows safety requirements verification of a model of

the system and the actual agent code used to set up the autonomous vehicles. They verified the agent behavior using AJPPF, and they used Uppaal model checker to verify the real-time requirements, where a system is represented as timed automata generated by a translation algorithm.

In 2019 Rehman et al. [22] modeled and verified a smart office system. Because people spend 8 to 12 hours working in their office every day under normal circumstances, increasing performance and efficiency of the employees is an important issue. The modeling of smart offices supports visualizing and understanding of an office management system from many viewpoints. The modeling and verification are done using UML diagrams, a formal specification language VDM-SL**,** and automata theory. UML diagrams are used for behavioral and operational models. In term of states, automata-based models are used to model the behavior of the system. And finally, formal methods are used to achieve reliability, accuracy, and consistency of the smart office system. For verification and validation of the model, they used the Vienna Development Method and Specification Language (VDM-SL).

Figure 1 shows the system's transaction flow from one state to another state.

In this work, they stated the behavior of the system using formal methods in a format of mathematical notations. And then they used the VDM-SL toolbox to verify these mathematical notations. The system properties are validated and verified using the VDM-SL toolbox to prove the correctness of the model.

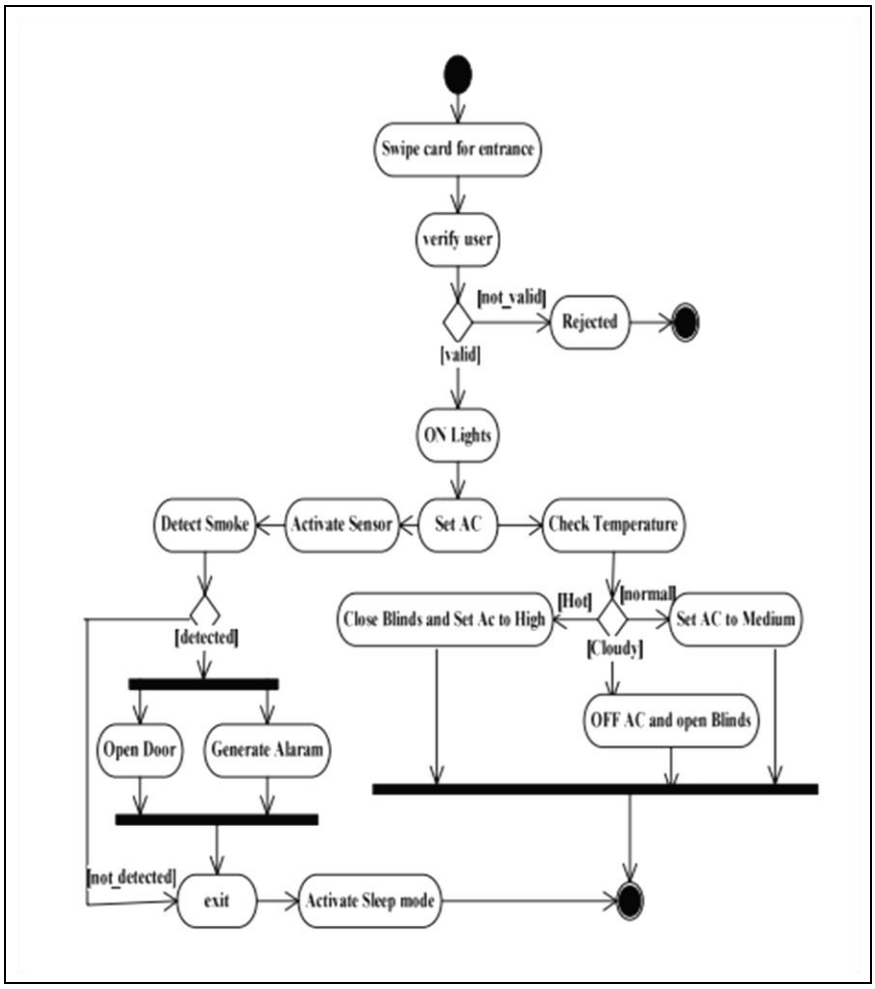


Figure 1: Activity Diagram for Smart Office [22]

1.6.2 Related Work Using TLA+

The examples above show the use of different specification languages for formal modeling. The next examples focus on using the TLA+ specification language for verification and validation for different kinds of applications.

In 2002 Tasiran et al. [8] worked on examining functional correctness during simulation, formally analyzing simulation runs and guiding them towards coverage gaps automatically**.** They verified the Compaq Alpha 21364 microprocessor's cache coherence engine. This work was a

collaboration with Intel Corporation, Microsoft Research Center, and Compaq Systems Research Center. Implementing a complex protocol on hardware and verifying the consistency using high-level specifications is a process that is very labor-intensive and impossible to be completed in practice. Hand-written test programs or random patterns simulation are the only existing tools for this purpose. Usually, a hardware description language is used to describe the design and a text document is used for high-level specification. During simulation, high-level specification is checked for violations by a program. This approach is not enough for a number of reasons. First, it is difficult to verify whether the specification is complete and consistent since it is informal. Second, the code itself that is written for checking for specification violations may have errors. Last, and most important, it is hard to quantify how well different features of the specification have been discovered, and to guide simulation runs towards unmapped areas. In this work, the authors used a formal language to write the high-level specification. In the implementation, they mapped the related simulation steps to state transitions in the specification using TLA+. And they used the TLC model checker to check the consistency of each state transition.

In 2006 Narayana et al. [6] used TLA+ with network protocols to check Denial-of-Service attack (DoS) vulnerability. Many researchers have proposed formal methods for vulnerability analysis and most current work focuses on security properties like correctness of authentication and perfect forwarding secrecy. The challenge is how to apply these approaches to analyze more challenging vulnerabilities like DoS attacks. In order to address this challenge, Narayana et al. proposed using TLA+ for checking DoS vulnerability automatically with completeness guarantee by developing new schemes to model attackers' abilities for finding real attacks and avoiding state space explosion while property checking. They successfully identified threats to IEEE 802.16 air interface protocols as their case study. They specified a network protocol using TLA+,

security properties to be checked, and an attack model. They used the TLC model checker to check the whole protocol state space and to find any possible attacks. This method can identify the attacks as well as faulty situations, and it can also guarantee completeness of the analysis. They showed how using TLA+ and TLC help with protocol design and improvements such as programming a modification or fix into TLA+ when a vulnerable design or fault is detected, then re-running TLC for problem solving verification. This work represents a first step towards checking network protocol vulnerability automatically with correctness and completeness guarantees.

In 2010 Zhang et al. [3] used TLA+ to express time specification. They presented a pattern-based method and introduced RealTimeNew which is a real-time module that contains commonly used time pattern definitions. A general framework was presented to differentiate system functionality with time constraints from the temporal characterizations. This method demonstrates the use of TLA+ in verifying and specifying time-sensitive systems. The real time module RealTime had been developed by Lamport to make time modeling applications easier to implement [7]. In time-sensitive systems, this module is not enough because it cannot specify the time intervals between actions, it only can specify the duration time of an action.

It is important to understand the difference between RealTime and RealTimeNew. The RealTime module is designed to be used in specification composition, while the RealTimeNew module is designed in to be used in a single specification, which means that it can be verified directly by deductive verification or the TLC model checker. RealTimeNew also can then be applied to more applications because it contains richer time patterns. The time formulas are divided into four types: the time interval between actions, the time duration of an action, the advanced time patterns, and time evolving. Zhang et al. used their module in a case study to demonstrate and

validate the module. The application used a simple answering machine case and they achieved a good result.

Also, in 2010, Zhang et al. [1] proposed a method using TLA+ to specify programmable logic controllers (PLC) systems formally. The framework of specification is generic. It separates the description of the controller itself from the environment and the PLCs' scan cycle mechanisms are consistent with the controller's structure. Specifications can be represented by a number of replicated components. In [1] they showed that TLA+ structuring mechanisms help to obtain well-organized, configurable, and clear specifications that they were able to verify using the TLC model checker.

As a case study to demonstrate their approach, they picked a controller for firefighting equipment in a ship dock. They used TLA+ for specifying the reaction cycle referring to the PLCs' scan cycle mechanism. A generic specification pattern was obtained per module that differentiates between actions of the controller, the user, and the plant feedback. These different modules are used to describe the overall system specification and the pattern is defined for a concrete PLC. The dock fire-fighting system that is shown in Figure 2 is a system used to fight fires that might happen at ship docks. The user controls the fire-fighting equipment and receives information about the present operating state. The dock has two berths and two water cannons that may be used only for firefighting. The pump supplies the cannons with water from a water tank. Different components are connected by several valves, for example, cannon1 can only be used for firefighting if both valve1 and valve2 are opened.

They successfully developed a format for the system specifications using TLA+, and successfully verified their work using the TLC model checker, which analyzed the results in a

reasonable time compared to other commonly used methods. They also noted the potential to apply their methods to additional real-time PLC system specifications in future.

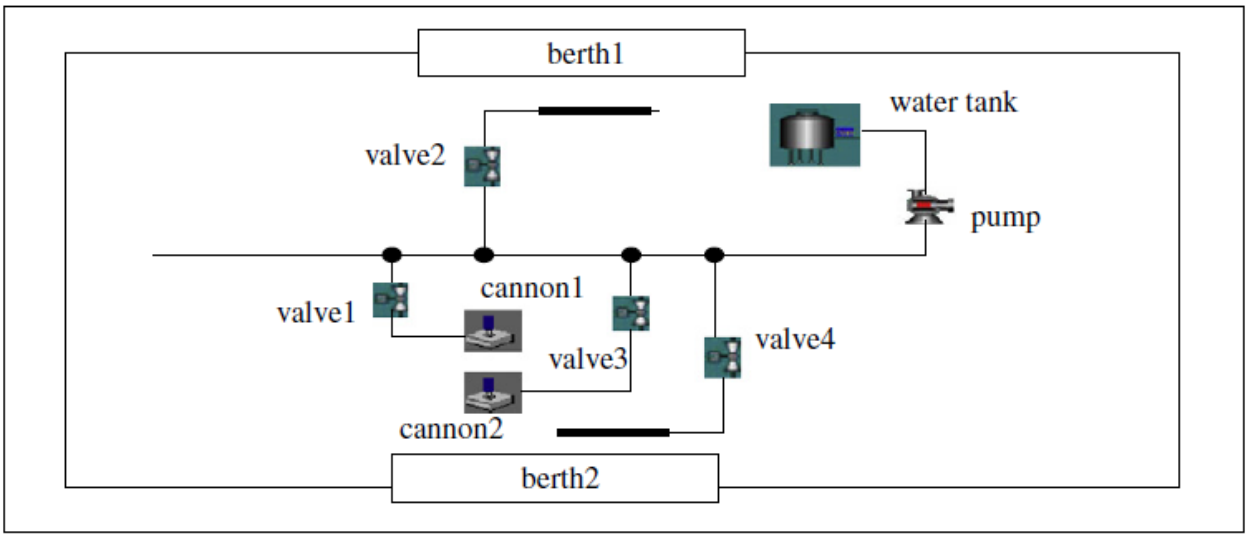


Figure 2: The physical configuration of the dock fire-fighting system [1]

In 2019 Latif et al. [26] used UML to describe an automata model, along with formal methods to propose a model for a smart parking system. The system represents smart objects that share information within a network by sensing, communicating, and also sending this information to additional IoT devices for analysis. They used UML based models to describe a real-world parking system and to show the working flow of the system. Then they converted these UML diagrams to an automated system using automata models which define the smart mechanisms of the parking system. Their model is represented by states and transitions, where every state is defined functionally and has a unique identity. Figure 3 shows the activity diagram for the smart parking system. There are many models for many operations in this system, for example, search shortest path towards empty slot, find free spaces, and car entrances and exits within a region.

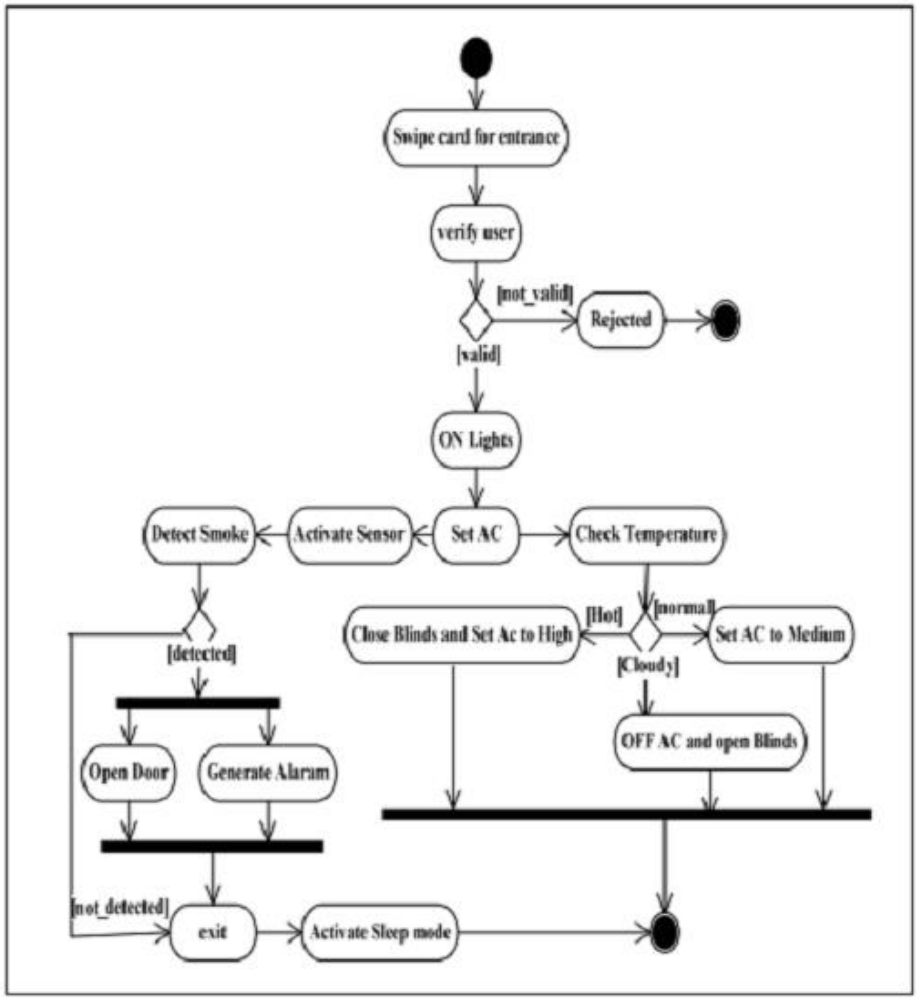


Figure 3: The Activity Diagram of the Smart Parking System [26]

They used the formal specification language TLA+ to verify their proposed model and the TLC model checker to capture the system behavior. They successfully integrated UML, finite state automata, and the TLA+ language to provide proof of correctness, verification, and validation of the proposed system.

In 2014 and 2015, Newcombe et al., who were working at Amazon, wrote an article [2] and a more detailed paper [4] explaining why they were using formal methods (in particular, TLA+) at

Amazon. At Amazon, they write their own complex algorithms that often deal with more than 1 million requests/sec. Verifying the correctness of these algorithms is very difficult. In 2011 Amazon started using formal methods (TLA+ as a specification language) in place of other verification techniques for many reasons, including:

- Formal methods are able to find bugs that cannot be found with any other technique used to verify system designs.
- Amazon found that they could efficiently and routinely apply formal methods to complex real-world software designs in applications such as public cloud services.
- They also found that mainstream software development is surprisingly feasible with formal methods and returns a good gain on investment.
- Over the lifetime of a system, formal specification writing pays dividends.
- They were able to use TLA+ on 10 big complex real-world systems (as of 2/2014);
- Formal methods can precisely describe the abstract design and its abstract operating environment.
- Formal methods can define correctness properties and specify what the system must do by applying:
    - Safety: which represents what the system is allowed to do. For instance, always; all committed data is correct and present.
    - Liveness: which represents what the system must finally do. For instance, whenever the system gets a request, it eventually must respond to that request.
- There is a second language accompanied by TLA+ called PlusCal, which is closer to the C programming language style but more expressive. It is used by Amazon's engineers as well.

Amazon implements many advanced distributed systems that process and store customers' data. To protect this data, Amazon depends on the correctness of an ever-growing set of algorithms in the areas of consistency, replication, fault tolerance, auto-scaling, concurrency-control, and other coordination activities. This challenge to achieve correctness in these areas led Amazon to adopt formal methods. A significant value has been added by TLA+, because it prevents serious tricky bugs from ruining production and also provides enough confidence and understanding to make amazing performance optimizations without losing correctness. Table 1 [4] shows an example of applying TLA+ to Amazon's more complex systems. At Amazon, sometimes it is necessary to verify not only individual algorithms themselves but also the interactions between algorithms.

System	Components	Line count	Benefit
S3	Fault-tolerant low-level network algorithm	804 PlusCal	Found 2 design bugs. Found further design bugs in proposed optimizations.
	Background redistribution of data	645 PlusCal	Found 1 design bug, and found a bug in the first proposed fix.
DynamoDB	Replication and group-membership systems (which tightly interact)	939 TLA$^+$	Found 3 design bugs, some requiring traces of 35 steps.
EBS	Volume management	102 PlusCal	Found 3 design bugs.
EC2	Change to fault-tolerant replication, including incremental deployment to existing system, with zero downtime	250 TLA$^+$ 460 TLA$^+$ 200 TLA$^+$	Found 1 design bug.
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence. Failed to find a liveness bug as we did not check liveness.
	Fault tolerant replication and reconfiguration algorithm	318 TLA$^+$	Found 1 design bug. Verified an aggressive optimization.

Table 1: Examples of applying TLA+ to some of Amazon's complex systems [4]

Amazon also evaluated other formal methods such as Alloy, which was not a good fit for their kinds of problems, and Microsoft VCC, which is good for low-level C programs, but whose abstraction features do not work well for verifying high-level designs. They also evaluated other formal methods such as Event-B, Coq, and PVS but after all these evaluations, they found that TLA+ was the best fit for their needs; it solves their problems and is simple to apply, simple to learn, and flexible. And on top of that, the TLC model checker works very well.

In section 1.6.1 and 1.6.2, we discussed many examples of applying formal methods in general and TLA+ in many different areas and applications, and we illustrated how useful it is to apply formal methods to verify a system. As part of our work focuses on using TLA+ to verify and validate different system applications, we will also discuss additional results in this area in the following chapters.

Chapter 2: Smart School Building System

2.1 Introduction

In the previous chapter, we illustrated how formal methods are efficient in designing different systems applications. We illustrated with many reasons why we picked TLA+ as the formal specification language to use in our work. In this chapter we will show how we used TLA+ and its TLC model checker to design a smart school building system. In the following chapters, we will illustrate how we used TLA+ to design different systems applications.

In our work, we did not focus on just designing a system, we focused on having a "safe" system, because we are dealing with human lives as part of our smart school building system. So safety is a very essential quality feature that we made sure to achieve in our designs. Also, achieving security is an important goal for us, since the secure system design will tend to increase the probability of safety also.

In this chapter we first provide a literature survey about previous work on designing school systems. Second, we provide an example of how to use TLA+. Then we will explain our work in designing a smart school building system (version I) using UML, TLA+, and the system validation using the TLC model checker. Finally, we will describe our work on the improved smart school building system (version II) using UML, TLA+, and the TLC model checker. In addition, we will show how the TLC model checker can find a security bug in the model.

2.2 Work Related to a Smart School Building System

In 2017 Pocero et al. [30] investigated improving energy efficiency in public school buildings. They presented an IoT hardware infrastructure that provides real-time monitoring in the school

buildings. Their system addressed some requirements related to collecting energy consumption and environmental data from school buildings, having a sensor network all around the school building (inside and outside) to monitor human comfort parameters, and ensuring that the sensor network used low-cost devices, that the IoT infrastructure would be extendable for new improvements in the system, and that hardware modules would be interoperable with the educational sector platform. Their design plan was based on an environmental comfort meter, a power consumption meter, and a set of IoT nodes that communicate through IEEE 802.15.4 [44].

In 2015 Amaxilatis et al. [31] designed a platform that allowed actuation and monitoring in many school buildings. This was based on research that was applied to a group of 12 school buildings in Greece. They installed IoT devices with a goal of achieving a more energy-efficient infrastructure and improved measurements in environmental parameters. They followed the approach outlined in [30], developed many user interfaces to provide tools for both the administrators and the educators, and applied their work to a real classroom in Greece.

In 2015 Borgan et al. [32] described an innovative methodology for energy management decision making in school buildings. It promises energy efficiency and energy savings. The project, called VERYSchool, was funded by the European Commission under their Competitiveness and Innovation Program. Based on the ISO-50001 standard [41], VERYSchool demonstrated effective energy action management and successfully connected many smart systems such as smart control functions for lighting and HVAC, smart meters, energy simulation modeling, and a complete energy action web-based navigator platform system. This innovative project had significant positive energy, socioeconomic, and environmental impacts.

In 2011 Hirsch et al. [33] and in 2016 Veeramanickam et al. [34] illustrated the importance of using IoT technology to create a smart campus for universities and to improve and support e-

learning. They also illustrated some of the challenges in applying this approach to an entire campus.

A few researchers modeled smart school buildings which focused mostly on power management and power consumption [30-32]. Others modeled a smart campus system [33,34] which focuses on E-learning, mobile-learning domains, cloud learning, and an environmentally aware campus. Our system is somewhat different due to differences between a university campus and an elementary school.

Our goal is to have a smart school building system that is safe and secure. Safety means, for example, having a smart building that uses IoT to connect sub-systems like smart lighting, smoke detection, and HVAC systems together. Secure means, for example, having a smart building that guarantees security for each student and employee in areas such as entering the system and having a secure file for each student and employee.

As we mentioned before, we will focus on TLA+ in our work. To learn TLA+ syntax and how to use TLA+ to solve problems, we used its tutorial [27] and we read the information in [5] and [7] as well.

TLA+ is a high-level (at the design level, above the code or hardware) modeling language that is used for modeling digital systems (including computer systems, algorithms, and programs). It has many tools to check these models, including the TLC model checker, which is the most important tool in TLA+. In order to model using TLA+, you need to think in a different way. You need to simplify the system and think abstractly. Abstraction means simplifying by removing details. We cannot get complex systems right without understanding them. Abstraction helps us understand them, and TLA+ helps us understand abstraction. TLA+ can

specify high-level designs and algorithms. Specifications mean high-level models. Architecture represents a high-level specification, higher than the code level. TLA+ specifications represent the architecture, and we use TLA+ tools to debug this architecture. TLA+ was designed for modeling distributed and concurrent systems [42]. It can find and correct hard design errors that cannot be found by testing before writing the code or implementing the hardware. It is used to check the design precisely. TLA+ can also reduce the code size. Using TLA+ to develop the operating system OpenComRTOS in The European Space Agency's Rosetta spacecraft [28] reduced the code size by a factor of 10, which is so crucial in such applications. We use TLA+ to ensure that the system is "working right", which means the system satisfies specific properties (these properties represent conditions on individual executions). TLA+ considers the system execution to be a sequence of discrete steps. TLA+ describes these steps as state changes. Some people describe their systems using state machines [9] and TLA+ is an extremely expressive, elegant language for describing state machines.

2.3 Example Using TLA+

In this section, we will illustrate an example using TLA+ in order to understand more details about TLA+ and how it works. This is a method Lamport used to explain how TLA+ models the states of a system [7].

TLA+ can be used to solve specific problems that might arise in daily life. Our first example is simple but interesting challenge; if you are given a 3-gallon jug, a 5-gallon jug, and a water faucet, how can you put exactly 4 gallons of water in a jug? The informal solution for this problem is as follows: we start with 2 empty jugs (3 and 5 gallons), then we fill out the 5 gallon

jug with water, then empty into the 3 gallon jug, then we will empty the 3 gallon jug into the 5 gallon jug, then we will fill the 3 gallon jug and empty it again into the 5 gallon jug, so we will have 5 gallons full and 1 gallon of water in the 3 gallon jug, then we will empty the 3 gallon jug (which has 1 gallon of water in it at this level) into the 5 gallon jug, and finally, we fill the 3 gallon jug with water and empty it into the 5 gallon to get exactly 4 gallons of water in the 5 gallon jug. To solve this problem formally, we will use TLA+. TLA+ works on any problem by defining a sequence of states (behaviors). Figure 4 shows the TLA+ specification of this problem.

As shown in figure 4, we created a module called JugModule. In this module, we declared the variables small (for the 3-gallon jug), and big (for the 5-gallon jug). We declared the initial state formula (Init) to give initial values of zero for small and big. We declared (TypeOK) invariant which represents a formula that asserts type correctness that is checked by TLC to be always true (asserts that each variable has a reasonable value in our example). And then we declared each possible step and had them all executed (one at a time) in the next state formula (Next) which describes all permitted steps. In TLA+, we must declare Init and Next state-formulas in every specification.

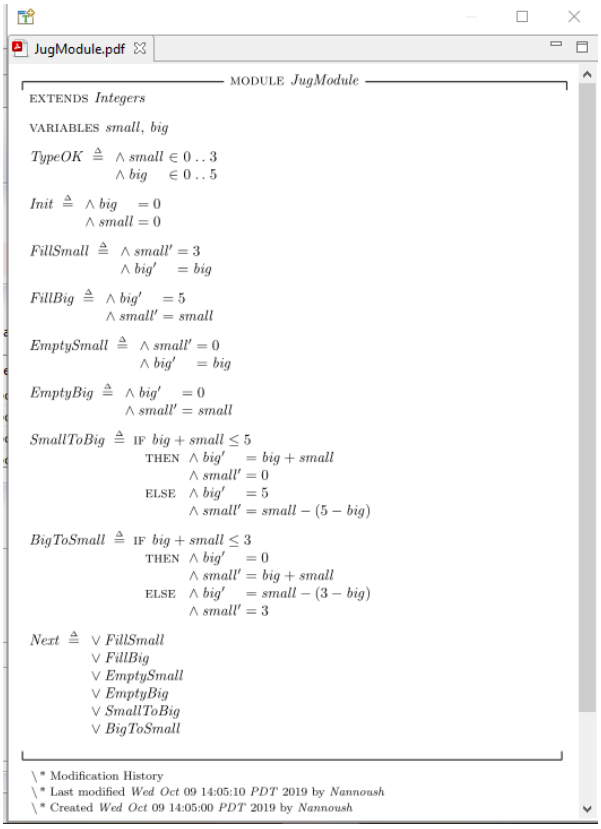


Figure 4: TLA+ Specification for jug problem

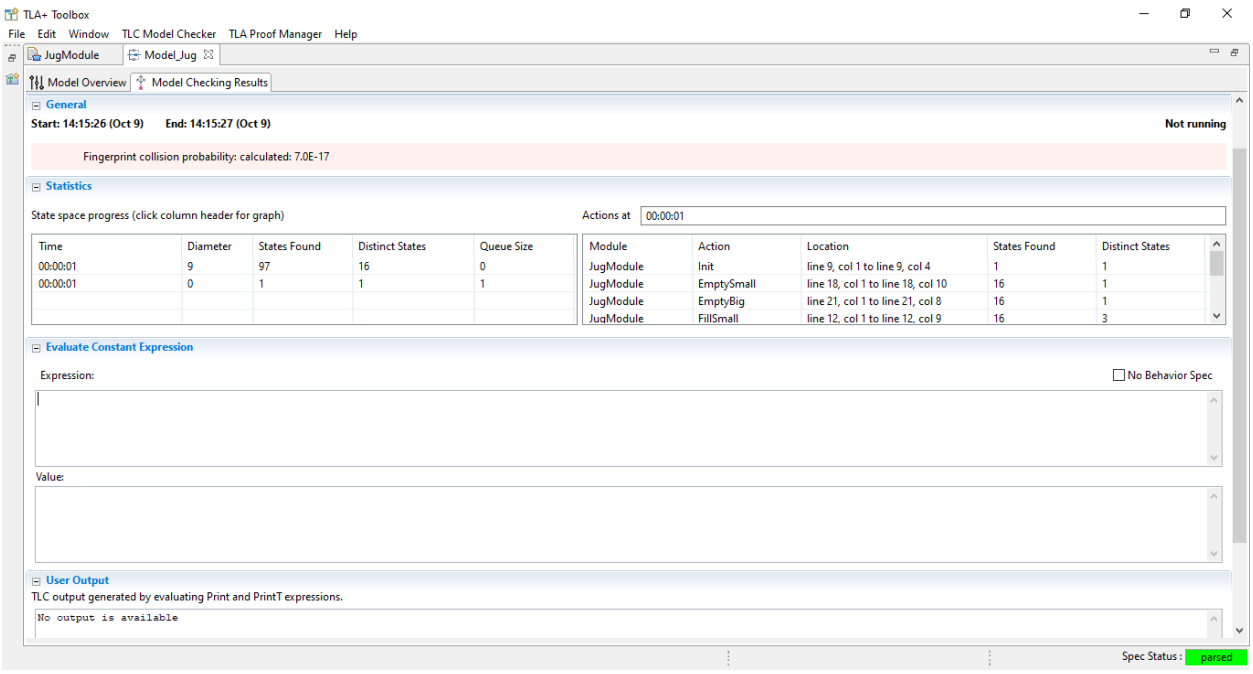


Figure 5: TLC Model Checking Results for JugModule problem

To check our module, we run the TLC model checker that computes all possible behaviors allowed by the specification. It also checks type correctness, which means every reachable state satisfies the formula TypeOK. Figure 5 shows the model checking results page in TLC with no errors; 16 reachable states are found using behavior allowed by the spec as shown in the figure under "States Found" column.

The following section illustrates a more realistic use of these tools. It explains how we specify a smart school building using the Unified Modeling Language (UML) and TLA+.

2.4 Smart School Building Model

We capture the smart school system objects, sequences of actions, and system behavior using the Unified Modeling Language (UML) [29] and TLA+ [7, 18]. UML is an informal object-oriented modeling method. It provides different diagrammatical modeling techniques such as object interaction diagrams, state diagrams, and object diagrams. It encompasses many diagrams, presentation conventions, and notations that have emerged in the structured methods and object-oriented domain. UML is defined as a set of graphical models that express different properties of an object-oriented design. The two most important model types are the behavioral and structural models [29]. In this work we use behavioral models which represent the dynamic behavior of the system.

Modeling a smart system requires an integration of different sub-systems to have a complete smart system. The verification and validation of such a smart system is not an easy thing to accomplish. There are many methods for verification and validation. Using formal methods is one of the ways welcomed by many researchers due to their effectiveness and the ability to

design almost bug-free systems. We have used formal specification of our proposed smart school building model. We use the TLA+ specification language for this purpose. The system automation relies on several different kinds of sensors. Using TLA+ allows us to represent all system operations as in [35] which represents the smart school building system (version I), and in [45] which represents the smart school building system (version II). In version II, we used the TLC model checker to validate the TLA+ module and then to find a security bug in the module.

Our system is mainly for elementary schools which are of medium size (with average of 450 students) with the ability to be enlarged to become a big school (more than 600 students). The reason we chose an elementary school is because this design is an initial design of such a system and focuses on the overall basic smart school system design without too many complicating details. We worked on achieving many system design goals such as having a "smart" system which includes many sub-systems working together as one smart system, having a "safe" system design, having a secure login system, and controlling the power usage in the building via controlling the lights and the HVAC sub-systems.

The smart school building system has integrated sub-systems that work together to achieve the design goals. The system has a secure login sub-system that requires the employee, student, and visitor to enter a valid username and password if they want to enter the school building. These usernames and passwords are given through the school district based on their own criteria which may be based on the registration numbers, names, grades, ...etc. Once a person enters a valid username and password, the main door will open automatically for him/her to enter. If the username and/or the password is invalid, the login sub-system will give the person the option to re-try entering them. The person will always have the option to contact the reception employee to check if there is any problem in the login sub-system and, if so, to provide an alternate

identification method to let the person enter the school building once he/she has an identity authentication. Once the first person enters the school building, the lighting and HVAC sub-systems will automatically work. The lighting sub-system will work by taking the data of the light outside the building via a light sensor. If the light sensor senses that its clear outside, the sub-system will turn on the lights inside the building in a low_mode which means that the lights will be reduced. While, if the light sensor senses that its cloudy outside, the sub-system will turn on the lights inside the building in a high_mode which means that the lights will work using the normal level of power. This smart sub-system is designed this way to control the power in order to have efficient power consumption, which is one of the goals of our smart school system design. (In practice there will be multiple subsystems, since different areas will have different exposures to external light and therefore different lighting needs).

The HVAC sub-system will automatically start working once the first person enters the school building, along with the lighting sub-system. The HVAC sub-system takes the input data from temperature sensors inside the building. If the temperature sensors sense that the temperature inside the building is 69 F or less, the HVAC sub-system will turn on the heat in the building. If the temperature sensors senses that the temperature inside the building is 74 F or more, the HVAC sub-system will turn on the AC in the building. This smart sub-system is designed this way to control the power in order to have efficient power consumption, which is one of the goals of our smart school system design. (In practice, there may again be multiple subsystems, as with the lighting).

The fourth smart sub-system is the smoke detection sub-system. This sub-system is different than the other sub-systems since it is working all the time (not only when there are people inside the building). This sub-system takes its input data from smoke detection sensors which sense

smoke/fire in the school building and once they sense smoke, a fire alarm will be turned on and all exit doors will automatically open to let all people to leave the school building immediately. This sub-system works 24/7 to enhance safety in the school building. This smart sub-system is designed this way to enhance safety in the school building, which is one of the goals of our smart school system design.

We designed two models of this smart school building system. The initial design model of our work (version I) was published in [35] and the improved design model (version II) was published in [45]. The initial model [35] is described in section 2.5. The improved design model is described in section 2.6.

2.5 The Initial Smart School Building Model (version I)

2.5.1 Unified Modeling Language (UML)

This section represents our designs using UML. UML stands for Unified Modeling Language that is used to model the system's abstract model. It used to capture system properties by providing graphical notations [29]. Figure 6 represents the smart school system entities which are the variables of the initial smart school model. The system inputs are taken from three different sensors, the light, the temperature, and the smoke sensors. Also, the username and password for anyone desiring entry is an input through the login sub-system. The system actions are represented as outputs based on the inputs; temperature in the building, light brightness, valid/invalid login, or smoke detection.

The activity diagram of the system is shown in figure 7. The figure makes the system easier to understand by showing the sequence of actions. First, school employees, students, and visitors

enter the school. Each employee or student has a unique username and password. If they enter the username and the password correctly, the main door will open for them to enter. Visitors need approval from the reception employee to enter the building. In addition, if the visitor is allowed to enter the building, the reception door only will open. In case the visitor is not allowed to enter the building, all doors will remain closed. All sub-systems will start working automatically when the first person enters the building. Otherwise, if there are no people in the building, all sub-systems will be turned off. In the model designed initially, the temperature sensor senses the temperature in the building and the system will automatically turn the heat on in the building if the temperature is below 68 °F and will automatically turn the AC on if the temperature is above 73 °F. The lighting sub-system will control the lighting inside the building. If its cloudy outside, the lights will be turned on in the high_light mode, and if its sunny outside, the lights will be turned on in the low_light mode. The lighting sub-system works in this way to have low power consumption in the system. The smoke detection sub-system will sense the smoke in the school building and in case there is fire or smoke, it will automatically send an alarm and open all doors in the building for exiting to ensure safety in the school building.

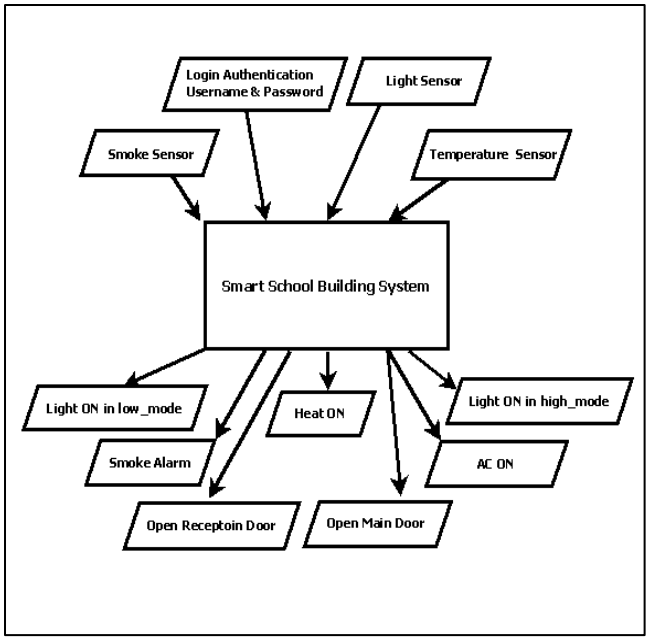


Figure 6: Smart school building system inputs and outputs [35]

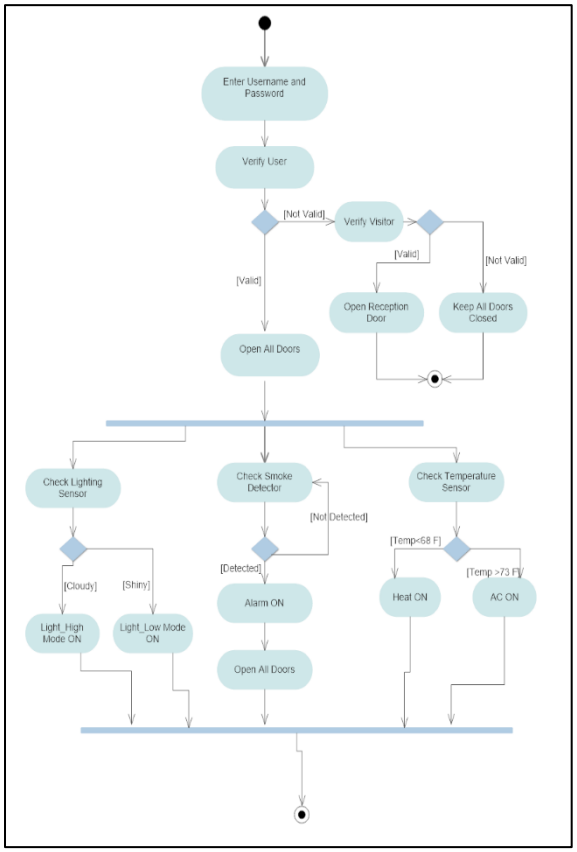


Figure 7: Activity diagram of the smart school building system [35]

2.5.2 Formal Specifications Using TLA+

The formal specifications for the initial smart school model are illustrated in this section. We used the formal specification language TLA+ which lets us to represent all system operations.

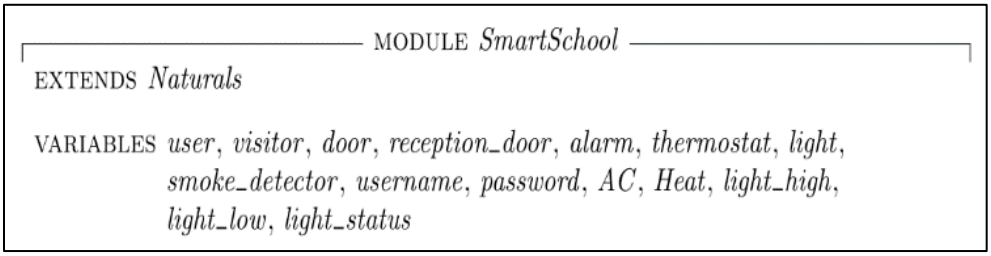


Figure 8: SmartSchool module with variables [35]

Figure 8 shows the top module of our initial smartSchool system. The module has a set of variables as shown in the figure. For example, *user* variable represents any person who may enter the school building and it has three values (employee, student, and visitor). In this initial model, the *visitor* does not need a username or password to enter the school, he/she needs an approval from the reception employee in the school. Other variable examples in this module are *username* and *password* which must have a unique value. These values are usually given by aspecific criteria from the school district or the school itself based on the personal information of each employee and student, that's why they can't be shown in the module with specific values.

The system invariants represent the strictions and conditions that the system must follow. In any TLA+ module, we need to declare the invariants. For example, in this initial model the *user* must have one of three values (1 for student, 2 for employee, and 3 for visitor). Also, a *student* and the *employee* are not able to enter the school without having a valid *username* and *password*. Figure 9 shows the invariants in the module.

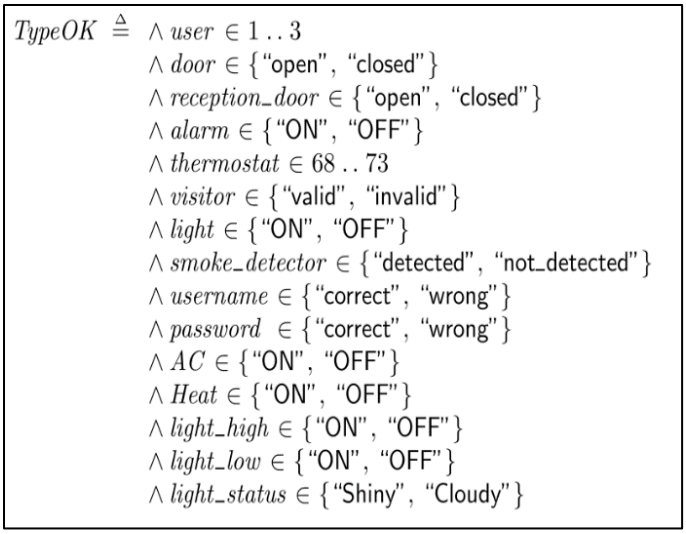


Figure 9: SmartSchool invariants [35]

The *Init* and *Next* functions are essential functions in a TLA+ module. As shown in figure 10, the *Init* function gives initial values for the variables in the module. For example, the *door* variable which represents the main door of the school building will be closed in the default case.

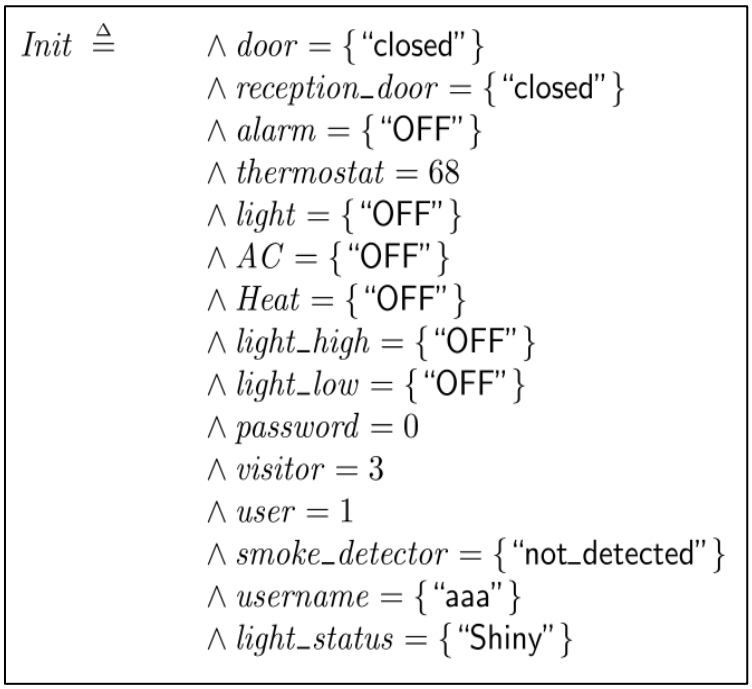


Figure 10: SmartSchool Init function [35]

Figure 11 shows the *EnterSchool* function. This function represents entering the school by the employees and the students. This function takes the input as the type of user (1 for student, 2 for employee) and the username and the password as well. The output is to open the doors for valid users.

$$
\begin{aligned}
EnterSchool \triangleq\ & \wedge user = \{1, 2\} \\
& \wedge username = \{\text{“correct”}\} \\
& \wedge password\ = \{\text{“correct”}\} \\
& \wedge door = \{\text{“OPEN”}\} \\
& \wedge reception\_door = \{\text{“open”}\}
\end{aligned}
$$

Figure 11: EnterSchool function [35]

Figure 12 shows the *VerifyVisitor* function. This function authorizes the visitors to the school. It will open the reception door for valid users only.

$$
\begin{aligned}
VerifyVisitor \triangleq\ & \vee \wedge user = 3 \\
& \quad \wedge visitor = \{\text{“valid”}\} \\
& \quad \wedge reception\_door = \{\text{“open”}\} \\
& \quad \wedge door = \{\text{“closed”}\} \\
& \vee \wedge user = 3 \\
& \quad \wedge visitor = \{\text{“not\_valid”}\} \\
& \quad \wedge reception\_door = \{\text{“closed”}\} \\
& \quad \wedge door = \{\text{“closed”}\}
\end{aligned}
$$

Figure 12: VerifyVisitor function [35]

Figure 13 shows *SetLight* function. This function starts working once the school open its door for the first person. It takes the input from the light sensor to determine the brightness of the lights in the building.

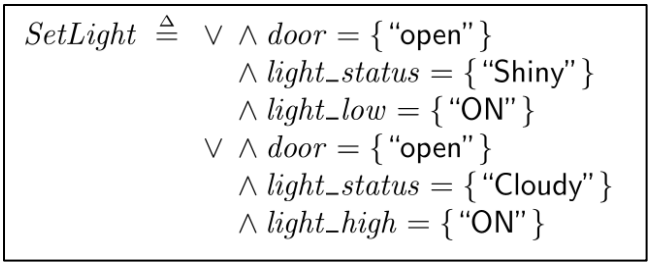


Figure 13: SetLight function [35]

Figure 14 shows the *SetTemptature* function. This function takes the input from the temperature sensor. The output depends on the temperature readings.

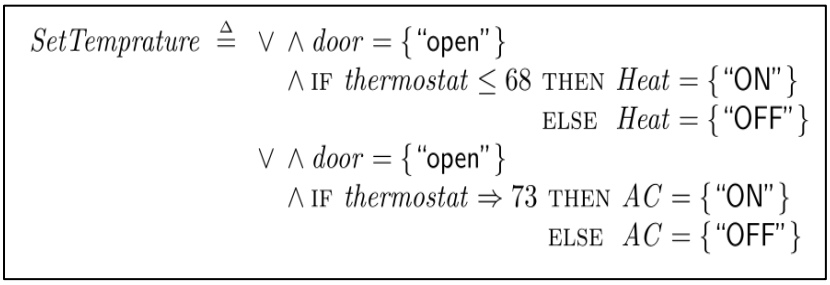


Figure 14: SetTemptature function [35]

Figure 15 shows the *DetectSmoke* function. This function takes its input from one or more smoke sensors. The output of this function is to turn on the fire alarm and open all doors in the building to let everyone in the building to exit immediately if smoke is sensed. This function enhances the safety in the system.

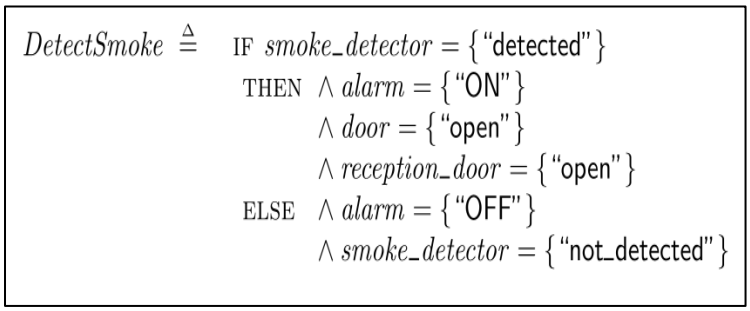


Figure 15: DetectSmoke function [35]

Figure 16 shows the *Next* function. It is an essential function in any TLA+ module. It represents the collection and execution of all functions to move to the next state in the system after initialization. It describes the choices for the next-state action.

$$
\begin{aligned}
Next \triangleq\ & \lor EnterSchool \\
& \lor\ SetLight \\
& \lor SetTemprature \\
& \lor DetectSmoke \\
& \lor VerifyVisitor
\end{aligned}
$$

Figure 16: Next function [35]

Figure 17 shows the *Spec* function. This function is the last statement in the module and is responsible for the main running of the system. That means it runs all system specifications. These specifications will be verified using the TLC model checker.

$$Spec \triangleq TypeOK \land Next$$

Figure 17: Spec function [35]

2.5.3 TLA+ Model Analysis

We used the TLA+ toolbox to write the initial smart school building specifications. One of the reasons why TLA+ is a powerful specification language is because of its powerful TLC model checker as we mentioned before in chapter 1. We used the TLA+ toolbox to verify that the

systems' specifications are correct and have no syntax errors. Figure 18 represents the parsed model. As shown in the figure, there is a green button on the right corner which means that the model is parsed correctly.

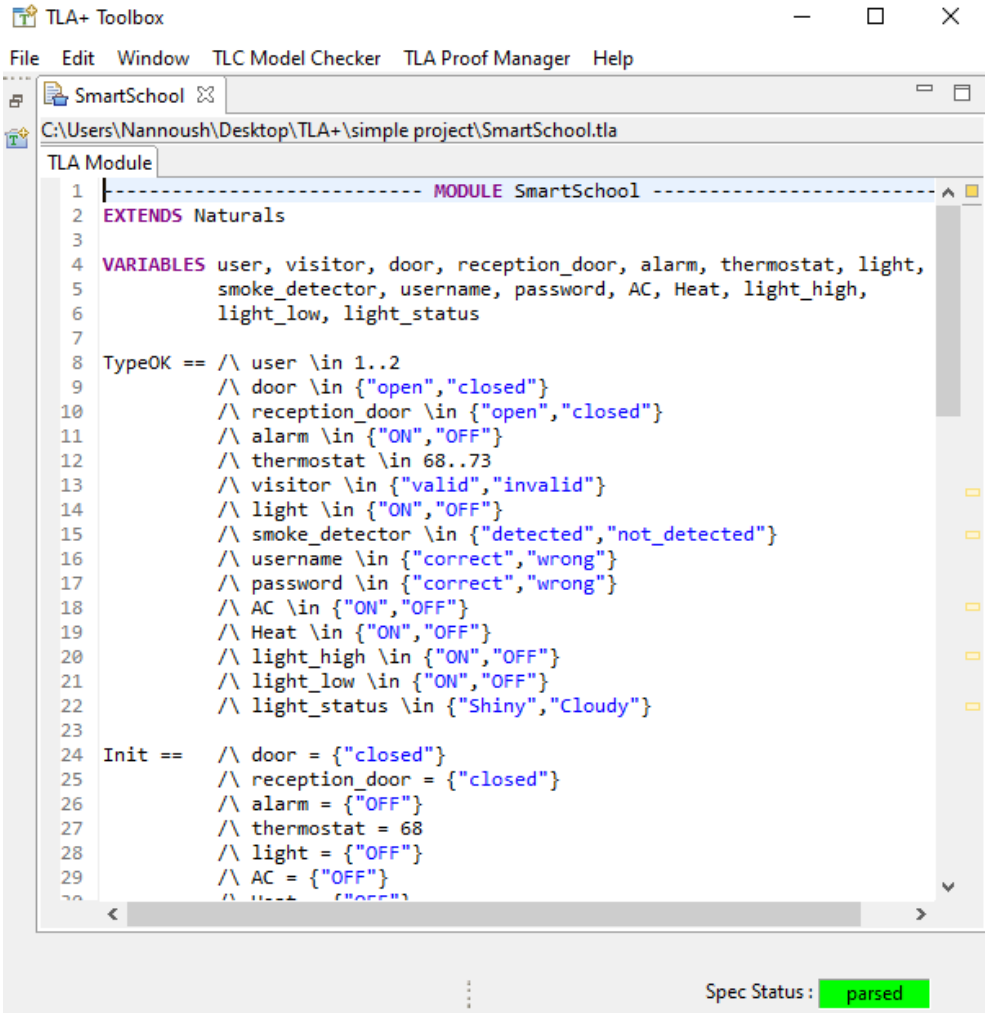


Figure 18: SmartSchool parsed model [35]

This model was not validated using the TLC model checker. Before using TLC to validate our system, we decided to improve the model first. In [35] we published the version I model as we mentioned before, and we published version II in [45] where it is shown to be validated using the TLC model checker. Section 2.6 will illustrate the improved smart school model (version II).

2.6 The improved smart school model (version II)

2.6.1 Introduction

Smart systems have been getting more attention by researchers recently since they are very useful and can make life easier. Smart systems contain functions of actuation, sensing, and controlling in order to analyze and describe a situation and make decisions based on the available input data to perform a smart action [47]. The challenge in these smart systems is to integrate sub-systems or materials together to build a smart system that works efficiently [47].

We improved our previous smart school building system model in [35], adding more requirements and new specifications. The improved model is better, more efficient, safer, and more secure.

Here's a list of the improvements in the new model:

- The new login system requires any person to have a username and password (even the visitors) which enhances security and safety in the system.
- The login system allows a "re-try" option to login in case a username and/or a password was wrong.
- To enhance safety in the system, we added "safe" invariant to the specifications to ensure that the smoke detection sub-system works all the time even if there is nobody in the building, and it sends a fire alarm immediately throughout the building and it can also send an alarm to the nearest fire station.
- To enhance safety in the system, we added a surveillance camera (or cameras) in the building.

- To enhance safety in the system, we added "surveillance" invariant to the specifications to ensure that the surveillance camera or cameras work all the time even if there is nobody in the building.
- The temperature thresholds were changed to reduce the power consumption in the building.
- The capacity of the school is determined to be small to medium, a size preferable for elementary schools. If we want to expand the school size, we will need to add more sensors, cameras, and lights in the building.
- The improved model was validated using the TLC model checker.
- The results were published in *ASTESJ. Advances in Science, Technology and Engineering Systems* as shown in [45].

In the following sub-sections, we will illustrate the improved model using UML and TLA+, and we will discuss the results using the TLC model checker.

2.6.2 UML Modeling for the improved Smart School Building System

We again use the Unified Modeling Language (UML) to represent the smart school system's informal abstract model. It is one of the most common languages that has been used for this purpose. There are many other modeling languages like SysML [48]. But for our current informal modeling, since we do not need to use differential equations for what we are modeling, the UML model is sufficient. UML provides graphical notations and captures the system properties we are interested in. Figure 19 shows the use case diagram of the modified system [45]. In this use case diagram, when the actors (employee, visitor, and student) enter the username and password correctly to login to the system, the main door in the building will be

opened for the person to enter the building, and the actor will have access to all sub-systems in the system.

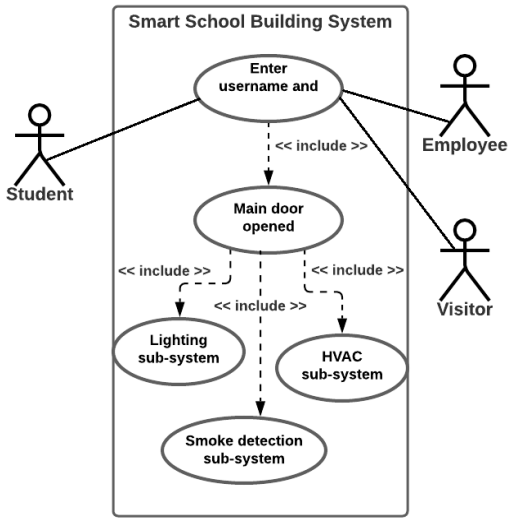


Figure 19: UML Use case diagram of the smart school system [45]

Figure 20 shows the UML sequence diagram of the system. This diagram demonstrates the sequence of actions which define how the system works.

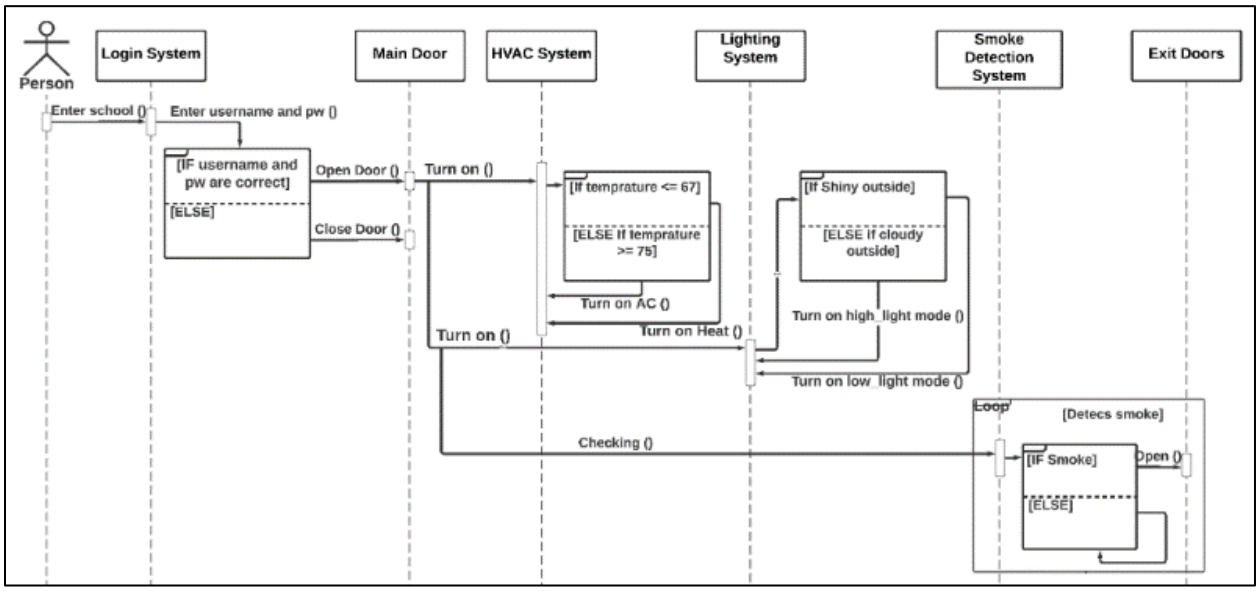


Figure 20: Sequence diagram for the smart school building system [45]

These diagrams define the UML informal modeling of the improved smart school building model. The following sub-section will demonstrate the formal specifications using TLA+.

2.6.3 Formal Specifications of the Improved Model Using TLA+

This section demonstrates the formal specification of the improved smart school system model. We wrote the specifications using the TLA+ toolbox and we validated the model using the TLC model checker. We used TLA+ to represent all the system's operations.

Figure 21 shows the TLA+ definition of the revised smart school system. The figure shows all the module variables that we will use in the specifications.

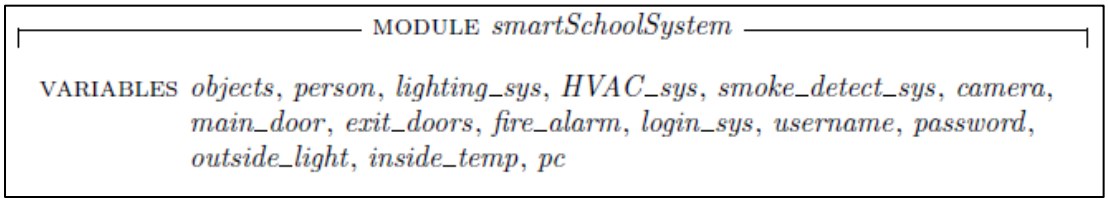


Figure 21: smartSchoolSystem module variables

smartSchoolSystem represents the top module of the system. As shown in figure 21, there are many additional variables in this module. For example, *camera* variable represents the outside camera that is supposed to be working all the time. It records whoever enters and leaves the building to enhance the safety in the system. Another addition is the *inside_temp* variable which represents the inside temperature of the building which is measured by the temperature sensor and represents the input for the HVAC sub-system. Also, there is a variable called *pc* which takes the value of the current state, and *pc'* will represent the next state in the module to keep track of the module states.

Figure 22 shows the *Init* function of the *smartSchoolSystem* module. The *init* function declares all possible values of the system's variables. For example, *login_sys* variable has the values of "ready" when it is ready to take inputs from the users, and "reTry" when a user enters an invalid username and/or password. Another example is the password variable which will have the values of "correct" or "wrong".

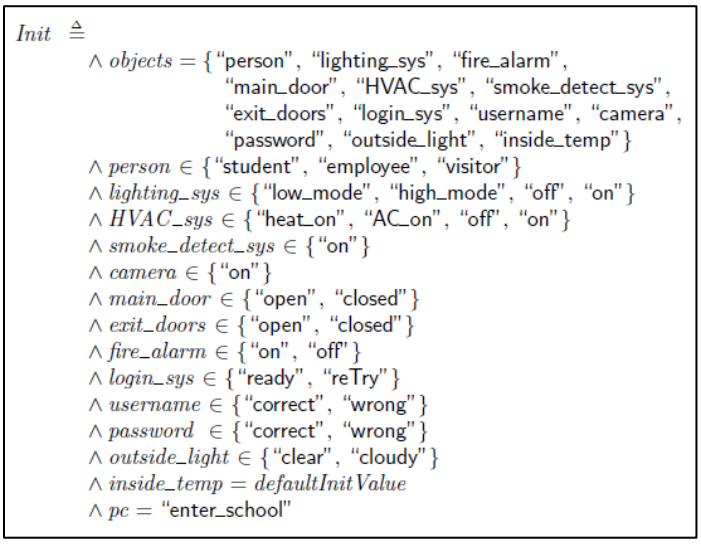


Figure 22: Init function

To enhance safety in the system, we set up two important invariants in both the specifications and the TLC model checker. The *safe* invariant, which is shown in figure 23, guarantees the continuous working of the smoke detection sub-system. In addition, we set up the *surveillance* invariant to guarantee that the camera will also work continuously.

$$safe \triangleq smoke\_detect\_sys = \text{“on”}$$
$$surveillance \triangleq camera = \text{“on”}$$

Figure 23: System invariant

Figure 24 shows the *enter_school* function which represents the login sub-system. This function needs a correct username and password as an input. If the person who wants to enter the building enters an invalid username and/or password, the function will give him/her an option to retry.

```
enter_school == /\ pc = "enter_school"
                /\ IF person  = "student" /\ username = "correct" /\ password = "correct"
                      THEN /\ main_door' = "open"
                           /\ UNCHANGED login_sys
                      ELSE /\ IF person = "student" /\ username = "wrong"
                                 THEN /\ main_door' = "closed"
                                      /\ login_sys' = "reTry"
                                 ELSE /\ IF person = "student" /\ username = "correct" /\ password = "wrong"
                                            THEN /\ main_door' = "closed"
                                                 /\ login_sys' = "reTry"
                                            ELSE /\ IF person  = "employee" /\ username = "correct" /\ password = "correct"
                                                       THEN /\ main_door' = "open"
                                                            /\ UNCHANGED login_sys
                                                       ELSE /\ IF person = "employee" /\ username = "wrong"
                                                                  THEN /\ main_door' = "closed"
                                                                       /\ login_sys' = "reTry"
                                                                  ELSE /\ IF person = "employee" /\ username = "correct" /\ password = "wrong"
                                                                             THEN /\ main_door' = "closed"
                                                                                  /\ login_sys' = "reTry"
                                                                             ELSE /\ IF person  = "visitor" /\ username = "correct" /\ password = "correct"
                                                                                        THEN /\ main_door' = "open"
                                                                                             /\ UNCHANGED login_sys
                                                                                        ELSE /\ IF person = "visitor" /\ username = "wrong"
                                                                                                   THEN /\ main_door' = "closed"
                                                                                                        /\ login_sys' = "reTry"
                                                                                                   ELSE /\ IF person = "visitor" /\ username = "correct" /\ passwor
                                                                                                              THEN /\ main_door' = "closed"
                                                                                                                   /\ login_sys' = "reTry"
                                                                                                              ELSE /\ TRUE
                                                                                                                   /\ UNCHANGED << main_door,
                                                                                                                                   login_sys >>
                /\ pc' = "Smoke"
                /\ UNCHANGED << objects, person, lighting_sys, HVAC_sys,
                                smoke_detect_sys, camera, exit_doors,
                                fire_alarm, username, password, outside_light,
                                inside_temp >>
```

Figure 24: enter_school function

Figure 25 shows the *smoke* function which represents the output in case of smoke/fire to turn on the fire alarm which can also be sent to the nearest fire station. In addition it will open all exit doors to let all people to exit the building immediately for their safety.

$$
\begin{aligned}
Smoke \triangleq\ & \wedge pc = \text{“Smoke”} \\
& \wedge \text{IF } main\_door = \text{“open”} \\
& \quad \text{THEN } \wedge \text{IF } smoke\_detect\_sys = \text{“smoke”} \\
& \qquad \text{THEN } \wedge exit\_doors' = \text{“open”} \\
& \qquad\quad \wedge fire\_alarm' = \text{“on”} \\
& \qquad \text{ELSE } \wedge \text{TRUE} \\
& \qquad\quad \wedge \text{UNCHANGED } \langle exit\_doors, fire\_alarm \rangle \\
& \quad\quad \wedge pc' = \text{“Smoke”} \\
& \quad \text{ELSE } \wedge pc' = \text{“Light”} \\
& \quad\quad \wedge \text{UNCHANGED } \langle exit\_doors, fire\_alarm \rangle \\
& \wedge \text{UNCHANGED } \langle objects, person, lighting\_sys, HVAC\_sys, \\
& \qquad smoke\_detect\_sys, camera, main\_door, login\_sys, \\
& \qquad username, password, outside\_light, inside\_temp \rangle
\end{aligned}
$$

Figure 25: Smoke function

Figure 26 shows the Light function. This function is responsible for controlling the lighting sub-system. It starts working once the first person enters the school building. This function takes its input from the lighting sensors and sends its output to the lights in the building. It is designed in a way to reduce the power consumption in the building which is one of our design goals.

$$
\begin{aligned}
Light \triangleq\ & \wedge pc = \text{“Light”} \\
& \wedge \text{IF } main\_door = \text{“open”} \wedge outside\_light = \text{“clear”} \\
& \quad \text{THEN } \wedge lighting\_sys' = \text{“low\_mode”} \\
& \quad \text{ELSE } \wedge \text{IF } main\_door = \text{“open”} \wedge outside\_light = \text{“cloudy”} \\
& \qquad \text{THEN } \wedge lighting\_sys' = \text{“high\_mode”} \\
& \qquad \text{ELSE } \wedge \text{TRUE} \\
& \qquad\quad \wedge \text{UNCHANGED } lighting\_sys \\
& \wedge pc' = \text{“HVAC”} \\
& \wedge \text{UNCHANGED } \langle objects, person, HVAC\_sys, smoke\_detect\_sys, \\
& \qquad main\_door, exit\_doors, login\_sys, username, password, \\
& \qquad outside\_light, inside\_temp \rangle
\end{aligned}
$$

Figure 26: Light function [45]

Figure 27 shows the HVAC function. This function is responsible for controlling the HVAC sub-system. It adjusts the temperature in the building. The HVAC sub-system will automatically work once the first person enters the school building. The HVAC function takes its input from the temperature sensor in the building and sends the output to the heating or AC systems based on the temperature. If the temperature inside the school building is 74 °F or more, the AC will work. If the temperature inside the school building is 69 °F or less, the heat will work.

$$
\begin{array}{l}
HVAC \triangleq \ \land\, pc = \text{"HVAC"} \\
\quad \land \text{IF } main\_door = \text{"open"} \land inside\_temp \geq 74 \\
\quad\quad \text{THEN } \land\, HVAC\_sys' = \text{"AC\_on"} \\
\quad\quad \text{ELSE } \land \text{IF } main\_door = \text{"open"} \land inside\_temp \leq 69 \\
\quad\quad\quad\quad \text{THEN } \land\, HVAC\_sys' = \text{"Heat\_on"} \\
\quad\quad\quad\quad \text{ELSE } \land \text{IF } main\_door = \text{"open"} \land inside\_temp > 69 \land inside\_temp < 74 \\
\quad\quad\quad\quad\quad\quad \text{THEN } \land\, HVAC\_sys' = \text{"off"} \\
\quad\quad\quad\quad\quad\quad \text{ELSE } \land \text{TRUE} \\
\quad\quad\quad\quad\quad\quad\quad\quad \land \text{UNCHANGED } HVAC\_sys \\
\quad \land\, pc' = \text{"Done"} \\
\quad \land \text{UNCHANGED } \langle objects, person, lighting\_sys, smoke\_detect\_sys, \\
\quad\quad\quad\quad\quad\quad main\_door, exit\_doors, login\_sys, username, password, \\
\quad\quad\quad\quad\quad\quad outside\_light, inside\_temp \rangle
\end{array}
$$

Figure 27: HVAC function [45]

Figure 28 represents the housekeeping functions that are essential to write a good TLA+ specification for a system. The *Next* function is one of the functions that must be written in any TLA+ specification. This function enables collection and execution of all functions in the spec and moving to the next state in the system after initialization. The Termination function guarantees the termination when pc reaches Done state. In TLA+, the Spec function is the main function that is responsible to run all system specifications in the main execution of the system.

$$Terminating \triangleq pc = \text{“Done"} \land \text{UNCHANGED } vars$$

$$Next \triangleq enter\_school \lor Smoke \lor Light \lor HVAC \lor Terminating$$

$$Spec \triangleq Init \land \Box[Next]_{vars}$$

$$Termination \triangleq \Diamond(pc = \text{“Done"})$$

Figure 28: Terminating, Next, and Spec functions

```
180                ELSE /\ pc' = "Light"
181                     /\ UNCHANGED << exit_doors, fire_alarm >>
182          /\ UNCHANGED << objects, person, lighting_sys, HVAC_sys,
183                          smoke_detect_sys, camera, main_door, login_sys,
184                          username, password, outside_light, inside_temp >>
185
186 Light == /\ pc = "Light"
187          /\ IF main_door = "open" /\ outside_light = "clear"
188                THEN /\ lighting_sys' = "low_mode"
189                ELSE /\ IF main_door = "open" /\ outside_light = "cloudy"
190                           THEN /\ lighting_sys' = "high_mode"
191                           ELSE /\ TRUE
192                                /\ UNCHANGED lighting_sys
193          /\ pc' = "HVAC"
194          /\ UNCHANGED << objects, person, HVAC_sys, smoke_detect_sys, camera,
195                          main_door, exit_doors, fire_alarm, login_sys,
196                          username, password, outside_light, inside_temp >>
197
198 HVAC == /\ pc = "HVAC"
199         /\ IF main_door = "open" /\ inside_temp >= 74
200               THEN /\ HVAC_sys' = "AC_on"
201               ELSE /\ IF main_door = "open" /\ inside_temp =< 69
202                          THEN /\ HVAC_sys' = "Heat_on"
203                          ELSE /\ IF main_door = "open" /\ inside_temp > 69 /\ inside_temp < 74
204                                     THEN /\ HVAC_sys' = "off"
205                                     ELSE /\ TRUE
206                                          /\ UNCHANGED HVAC_sys
207         /\ pc' = "Done"
208         /\ UNCHANGED << objects, person, lighting_sys, smoke_detect_sys,
209                         camera, main_door, exit_doors, fire_alarm, login_sys,
210                         username, password, outside_light, inside_temp >>
211
Desktop\Practicing TLA+ Modules\improvedSmartSchool.tla ]   129 : 45 : 4584   Spec Status : parsed
```

Figure 29: TLA+ Parsed model for smart school system

After finishing writing the specifications of the system, we need to make sure that these specifications are correct and have no syntax errors, which is the first verification step. The

TLA+ toolbox has the option to check the specification correctness by parsing the model. If the model is parsed correctly with no errors, a green button would appear on the bottom right corner in the TLA+ toolbox to verify that as shown in figure 29.

After parsing the model correctly, the next step is to validate the model using the TLC model checker. This is a big step in the improved smart school model that we did not carry out in the initial model. Sub-section 2.6.4 will illustrate the improved smart school building system validation process using the TLC model checker.

2.6.4 Formal Verification Using TLC

As we illustrated above, we successfully wrote and verified the improved smart school building system using the TLA+ toolbox. The model was parsed correctly. The next step is to validate our work. We will use the powerful TLC model checker in order to validate our model. The TLC checks the invariance properties of the finite state model of the specification [6]. It also checks for deadlock and the invariants in the system. In our TLC model, we set up two invariants as shown in figure 30. The safe invariant guarantees that the smoke detection sub-system will keep working all the time to guarantee safety in the system. And the surveillance invariant guarantees that the camera will keep recording all the time to enhance safety in the system as well. The system's safety is one of the main goals in our design. In this TLC model, we included the deadlock option to make the TLC check if there is a deadlock in the specifications.

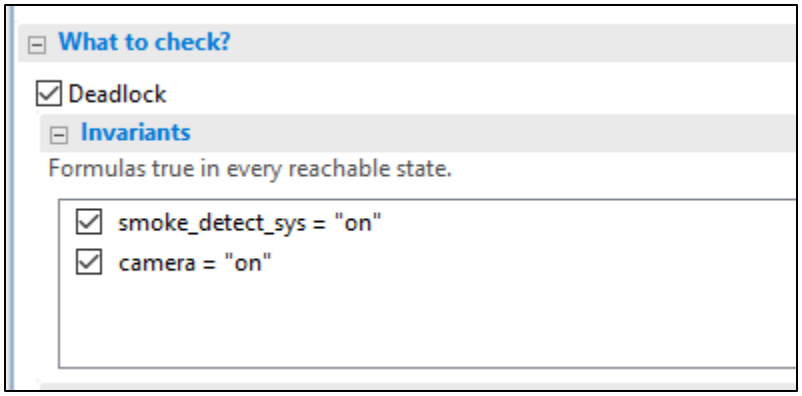


Figure 30: Safety invariants setup in TLC

After finishing setting up the TLC model checker, we clicked on a run button in the TLC model to run the model and verify it. Figure 31 shows the TLC model checker while running.

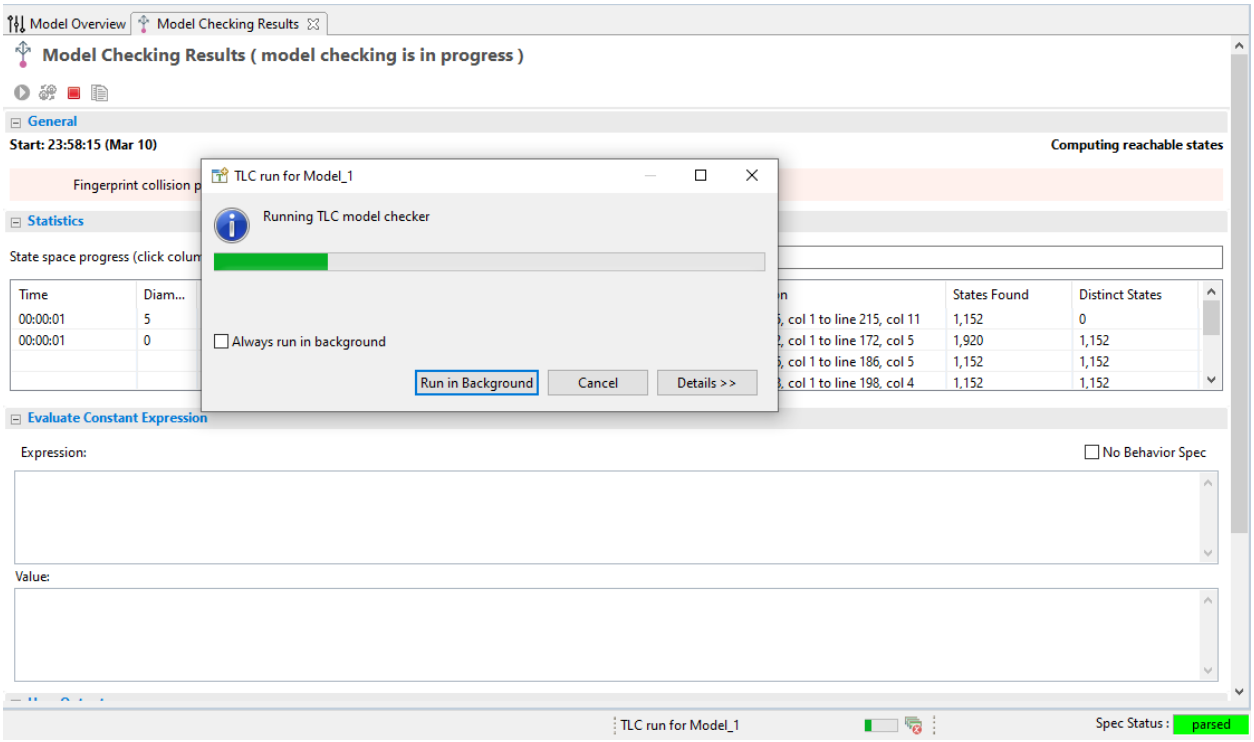


Figure 31: TLC model checker while running

Figure 32 shows the *smartSchoolSystem* final result. The figure shows that the TLC model checker ran the model, checked the invariants and deadlock, and parsed it. That means that the TLC model checker verified and validated the *smartSchoolSystem* model successfully. This proves our system's validation which achieves the most important goal for our system.

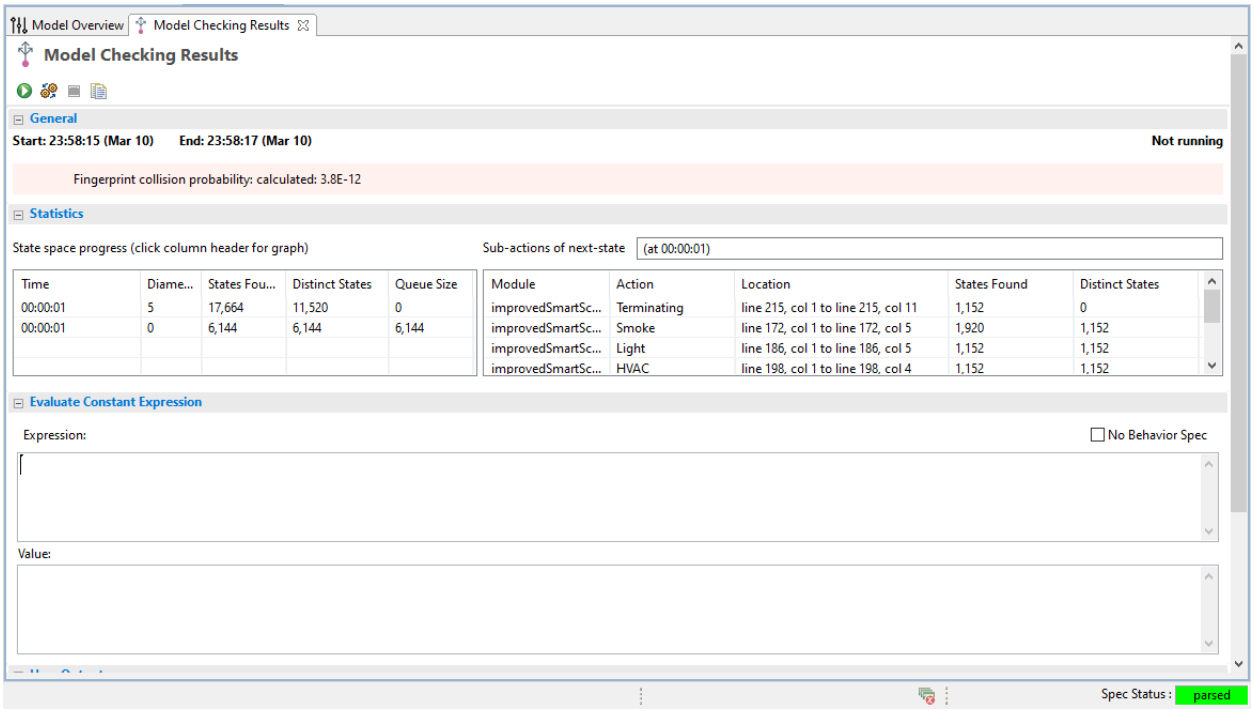


Figure 32: TLC verification model

2.7 TLC Finds Security Break

As we mentioned before, one of our goals is to design a smart school that is safe and secure. We designed the improved smart system and validated our module successfully using the TLC model checker as we have shown. But what will happen if a hacker decides to break the system's security and safety subsystems.

To check, we decided to test the system in one case where we break one of the system's invariant. As we illustrated before, there are two invariants in the improved smart school module as was shown in figure 23. One of the invariants "surveillance" guarantees that the camera in the school building should be "on" all the time to record everything happening in order to achieve safety in the system. The invariant was:

```
surveillance == camera = "on"
```

We decided to run a case of breaking security by turning the camera "off" when the light function, which is responsible for the turning the lights on in the school building once the first person enters the building, started to work.

The modified light function is shown in figure 33. The modified statement is in line 179.

```
176 Light == /\ pc = "Light"
177          /\ IF main_door = "open" /\ outside_light = "clear"
178                THEN /\ lighting_sys' = "low_mode"
179                     /\ camera' = "off"
180                ELSE /\ IF main_door = "open" /\ outside_light = "cloudy"
181                           THEN /\ lighting_sys' = "high_mode"
182                           ELSE /\ TRUE
183                                /\ UNCHANGED lighting_sys
184                     /\ UNCHANGED camera
185          /\ pc' = "HVAC"
186          /\ UNCHANGED << objects, person, HVAC_sys, smoke_detect_sys,
187                          main_door, exit_doors, fire_alarm, login_sys,
188                          username, password, outside_light, inside_temp >>
189
```

Figure 33: the modified Light function that turns the camera off

We decided to run the TLA+ toolbox to check if it will throw an error for breaking the "surveillance" invariant but Surprisingly the module was parsed correctly with no errors.

Then, we ran the TLC model checker to see if it would validate the system. The TLC model checker throws an error as shown in figure 34. The figure shows that the TLC checker produced a message on the upper right side of the window that says, "invariant camera = "on" is violated", and in the bottom right side of the window it shows the error-trace tool that the TLC model checker has and allows us to trace the error. TLC also highlighted the states that were never visited due to the invariant violation which are in this case are the HVAC and the Termination states.

This demonstrates that the powerful TLC model checker can find errors and bugs which may be parsed correctly using the TLA+ toolbox but cannot be parsed through the TLC. The TLC model checker can find the security and safety breaks in the system.

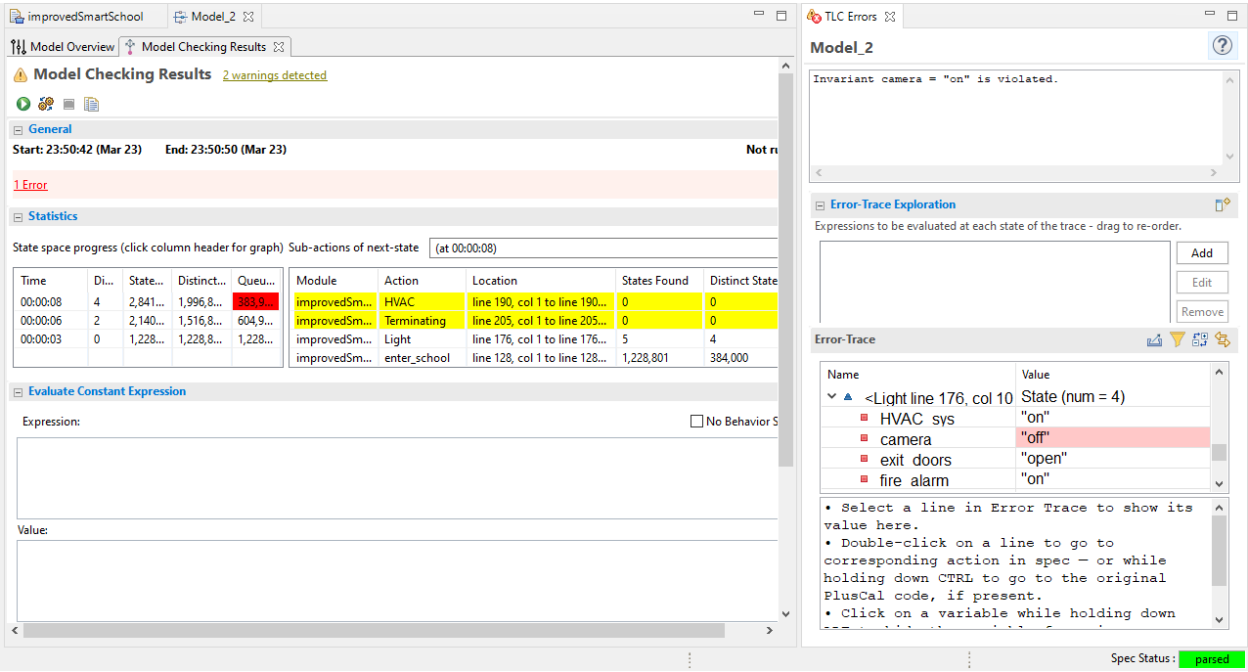


Figure 34: TLC model checker found the security hacking error

2.8 Conclusion

This chapter demonstrates the smart school building model. The system has many sub-systems that are integrated together to work as one smart system. There is a login sub-system, a smoke detection sub-system, an HVAC sub-system, and a lighting sub-system. The model shows in detail how each sub-system works and how we achieved the intended goal that we mentioned in section 2.6.1 for our system. We started by modeling an initial smart school system (version I) and published it in [35]. Then, we improved our work with new specifications and a new improved model (version II) which was published in [45].

In this work, we used formal methods to validate our design and to make sure there are no errors. In the design, the informal model was designed using UML. The specifications were written, verified, and the system's behavior was captured using the TLA+ toolbox. The final model was validated using the TLC model checker.

In this design a failure may still happen. For example, if we simulated the design and built the school from this model, we could have a failure in the lighting system because in practice we will need a light sensor in each room. But our simplified design assumes that the light will be the same in the whole building. In addition, many sensors will be needed throughout the building especially for larger school sizes. This design is an initial and general design to show and explain our methodology, details such as the more complex lighting system would be needed in the final design.

To enhance security and safety in the system, the system requires each person who enters the building to login by entering a valid username and password. To enhance the safety as well, the smoke detection sub-system is working all the time. In addition, we added a surveillance camera that records everyone who enters and leave the building. To control the power in the system, the

lighting sub-system will use natural light when possible to reduce the power consumption in the building. In addition, the way the HVAC sub-system works will reduce the power consumption in the building since it controls the heat and AC systems and turns them on only when necessary based on the temperature inside the building.

We ran a security hacking case in this model, which consisted of breaking one of the system's invariants. The hacking action was turning the camera off when turning on the light in the building. The modified module ran correctly through the TLA+ toolbox but the TLC model checker found the security break and threw an error. This showed that the TLC model checker is powerful and able not only to validate TLA+ modules, but also to find errors and bugs in the system that are hard to find, especially in the cases of security and/or safety errors in the system.

Chapter 3: Automatic Dependent Surveillance – Broadcast (ADS-B)

3.1 Air Traffic Control Surveillance

In Air Traffic Control (ATC), surveillance is very important. It determines the location of aircraft accurately and reliably, which directly influences the aircraft required separation distances as in the separation standards, which affects how to utilize a given airspace efficiently [49].

Where there are no electronic surveillance systems, ATC must rely on pilots reporting their positions verbally. This results in requiring a relatively large separation distance between aircraft because of the estimated position uncertainty and the information timeliness. In contrast, where there is a reliable accurate electronic surveillance system and the aircraft positions are reported more often, there will be more efficient use of airspace along with a safe higher density of aircraft. Also, this will allow aircraft vectoring for reasons of safety, efficiency, and capacity [49].

As explained in [49], ATC surveillance helps in closing the gap between ATC expectations of aircraft movements based on instructions or clearances issued to pilots, and the real trajectories of these aircraft. This means that ATC needs to provide surveillance or "blunder detection" as an important safety function when expectations are not matched.

There is a demand for airspace users to have increased flexibility. This achieved by reducing fixed routes flying restrictions which requires improving the navigation capability on aircraft. At the same time, accurate surveillance is needed to help in the resolution and detection of any potential conflicts related to the flexible use of airspace, which is likely to create a more dynamic environment.

An accurate surveillance system can be used as a base for automatic alerting systems. Using the ability to actively track aircraft will alert the ATC when an aircraft is detected deviating from its assigned route or altitude, or when there is a conflict in predicting the aircraft's future position. An accurate surveillance system also supports minimum area danger warnings, safe altitude warnings, and other alerts.

Surveillance updates flight plans, improves future waypoints estimations, and removes the workload for pilots when they have to provide voice reports when they reach waypoints.

The most important function of a surveillance system is to periodically provide an accurate estimation of the altitude, position, and identity of aircraft. There will also be other requirements of the system based on the ATC applications meant to be supported by the surveillance system.

Here is a list of parameters that may characterize a surveillance system [49]:

1. Coverage volume, which represents the airspace volume when the system works to specification.
2. Accuracy, which represents a measuring difference between the true and estimated aircraft position.
3. Integrity, which represents a clue that the aircraft's estimated position is within a declared containment area of its true position. Integrity contains the concept of generating an alarm if this ceases to be the case, in a defined time to alarm. Also, integrity may be used to indicate if the system operates normally.
4. Update rate, which represents the rate of the updates of the aircraft's position relative to other users.

5. Reliability, the probability within a defined period that the system will remain operating to specification. This is called continuity sometimes.
6. Availability, which represents the total operating time percentage while the system is executing according to specification.

When designing an ATC surveillance system, we need to consider other issues such as:

1. The ability to identify targets uniquely.
2. The loss of surveillance impact of individual aircraft in both the long term and the short term (few seconds).
3. The loss of surveillance impact over an extended area.
4. Applying emergency procedures or backup in cases of ground system failure or of unexpected aircraft events.
5. The ability to operate to specification with the traffic density expectation.
6. The ability to work in harmony with other systems like the Airborne Separation Assistance Systems (ASAS) and the Airborne Collision Avoidance Systems (ACAS).
7. The ability to gain Aircraft Derived Data (ADD).
8. The interaction between navigation, communication, and surveillance functions [49].

3.2 Surveillance Technologies

Knowledge of the aircraft position is crucial to an Air Traffic Controller in providing most air traffic services. A certain knowledge of the aircraft position is necessary to provide separation services. Knowledge of the aircraft position is referred to as surveillance. Pilots can provide

position reports which then provide aircraft position knowledge to a controller. However, scope for misunderstanding errors, the infrequent updates, and the inherent inaccuracy mandate huge spacing in between aircraft in order to maintain safety. This is called a procedural separation technique.

Currently there are four main classes of surveillance technology to support air traffic control services — PSR, SSR, MLAT, and ADS-C [49]. These are described in the following sections.

3.2.1 Primary Surveillance Radars (PSR)

Figure 35 shows how PSR uses the plane's radio waves reflection to detect presence of planes. It supports the controller with a trustworthy, accurate plan view on-screen of the position of aircraft in real-time. The antenna of the radar which usually rotates at 5-12 rpm emits a pulse of radio wave. When it reaches an aircraft, the wave will be reflected and some of the energy will be returned to the antenna [50].

The radar uses the elapsed time between reception and transmission of the reflection to determine the position of the aircraft. The aircraft direction is the same as the direction that the narrow beam antenna of the radar is facing [49].

The polar coordinate system is used by the PSR output data. It provides bearing and range of the targets found based on antenna position. The slant distance from the antenna determines the range, not the horizontal distance [50].

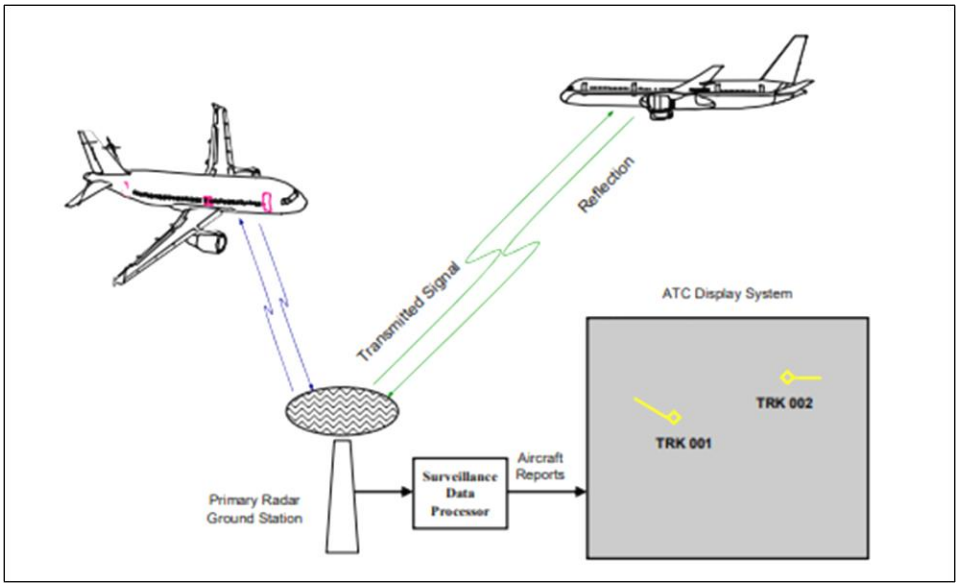


Figure 35: PSR Principle of Operation [49]

Table 2 [49] shows the strength and limitations of PSR.

Strengths of PSR	Limitations of PSR
Installing PSR does not need a transponder or operating on aircraft so allowing the management and detection of faulty/non-equipped aircraft or non-co-operative aircraft.	Identity cannot be provided by PSR.
If display of weather is needed, it can give a weather channel output.	Altitude is not provided by PSR
Well fitted for aerodrome surface surveillance	Position is not based on true range measurement, it is based on slant range measurement (this makes few difficulties for multi-radar tracking systems).
	Can report false targets sometimes (weather, ground vehicles, birds etc.).
	Poor performance detection in the presence of weather and ground clutter specially for the radar flight tangential
	Expensive in comparison to Secondary Surveillance Radar (SSR).
	The update rate longer than ADS-B or traditional multilateration which is between 4 - 12 seconds.
	High transmitter power needed for long range performance. It brings environmental and interference concerns.
	Systems are very expensive to maintain and install
	Systems require with unobstructed view to aircraft with optimum site, and with the minimum of ground clutter detectable by the radar.
	Cannot solve two aircraft at the same range at a similar location, because of weak azimuth resolution performance.

Table 2: Strength and Limitations of PSR [49]

3.2.2 Secondary Surveillance Radar (SSR)

Figure 36 shows the SSR which contains two elements, an aircraft transponder and a ground-based receiver/interrogator. The aircraft's transponder responds to interrogations from the ground station, and this enables the aircraft's range and bearing to be determined by the ground station [49]. SSR not only measures and detects the bearing and range of aircraft, it also requests further information, such as altitude and identity, from the aircraft itself [51].

Military Identification Friend or Foe (IFF) systems evolved the development of SSR and allowed some services for civil aviation such as the use of the Mode A/C. It has been significantly developed since then to include the Mode S service. The SSR frequencies of 1090 and 1030 MHz continue to be shared with the military. Most of the times PSR is co-located with SSR, usually with the SSR mounted on the PSR antenna [49].

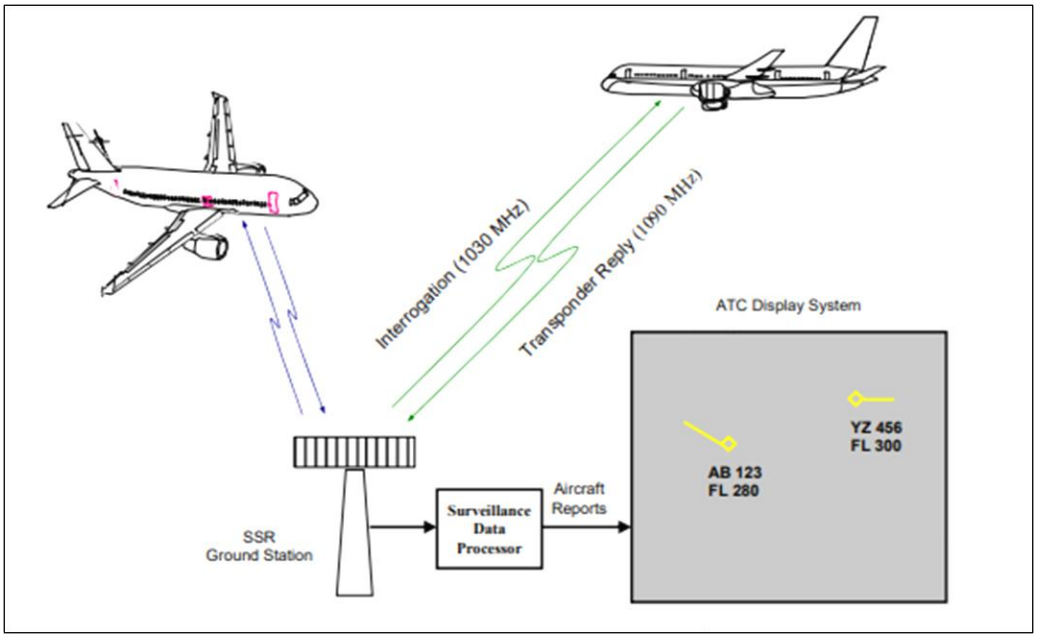


Figure 36: SSR Principle of Operation [49]

Table 3 [49] shows the strength and limitations of SSR.

Strengths of SSR	Limitations of SSR
SSR allows communication of identity with (4 digit octal codes). This held by the ground system when matched with flight plan info	Poor azimuth resolution and accuracy (mainly for classical SSR)
Allows communication of emergency states and altitude to ground system.	Can report false position or targets sometimes (multipath, reflections).
Provides good detection ability independent of weather and clutter.	Can confuse Mode A sometimes (i.e., identification) and replies as Mode C (i.e., altitude), and vice versa. Can report false 4-digit code or altitude sometimes
Provides relatively high update rate.	In altitude and downlinked 4-digit code, it does not provide error detection from Mode C transponders.
Provision of altitude permits slant range error correction.	Systems are expensive to maintain and install
	Systems need optimum site with clear view to aircraft.
	At the same location, it cannot resolve two aircraft (resolution performance/garbling)
	Depends on aircraft avionics.
	Because of transponder delay uncertainty, it is not accurate for aerodrome surface applications

Table 3: Strength and Limitations of SSR [49]

3.2.3 Multilateration (MLAT)

MLAT is a system used to calculate 2D or 3D positions using aircraft transponder transmissions. Figure 37 shows the Transponder MLAT Principle of Operation. MLAT is an important system that is used in large airports as an important identification and surveillance system. A number of measurement stations (e.g., 15-20) make up a typical MLAT system which will be capable of time-tag, receive, and transmit, replies, and squitters to the Central Processing Station (CPS) over a Local Area Network (LAN) in the airport. Furthermore, one or more Reference Transponders whose basic function is to transfer Mode S "squitter" signals (hence the Squitter Generation Unit name) allow monitoring and synchronization of the whole system. The times-of-

arrival of squitters/replies because of the Reference Transponders, equipped vehicles, and SSR equipped aircraft are forwarded to the Central Processing Station (CPS) where the mobiles are located by multilateration algorithms. Due to the airport traffic increasing (particularly on channel 1090 MHz), both the robustness to interference and the MLAT resolution/accuracy have to be improved without the number of MLAT stations increasing too much. The "non transponder device" and the SSR Mode S transponder has the same main functionalities with a smaller cost, lower power, and with no "flyability" certification. The total MLAT position error is usually below 7.5 m for 95% of time, as given in [49][52][53][54].

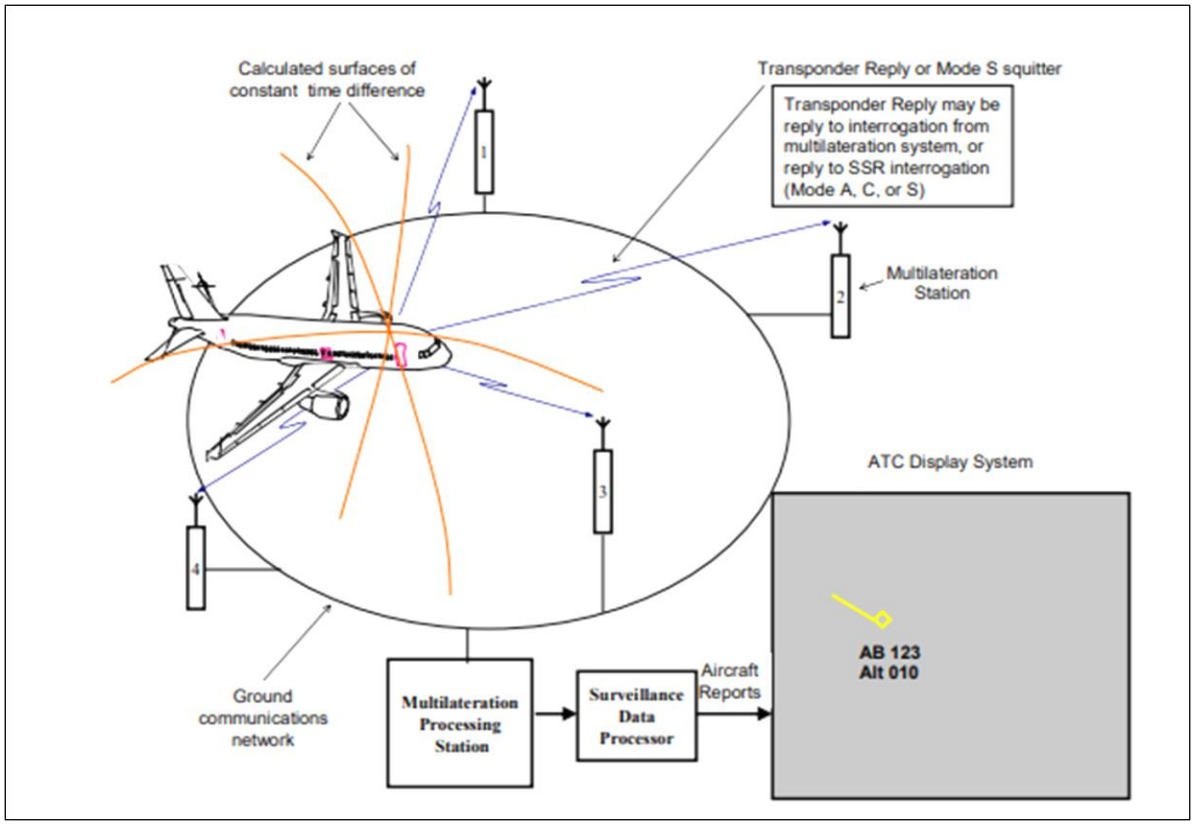


Figure 37: Transponder MLAT Principle of Operation [49]

Table 4 [49] shows the strength and limitations of MLAT.

Strengths of MLAT	Limitations of MLAT
Provides aircraft identification to ground system using Flight Identity (Mode S or ADS-B based), 24-bit Mode S codes, or 4-digit octal codes.	Needs multiple sites
From aircraft, it allows communication of altitude, identity, and emergency states downlinked	Needs many communication links
Provides good detection ability independent of weather and clutter.	Reports false targets sometimes (multipath, reflections)
Capable to give a high update rate.	There is no error detection supported in Mode C transponders altitude and downlinked 4-digit code.
At the same location, it can resolve two aircraft (resolution performance/ garbling) using selective address interrogation if aircraft are Mode S capable.	Moderately, systems are expensive to maintain and install
Can work as a set of multiple ground stations of ADS-B.	Systems need many sites with clear view to aircraft. This may cause a big problem in some areas and environments
Able to be installed at a shared site with other users.	Not yet accepted by ICAO.
Before occurring of widescale ADS-B equipage in some States, it was an attractive transition path.	Needs many transmitter sites for big coverage.
Costs less than the radar.	
Data feed may re-made to look like resemble radar data.	
Whenever there is infrastructure is available, in some locations, the systems may be inexpensive to maintain and install in comparison to alternative systems	

Table 4: Strength and Limitations of MLAT [49]

3.2.4 Automatic dependent surveillance (ADS-C)

Figure 38 shows the ADS-C system. ADS-C supports an exchanged agreement between the aircraft and the ground system using a data link, specified under ADS-C conditions. ADS-C reports these conditions and determines what would be initiated, and what information would be included in the reports. To determine its velocity, position, and other data, the aircraft with ADS-C uses on-board navigation systems and reports this data to the in-charged air traffic control center [49][50].

Reports with ADS-C are sent by point-to-point VHF data links or satellite. Service providers are typically providing the data links. There is a transmission fee for each message that airlines

borne most of it, which makes using ADS-C for more than (10-15 minutes) between messages is a reluctance. There is a limitation of using HF datalink sometimes, just when there is a reduced performance. With ADS-C the ground systems and the airborne aircraft negotiate the conditions under the aircraft reports submissions (i.e., event reports, periodic reports, emergency reports, and demand reports). When the ground system has received the reports, these reports are processed to track the aircraft using the ATC display as the same as surveillance data gained from SSR. ADS-C is normally used in remote and oceanic areas with no radar, and therefore it is mostly fitted to aircraft with long range air transport. The aircraft avionics picks VHF communication when it costs less with improved performance. At other times, when the aircraft is over the ocean, satellite data-communications are used. Typically, transmitting messages happens infrequently (~ each 15 minutes). a "figure of merit" value is accompanied by the positional data which indicates the accuracy, not the integrity value [49].

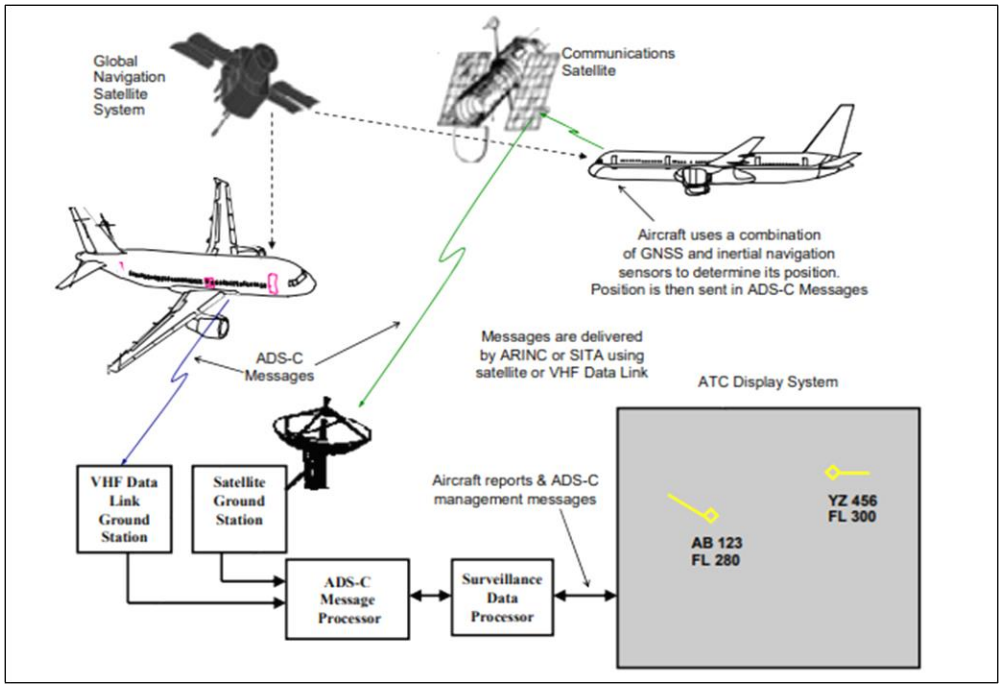


Figure 38: ADS–C Principle of Operation [49]

Table 5 [49] shows the strength and limitations of ADS-C.

Strengths of ADS-C	Limitations of ADS-C
Supports surveillance coverage over oceans and very remote regions excep the polar areas.	High costs for each report (service provider).
Supports some of the safety net applications(ARCW11: ADS Route conformance warning RAM: Route adherence monitoring, and CLAM: Cleared Level Adherence Monitoring but it is not able to support tactical alerts such as STCA	Not able to offer radar such as separation services.
ANSP low capital cost	Minimal reporting rates.
Low maintenance costs.	High-cost avionics fitment.
	Although it will give higher reporting rates, ATN variant is not matured
	There is long latency when using satellite communication link.
	Not available as other systems (duplication is not for all elements).
	Ability to overload/failure at satellite ground stations.
	Relatively low reliability in message delivery.

Table 5: Strength and Limitations of ADS-C [49]

3.2.5 Automatic Dependent Surveillance – Broadcast (ADS-B)

ADS-B is a new surveillance system designed to improve the air transportation system. It supports foundational technology for developments related to (ATM) Single European Sky Air Traffic Management Research Program (or SESAR) and Next Generation Air Transportation System (NextGen). NextGen indicates the effort that the U.S. Federal Aviation Administration (FAA) is making to change the ATC system to service larger airplane volume more efficiently. SESAR in Europe is making a similar effort [55].

ADS-B uses transmissions from aircraft to give geographical position, positional integrity measures, 24-bit aircraft address, flight identity, velocity, pressure altitude data, and other information which has been determined by airborne sensors [49]. Then, the ground receivers transmit that data to cockpit and controller screens which display on aircraft included with ADS-B avionics [56].

ADS-B is certified and developed as a workable low-cost replacement for typical radar. ATC is allowed to control and monitor airplanes with greater accuracy, and it covers a much larger percentage of the earth's surface, much greater than when using ADS-B. For example, large expanses of Hudson Bay in Canada and in Australia do not have any radar coverage currently, but these are now apparent on ATC screens after the replacement of ADS-B receiving stations. For SESAR and NextGen, ADS-B is one of the best underlying technologies in the ATC plan transforming from the current surveillance radar to surveillance using satellite-based the global positioning system (GPS). Furthermore, the FAA declares that ADS-B will be the cornerstone for the transformation, bringing the reliability and precision of satellite-based surveillance technology to the nation's skies [55].

3.2.5.1 How ADS-B works

ADS-B uses a combination of transmitters, receivers, and satellites to provide both ground control personnel and flight crews with very specific data about the speed and location of airplanes in the area as shown in figure 39. There are two aspects to ADS-B from the airplane perspective. The transmitting airplanes send ADS-B Out signals to receivers which can be other

airplanes or located on the ground. These signals travel from transmitter to receiver as a line-of-sight. Then the ATC ground stations receive these ADS-B Out signals to display the traffic to air traffic controllers. Also, other airplanes receive ADS-B Out signals if they are nearby the transmitting airplanes. After the receiving airplane receives the ADS-B signals, the lateral position (longitude and latitude), the transmitting airplane flight number, altitude, and velocity are given to the receiving airplane pilot on CDTI, a Cockpit Display of Traffic Information. The ADS-B received signal known as ADS-B In. The maximum range between the receiving and the transmitting and airplanes is more than 100 nautical miles (nmi), which allows the CDTI to display both far and near traffic. Navigation satellites send accurate timing information that lets airplanes equipped with GPS receivers or a global navigation satellite system (GNSS) define their own velocity and position. The ADS-B Out equipped airplanes broadcast accurate velocity and position to the ground ADS-B receivers and to other airplanes using a digital datalink (1090 megahertz) as well as sending other information, such as the emergency status and the airplane's flight number. The ADS-B receivers which are installed on other airplanes (i.e., ADS-B In) or integrated on the ground into the ATC systems provide users with a precise description of real-time aviation traffic. In contrast to traditional radar, ADS-B works on the ground and at low altitudes which makes it able to be used to monitor the traffic on the runways and taxiways of an airport. In remote areas with limited radar coverage or no radar coverage, ADS-B is also effective [55].

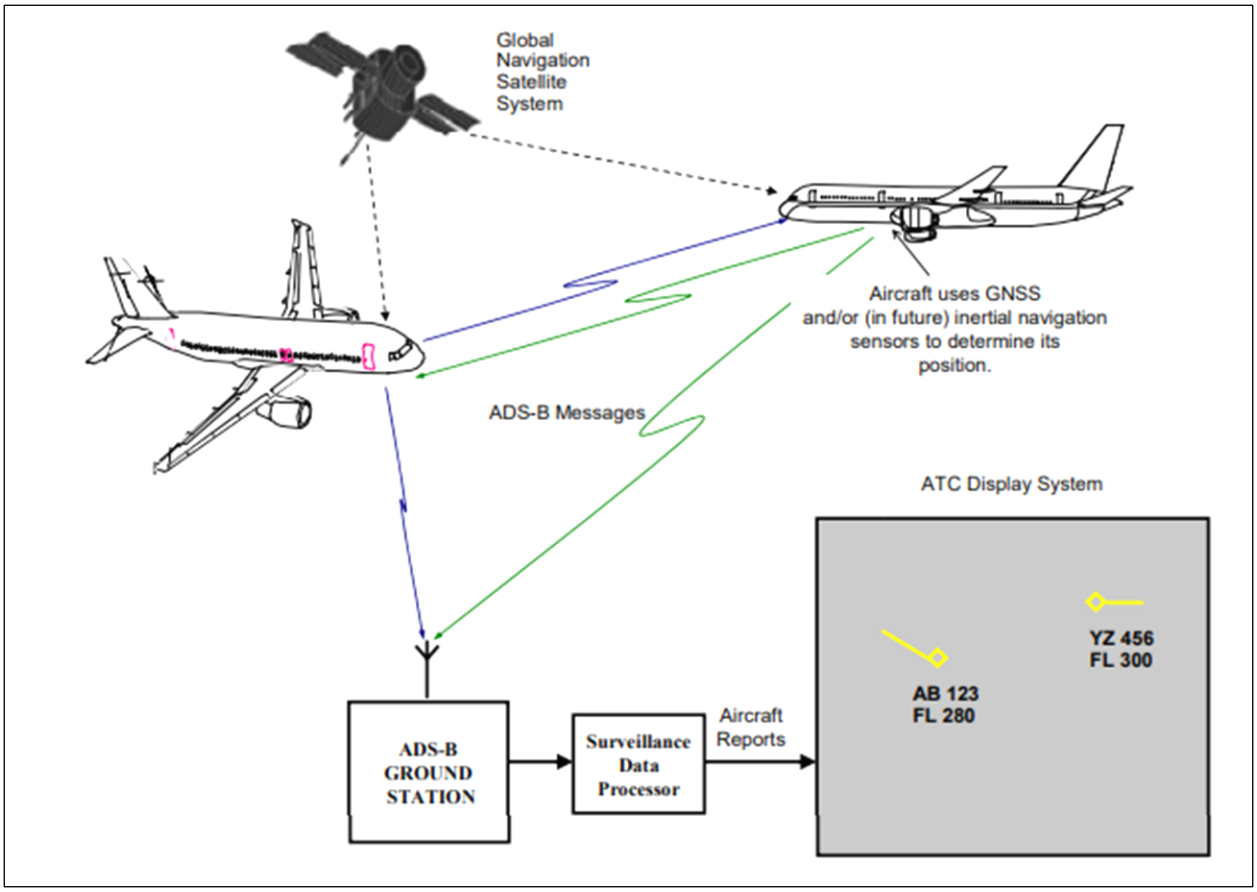


Figure 39: ADS–B Principle of Operation [49]

3.2.5.2 The benefits of ADS-B to airlines

With appropriate operational procedure readiness, and airborne and ground equipage updates, ADS-B may support airlines with many benefits, including:

1. **Safety**:

   ADS-B provides the aviation industry with the capability to improve or maintain existing safety standards along with increasing system capacity and efficiency.

- ADS-B develops flight crews' situational recognition significantly because they are aware of their relation to other airplanes.
- It gives a common real-time surveillance picture in sharing data quickly when participating airplanes are deviating from their allocated flight paths.
- It offers more commonly shared and accurate traffic info. This makes all users have a common operational picture.
- It provides more timely surveillance and accurate information than radar. ADS-B updates the information more frequently than radar.
- It displays both ground and airborne traffic.
- It allows much greater margin in implementing conflict resolution and detection than what is available with any other system. This happens because it provides high accuracy with an effective range of more than 100 nmi.
- It immediately and clearly indicates changes such as when conflicting traffic accelerates, turns, climbs, or descends.
- ADS-B In applications can give automatic traffic warnings or callouts of imminent runway incursions.

2. **Capacity:**

ADS-B can provide a fundamental increase in the ATC system number of flights that it can accommodate. Many more airplanes can use a given airspace at the same time if separation standards are decreased, and the increased accuracy of ADS-B enables significantly reduced separation standards along with providing safety. Not only does ADS-B increase the integrity and accuracy of the position reports, but it also increases

the frequency of the reports, providing a better understanding of the air traffic environment on the ground and in the air.

ADS-B also:

- Improves arrival precision to the metering fix along with increasing runway capacity.
- Helps sustain runway approaches by using cockpit display of traffic info in minimal visual weather conditions.
- Allows using the same runway for more airplanes by enhancing visibility of all airplanes in the area.
- Compared to current procedural separation, it allows 5 nmi of separation in (NRA) non-radar airspace, and in radar airspace it allows a potential reduction of separation from 5 to 3 nmi.

3.2.5.3 Equipment required for ADS-B

Special equipment is needed both on the ground and on board airplanes to receive and transmit ADS-B signals.

Airborne components for ADS-B Out: A GNSS (Global Navigation Satellite System) with associated antennas and receiver on board allows the airplane to process and receive GNSS satellite signals to send the airplane's velocity and position. The velocity and position information are received by the ATC transponder that creates ADS-B Out messages and the ATC antennas broadcast them.

Airborne components for ADS-B In: An airborne collision avoidance traffic/system alert and associated antennas and collision avoidance system unit are used to receive the ADS-B Out

message from a target airplane. Then this target airplane information is processed and sent to a cockpit display of traffic info (CDTI) for display to the flight crew. Other airborne systems that could be affected based on the ADS-B In application requirements include control panels, flight management computer, electronic flight bag, associated wiring, and displays.

Ground components: The ATC system must have ADS-B ground stations in order to receive the airplanes' ADS-B Out messages. ADS-B ground stations contain an ADS-B antenna receiver with a clear view toward the horizon, power supply, an ADS-B receiver, physical and data security, and a communications link (terrestrial or satellite) [55].

Table 6 [49] shows the strengths and limitations of the ADS-B system.

Strength of ADS-B	Limitations of ADS-B
The ground station design is simple and does not need a transmitter.	Dependent on aircraft avionics. This can be a major issue in some environments.
Can be installed at sites that shared with other users.	Equipage rates are relatively low at this stage (2007)
The ground station cost is very low (however, variable ADS-B avionics setup cost is high).	Systems require optimum site with unobstructed view to aircraft
The update rate is very high.	Some outages expected due to poor GPS geometry when satellites out of service, although exposure expected to reduce in the future with use of GNSS augmentation & internal support
Almost perfect resolution.	ADS-B has the capacity to evolve towards the broadcast and use of other data, such as Trajectory Change Point (TCP) or others, already defined in the standard
High accuracy and integrity (airborne measurements).	
Higher performance velocity vector measured by avionics and then broadcast, rather than determined from positional data received on the ground.	
Accuracy not dependent on range from ground station	
Facilitates exchange of surveillance data across FIR boundaries	
Can be easily deployed for temporary use (emergency, special events etc)	
Facilitates future provision of innovative ATM services based on air-to-air ADS-B.	

Table 6: strength and limitations of ADS-B system [49]

3.2.5.4 ADS-B Critical issue

ADS-B has a critical issue in requiring ADS-B avionics which include GPS or a similar system in practicing aircraft. While a lot of airliner manufacturers manufacture aircraft with ADS-B out avionics, there still a need for equipping a large legacy fleet. In different areas of the world, this situation is different. Some countries have rapidly growing new airliner fleets, and the ADS-B is fitted in the new aircraft. In other countries, huge numbers of legacy aircraft stay unequipped. In different aviation segments. the situation is also different.

Few regional airliners are equipped while large aircraft are typically being better equipped. And an additional potentially problematic area is General Aviation (GA). The cost is low to equip the GA fleet in some countries compared to other countries where it may be very expensive. Some countries predict subsidies in assisting GA equipage. Some countries also predict the needed fitment of ADS-B with/without subsidies. For many countries, timing of transition will be critical to match aircraft equipage of ADS-B. However, the benefits of ADS-B equipage are significant and might allow other surveillance systems to support delivery and be decommissioned from air-to-air surveillance applications. The ADS-B application is most likely supported by ADS-B avionics in all locations where the aircraft travels [49].

3.2.5.5 ADS-B: Economic Point of View

Aviation is a key sector in the economy. In gross domestic product (GDP), its contribution is at least 3% in the US and UK. Currently, there is a three-month delay in publishing airline

performance statistics. However, aircraft now use ADS-B systems to broadcast their real-time location. Since July 2016, Sam et. al. analyzed a flights global dataset [57]. First, they showed that there is a possibility to use ADS-B to accurately estimate airline fight volumes accurately, which is immediately available. Then, they explained that the knowledge of fight volumes in real-time could be a main indicator for aviation's immediate contribution to GDP in both the US and the UK. Therefore, using ADS-B data might help moving towards real-time estimation of GDP, this will equip policymakers to respond to shocks from this information more quickly [57].

The cost of deploying and maintaining surveillance systems is high. The cost not only includes the ground based electronic equipment, but it also includes much more than that. When examining the total cost of many systems, consideration of the following points is required:

1. **Aircraft operator/owner costs:**

In comparison to the total cost of surveillance systems, some consideration should be given such airborne equipment requirements, that are considerable for some technologies. Some points are noted when take under consideration for many ground-based surveillance technologies:

- Main radar surveillance does not need avionics that is deployed in aircraft.
- Multilateration surveillance be able to work with ADS-B avionics, Mode S, or Mode C. But it works better when aircraft are ADS-B equipped or Mode S.
- Mode C based surveillance needs either Mode C or Mode S transponders on board aircraft.

Any aircraft equipage program associated cost (for retrofit as well as new production aircraft) will be airframe dependent and highly variable. Hundredfold cost variations for fitting the same avionics to various aircraft types are common. Aircraft type is highly sensitive for determining the operating costs, fleet nature and size of operation but include:

- Costs of engineering support.
- Scheduled and unscheduled maintenance.
- Costs of flight crew training.
- Aircraft simulator upgrades costs.

For these reasons avionics costs related to each surveillance technology should be very FIR (Flight Information Region) or ANSP (Air Navigation Service Provider) specific. However, the aviation industry nature (in particular, international operations and cross FIR, and the prevalence and fleet turnover of aircraft leasing) mean that it is unhelpful and impossible to refer the avionics equipage total cost to any one ANSP, FIR, or surveillance system. It should be noted that a few of the avionics needed to support surveillance, particularly, ADS-C and ADS-B, have other applications and therefore benefits to operators.

2. ANSP costs:

ANSP costs include:

- Equipment purchase.
- Costs of installation and system testing.
- Project costs including procurement activities, planning, etc.
- Site costs.
- Operating costs.

Taking the above points under consideration, and using experience of the technologies, the cost of surveillance to support TMA airspace and enroute and is shown in table 7. Table 7 supposed that the selected areas are NOT “Greenfield” areas so do not include environmental clearance,

land purchase, road and shelter building costs. Also, Table 7 does NOT include avionics costs [49].

Surveillance Technologies	Major cost factors	Cost for TMA (60NM radius) $ Australian	Cost for Enroute (200NM radius) $ Australian
Primary radar	Site costs, capital cost, ongoing maintenance & management costs, large UPS & power supply especially for antenna. No avionics required.	$8M	$10-14M
SSR	Site costs, capital cost, large UPS & power supply especially for antenna, ongoing maintenance & management costs. SSR Mode C avionics required. In some regions the majority are required to be Mode C equipped.	$6M	$6M
Mode S	Same as SSR. Most vendors offer Mode S radars as"standard" at a similar price to SSR. Mode S avionics required.. Air transport already have Mode S to support ACAS in most parts of the world	$6M	$6M
ADS-B	Apart from avionics installation, major items are site related costs and ongoing telecommunication costs. ADS-B avionics fitment is not included in estimate here because of difficulty in attribution of cost especially for international aircraft where the fitment supports surveillance in all ADSB capable FIRs/ANSPs. Major airframe manufacturers fit ADS-B in the factory. IATA has recommended at APANPIRG that APANPIRG members ignore fitment costs be ignored in business case development.	$380K	$380K
MLAT	Major items are site related costs and ongoing telecommunication costs. If the operational requirement does not demand coverage extended range coverage (e.g., say coverage is only required to 40 nautical miles) thenmultilaterationis a stronger competitor. i.e., it may not be warranted paying the extra costs for long range performance provided by radar.	>$1-$3 M	$2M - $5M
ADS-C	No sensor cost. Minimal setup cost for ANSP. Large cost of FANS1/A avionics and associated equipment for new aircraft (and very large for retrofit).	N/A	N/A

Table 7: Surveillance technologies cost to support TMA airspace and enroute [49]

As shown in table 7, ADS-B is much more economical than the current radar system. It costs around $6 M using SSR, $10-14 M using PSR, and only $380,000 using ADS-B to monitor 200 nautical miles of air space [49]. However, although ADS-B is cost effective, it has weaknesses in security and safety.

3.2.5.6 Security and Safety of the ADS-B

Under the 2020 ADS-B mandate, the FAA has specified that almost all private traffic and commercial traffic must be compatible with ADS-B through software and hardware updates by

the date of January 1. 2020 [58,59]. The ADS-B system is intended to replace legacy approaches like secondary and primary radars by using global satellite navigation systems to make accurate air pictures for air traffic management. The ADS-B security is a main concern due to broadcasting detailed info about aircraft, their velocities, positions, and other data through unencrypted data links, which makes it easy to launch message modification attacks, eavesdropping, and jamming on aircraft in flight [60].

In the ADS-B protocol, the inherent lack of security measures has long been an important topic in both the academic community and aviation circles. Because of the recent published proof-of-concept attacks, this topic is becoming even more pressing, specifically with the deadline for compulsory implementation in most airspaces. It looks like the solutions that are under consideration now (and some of them, such as multilateration, are in use) can only be supported by a rapid improvement of the current system's security. For overall security (and maybe privacy), completely new protocols and/or new message types are required. Work on generating a long-term security solution in dependent surveillance air traffic should consider the impact of both secure location verification and secure broadcast authentication approaches. To avoid hard new challenges in the near future, this should include a detailed analysis of the expected traffic density on today's wireless navigation channels, and the possible impact of the message and communication overhead of a new protocol [61].

After evaluating realistic ADS-B attacks systematically, there are different threats and factors that impact the success of attacks. These attacks on ADS-B can be highly successful and inexpensive. There are some insights from a real-world feasibility analysis using a controlled experimental design that led to the conclusion that any decision process related to safety-critical air traffic should not depend on the ADS-B system exclusively [62].

3.3 Our Work

3.3.1 Background and Related Work

As we explained above, security is a big challenge in an ADS-B system. Now efforts are ongoing to add security to the existing ADS-B system. But it has been shown that security should be implemented from the design level in any system [63]. Adding security later will likely result in some underestimation of running security requirements.

The domain of ADS-B is not limited to 1090/978 MHz. In [64], the authors described a future in which the communications backbone is the Internet Protocol for this system, even for aircraft flying voice communication. for a long time, the radio was not the only point of communication for an aircraft. Now an aircraft component subsystems 'talk' independently to route and maintenance the operator planning departments using Aircraft Communications Addressing and Reporting System [65]. To take this communication load, IP is also slated.

The goal of our work is to rethink the ADS-B system from a security point of view. Descriptions of how ADS-B subsystems interact with each other in case of an attack exist [66].

Our methodology is to pick an attack, and a solution for this attack. Then, we will use a UML use case diagram to informally model the attack and response, a UML sequence diagram to show the sequence of actions needed, and a UML state diagram to show the states and what will happen in each state. We will then formally model the system using TLA+ and validate the model using the TLC model checker as the final step.

We chose UML [68] to informally model the system. Also, in the model, we don't need too many details in the design. The author in [67] modeled the Controller Area Network Bus (CAN-Bus) using UML. However, reserchers in [69,70] used UML to create security parameters. So UML can be used to model cyber-physical, software/hardware systems.

As we mentioned before, ADS-B system has many security issues that need to be solved, including lack of authentication, lack of encryption, and other vulnerabilities. It is vulnerable to some common attacks suck as ghost contacts, hybrid attacks, legacy systems, and physical access [46]. In addition, ADS-B uses GNSS, which also causes some risks. Using GNSS in aviation has many risks related to survivability, deliberate shutdown, jamming, and spoofing [46].

In this work, we picked TLA+ to show how a "built-in security" ADS-B system model will look and behave. It will give a good vision for reserchers to model their own systems or improve the current ADS-B models.

Section 3.3.2 will illustrate our work using UML diagrams to informally model ADS-B with some possible attacking scenarios. The work in section 3.3.2 and section 3.3.3 was published in [46].

3.3.2 UML Models for ADS-B security

For such a big and distributed system as ADS-B, it should be known that there is no single solution to security problems. In [60], the authors tested the possibility of many solutions and discovered that none of these solutions is without a loophole. In this section we will define UML diagrams to deploy various cybersecurity tools which can be used alone or in combination to ensure protection.

Figure 40 shows the use case diagram convention we use.

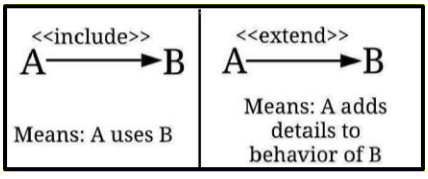


Figure 40: UML Use Case Diagrams Convention [46]

In Figure 41, multilateration is used to estimate the transmitter's location by using the time delay of signal arrival at multiple points (>2) [60]. Time stamps (on any message) are used to calculate the Time Delay of Arrival (TDOA) of the signal. Transmission and reception comparison of timestamps gives a rough idea of the travel time, so the distance can be roughly calculated. If there are 3 receivers, estimated X-Y-Z coordinates can be back-calculated. This is close to the SONAR equation.

If the GNSS transmitter's reported coordinate is reasonably close to this back-calculated position, the message sender allegiance can be confirmed. This multilateration could be used to combat message modification, man-in-the-middle attack, ghost aircraft, and, if PSR was absent, it can be used in providing a GNSS breakdown backup.

Figure 41 is the most important here and will be extended into a state diagram, a sequence diagram, and a TLA+ specification. The figure shows a use case diagram of multilateration and group verification that can protect against ghost aircraft injections. It matches the physical source of the signal to the reported one.

In multilateration, using static ground posts as receivers has some weaknesses such as the system might be fooled by the attacker. If the attacker knows these ground antennas' locations, he/she can customize his transmission time stamps to give a 'false but accurate' image to the multilateration systems. One solution to this is trusted aircraft in the airspace as multilateration.

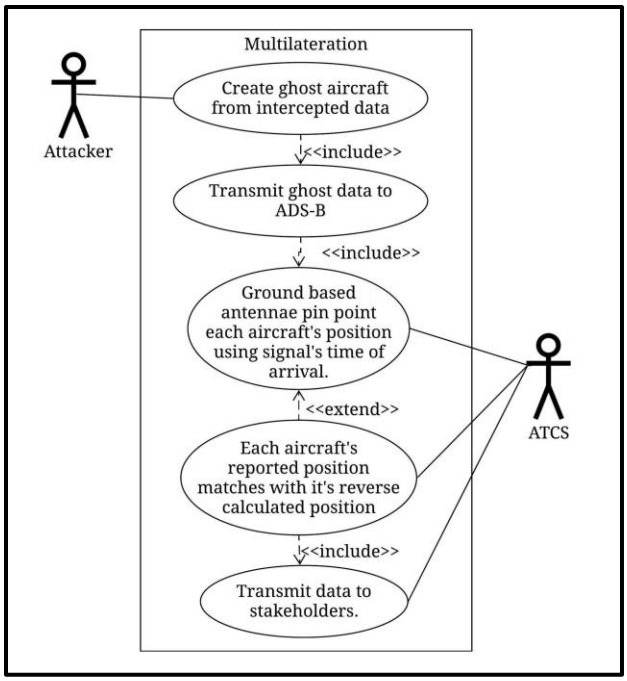


Figure 41: Use Case Diagram of Multilateration and group verification [46]

Figure 42 shows multilateration multi-point which may help with GNSS unreliable position reports and spoofing. Group verification [60] is a type of multilateration executed in the air. Aircraft already use TDOA to multilaterate each other which works exactly like the Traffic Collision Avoidance System (TCAS). The algorithms and equipment are already available on board, and this is matched with our philosophy in causing minimal modification.

Each aircraft use TDOA to locate each member of received ADS-B IN signals and then estimates the other aircraft positions. If there is a difference in the calculated position from the reported aircraft, a 'suspect' airframe will be identified, and all group members fly away from it. This cannot be successful unless 100% of ADS-B implementation is achieved. Fixing location from various sources is independent of GNSS. The fusion of this information can be another method to know aircraft location. This will counter GNSS jamming and GNSS spoofing attacks.

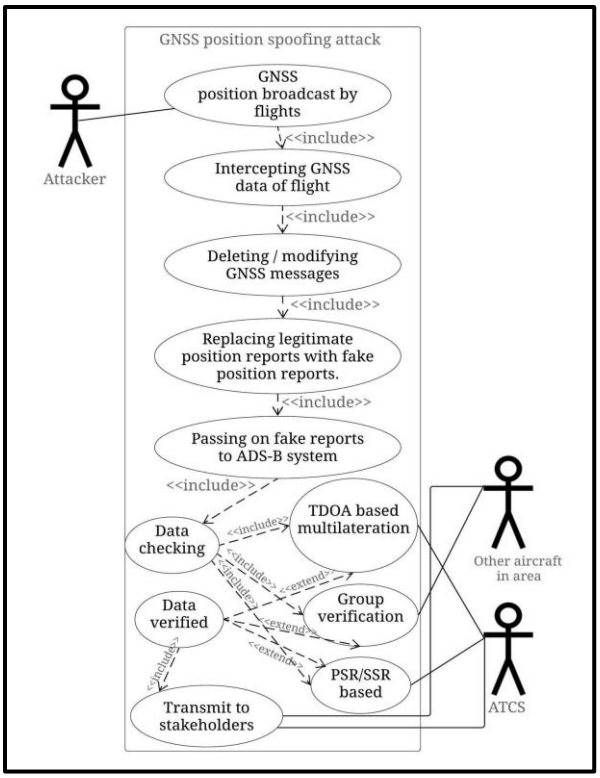


Figure 42: Multilateration might help with GNSS unreliable position reports and spoofing [46]

3.3.3 TLA+ Work

3.3.3.1 Conversion

TLA+ allows writing formal specifications for different system applications. Here we will include a TLA+ formal specification for the ADS-B system and its components with included security protocols. Every system has states that could be defined by a set of specific values. When the values are changed, the states will be changed too.

The whole system could be modeled as a superset of all potential state sets. We do not dig so deep into the details of each state, we focused on “what” rather than “how”. Abstraction is a very important skill which must be mastered before modeling in TLA+. While computer overhead and economics are major factors in systems design, we ignore them while dealing with TLA+. Our

work can be presented as a guide to designing large systems in future with analysis comparison and model-driven design extensions. GNSS is responsible for determining the aircraft position and reporting it. Aircraft reports their position to (ATCS) air traffic control.

Aircraft get aerodrome and weather information, and, for preventing collision, aircraft share position information with each other. All this communication occurs over the 978/1090 MHz spectrum. So, there are some independent operations as well as some independent failure points. These systems must be included in any formal method examination related to the ADS-B system. The solutions provided for responding to attack are the work of other researchers as mentioned before. In this work, we will model a security solution for the ADS-B system for achieving security and safety by by designing a procedure, simulating it, and validating it.

TLA+ defines a system as a collection of states. A system can have many states, each defined as a 'step', or a sequence of events, in going from one state to another. We developed a sequence diagram first, which illustrates all the steps between states, and second, a state diagram which illustrates the states themselves. After that, we draw a true TLA+ model.

We will apply this methodology as we did in [45] and [46] to GNSS spoofing issues and ghost aircraft as shown in figure 42.

3.3.3.2 ADS-B response to GNSS spoofing and Ghost Aircraft

In the previous sections, we talked about security issues if an aircraft is either a fake contact or reports untruthful position info about itself to ADS-B, by GNSS spoofing. Because ADS-B envisions doing away with SSR/ PSR, we examined a few other backup systems that may be able

to physically locate an aircraft in case of unreliable GNSS reports. Figure 42 was the UML use case diagram for this. Now, figure 43 will illustrate the sequence diagram for this situation.

GNSS reports any aircraft position to airplanes, which includes the ‘good’ and ‘bad’ airplane. Then, all aircraft will report their position to ATCS. From all aircraft, a time delay analysis will be performed on the signals, and this will be used in physical space to fix a location of the aircraft. Also, aircraft will be able to do triangulation and multilateration on each other in airspace in a ‘4-ship-cell’ for instance. Also, this fixes location for each other’s locations.

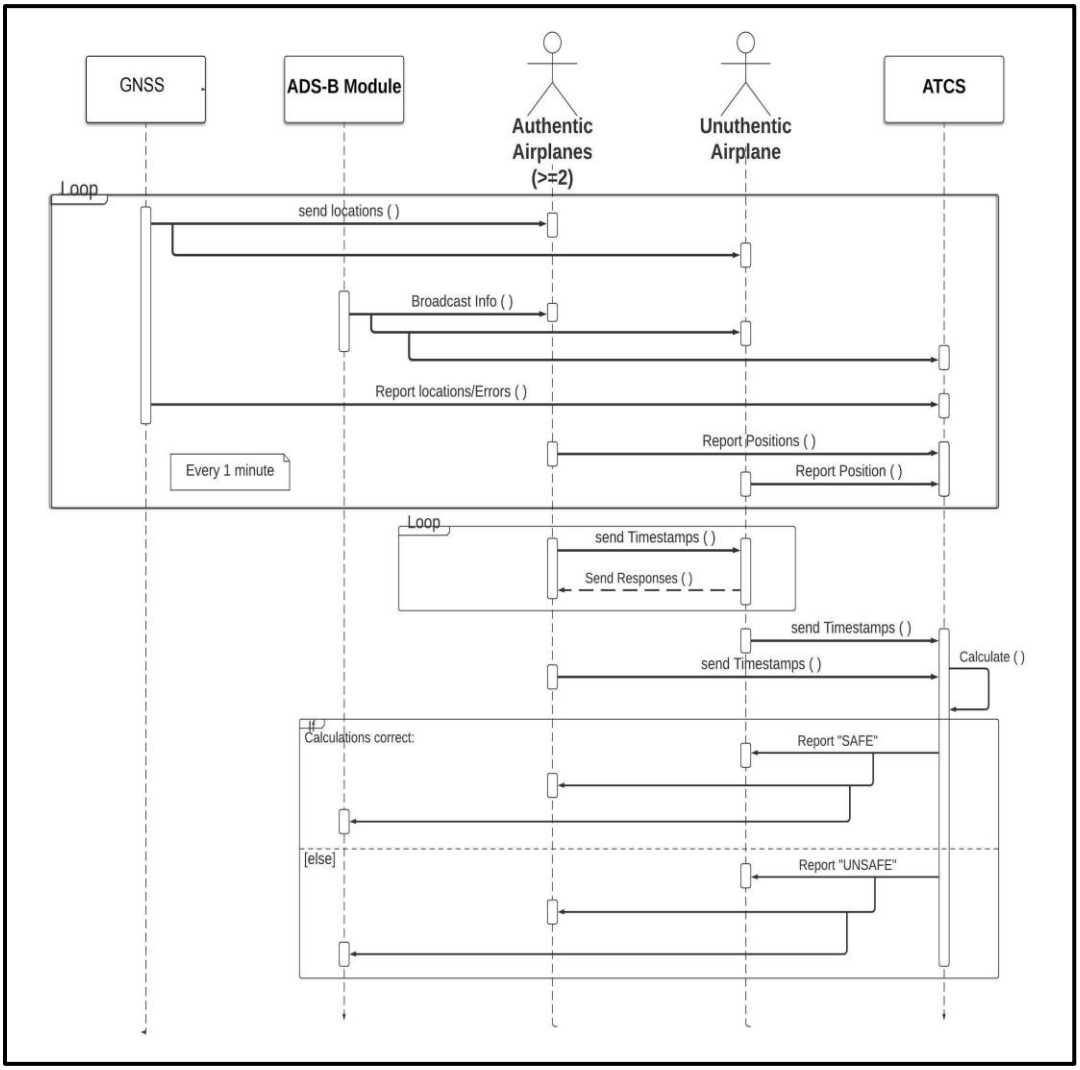


Figure 43: Sequence diagram: the aircraft’s true position reverse calculation that can be done using time-delay analysis of transmitted signals [46]

GNSS reports position information which will be compared to the back-calculated position data to determine which aircraft is reporting true information and which is not. Based on this, the aircraft will be considered as “SAFE” or “UNSAFE”. Then, this information will be broadcast to ATCS and all aircraft in the area. Now, we will explain figure 44 which represents the UML state diagram for the system.

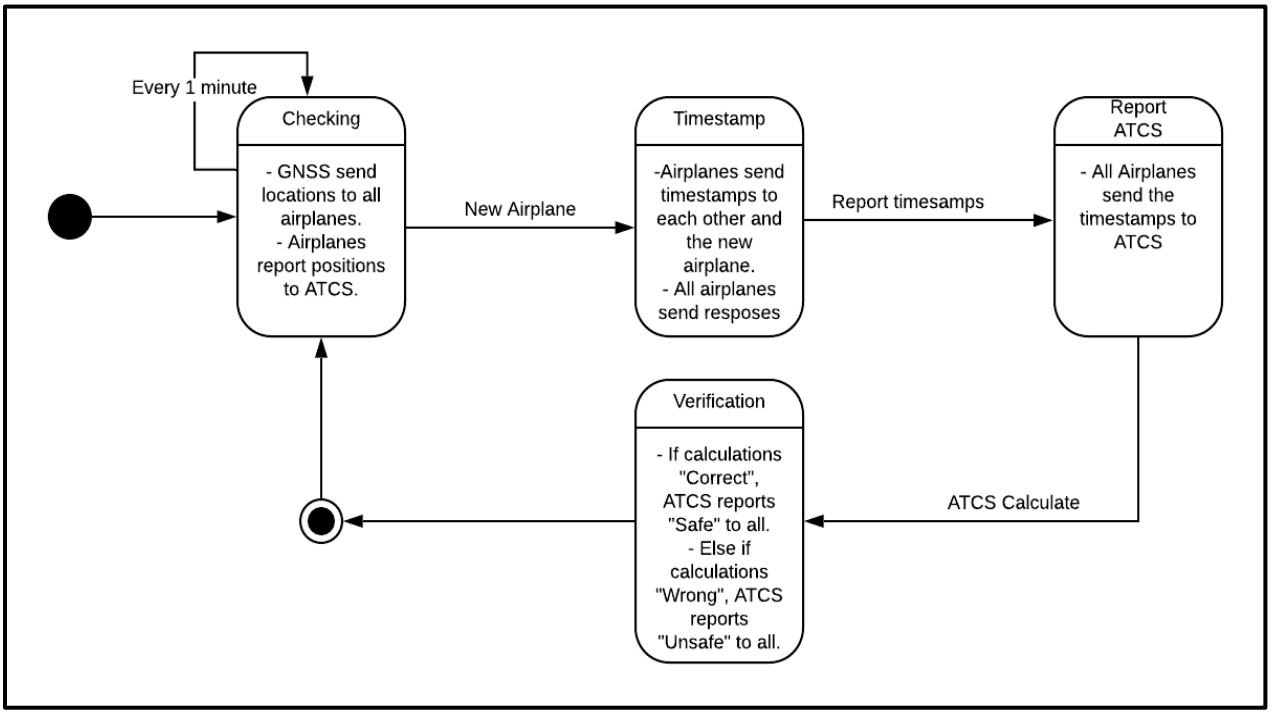


Figure 44: UML State Diagram of the system under Figure 4 conditions [46]

This assigns a one-minute interval foe each whole cycle repetition. This is arbitrary, and it will be practically a decision made by radio traffic and economic factors. Due to the aircraft’s fast movement in all 3 dimensions, this interval cannot be very long. Furthermore, when we say ‘ATCS’, this means a regional or local airspace manager. We do not mean a global central ATCS system.

The next step is to use the 3 inputs, Figures 42, 43 and 44 to develop the TLA+ formal specification and then the TLC model checker validation for the above response.

3.3.3.3 TLA+ Specifications

Now, we present the TLA+ specification for the attack response model which was sequenced in Figure 43 and stated in Figure 44. The following will be a block-by-block explanation.

In this module, the variables are represented by the keyword "VARIABLES", where "vars" is the tuple of variables:

<objects, sender1, receiver1, sender2, receiver2, ATCS, Calculations, Report, pc >

In any TLA+ module, there is an "Init" function/state and a "Next" function/state. The variables get their initial values In the Init state, while the Next state visits all possible next states. In this module, the Next state is:

Next == checking \/ Timestamp \/ Report_ATCS \/ Verification\/ Terminating

In this module there are 5 different states. The 'Checking' state will be entered when the GNSS sends positioning information to all aircraft, and then all aircraft report their locations to ATCS.

Figure 42 showed the group verification strategy. Airplanes triangulate each other with time delay of transmission using timestamps on exchanged signals. This communication between airplanes is represented by the 'Timestamp' state. When a new airplane that might be an attacker

appears, it should communicate with other trusted airplanes (even if only for TCAS) and sends timestamps.

In the ‘Report_ATCS’ state, all airplanes report to ATCS as well with timestamps as a routine process, and these timestamps could be used to verify the position from a ground point of view.

Then the system must enter the ‘verification’ state. In this state, the ATCS calculates the time delay and radio directional on these signals and then compares them to the very first GNSS location reports that were sent from all airplanes.

The ATCS will identify which airplanes are faithfully in their reporting position and which is not. Then, ATCS will broadcast a reported result with a safe/unsafe field to everyone in the airspace.

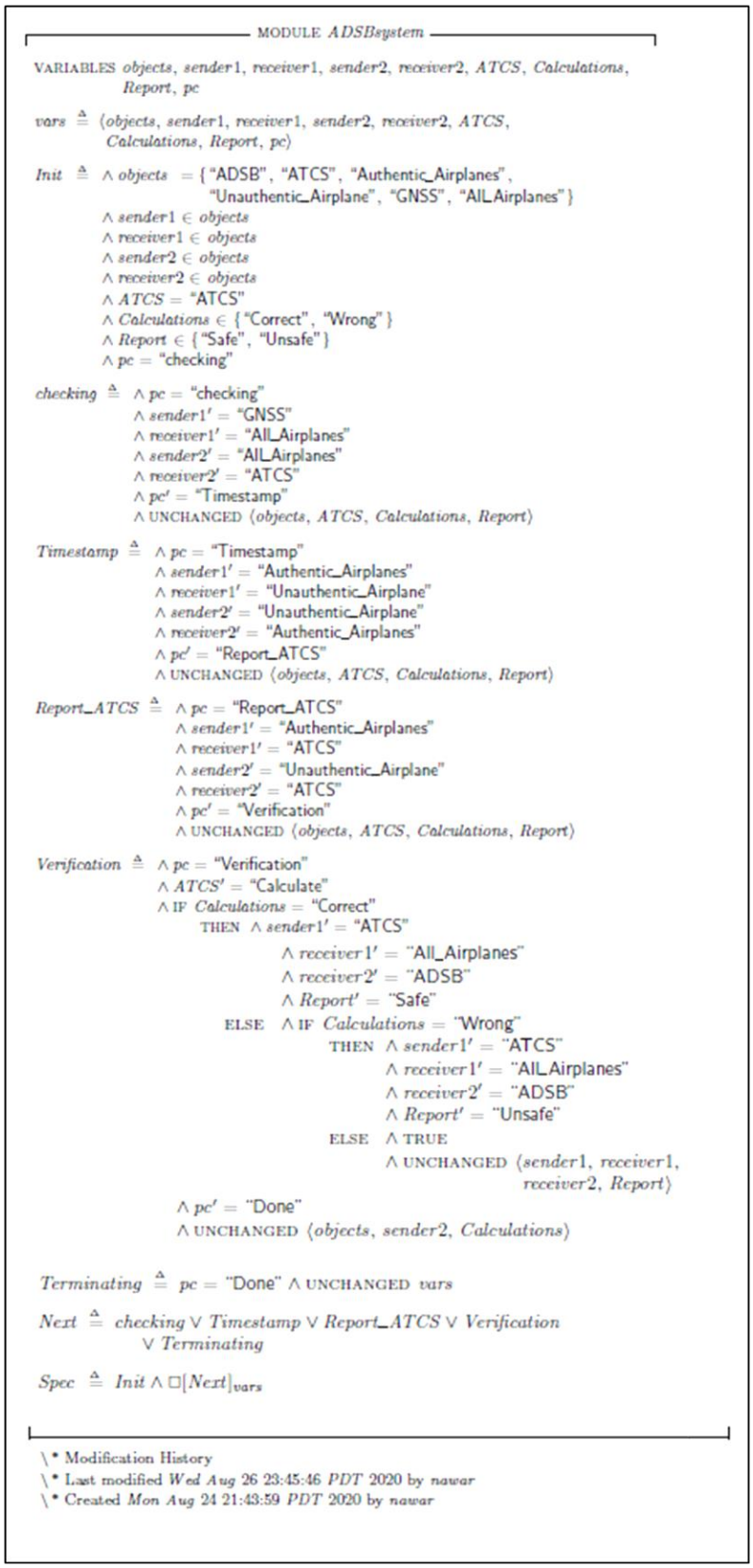
To prevent deadlock on the termination, the termination state allows infinite stuttering (when pc = “Done”).

The *Spec* statement,

$$Spec == Init /\ [][Next]\_vars$$

represents the specification of the entire system, where *Init* represents the starting state and *[]Next_vars* represents *Next* state which need be true for the entire behavior with keeping all variables unchanged.

Figure 45 shows the TLA+ ADSBsystem module (TLA+ specifications) of the system.



```
MODULE ADSBsystem

VARIABLES objects, sender1, receiver1, sender2, receiver2, ATCS, Calculations,
          Report, pc

vars ≜ ⟨objects, sender1, receiver1, sender2, receiver2, ATCS,
        Calculations, Report, pc⟩

Init ≜ ∧ objects = {"ADSB", "ATCS", "Authentic_Airplanes",
                    "Unauthentic_Airplane", "GNSS", "All_Airplanes"}
       ∧ sender1 ∈ objects
       ∧ receiver1 ∈ objects
       ∧ sender2 ∈ objects
       ∧ receiver2 ∈ objects
       ∧ ATCS = "ATCS"
       ∧ Calculations ∈ {"Correct", "Wrong"}
       ∧ Report ∈ {"Safe", "Unsafe"}
       ∧ pc = "checking"

checking ≜ ∧ pc = "checking"
           ∧ sender1′ = "GNSS"
           ∧ receiver1′ = "All_Airplanes"
           ∧ sender2′ = "All_Airplanes"
           ∧ receiver2′ = "ATCS"
           ∧ pc′ = "Timestamp"
           ∧ UNCHANGED ⟨objects, ATCS, Calculations, Report⟩

Timestamp ≜ ∧ pc = "Timestamp"
            ∧ sender1′ = "Authentic_Airplanes"
            ∧ receiver1′ = "Unauthentic_Airplane"
            ∧ sender2′ = "Unauthentic_Airplane"
            ∧ receiver2′ = "Authentic_Airplanes"
            ∧ pc′ = "Report_ATCS"
            ∧ UNCHANGED ⟨objects, ATCS, Calculations, Report⟩

Report_ATCS ≜ ∧ pc = "Report_ATCS"
              ∧ sender1′ = "Authentic_Airplanes"
              ∧ receiver1′ = "ATCS"
              ∧ sender2′ = "Unauthentic_Airplane"
              ∧ receiver2′ = "ATCS"
              ∧ pc′ = "Verification"
              ∧ UNCHANGED ⟨objects, ATCS, Calculations, Report⟩

Verification ≜ ∧ pc = "Verification"
               ∧ ATCS′ = "Calculate"
               ∧ IF Calculations = "Correct"
                 THEN ∧ sender1′ = "ATCS"
                      ∧ receiver1′ = "All_Airplanes"
                      ∧ receiver2′ = "ADSB"
                      ∧ Report′ = "Safe"
                 ELSE ∧ IF Calculations = "Wrong"
                        THEN ∧ sender1′ = "ATCS"
                             ∧ receiver1′ = "All_Airplanes"
                             ∧ receiver2′ = "ADSB"
                             ∧ Report′ = "Unsafe"
                        ELSE ∧ TRUE
                             ∧ UNCHANGED ⟨sender1, receiver1,
                                          receiver2, Report⟩
               ∧ pc′ = "Done"
               ∧ UNCHANGED ⟨objects, sender2, Calculations⟩

Terminating ≜ pc = "Done" ∧ UNCHANGED vars

Next ≜ checking ∨ Timestamp ∨ Report_ATCS ∨ Verification
       ∨ Terminating

Spec ≜ Init ∧ □[Next]_vars

\* Modification History
\* Last modified Wed Aug 26 23:45:46 PDT 2020 by nawar
\* Created Mon Aug 24 21:43:59 PDT 2020 by nawar
```

Figure 45: TLA+ model specification [46]

3.3.3.4 TLC Result

The TLC model checker is a TLA+ tool that is used to check the TLA+ module for any errors and validate it. For our TLA+ ADS-B module, we ran the TLC model checker as shown in figure 46 and the result shows that the model is valid with no errors and no deadlock. Also, as shown in the figure, the TLC model checker shows the number of times each state was visited in the TLA+ module, and which values changed to reach the next state.

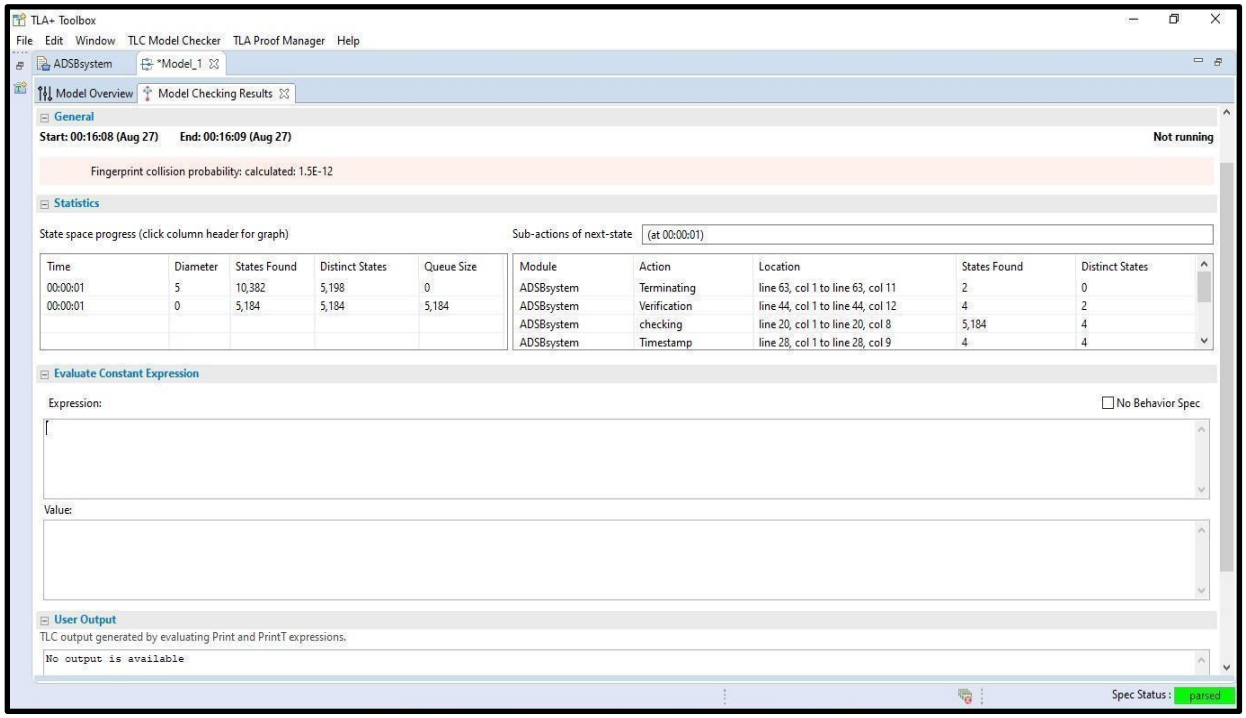


Figure 46: TLC Model checker Validation for TLA+ spec in Figure 45 [46]

3.4 Conclusion

We described Air Traffic Control Surveillance and then we talked about each surveillance technology; PSR, SSR, MLAT, ADS-C, and ADS-B. We talked about the strengths and weaknesses of each one of these technologies. We compared them and found out that the ADS-B is the best system to use because of its strength and cost effectiveness compared to the other surveillance systems.

Although ADS-B is efficient, it has security issues that need to be addressed. The current security solution is added to the system on the design level, but after the initial design has been completed. This makes it harder to find a really strong solution for solving the security issues. In this work, we highlighted the current solutions available, then we picked one of them to be modeled, designed, and finally validated using UML, TLA+, and TLC model checker.

We used multi-level modeling to model a large system like ADS-B; the UML modeling (use case then sequence then state diagrams), then TLA+ specification module, and finally the TLC model. We showed how the system can be modeled and what could fail in it since what TLC does is to go through each iteration and find any failure point. At this level, we can conclude that our procedure in modeling the ADS-B and the ability to test the model is valid. It is always much better to design security into a system at the beginning of the design process. Hopefully, our method of using UML state diagrams, UML sequence diagrams, and TLA+ can provide the reader with a practical and stable ADS-B model that is realistic and secure. Also, it can serve as a model of how to design security into other discrete systems in the future.

There is a weakness in our model in that it is not detailed enough. A real ADS-B system model will be much larger with all subsystems and security systems and many attackers. We only

modeled one attack in this work, a real-world model will have all security systems integrated and all attacks defined and will test them in many ways to see where and when two security systems collide. This is what will be our future work.

Chapter 4: Supervisory Control and Data Acquisition (SCADA) Systems

4.1 Introduction

SCADA is an abbreviation for Supervisory Control and Data Acquisition. SCADA systems are used to control and monitor equipment or plants in industries such as energy, telecommunications, transportation, oil and gas refining, manufacturing, and water and waste control. These systems include the transfer of information between several Remote Terminal Units (RTUs) and a SCADA central host computer or/and Programmable Logic Controllers (PLCs), as well as the operator terminals and the central host. A SCADA system gathers data (such as where a pipeline leakage has occurred), transfers this data back to a central site, and alerts the home station about that leakage, carrying out necessary control and analysis, such as deciding if this leak is critical, and presenting the data in an organized and logical fashion. SCADA systems vary from a relatively simple SCADA system, such as an environmental condition monitoring system of a small office building, to very complex SCADA systems, such as a monitoring system for all the activity of a community water system or the activity in a nuclear power plant. For monitoring purposes, SCADA systems usually use a Public Switched Network (PSN). Today a lot of systems are monitored using a corporate Wide Area Network (WAN) / Local Area Network (LAN) infrastructure. For monitoring purposes, wireless technologies are being widely used now [72]. SCADA systems contain of:

- One or more RTUs or PLCs which connect with data interface devices, they used to sense devices, control valve actuators and switchboxes.

- A communications system to transfer data between control units and field data interface devices and the computers in the SCADA main host. The system could be telephone, radio, satellite, cable, etc., or a combination of any of these.
- A main host computer server (sometimes called a SCADA Center, Master Terminal Unit (MTU), or master station).
- A collection of standard and/or custom software (sometimes called Man Machine Interface (MMI) software or Human Machine Interface (HMI) software) systems used in providing the operator terminal application and SCADA central host, supporting the communications system, and controlling and monitoring field data interface devices remotely [72]. Figure 47 shows a classic SCADA system.

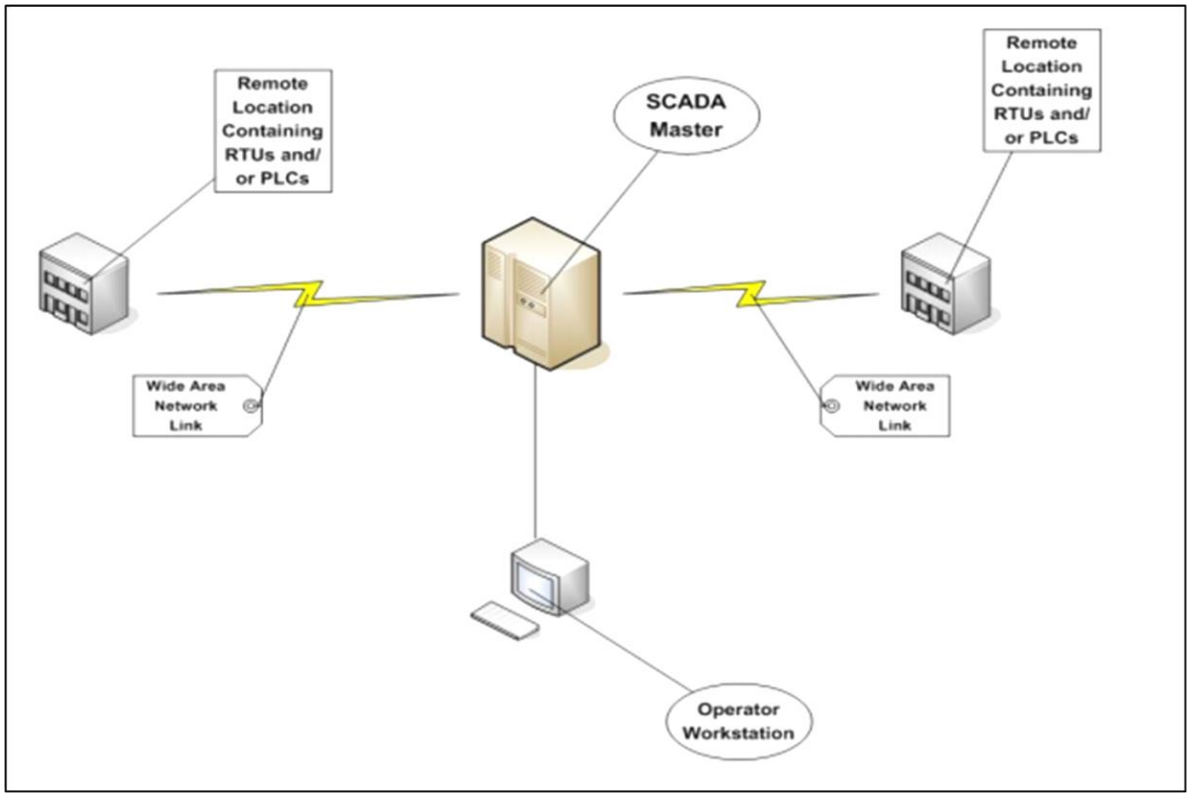


Figure 47: Classic SCADA System [72]

At first, the goal of a SCADA system focused on efficient and accurate process execution in a specific location, for example, a manufacturing plant, without having an emphasis on securing network information. Today, due to the increase in interconnectivity of networks and the remote systems accessibility on a SCADA network, there is increased danger from different cyber-attacks and vulnerabilities. It has become necessary to include adequate safety measures to improve SCADA network security. General safeguards include strong cryptography, patch management, restricted perimeters, and most importantly, control network and corporate network separation, along with in depth security mechanisms. However, these security guards are challenging to apply due to inherited security weaknesses in legacy systems as well as the high risk of exploitation during real-time communication [73].

4.2 SCADA Protocols

Protocols are an essential part of SCADA systems, because they ensure efficient and correct communication between the RTUs and the central station and also between RTUs. Since the advent of SCADA systems, protocols have been generated [74] to optimize communication.

Normally, SCADA systems protocols are specifically designed for the SCADA system use. Within the SCADA system, they ensure efficient communication, but at the same time they represent a hacker's possible attack points. There are a number of known SCADA protocols attacks such as man-in-the-middle attacks [75] and also denial-of-service attacks [76]. Hence, it is essential that the protocols have security features. Additionally, they should not be very complex, because having more complex protocols will increase the risk of mistakes made in the

system. If the system has more decentralized control, it may also be essential to let remote devices communicate directly with each other. Therefore, protocols have to also allow peer-to-peer communication. It also has to be mentioned that the system state might not be consistent when applying peer-to-peer communication, as pointed out in [77]. When designing a protocol, the possible possession of per-to-peer capabilities thus has to be considered. Furthermore, protocols may need to be capable to deal with smart grid security [78] and the Internet of Things (IoT) [79]. Among all the most used SCADA protocols, there is no protocol that is obviously better than the others. Moreover, these protocols might not even be suitable for working with smart grids or the IoT as concluded by [78] where the protocol MQTT (Message Queuing Telemetry Transport) is proposed as a potential alternative. The regularly used DNP3 protocol, the legacy protocol Modbus, and IEC 61850 were tested in [80] and the authors concluded that an IEC 61850 extended version could be useful more than the traditional SCADA protocols. In [77], the authors found out through experiments that neither TCP nor UPD based DNP3 is quick enough in meeting the delay requirements in the smart grids that are controlled by decentralized systems.

4.3 SCADA Systems Security and Vulnerability

4.3.1 Introduction

SCADA systems have developed recently and are often based on open COTS products and open standards these days. Most SCADA hardware and software vendors have supported Internet Protocol (TCP/IP)/Transmission Control Protocol and Ethernet communications, and many of them have used TCP/IP packets to encapsulate their proprietary protocols. All of this evolution

in having more open-based standards made things easier for the industry to integrate many diverse systems, but, on the other hand, it increased the risks of less technical methods of gaining access and less control of industrial networks [72].

4.3.2 Attacks Against SCADA Systems

In today's corporate environment, all corporate communications are typically done using internal networks, including SCADA. Therefore, SCADA systems are vulnerable to most of the same threats that any TCP/IP-based system may face. Industrial Systems Analysts and SCADA Administrators are often tricked into thinking that they are safe against outside attacks because their industrial networks are using separate systems from the ones that the corporate network is using. RTUs and PLCs are usually selected by a third-party vendor-specific protocols and networks such as RS-485, RS-232, DNP, and MODBUS, which are usually done over satellite systems, phone lines, spread and licensed spectrum radios, leased private frame relay circuits, and other token-ring topology bus systems. This frequently gives a false sense of security to the SCADA System Administrators since they assume a protection for these end devices using these non-corporate network connections [81].

In an industrial network, security can be compromised in many ways in the system, and it is most easily compromised at the SCADA control room level or host. SCADA computers logging out the data to certain back-office database repositories should be with the back-end database systems on the same physical network or have a path for accessing these database systems. This means that there will be a path back to the SCADA systems as well as to the end devices and eventually through their corporate network. When the corporate network is compromised, any computer system or IP-based device can be accessed. To allow full-time logging, these

connections are open 24/7, which gives an opportunity to attack the SCADA host system using any of the following kinds of attacks:

- A Denial of Service (DoS) attack can be used to make the SCADA server leading to a shutdown condition (Loss of Operations and System Downtime)
- Deleting the system files on the server (Loss of Operations and System Downtime)
- Taking complete control of system or planting a Trojan
- Preparing for future take down by logging keystrokes from operators to get usernames and passwords
- Logging any company-sensitive operational information for competition or personal usage
- Deceiving operators or changing data points to make it seem that the control process is not functioning and should be shut down (Loss of Corporate Data and Downtime).
- Modifying any logged information in a remote database system (Corporate Data Loss).
- IP spoofing: using the SCADA Server as a starting point to compromise and deface other system elements within the corporate network [82].

In the following section, we will illustrate our work in SCADA system. We used informal and formal languages to design the system and validate it. Also, we will run a security hacking case in our module and show how to find it.

4.4 Our Work

4.4.1 Introduction

In this section we will illustrate our work in using UML, TLA+, and the TLC model checker as an extension of work in [71]. That work focused on building a SCADA system. They did an impressive job building a testbed using one of the University of Cincinnati's labs. The testbed is shown in figure 48.

Figure 48: The PLC/HMI Network and the HMI Output Screen [71]

The question is, what is the benefit of building this SCADA testbed and running simple examples on it? Testing is the answer. SCADA systems are widely used in industrial processes (fabrication, manufacturing, refining), as well as in infrastructure processes (oil and gas pipelines, water treatment, power distribution and generation). Down time in these kinds of systems may causes physical damage, profit loss, and even life loss. This means that, after installation, some SCADA systems may have little to no down time. And when there is down

time, this time will not be used for security testing or system upgrading, it will typically be used for system maintenance. The SCADA testbed is useful because a testbed is a precise model of a real-world process and is reconfigurable and modular. With few hardware devices, it can simulate a large network. Other important uses for the testbed are penetration testing and vulnerability assessment which we will illustrate later.

In [71] the SCADA testbed was built so as to in ease the learning process by breaking up the construction into many different parts. First the hardware was set up and then the latest version of the firmware was installed. Specifically, an Allen-Bradley PanelView 5310 HMI and an Allen-Bradley 1756-A7 PLC were used. After setting up both systems. Ethernet on a simple LAN network was used to integrate them together. Next, a PLC basic program was installed using Studio 5000 Logix Designer to show the connection. This setup is shown in figure 48. And the final SCADA testbed network diagram setup is shown in figure 49.

The PLC is set up to provide the end user with as much data as possible. This is very helpful for PLCs set up after installation where physical access is limited because equipment is in remote locations. The PLC responds with the default configurations to service scans, ARP requests, ping requests, and TCP port scans. This provides data including applications that run on the PLC, MAC address, and the operating system of the devices. Furthermore, the PLC is hosting a website as well that has data about the firmware version, program running, controller mode, fault status, and controller status as shown in Figure 50. This valuable information could be used to exploit the system internally using the MAC address or externally using the website itself.

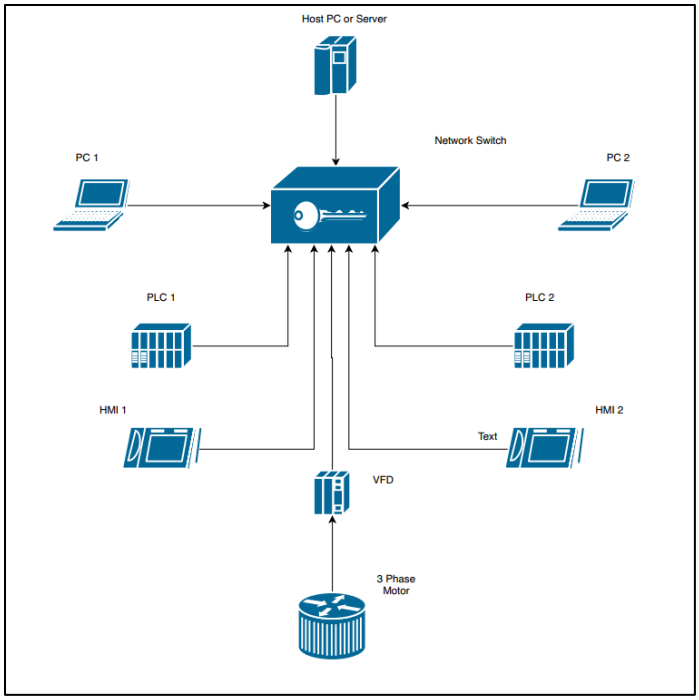


Figure 49: Network diagram of final SCADA testbed setup [71]

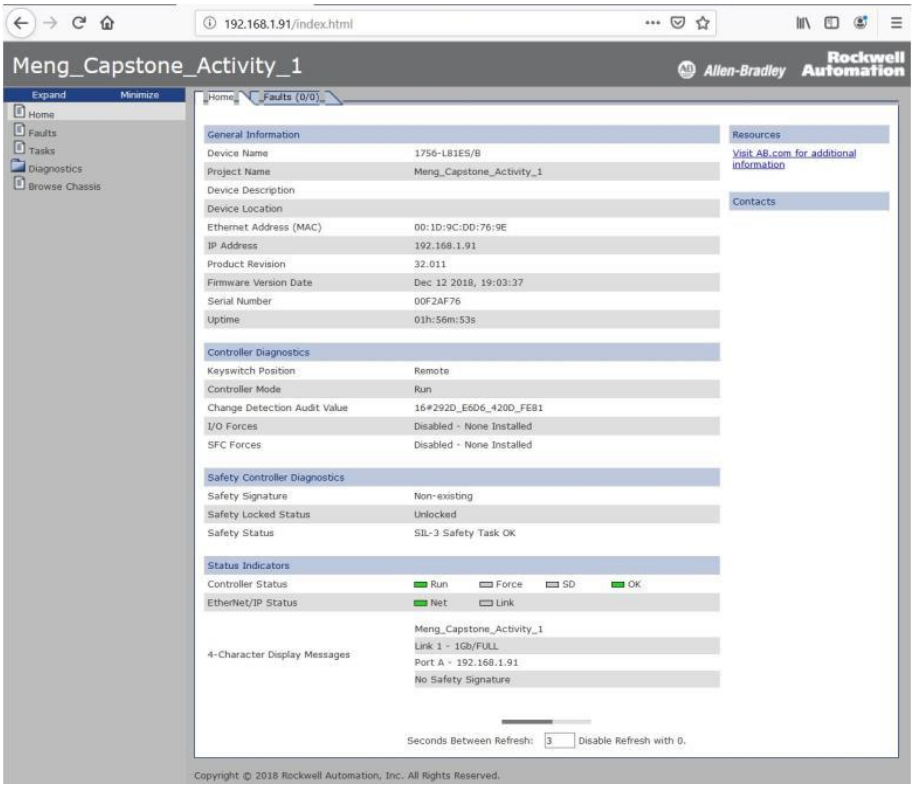


Figure 50: Homepage of local server hosted by the PLC [71]

In [71], Pycomm was used to access the PLCs in the system. It is available at https://pypi.org/project/pycomm/. This version compatible with Python 2.7 and has an ab-comm module which can be interfaced with the Allen Bradley PLCs by the Ethernet/IP protocol.

They used one laptop to download the Pycomm software, plugged it into the switch, and then connected it to the local network to communicate with the PLC. In a real production environment, wireless connection or this type of connection is exactly how engineers would communicate with their PLC controller.

They used Pycomm to write a code to do the following:

1) Use IP address to communicate with the PLC controller.

2) Read the word's tag value.

3) Print that value.

4) Write a value to the PLC tag.

5) Close the connection.

As shown above, the authors in [71] built and tested the SCADA testbed and they ran a dishwasher example using it. The following section will illustrate the dishwasher example in detail. First, we used the dishwasher example as a basis for our work. Then we modeled it using a UML state diagram, and a TLA+ specification module. Next, we used the TLC model checker to validate our module.

4.4.2 Implementing a Dishwasher Example

4.4.2.1 UML Modeling

In this section, we will illustrate how we modeled the SCADA testbed dishwasher example using the UML state diagram.

The dishwasher example sequence of actions as illustrated [71] were:

1. Open the soap solenoid for 4 seconds.
2. Open the hot water input valve for 5 minutes.
3. Open the washer impeller for 12 seconds.
4. Open the rainwater valve for 1 second.
5. Open the drain for 3 seconds.
6. Turn on the heat for 6 seconds.

We used a UML state diagram to show these steps as shown in figure 51.

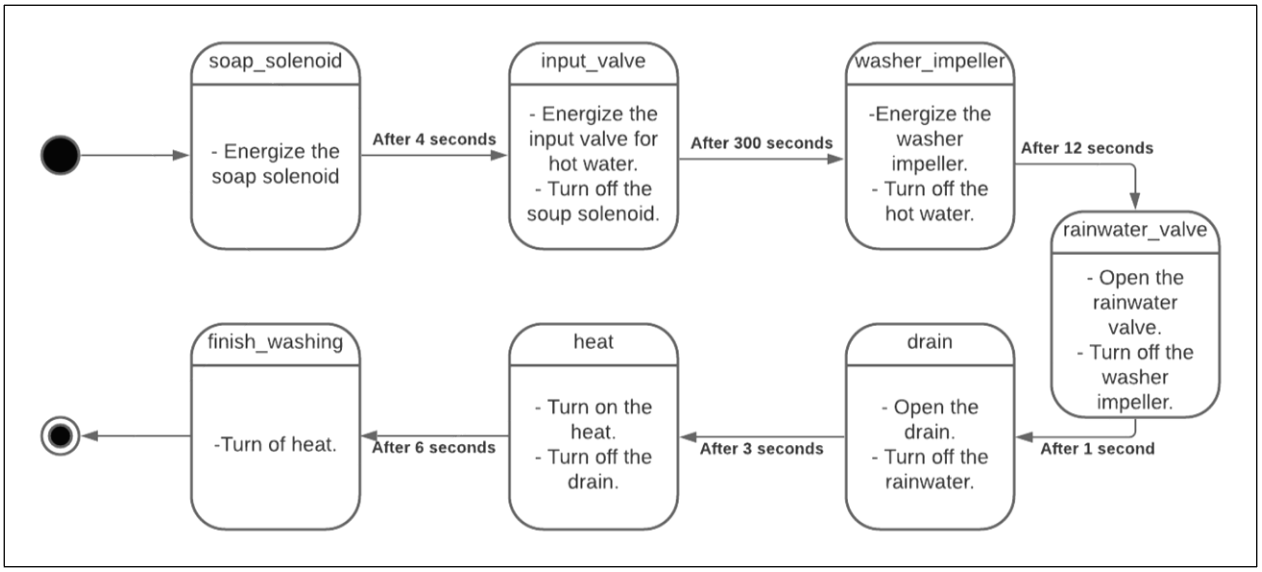


Figure 51: UML state diagram for the dishwasher example

The figure illustrates what is happening in each state. In the soap_solenoid state, the soap solenoid will be opened. After 4 seconds, we will move to the next state which is the input_valve state. In this state, the soap solenoid will be turned off and the hot water input valve will be turned on. After 5 minutes (300 seconds), we will move to the water_impeller state. In this state, the hot water will be turned off and the washer impeller will be turned on. After 12 seconds, we will move to the rainwater_valve state. In this state, the washer impeller will be turned off and rainwater turned on. After 1 second, we will move to the drain state. In this state, the rainwater will be turned off and the drain will be opened. After 3 seconds, we will move to the heat state. In this state, the drain will be turned off and the heat will be turned on. After 6 seconds, we will move to the finish_washing state when the heat will be turned off. This is the last state in this state diagram. This state diagram will help in implementing the TLA+ specifications in the next section.

4.4.2.2 TLA+ Specifications

In this section, we modeled a TLA+ module called *scadaTest* to implement the dishwasher example in [71].

MODULE *scadaTest*

EXTENDS *Integers*, *TLC*

VARIABLES *objects*, *soap_solenoid*, *input_valve*, *washer_impeller*, *rainwater_valve*, *drain*, *heat*, *whole_washing_time*, *time_in_seconds*, *pc*

1. Energize the soap solenoid first for four seconds
2. Energize the input valve for hot water for five minutes
3. Energize the washer impeller for twelve seconds
4. Open the rainwater valve for one second
5. Open the drain for three seconds
6. Turn on the heat for six seconds.

$timer \triangleq time\_in\_seconds \leq 300$

$total\_time \triangleq whole\_washing\_time \leq 326$

Figure 52: scadaTest TLA+ module, variables, and invariants

Figure 52 shows the *scadaTest* module, the variables, and the invariants in the module. These variables include the module states along with other variables such as *time_in_seconds* variable which represents the time (in seconds) needed for each state, and the *whole_washing_time* variable which represents the total time needed for the dishwasher to finish the whole washing cycle from start to the end.

The invariants in this module are represented by *timer* which guarantees that the time required in each state should not exceed 300 seconds. Also, the *total_time* invariant in this module guarantees that total washing time per cycle should not exceed 326 seconds.

$$
\begin{array}{lll}
vars & \triangleq & \langle objects,\ soap\_solenoid,\ input\_valve,\ washer\_impeller,\\
 & & \ rainwater\_valve,\ drain,\ heat,\ whole\_washing\_time,\ time\_in\_seconds,\\
 & & \ pc\rangle \\
Init & \triangleq & \land\ objects = \{\text{“soap\_solenoid”},\ \text{“input\_valve”},\ \text{“washer\_impeller”},\\
 & & \qquad\qquad \text{“rainwater\_valve”},\ \text{“drain”},\ \text{“heat”},\ \text{“timer”},\\
 & & \qquad\qquad \text{“time\_in\_seconds”}\} \\
 & & \land\ soap\_solenoid \in \{\text{“open”},\ \text{“closed”}\} \\
 & & \land\ input\_valve \in \{\text{“open”},\ \text{“closed”}\} \\
 & & \land\ washer\_impeller \in \{\text{“open”},\ \text{“closed”}\} \\
 & & \land\ rainwater\_valve\ \in \{\text{“open”},\ \text{“closed”}\} \\
 & & \land\ drain \in \{\text{“open”},\ \text{“closed”}\} \\
 & & \land\ heat \in \{\text{“open”},\ \text{“closed”}\} \\
 & & \land\ whole\_washing\_time = 0 \\
 & & \land\ time\_in\_seconds \in 0\ldots 300 \\
 & & \land\ pc = \text{“soap\_solenoid\_”}
\end{array}
$$

Figure 53: scadaTest module vars and Init function

Figure 53 shows the *vars* which represents a tuple of the variables, and the *Init* function which represents the initial values of the variables. In the *Init* function, we also can declare all possible values as shown in the figure.

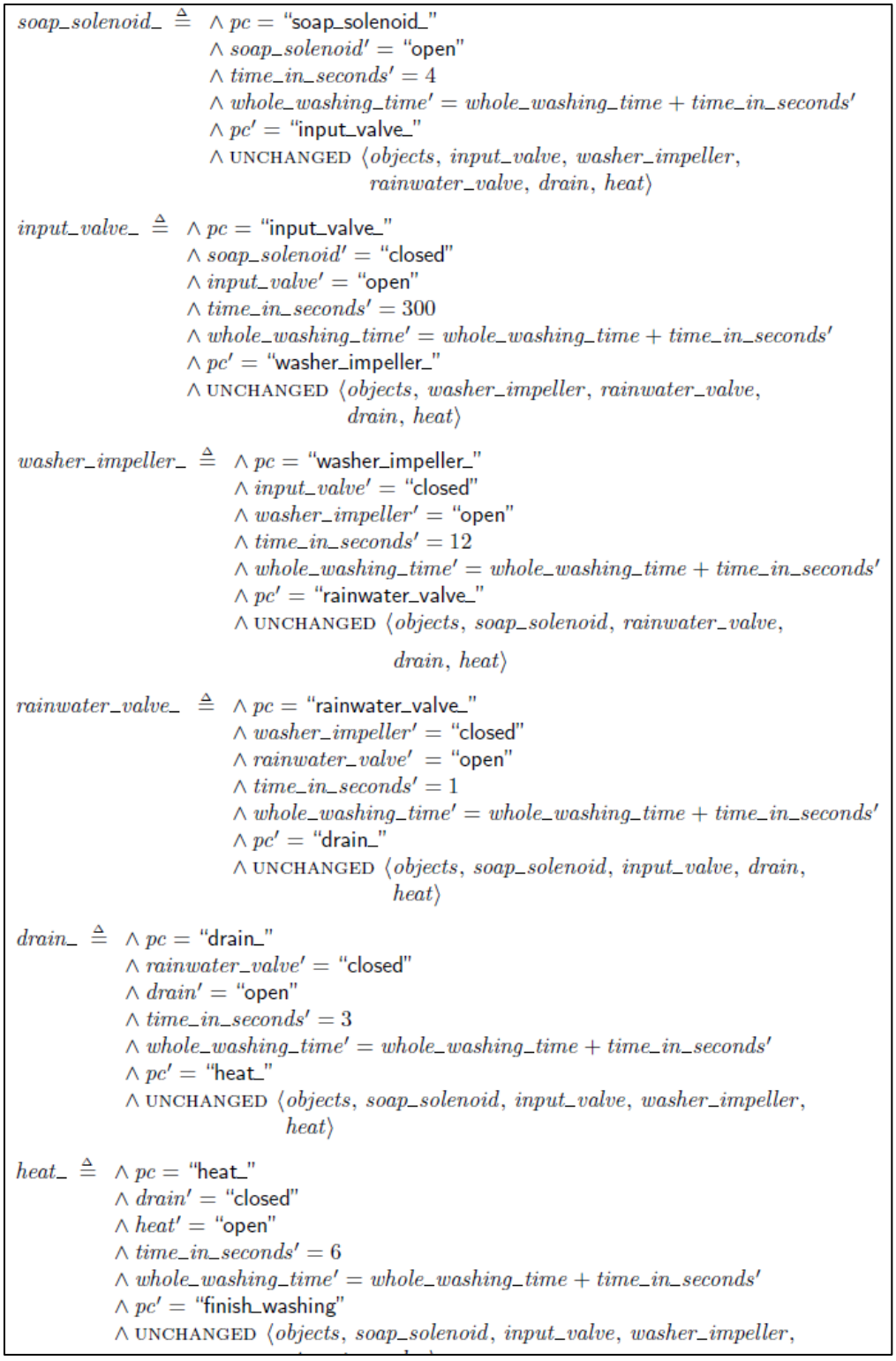


Figure 54: scadaTest module functions

Figure 54 represents the *soap_solenoid_*, *input_valve_*, *washer_impeller_, rainwater_valve_, drain,* and *heat* functions in the *scadaTest* TLA+ module. In the *soap_solenoid_* function, the soap solenoid will be opened for 4 seconds, the whole_washing_time will be calculated as the previous washing time (which was initially 0) and the time needed by each state to be finished which will be 4 seconds in total by the end of this state. The next state will be *input_valve_,* and no other variables will be changed.

In the *input_valve_* function, the soap solenoid will be closed, and the hot water input valve will be opened for 300 seconds (5 minutes), the *whole_washing_time* will be calculated as the previous washing time (which was 4) and the time needed by this state to be finished which will be 304 seconds in total by the end of this state. The next state will be *washer_impeller_,* and no other variables will be changed.

In the *washer_impeller_* function, the hot water input valve will be closed, and the washer impeller will be opened for 12 seconds, the *whole_washing_time* will be calculated as the previous washing time (which was 304) and the time needed by this state to be finished which will be 316 seconds in total by the end of this state. The next state will be *rainwater_valve_,* and no other variables will be changed.

In the *rainwater_valve_* function, the washer impeller will be closed, and the rainwater valve will be opened for 1 second, the *whole_washing_time* will be calculated as the previous washing time (which was 316) and the time needed by this state to be finished which will be 317 seconds in total by the end of this state. The next state will be *drain,* and no other variables will be changed.

In the *drain* function, the *rainwater_valve_* will be closed, and the drain will be opened for 3 seconds, the *whole_washing_time* will be calculated as the previous washing time (which was 317) and the time needed by this state to be finished which will be 320 seconds in total by the end of this state. The next state will be *heat,* and no other variables will be changed.

In the *heat* function, the drain will be closed, and the heat will be opened for 6 seconds, the *whole_washing_time* will be calculated as the previous washing time (which was 320) and the time needed by this state to be finished which will be 326 seconds in total by the end of this state. The next state will be *finish_washing,* and no other variables will be changed.

$$
\begin{aligned}
finish\_washing \triangleq\ & \land\ pc = \text{“finish\_washing”} \\
& \land\ heat' = \text{“closed”} \\
& \land\ time\_in\_seconds' = 0 \\
& \land\ whole\_washing\_time' = whole\_washing\_time + time\_in\_seconds' \\
& \land\ pc' = \text{“Done”} \\
& \land\ \text{UNCHANGED}\ \langle objects,\ soap\_solenoid,\ input\_valve, \\
& \qquad\qquad\qquad\qquad washer\_impeller,\ rainwater\_valve,\ drain \rangle
\end{aligned}
$$

$$
Terminating \triangleq pc = \text{“Done”} \land \text{UNCHANGED}\ vars
$$

$$
\begin{aligned}
Next \triangleq\ & soap\_solenoid\_ \lor input\_valve\_ \lor washer\_impeller\_ \\
& \lor rainwater\_valve\_ \lor drain\_ \lor heat\_ \lor finish\_washing \\
& \lor Terminating
\end{aligned}
$$

$$
Spec \triangleq Init \land \Box[Next]_{vars}
$$

$$
Termination \triangleq \Diamond(pc = \text{“Done”})
$$

Figure 55: finish_washing, Terminating, Next, Spec, and Termination functions

Figure 55 represents *finish_washing, Terminating, Next, Spec,* and *Termination* functions in the *scadaTest* TLA+ module.

In the *finish_washing* function, the heat will be turned off, the *whole_washing_time* will be calculated as the previous washing time (which was 326) and the time needed by this state to be finished which will be 326 seconds in total by the end of this state since this state will be responsible for finishing the washing cycle. The next state will be Done state, and no other variables will be changed.

As we mentioned in the previous chapters, the Next function represents all states in the Spec. The Spec is the main function that runs all module specification. And the Termination state is responsible for terminating the module.

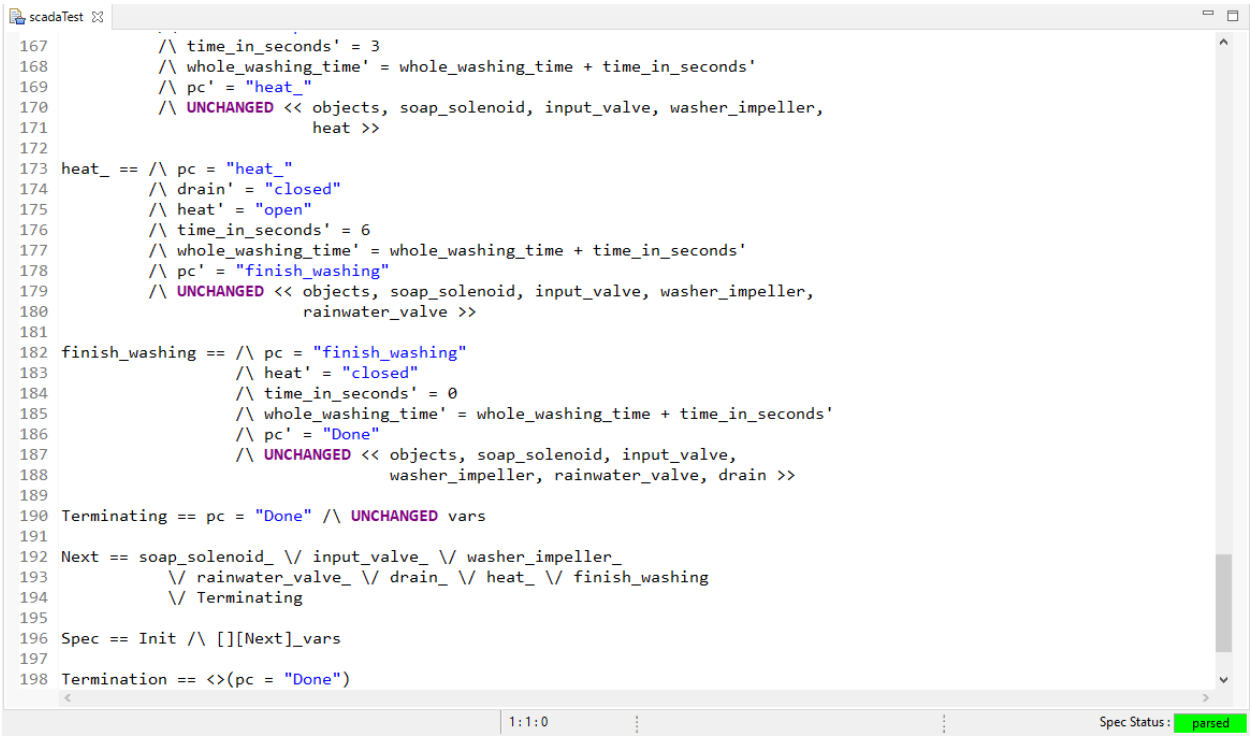


Figure 56: scadaTest parsed moule

Figure 56 represents the *scadaTest* module which was parsed successfully using the TLA+ Toolbox as shown by the green button on the bottom right of the screen. In order to validate our module, we ran the TLC model checker. The result will be represented in the next section.

4.4.2.3 TLC Model Checker

In the previous section, we successfully parsed the TLA+ scadaTest module. In this section we will prove that our module is validated using the TLC model checker.

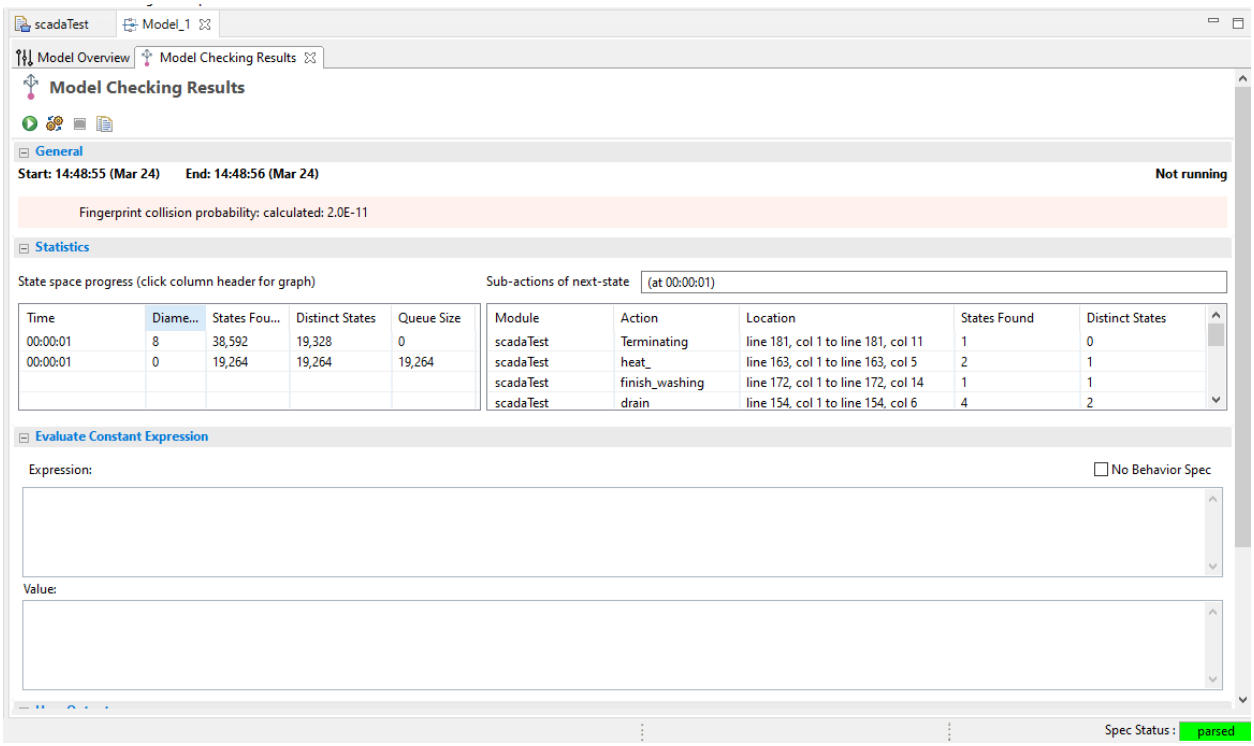


Figure 57: The TLC model checker results

Figure 57 shows that the module was validated using the TLC model checker with no errors or bugs. This is the final result for the module. But there is an important question, what if this system is vulnerable? We will find the answer in the following section.

4.4.3 TLC Finds Security Break

As was mentioned before, SCADA systems are vulnerable. Authors in [71] tried a case of security breakage in their SCADA dishwasher example. In this section we will illustrate a security breaking case in our module, and we will illustrate how to find this bug in the module.

As we mentioned before in our *scadaTest* module, the time required for the *input_valve_* state to be finished is 300 seconds which is the longest time needed in all states of the module. Also, we set up an invariant in the module called "timer" which guarantees that the time needed for any state should be less than or equal to 300 seconds.

As a security hacking case, we changed the time needed in the *input_valve_* state to 600 seconds. We change it manually in the TLA+ specifications as shown in figure 58.

```
input_valve_ == /\ pc = "input_valve_"
                /\ soap_solenoid' = "closed"
                /\ input_valve' = "open"
                /\ time_in_seconds' = 600
                /\ whole_washing_time' = whole_washing_time + time_in_seconds'
                /\ pc' = "washer_impeller_"
                /\ UNCHANGED << objects, washer_impeller, rainwater_valve,
                                drain, heat >>
```

Figure 58: time_in_seconds variable is changed to break security

We ran the TLA+ toolbox with breaking the "timer" invariant. Surprisingly, the TLA+ module ran successfully and parsed correctly with no errors. But how we will find this error then?

We ran the TLC model checker, but first, we set up the invariant in the TLC model checker as shown in figure 59. These invariants are the same in the TLA+ module, but the TLA+ toolbox was not able to find the error.

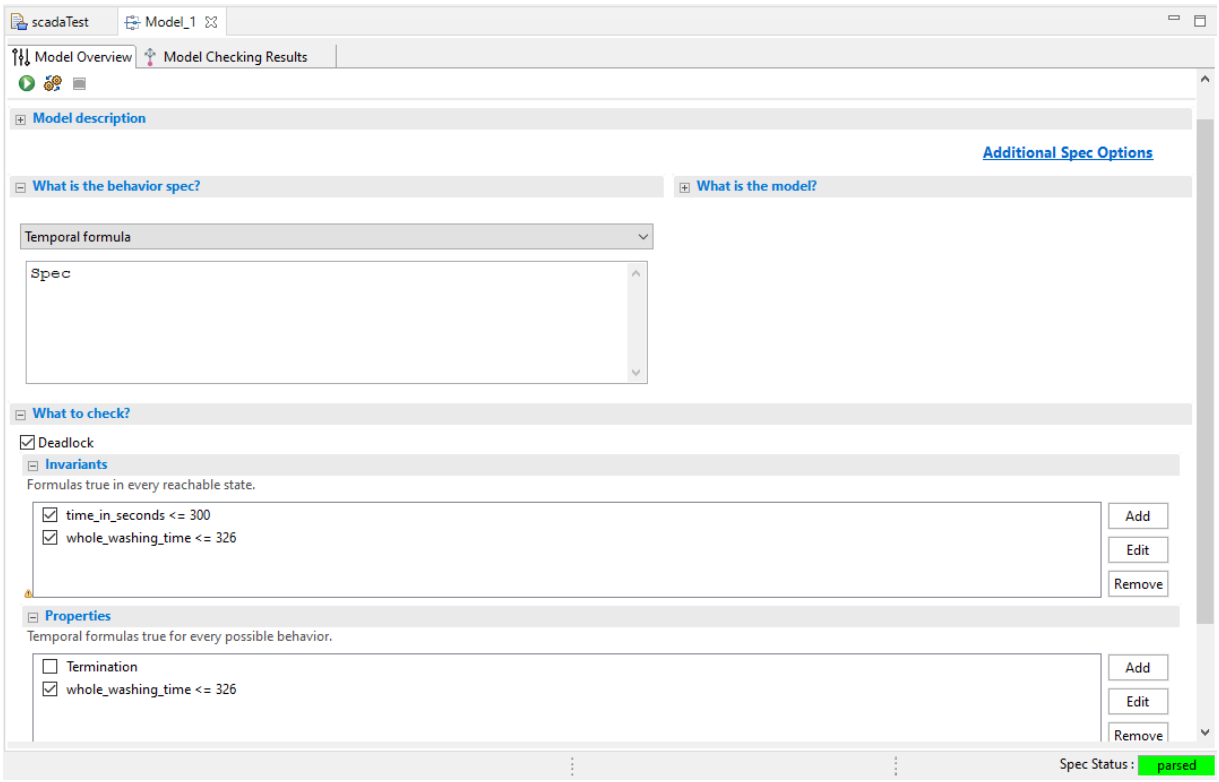


Figure 59: TLC model checker invariants

After setting up the invariants, we ran the TLC model checker and it threw an error, and error tracing (in the bottom right side), and a message (in the top right side) that said, “Invariant time_in_seconds <= 300 is violated”. The yellow highlighted states will not be visited since this invariant violation was found before reaching these states. All this is shown in figure 60.

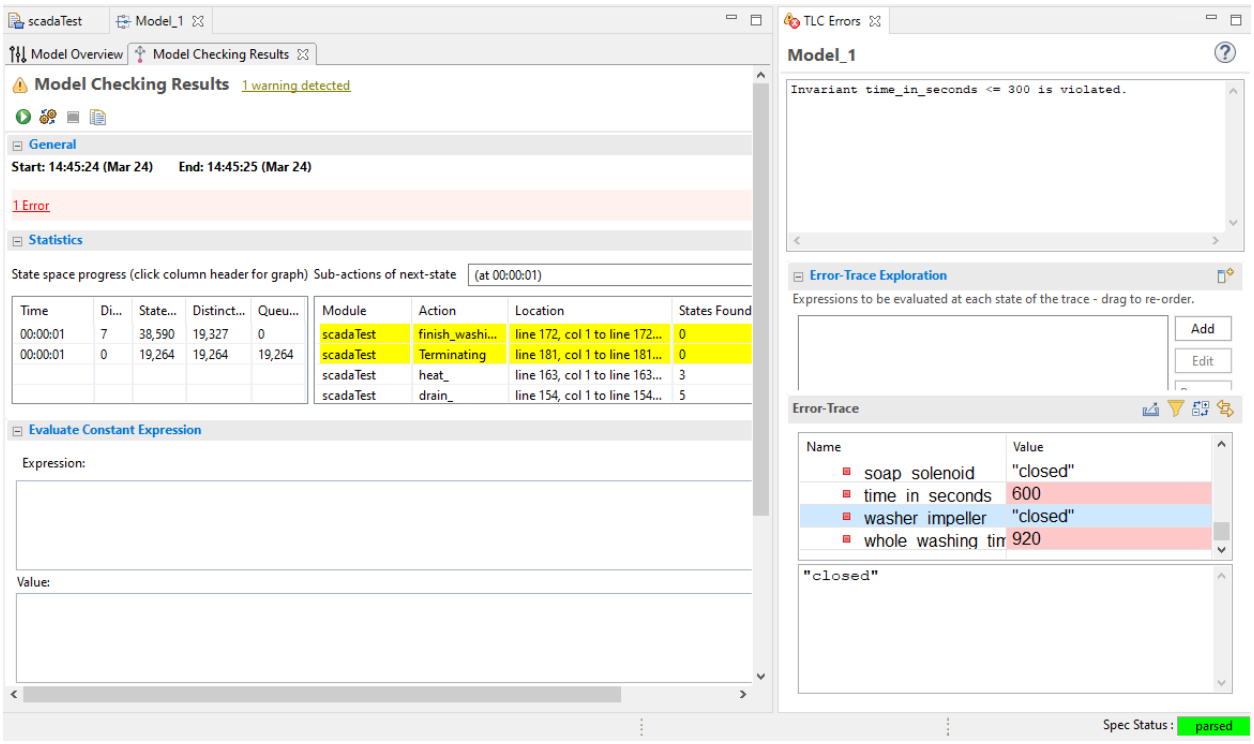


Figure 60: TLC model checker found the security hacking error

We analyzed in a particular security hack when *time_in _seconds* variable equals 600, which exceeds the maximum needed for any washing cycle. We set the TLC invariant to find an error in this particular case (when time_in_seconds >= 300). What if someone hacked the system and changed any of the washing cycle times? Any time could be made larger or smaller by a hacker. As an example, we chose the case where the hacker changed the *time_in_seconds* in the *washer_impeller_* state from 12 to 120 seconds, which is still less than 300 seconds and is hard to detect through the current invariants.

We implemented this security hack as shown in figure 61. The change was set up manually (line 141).

```
138 washer_impeller_ == /\ pc = "washer_impeller_"
139                     /\ input_valve' = "closed"
140                     /\ washer_impeller' = "open"
141                     /\ time_in_seconds' = 120
142                     /\ whole_washing_time' = whole_washing_time + time_in_seconds'
143                     /\ pc' = "rainwater_valve_"
144                     /\ UNCHANGED << objects, soap_solenoid, rainwater_valve,
145                                     drain, heat >>
```

Figure 61: *time_in_seconds* Variable is Changed to 120 to Break Security

Before running the TLA+ toolbox, we changed the timer invariant to be:

```
timer == time_in_seconds \in {4, 300, 12, 3, 6, 1, 0}
```

We then ran the TLA+ toolbox and it parsed the module successfully but was unable to find the security hack. In order to find the security hack, we re-set up the TLC model invariant to the modified timer invariant as shown in figure 62. This updated invariant can catch any security hack that has changed any time needed for any of the cycles in the system.

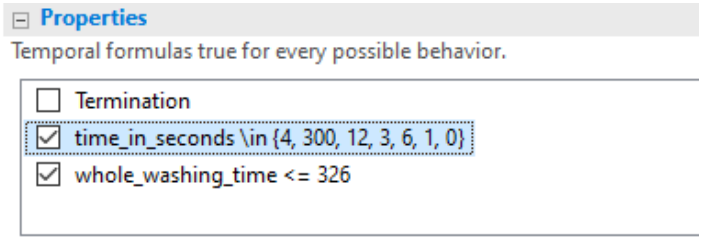


Figure 62: TLC Updated Invariants for *time_in_seconds* variable

After setting up the TLC invariants, we ran the TLC model checker, and the TLC threw an error. Error tracing (in the bottom right side), and a message (in the top right side) said, “Invariant time_in_seconds \in {4, 300, 12, 3, 6, 1, 0} is violated”. The yellow highlighted states will not be visited since this invariant violation was found before these states were reached. All this is shown in figure 63.

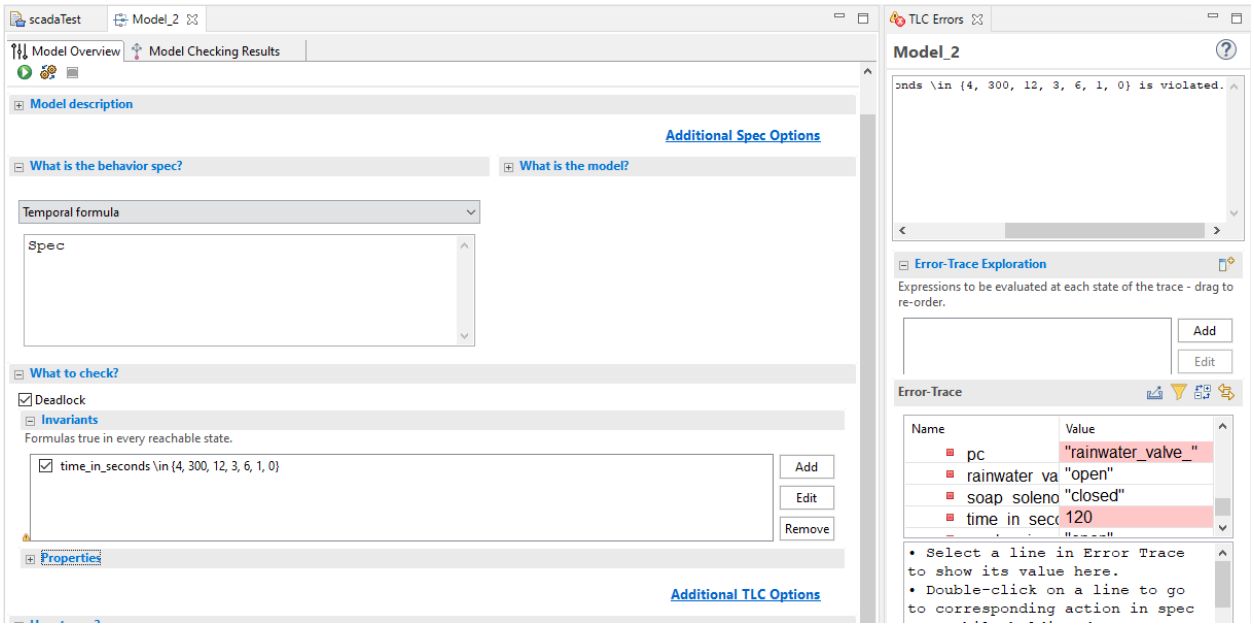


Figure 63: TLC model checker found the security hacking error

As a result, the TLC model checker is not only validating the TLA+ modules but is also finding the errors and bugs in the design as shown before. This model is for a dishwasher, but similar models to check security could be developed for similar but more complicated systems such as transportation systems (railways, highways, city traffic, etc.), water purification systems, electric power systems, material storage and retrieval systems, and other SCADA systems.

4.5 Conclusion

In this chapter, we discussed SCADA systems and their vulnerabilities, and we also discussed specific possible attacks on SCADA systems.

We also described the work in [71] and showed how the authors built a SCADA testbed in their lab. We described the system and how it works, and we also explained the dishwasher example.

As an extension of this work, we used the dishwasher example to first, informally model it using the UML state diagram. Then we used the formal TLA+ specification language to design it and simulate it successfully. The last step was to validate our model using the TLC model checker which validated it successfully.

Because an important part of our research is to include security in our systems, we implemented a case of security hacking in our dishwasher module, and we proved that the TLC model checker is powerful since it not only validated the TLA+ modules, but also found the security errors and bugs implemented in the modules.

Our future work is to analyze more complex SCADA examples, and to find more security issues and how to harden these systems against them.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

In this work, we were working on designing safe and secure systems using formal methods. Our methodology was to design a system informally first using UML, and then to design the system module using the TLA+ specification language and its TLC model checker to validate our module. To improve system safety, we designed the model to have some safety invariants in the model specifications and worked on the sub-systems to achieve the system's desired quality. To achieve security in our models, we included some security invariants. We also defined sub-systems in some modules to support system quality. Moreover, we ran some security break cases in different modules and proved that the TLC model checker can find these security breaks. This was an additional step in modeling and designing secure systems using formal methods.

We have made a number of contributions. First, we showed by examples that formal methods are powerful tools for designing different systems applications. We designed three different systems using UML and TLA+, the smart school building system (two versions [35, 83]), the ADS-B system, and a SCADA system. In all these systems, we designed, modeled, and parsed our models successfully using the TLA+ Toolbox. Then, we proved and validated these models using the TLC model checker. Another important contribution is the use of the TLC model checker to find the security break in the smart school model (version II), and the SCADA system. This showed how the TLC model checker is not only powerful in validation, but also in finding errors, bugs, and security breaks in the model.

Our contribution in each system was different. In the smart school building system [45], we designed the model to be smart and concentrated on integrating different sub-systems that work together to define a one smart module that also achieves safety and security as one of the main goals. Most reserchers did not design smart schools or they only focused on having a design that reduced the power consumption in the building, which was one of our goals as well. Moreover, we proved the efficiency of using formal methods in designing smart systems as well as in finding security breaks in the system using TLA+ and the TLC model checker. The smart school building system has different sub-systems: the login sub-system which requires a username and password to enter the building from anyone (student, employee, and visitor). This sub-system increases the security and safety as a result in the design. The lighting sub-system which is responsible for controlling the lights in the building, is designed to be smart and to reduces the power consumption in the building. The HVAC sub-system, which is responsible for controlling the temperature in the building, is designed to be smart and to reduces the power consumption in the building as well. The smoke detecting system, which is responsible for sensing the smoke/fire in the building, is designed in a way to be smart and to achieve safety in the building by working 24/7. We also added another component to the system which is a surveillance camera to increase safety by recording everything. This smart school building system was designed using UML diagrams and a TLA+ specification module and validated using the TLC model checker. To prove the security in our model, we considered a case where the camera, which was designed to be on 24/7, was turned off, and the TLC model checker found this bug, demonstrating that it can catch errors and bugs that may be hard to identify. And it can find security weaknesses in a design. Our work on the school system has been published in [35] and [45].

Another system that we worked on was the ADS-B avionics system. This system is already built and in use, but it is a vulnerable system since when it was designed, the designers did not add security to it. Many researchers have tried to find a solution for the security issues in this system. We redesigned the ADS-B, adding a security solution. The solution was to make the airplanes send timestamps to each other and to the ATCS. At the same time, the GNSS will broadcast the locations of all airplanes which will reached to the ATCS. The ATCS will then calculate the locations and compare the information to determine if an airplane is safe or it is a malicious one. It will then broadcast the result to all airplanes. This design was implemented using UML diagrams first. And then the TLA+ specifications module was used. The module was validated using the TLC model checker. This work was published in [46].

We also worked on a SCADA system. We extended the work in [71] which described building a SCADA testbed using hardware and software components. That work implemented a dishwasher example using the SCADA testbed. We designed this dishwasher example using a UML state diagram and the TLA+ specifications module. The model was validated using the TLC model checker. To prove that the TLC model checker can find the security flaws in the model, we ran a case where the TLA+ module included hacking and showed how the TLC model checker found this bug. This proves that using formal methods is efficient in finding security flaws in a design. This work is being submitted for publication.

5.2 Publications Resulting from the Dissertation

This research has produced the following publications:

1. **Conference paper — Smart school building system (version I)**
   Nawar Obeidat, and Carla Purdy. "Modeling a Smart School Building System Using UML and TLA+." In 2020 3rd International Conference on Information and Computer Technologies (ICICT), pp. 131-136. IEEE, 2020. **(Published)**

2. **Journal paper — Smart school building system (version II)**
   Nawar Obeidat, and Carla Purdy. " Using Formal Methods to Model a Smart School System via TLA+ and its TLC Model Checker for Validation " 2021 ASTESJ Advances in Science, Technology and Engineering Systems Journal. ASTESJ, 2021. **(Published)**

3. **Journal paper — ADS-B system**
   Pranay Bhardwaj, Nawar Obeidat, and Carla Purdy. "A novel way to design ADS-B using UML and TLA+ with security as a focus." 2020 ASTESJ Advances in Science, Technology and Engineering Systems Journal. ASTESJ, 2020. **(Published)**

4. **Conference paper — SCADA system**
   Nawar Obeidat and Carla Purdy. “Improving security in SCADA systems through model-checking using TLA+” 2021 64th International IEEE Midwest Symposium on Circuits and Systems, Lansing Michigan, August 2021. **(Submitted)**

5.3 Future Work

In the smart school building system, future work can be in adding more sub-systems to the school design. Also, adding more security features like sending an alarm in case of fire can be added. In addition, we can add a method for specifying the number of sensors needed related to the size of the building and the number of rooms. For example, the building in this current system work is a theoretical model that uses one sensor for light in the building, which is in real life is not enough. The building needs more light sensors in different locations inside the

building. Can we determine an optimal placement? These kinds of design details can be added to the current design to make it more realistic, secure, and safe. Additional work that can be done in future is to add a number of hardware inputs and outputs to the system along with the specifications for variables such as time rates of updates for the input/output ports.

In the ADS-B system design, a weakness in our model is that it is not detailed enough. A real ADS-B system model will be larger and much more detailed. It will have more subsystems and security systems, and many attackers. We only modeled one attaching scenario. In future, the goal is to design a more detailed model that can handle different kinds of attaches and be closer to the real ADS-B system.

In the SCADA system, we modeled a dishwasher example and demonstrated one security case. In future, we can work on modeling a more complicated SCADA system and making it hardened against different kinds of attacks, not only one case.

References

[1] Zhang, Hehua, Stephan Merz, and Ming Gu. "Specifying and verifying PLC systems with TLA+: A case study." Computers & Mathematics with Applications 60, no. 3 (2010): 695-705.

[2] Newcombe, Chris, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. "How Amazon web services uses formal methods." Communications of the ACM 58, no. 4 (2015): 66-73.

[3] Zhang, Hehua, Ming Gu, and Xiaoyu Song. "Specifying time-sensitive systems with TLA+." In 2010 IEEE 34th Annual Computer Software and Applications Conference, pp. 425-430. IEEE, 2010.

[4] Newcombe, Chris. "Why amazon chose TLA+." In International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z, pp. 25-39. Springer, Berlin, Heidelberg, 2014.

[5] Bjørner, Dines, and Martin C. Henson, eds. Logics of specification languages (The Specification Language TLA+) Page 401- 448 written by Stephan Merz. Springer Science & Business Media, 2007.

[6] Narayana, Prasad, Ruiming Chen, Yao Zhao, Yan Chen, Zhi Fu, and Hai Zhou. "Automatic vulnerability checking of IEEE 802.16 WiMAX protocols through TLA+." In 2006 2nd IEEE Workshop on Secure Network Protocols, pp. 44-49. IEEE, 2006.

[7] Lamport, Leslie. Specifying systems: the TLA+ language and tools for hardware and software engineers. Addison-Wesley Longman Publishing Co., Inc., 2002.

[8] Tasiran, Serdar, Yuan Yu, Brannon Batson, and Scott Kreider. "Using formal specifications to monitor and guide simulation: Verifying the cache coherence engine of the Alpha 21364 microprocessor." In In Proceedings of the 3rd IEEE Workshop on Microprocessor Test and Verification, Common Challenges and Solutions. 2002.

[9] Börger, Egon. "The abstract state machines method for high-level system design and analysis." In Formal Methods: State of the Art and New Directions, pp. 79-116. Springer, London, 2010.

[10] Verification and Validation. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/Verification_and_validation.

[11] Wang, Hsiao-Hsuan, and William E. Grant. "How good ("valid") are models?." In Developments in Environmental Modelling, vol. 31, pp. 191-214. Elsevier, 2019.

[12] Zamani, Majid, and Damien Zufferey. "Numerical Software Verification." 2019.

[13] Thacker, Robert A., Kevin R. Jones, Chris J. Myers, and Hao Zheng. "Automatic abstraction for verification of cyber-physical systems." In Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems, pp. 12-21. ACM, 2010.

[14] Baheti, Radhakisan, and Helen Gill. "Cyber-physical systems." The impact of control technology 12, no. 1 (2011): 161-166.

[15] Formal Methods. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/Formal_methods

[16] Specification Languages. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/Specification_language

[17] RAISE. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/Rigorous_Approach_to_Industrial_Software_Engineering

[18] TLA+. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/TLA%2B

[19] Z. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/Z_notation

[20] VDM. Wikipedia. [Online], Available 905/13/2021):

https://en.wikipedia.org/wiki/Vienna_Development_Method

[21] B. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/B-Method

[22] Rehman, Aniqa, Saba Latif, and Nazir Ahmad Zafar. "Formal Modeling of Smart office using Activity Diagram and Non Deterministic Finite Automata." In 2019 International Conference on Information Science and Communication Technology (ICISCT), pp. 1-5. IEEE, 2019.

[23] Zafar, Nazir Ahmad. "Modeling and formal specification of automated train control system using Z notation." In 2006 IEEE International Multitopic Conference, pp. 438-443. IEEE, 2006.

[24] Afzaal, Hamra, and Nazir Ahmad Zafar. "Formal modeling and algorithm of subnet-based backup assigning in WSAN." In 2015 International Conference on Information and Communication Technologies (ICICT), pp. 1-6. IEEE, 2015.

[25] Kamali, Maryam, Louise A. Dennis, Owen McAree, Michael Fisher, and Sandor M. Veres. "Formal verification of autonomous vehicle platooning." Science of computer programming 148 (2017): 88-106.

[26] Latif, Saba, Aniqa Rehman, and Nazir Ahmad Zafar. "NFA Based Formal Modeling of Smart Parking System Using TLA+." In 2019 International Conference on Information Science and Communication Technology (ICISCT), pp. 1-6. IEEE, 2019.

[27] The TLA+ Video Course. By: Lamport. [Online], Available (05/13/2021): http://lamport.azurewebsites.net/video/videos.html

[28] Verhulst, Eric, Raymond T. Boute, José Miguel Sampaio Faria, Bernhard HC Sputh, and Vitaliy Mezhuyev. Formal Development of a Network-Centric RTOS: software engineering for reliable embedded systems. Springer Science & Business Media, 2011.

[29] Clark, Anthony, and Andy Evans. "Foundations of the Unified Modeling Language." In Proceedigs of the 2nd Northern Formal Methods Workshop. Springer, 1997.

[30] Pocero, Lidia, Dimitrios Amaxilatis, Georgios Mylonas, and Ioannis Chatzigiannakis. "Open source IoT meter devices for smart and energy-efficient school buildings." HardwareX 1 (2017): 54-67.

[31] Amaxilatis, Dimitrios, Ioannis Chatzigiannakis, and Georgios Mylonas. "Design and Implementation of a Platform for Smart Connected School Buildings." In AmI (Workshops/Posters). 2015.

[32] Brogan, Mike, and Alfio Galata. "The VERYSchool Project: Valuable EneRgY for a Smart School-Intelligent ISO 50001 Energy Management Decision Making in School Buildings." In AIAI Workshops, pp. 46-58. 2015.

[33] Hirsch, Benjamin, and Jason WP Ng. "Education beyond the cloud: Anytime-anywhere learning in a smart campus environment." 2011 International Conference for Internet Technology and Secured Transactions. IEEE, 2011.

[34] Veeramanickam, M. R. M., and M. Mohanapriya. "Iot enabled futurus smart campus with effective e-learning: i-campus." GSTF journal of Engineering Technology (JET) 3.4 (2016).

[35] Obeidat, Nawar H., and Carla Purdy. "Modeling a Smart School Building System Using UML and TLA+." In 2020 3rd International Conference on Information and Computer Technologies (ICICT), pp. 131-136. IEEE, 2020.

[36] Bhardwaj, Pranay, and Carla Purdy. "System Design Methodologies for Safety and Security of Future Wireless Technologies in Aviation." In 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 235-238. IEEE, 2019.

[37] Prince, B. "Air traffic control systems vulnerabilities could make for unfriendly skies [black hat]. Security Week (2012)."

[38] Alloy. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/Alloy_(specification_language)

[39] Microsoft VCC. Github. [Online], Available (05/13/2021):

https://github.com/microsoft/vcc

[40] SCADA. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/SCADA

[41] ISO 50001. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/ISO_50001

[42] Agha, Gul A. Actors: A model of concurrent computation in distributed systems. Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1985.

[43] Pierre, Laurence, and Thomas Kropf, eds. Correct Hardware Design and Verification Methods: 10th IFIP WG10. 5 Advanced Research Working Conference, CHARME'99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings. Springer, 2003.

[44] IEEE 802.15.4. Wikipedia. [Online], Available (05/13/2021) :

https://en.wikipedia.org/wiki/IEEE_802.15.4

[45] Nawar Obeidat, and Carla Purdy. " Using Formal Methods to Model a Smart School System via TLA+ and its TLC Model Checker for Validation " 2021 ASTESJ Advances in Science, Technology and Engineering Systems Journal. ASTESJ, 2021.

[46] Pranay Bhardwaj, Nawar Obeidat, and Carla Purdy. "A novel way to design ADS-B using UML and TLA+ with security as a focus." 2020 ASTESJ Advances in Science, Technology and Engineering Systems Journal. ASTESJ, 2020.

[47] Smart System. Wikipedia. [Online], Available (05/13/2021):

https://en.wikipedia.org/wiki/Smart_system

[48] Friedenthal, Sanford, Alan Moore, and Rick Steiner. A practical guide to SysML: the systems modeling language. Morgan Kaufmann, 2014.

[49] Cao, I. "Guidance material on comparison of surveillance technologies (GMST)." International Civil Aviation Organization (2007).

[50] Primary Surveillance Radar (PSR). [Online], Available (05/13/2021):

https://www.skybrary.aero/index.php/Primary_Surveillance_Radar_(PSR)

[51] Schejbal, Vladimir, Pavel Bezousek, Jan Pidanic, and Milan Chyba. "Secondary surveillance radar antenna [Antenna Designer's Notebook]." IEEE Antennas and Propagation Magazine 55, no. 2 (2013): 164-170.

[52] Galati, G. "Sistema di sorveglianza ad alta precisione mediante multilaterazione di segnali SSR." Italian patent RM (2004): A000249.

[53] Galati, Gaspare, Mauro Leonardi, Patrizio De Marco, Luca Menè, Pierfrancesco Magarò, and Maurizio Gasbarra. "New time of arrival estimation method for multilateration target location." Proc. JISSA (2005): 1-11.

[54] EUROCAE, "Minimum Operational Performance Specification for Mode S Multilateration System for use in A-SMGCS", ED-117, April 2003.

[55] New Air Traffic Surveillance Technology. [Online], Available (05/13/2021):

http://www.boeing.com/commercial/aeromagazine/articles/qtr_02_10/pdfs/AERO_Q2-10_article02.pdf

[56] Fact Sheet – Automatic Dependent Surveillance-Broadcast (ADS-B). [Online], Available (05/13/2021):

https://www.faa.gov/news/fact_sheets/news_story.cfm?newsid=7131

[57] Miller, Sam, Helen Susannah Moat, and Tobias Preis. "Using aircraft location data to estimate current economic activity." Scientific reports 10, no. 1 (2020): 1-7.

[58] FEDERAL AVIATION ADMINISTRATION, May 2010, Federal regulation 14 CFR 91.225S, [Online], Available (05/13/2021):

http://www.ecfr.gov/cgi-bin/text-idx?node=14:2.0.1.3.10#_top

[59] FEDERAL AVIATION ADMINISTRATION, May 2010, Federal regulation 14 CFR 91.227, [Online], Available (05/13/2021):

http://www.ecfr.gov/cgi-bin/text-idx?node=14:2.0.1.3.10#se14.2.91_1225

[60] Manesh, Mohsen Riahi, and Naima Kaabouch. "Analysis of vulnerabilities, attacks, countermeasures and overall risk of the Automatic Dependent Surveillance-Broadcast (ADS-B) system." International Journal of Critical Infrastructure Protection 19 (2017): 16-31.

[61] Strohmeier, Martin, Vincent Lenders, and Ivan Martinovic. "On the security of the automatic dependent surveillance-broadcast protocol." IEEE Communications Surveys & Tutorials 17, no. 2 (2014): 1066-1087.

[62] Schäfer, Matthias, Vincent Lenders, and Ivan Martinovic. "Experimental analysis of attacks on next generation air traffic communication." In International Conference on Applied Cryptography and Network Security, pp. 253-271. Springer, Berlin, Heidelberg, 2013.

[63] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, S. Ravi, “Security as a new dimension in embedded system design,” in Proceedings - Design Automation Conference, 2004, doi:10.1145/996566.996771.

[64] K. Sampigethaya, R. Poovendran, S. Shetty, T. Davis, C. Royalty, "Future E-enabled aircraft communications and security: The next 20 years and beyond," Proceedings of the IEEE, 2011, doi:10.1109/JPROC.2011.2162209.

[65] ICAO International Communications Group, April 2006, Introduction to ACARS messaging services as implemented via Iridium satellite link, [Online], Available (05/13/2021): http://www.icao.int/safety/acp/inactive%20working%20groups%20library/acp-wg-m-iridium-7/ird-swg07-wp08%20-%20acars%20app%20note.pdf

[66] C. W. Lin, A. G. Vincentelli, Security-aware design for cyber-physical systems, Springer, 2017, doi: 10.1007/978-3-319-51328-7.

[67] G. Kalakota, Hierarchical partition based design approach for security of CAN bus based automobile embedded system, Electronic Thesis or Dissertation, University of Cincinnati, 2018.

[68] Object Management Group, 2005, Introduction to OMG's Unified Modeling Language (UML®), [Online], Available (05/13/2021):

http://www.uml.org/what-is-uml.htm

[69] J. Vidal, F. De Lamotte, G. Gogniat, P. Soulard, J.P. Diguet, "A co-design approach for embedded system modeling and code generation with UML and MARTE," in Proceedings - Design, Automation and Test in Europe, DATE, 2009, doi:10.1109/date.2009.5090662.

[70] J. Jürjens, P. Shabalin, "Tools for secure systems development with UML," in International Journal on Software Tools for Technology Transfer, 2007, doi:10.1007/s10009-007-0048-8.

[71] Fall, Moustapha, Chris Chuvalas, Nolan Warning, Max Rabiee, and Carla Purdy. "Enhancing SCADA System Security." In 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 830-833. IEEE, 2020.

[72] Office of the manager national communications system. October 2004. "Supervisory Control and Data Acquisition (SCADA) Systems" (pdf). National communications system. Archived from the original (pdf) on 14 July 2015.

[73] Nazir, Sajid, Shushma Patel, and Dilip Patel. "Assessing and augmenting SCADA cyber security: A survey of techniques." Computers & Security 70 (2017): 436-454.

[74] Byres, Eric J., Matthew Franz, and Darrin Miller. "The use of attack trees in assessing vulnerabilities in SCADA systems." In Proceedings of the international infrastructure survivability workshop, pp. 3-10. Citeseer, 2004.

[75] Maynard, Peter, Kieran McLaughlin, and Berthold Haberler. "Towards understanding man-in-the-middle attacks on iec 60870-5-104 scada networks." In 2nd International Symposium for ICS & SCADA Cyber Security Research 2014 (ICS-CSR 2014) 2, pp. 30-42. 2014.

[76] Lu, Zhuo, Xiang Lu, Wenye Wang, and Cliff Wang. "Review and evaluation of security threats on the communication networks in the smart grid." In 2010-Milcom 2010 Military Communications Conference, pp. 1830-1835. IEEE, 2010.

[77] Wei, Mingkui, and Wenye Wang. "Toward distributed intelligent: A case study of peer to peer communication in smart grid." In 2013 IEEE Global Communications Conference (GLOBECOM), pp. 2210-2216. IEEE, 2013.

[78] Teodorowicz, Thomas. "Comparison of SCADA protocols and implementation of IEC 104 and MQTT in MOSAIK." Bachelorarbeit. Münster: University of Münster 15 (2017).

[79] Johnson, Chris. "Securing the participation of safety-critical SCADA systems in the industrial internet of things." (2016).

[80] Mohagheghi, Salman, J. Stoupis, and Z. Wang. "Communication protocols and networks for power systems-current status and future trends." In 2009 IEEE/PES Power Systems Conference and Exposition, pp. 1-9. IEEE, 2009.

[81] MODBUS Protocol is a messaging structure developed by Modicon in 1979, used to establish master-slave/client-server communication between intelligent devices more info. [Online], Available (05/13/2021): www.modbus.org

[82] Lee, Newton. "Counterterrorism and Cybersecurity: Total Information Awareness. [Online] Springer International, April 8, 2015."

[83] Asavoae, Mihail, Imane Haur, Mathieu Jan, Belgacem Ben Hedia, and Martin Schoeberl. "Towards Formal Co-validation of Hardware and Software Timing Models of CPSs." In Cyber Physical Systems. Model-Based Design, pp. 203-227. Springer, Cham, 2019.