Formal Specification and Verification of System of Systems Using UPPAAL: A Case Study of a Defensive Missile Systems

Joon-Ha Jang¹ and Jin-Young Choi²

¹Department of Computer Science and Engineering, Korea University, Seoul Korea
²Graduate School of Information Security, Korea University, Seoul Korea
Email: {jhjang, Choi}@formal.korea.ac.kr

Abstract—In this paper, we specify and verify System of Systems (SoS) using Formal Methods. As software evolved, its size and weight increased. This makes the embedded system more complex. This trend led to the concept of SoS. SoS is an integrated system, which has systems as components. This has a very large system complexity. At this time, the total system complexity is larger than each sum. Due to the complexity of the system it is very difficult to evaluate and develop the SoS appropriately. So we use system engineering. The advantages of system engineering are as follows. First, it is possible to clarify the requirements of the independent systems and the SoS requirements. Second, it is not only possible to demonstrate and evaluate independently the requirements for each system, but also to demonstrate and evaluate requirements in a SoS. At the same time, system engineering has the following challenges. First, it is difficult to simulate and model from the perspective of SoS. Second, it is difficult to verify the time constraints of SoS. In this work we try to solve this challenge using formal methods. We use model checking among Formal Methods. And we model and verify the system using a tool called UPPAAL. The reason for using UPPAAL is because the system definition of it is suitable for SoS. And because UPPAAL is suitable for verifying real-time systems. In this study, we do modelling and verify the behavior of Defensive Missile System (DMS). Through this we objectively verified the time constraints of SoS using formal methods. And we verified interoperability of DMS. And we have verified that the procedures of the DMS are logically error free and the time constraints that the DMS has. The DMS model, an implementation of our study, is reusable and extensible.

Index Terms—Systems of systems, system engineering, model checking, formal verification of missile defense system, UPPAAL, formal methods

I. INTRODUCTION

With advances in software technology, the proportion and importance of embedded SW in major industrial sectors is increasing every year. As of 2013, SW accounts for 53.7% of households, 52.7% of communications, 52.4% of automobiles, and 51.5% of industrial automation. In addition, SW functions in the automotive and aircraft sectors account for more than 90% of the major industries. [1] In addition to this, the scale also increased. This tendency is especially noticeable in embedded systems.

Due to the increase in the proportion of SW and the scale of SW, the system is becoming more complex. The system consists of many heterogeneous elements and elements interact with each other. This tendency has led to the formation of a complex system in which several systems constitute one system. This complex system is called SoS. System failures and errors cause massive human casualties and property damage.

So it is important to prevent and eliminate these system errors. System engineering is effective in preventing and eliminating system failures. However, there is always a possibility of system error or failure even if this method is used. To reduce this possibility, we must do our best to prevent and eliminate system failures.

In this paper, we model the DMS based on US MIM-104A Patriot. A DMS is a complex of embedded systems and human operators. It consists of multiple systems, which are radar (RS), controller (ECS) and missile launchers (LS). Each system performs its own process independently, and these processes are integrated into a single process. And this system has its own timing issues and resource constraints. And the DMS is in final decision stage in air defense system.

In the DMS, each system interacts through events. In this sense, this system is event driven architecture (EDA). Here, EDA can be defined as follows. EDA, also known as message-driven architectures, is a SW architecture pattern promoting the production, detection, consumption of, and reaction to events. An event can be defined as "a significant change in state". [2]

In this study, we try to use formal methods as a cornerstone to ensure that the model to be developed and verified has such predictability and accuracy. We applied model checking to the DMS. Our goal is to more objectively evaluate DMS behavior by modeling the behavior of the system and determining the time constraint of the system. Because the missile defense system is a SoS consisting of an embedded system and a human operator, we verify the time constraints by including people in the loop when modeling. In past studies on modeling and verifying the behavior of these SoS have been preceded, but these studies have limitations in verifying the time constraints of system behavior. [3]
Unlike previous studies, this study can more accurately and objectively verify the time constraints of system behavior by using formal methods. The results of this case study are SoS interoperability verification and development of a reusable system behavior verification model. At this time, interoperability is defined as an attribute indicating whether the individual systems constituting the complex system satisfy the system requirements and meet the requirements of the integrated system. [4] The first paragraph, we talk about SoS and the strengths and challenges of SoS engineering. Second, we describe the UPPAAL used as a modeling tool in this study, and explain why we chose it. Then, we describe the modeling target system, validation process, formal specification, and formal verification that have been conducted to verify system interoperability and time constraints. Next, we describe the contribution of our research in terms of interoperability, model reusability. Finally, the conclusions are discussed in terms of the implications of this study and future research.

II. CHARACTERISTICS OF THE SYSTEM OF SYSTEMS ENGINEERING

This section explains why the System of Systems Engineering is needed and what challenges are there.

First, we define System of Systems in this paper like below. System of Systems is an integrated system, which has systems as components. But it does not just mean a complex of systems. System can be called SoS when it has the following two attributes. The first component needs to be operationally independent and the second is to be managerially independent.

To explain, the independence of the operational aspect means that the function can be performed according to the original purpose even if the system component is separated from the complex system. It means that the process of the component system works independently and meets its own system requirements.

In addition, the independence of the managerial aspect means that the management of the component system is managed not only for the overall system purpose but also for the purpose of the system itself. That is, they are managed as one organism in the whole system, but each constituent system is independent. [5]

In order to develop and evaluate the SoS, it is important and essential to approach from a systems engineering perspective. The reason for this is as follows.

First, by engineering approach to designing the structure of a SoS, it is possible to clarify the requirements of the independent systems and to define the system requirements. Second, it is not only possible to demonstrate and evaluate independently the requirements for each system, but also to demonstrate and evaluate requirements in a complex system, [6] and [7].

The application examples of the system engineering procedure for the complex system are as follows. 1) A case applied to estimate the ability to track jointly from various units at the time of US-Air Force's Joint tracking management system development. 2) A case used to predict the engagement capability and to evaluate the actual development engagement capability for launching the missile defense system to defend the ballistic missile at sea through Aegis. 3) There is a case based on the results of system engineering analysis of the complex system in advance in order to decide whether or not to invest in the research project. When developing complex systems, if system engineering is not applied, system failures will eventually lead to system requirements revision and expensive redevelopment procedures.

The challenges that arise when applying system engineering to the development of SoS systems, are as follows. First, SoS require shared architecture and vision across different systems. A SoS is a complex system in which a plurality of systems operate as one, and the technical complexity is very high. The complexity of a system is greater than the sum of the complexities of each component system. Because the system interface must be compatible with each system, and the entire system is operated through the distribution of functions. In other words, for the development of complex systems, it is necessary to understand not only the complex interactions between the systems but also to share the vision of how the system should function.

Secondly, there are difficulties in that a large number of programs run simultaneously in many systems. As mentioned above, since the SoS have high complexity and mutual independence. So it is necessary to establish a concept and generalize the system requirements. The clear requirements reduce overall system complexity and improves mutual independence and system integration. This process was applied to Aegis's development process in the past.

The third challenge is that System component systems are not all in the same development phase. So, it is necessary to analyze and integrate requirements of each system. In the process of matching the tempo between the legacy system and the new system, it is important to explore and define the core concept in the SoS. It is also important to ensure backward compatibility. In this paper we call this compatibility is interoperability. To ensure interoperability, it is necessary to accurately model and verify the existing system.

Finally, it is a difficulty in testing and simulation. The process of testing and simulating a complex is logically much more complex. To solve this logical complexity, we use formal methods as a system modeling and verification method. The use of formal techniques can provide appropriate reliability for modeling at the individual
system level and integration level, and can further improve the objectivity of the verification results.

III. UPPAAL

A. UPPAAL’s Character and Strength Points

We used UPPAAL for formal verification. This tool provides user with an integrated tool environment that enables modeling, verification, and simulation. UPPAAL allows the user to model the behavior of systems and systems by switching between specified states and states. In this tool, the system is specified through timed automata. It also use the TCTL [8], [9] logic and the corresponding query statement to verify that the model has the appropriate attributes for the specified model. UPPAAL consists of a verifier and a stimulator. And this has the advantage that it can create counter examples when the attribute is unsatisfied. This character allows you to track logical errors during property validation and improve the model more efficiently. In addition, the UPPAAL simulator has the advantage that the relationship of each model, the transition of the state according to time, and the change of property can be confirmed.

The reason for choosing UPPAAL as a tool for modeling and verification of a DMS is as follows.

First, the definition of the system in UPPAAL is suitable for SoS.

Second, due to the non-determinism of the real-time properties of UPPAAL. This property allows an acceptable degree of abstraction when modeling and verifying system behavior. A detailed explanation of the above two reasons is as follows.

In UPPAAL A system is modeled as a network of Templates, Which consisted of Timed Automata (TA) [8] in parallel. This definition is well suited for modeling a System of Systems. The detailed syntax used in UPPAAL is as follows. [3]

Each TA is a finite state machine equipped with clock variables that acts as the semantics basis of Stateflow.

Let N be the set of natural numbers, X be a finite set of clocks, and \( \Phi(X) \) the set of formulas obtained as conjunctions of atomic constraints of the form \( x \sim n \), where \( x \in X \), \( n \in N \), and \( \sim \in \{<,\leq,=,>,\geq\} \). The elements of \( \Phi(X) \) are called clock constraints over X. A TA over clocks X and action Act is a tuple \( \{L, I, E, I, L\} \) where L is a set of locations, I is the initial location, E \( \subseteq L \times \Phi(X) \times Act \times P(X) \times L \) is the set of edges, I : L \( \rightarrow \Phi(x) \) assigns invariants to locations.[3]

In UPPAAL syntax Location is a set of states that has a time variable called clock. And this feature is available in TA. TA permits two-way synchronization on complementary input and output actions (marked as a? and a! respectively). UPPAAL’s two-way synchronization is suitable for inter-system interaction modeling.

A network of TA, NTA = TA1 . . . TAa over X and Act, is defined as the parallel composition of n TAs over X and Act. Semantically, the network of TAs (NTAs) describes the transition of time using time variables between components and synchronization through events, respectively.

IV. MODELING DMS BEHAVIOR USING UPPAAL

A. Defensive Missile System

Our modeling target is a DMS. From the air-defense system point of view, on the air side, this system is the final step in determining whether we can defend against an enemy missile attack. So this system is affected by time constraints. Therefore, the upper bound of the missile defense system’s action time is important in terms of air defense. So we want to objectively verify the upper bound of the minimum execution time of defense procedure by using UPPAAL. In this study, we modeled DMS behavior based on the MIM-104A Patriot, which is a typical example of DMS. And the target system is the SoS. So we used UPPAL and the system consists of Timed Automatas (TA). Because it consists of independent systems such as Engagement Control System (ECS), Radar Set (RS) and Launcher Station (LS), and each system operates independently and operates as a single system. When modeling, we implemented each component system as one model. Each model operates in parallel and eventually becomes a single system. In the modeling and specification, each system is an actor. Each actor in the DMS performs its own procedures independently and is implemented as a single model. One model can be simulated and verified independently, and can be simulated and verified in an integrated manner. We further implement a system monitor model (‘Proc’) to model and verify DMS as SoS.

In our DMS model, we limit actors to ECS, RS, LS, and System monitor when modeling a defensive missile system. And the number of launchers is limited to two, and the real-time error checking function of the ECS system is limited to two. However, we can change the number of error checking functions and the number of Missile Launchers to suit our purposes using the parameters and instances of UPPAAL model.

Our UPPAAL model targets the behavior of the DMS. And this model is implemented with three purposes. First, a model for verifying and simulating DMS from SoS perspective. Second, a model to verify that the defense procedures of the DMS are logically error free. Third, a model for verifying the time constraints of the system. For this modeling purpose, we use time variables such as clock, invariant, and channel. In addition, we use a non-deterministic approach that considers all possible cases in the modeling process. This allowed us to limit the degree of abstraction of modeling to acceptable limits and perform more objective verification.

B. Formal Specification of DMS Behavior

Fig. 1 shows the model we developed using UPPAAL. First, the Proc model on the left is a model for monitoring
the DMS system with SoS concept. On the right side, models composed of ECS, LS (0), LS (1), LS_que, and ErrorCheck systems are behavior models of the DMS system. The five models on the right side of Fig. 1 are the component systems that make up the DMS. The models work in parallel and interact with each other. It can be monitored by the integrated system process through the left monitor model.

As mentioned above, the ‘Proc’ model (Fig. 2) is a modeling of the DMS system process from the SoS perspective and is used for formal verification of the time constraints of the DMS system behavior. The ‘Proc’ model specifies and verifies the interactions between independent systems using UPPAAL’s time variable clock and a synchronization variable called action. In other words, we implement the ‘Proc’ model to monitor the system behavior of the DMS from a SoS perspective. The automata that exist in the ‘Proc’ model are called locations in the UPPAAL tool. We use this to monitor the processes of the systems that make up the DMS. And we use TCTL queries as a means of monitoring.

The ECS model (Fig. 3) represents the system behavior of ECS and RS in the MDS system. In this model, we implemented ECS's ability to control RS, LS equipment, and automatically check for equipment failures. We also use the local variable x in the ECS model as well as the global time variable now. Using x, we can distinguish between the total defense operation time and the ECS and RS procedure execution time separately in the DMS.

The error checking model (Fig. 4) represents a system for recovering random errors occurring in a DMS system. When modeling this system, we first implemented the interaction when the error occurred with bidirectional synchronization through the action variable. And we randomly implemented the occurrence of errors using non-determinism.

LS model (Fig. 5) shows that LS (0) and LS (1) are the same. In the DMS system, since LS operates in the FIFO (First in First Out) manner, we implemented LS (0) and LS (1) as instances with a common frame. The advantage of this implementation is that it can be used by reducing or increasing the number of LS's depending on the purpose.

The LS_que model (Fig. 6) monitors each LS operating independently in the system and interacting with each other. In this case, we implemented LS to be generated.
nondeterministically using parameters and functions. The LS (0), LS (1) and LS_que models use the local clock y and the global clock now. LS (0) and LS (1) have independent clock y, and each y is integrated through LS_que. Through y of LS_que, we can analyze the system execution time of LS independently. Also, through now, we can see whether the LS execution time is appropriate for the whole DMS system execution time.

In summary, we have implemented the DMS behavior model as ECS, LS_que, LS (0), LS (1), and ErrorCheck, which operate in parallel, and monitor model('Proc') to analyze and verify the implemented DMS behavior from the SoS point of view. To analyze the time constraints of the DMS, we used the global time variable Now and the local time variables x, y.

C. Simulation of Defensive Missile System Behavior Model

We constantly simulated to develop a model that correctly reflects the behavior of the DMS. Using a simulator, we tested whether the system models that make up the DMS work independently and that there are no deadlocks. In addition, we have implemented the DMS behavior model by dynamically validating that each system works well as single system in this model. We used the red line in Fig. 7 to analyze the interaction between each model. And through the intermediate <Global variables> menu, we analyzed the changes of time variables and variables according to the simulation process. And through the <Enabled Transitions> menu, we simulated whether or not each system was deadlocked. Table I below Fig. 7 shows the hardware environment and UPPAAL tool settings that we simulated.

---

TABLE I: SIMULATION ENVIRONMENT

| 1. CPU : Intel (R) Core (TM) i5-3470 3.20 GHz |
| 2. RAM : 12.0GB |
| 3. OS : window 10 x86 |
| 4. Diagnostic Menu : shortest path, fastest path |
| 5. Search order : random depth search |

D. Formal Verification of DMS Behavior

Table II shows the TCTL [8] we used to verify the correctness [10] of the model and the upper bound of the system time limit. And TCTL is translated into natural language.

---

TABLE II: PROPERTIES IN TCTL

1. A[] ECS.HMComplete imply Proc.Proc_Done
2. A[] ECS.StateGreen imply LS_que.Cm_h
3. A[] LS_que.Com_hot imply Proc.LS_done
4. A[] Proc.Proc_Done imply now<=NUMBER
5. A[] ECS.HMComplete imply ECS.x<= NUMBER

1. The monitor model ('Proc') correctly monitors the ECS model.
2. The ECS model correctly controls the LS_que models.
3. The LS_que model correctly controls LS (0) and LS (1).
4. Minimum procedure execution time is less than or equal to NUMBER.
5. Minimum ECS procedure execution time is less than or equal to NUMBER.
6. Deadlock verification of monitor model ('Proc')
"A [] ECS.HMComplete imply Proc.Proc_Done" We use this TCTL statement to verify that the procedure of the ECS model works horizontally in the monitor model. In short, this statement verifies that the Proc model correctly monitors the ECS model.

"A [] ECS.StateGreen imply LS_que.Cm_ht" This statement verifies the correctness of the interaction between the ECS model and the LS_que model. In other words, this verifies that the ECS model correctly controls the LS_que model. "A [] LS_que.Cm_ht imply Proc.LS_done" This statement verifies the correctness of the interaction between the LS model and the LS_que model. In statement 4, 5, the fourth statement verifies the minimum procedure execution time of the DMS behavior. The fifth query verifies the minimum procedure execution time of ECS. And the final query is "E <> Proc.Proc_Done" This is a statement to verify that the monitoring model does not have deadlocks.

Finally, our verification results and system model can be reused for evaluation of similar systems because they analyze and verify the behavior of existing SoS systems without additional expansion and implementation. As mentioned above (V), this model can be extended to air defense system behavior models through additional implementations.

E. Contribution of This Study

The contribution of our research is as follows. First, we have done accurate modeling and simulation and have verified SoS time constraints. These are the challenges of SOS engineering [6] and [7].

Second, our model verifies interoperability of DMS. And verify the reliability of the defense procedures performed by the DMS.

Fourth, we improved logical objectivity by system engineering using formal methods. Formal methods improve the logical objectivity of our DMS modeling and simulation because it limits the degree of abstraction in the modeling process.

Fifth, our model is scalable and reusable. Specifically, our model is scalable by parameter or instance manipulation. And our model can be used to improve or change defensive procedures and add new components to the DMS.

Fifth, our model is scalable and reusable. Specifically, our model is scalable by parameter or instance manipulation. And our model can be used to improve or change defensive procedures and add new components to the DMS.

V. CONCLUSION AND FUTURE WORKS

In this paper, we apply system engineering to SoS using formal specification and formal verification. As a case study of this study, we model and verify DMS at operator level. We use model-checking in formal methods to verify the time constraints of SoS and the soundness of the whole process. The results are used to verify interoperability of missile defense systems, which can be extended to Air-Defense systems.

The models and validation results developed in this study are more objective than previous studies. [10]-[14] This is because we modeled the target system with the SoS concept and verified the time constraints. This model can also be modified to improve current defensive missile system procedures and can be used to improve the model when adding a new system to a defensive missile system.

However, our research modeled the behavior of SoS system, it can be applied only to system process evaluation, attack response time improvement, and system extension evaluation. We performed the time constraint in 100 clock units. More detailed verification is possible if the clock unit is increased. However, we performed 100 units due to the hardware environment.
Future research based on the results of this study is as follows. First, there is research to model and verify SoS when developing a new missile defense system. On the other hand, there is a study that automatically generates code according to the model. Third, there are studies to model and verify embedded system or controller that constitute SoS.

REFERENCES


Joon Ha Jang was born in Seoul, Korea on April 9, 1990. He received BS in Electronic Engineering from Korea Air Force Academy, in 2014. He started MS program in Computer Science, Korea University, 2016. His research interests are formal methods and System Engineering, Software Engineering, Complex System, Embedded Software Engineering.

Jin-Young Choi was born in 1959 in Seoul, Korea. He received BS degree in Computer Science from Seoul National University in 1892. He received MS in Computer Science from Drexel University in 1986, and PhD in computer science from the University of Pennsylvania in 1993. His research interests are real-time computing, formal methods, and process algebras, SDN, and software engineering. He is a Professor in Graduate School of Information Security in Korea University.

©2017 Journal of Communications