Model checking For An Architecture Description: Where Does It Fit Into A Safety Case?

Omar Jaradat
PhD student in Safety Argumentation
Högskoleplan 1, 721 23 Västerås
Mälardalen University - Sweden
Telephone number +46-21 101 369
omar.jaradat@mdh.se

ABSTRACT
Safety critical system is responsible for ensuring safety to its users or at least helps to ensure the safety. However, due to the fact says Absolute safety is not an attainable goal. Thus, we must argue over the reasons of “why” the system is safe enough to be deployed, where safe enough means that despite of the available risks but the system is still considered safe. In this context, safety cases are used to structure the safety arguments. Safety cases are defined as a structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment. Evidences are based on observing, analysing, testing, simulating or estimating the properties of a system.

Since, model checking is a technique used to simulate and test some system properties, thus, the findings of this technique can be used to support claims in the safety case. In this paper, we are adopting a safety critical system from the real-world. The system was used by a case study for a verification technique by which we use the model checking approach to check some properties of the system, where the AADL (Architecture Analysis & Design Language) was used to design the architecture model of the system and its properties. Then we build a safety argument for the tackled system to show how the results obtained by the verification technique (i.e. model checking) can support claims in the safety argument.

Keywords
Safety Argumentation, Evidence, Claim, GSN, Model checking.

1. INTRODUCTION
Failures in safety critical systems may cause dire consequences that may harm humans or impact the environment. This makes these systems different than non-critical systems in terms of the greater consideration before, along and after development. Moreover, safety critical systems must be certified that they are safe before they can be put in-use. However, obtaining the certification is not an easy procedure and it demands a lot of work before getting the stamp of approval. Furthermore, absolute safety is out of reach, thus, we must instead argue why the system is safe enough to be deployed and to be certified accordingly.

The argumentation must be performed in a structured manner where the claims (objectives) are linked with the proper evidences. The evidences are based on observing, analysing, testing, simulating or estimating the properties of a system that provide the fundamental information from which safety can be inferred. The combination of these claims and evidences are called safety case which is a document encompasses the safety arguments and evidences to demonstrate that the system is safe enough to operate.

In safety critical system development, architecture design is the most critical phase, where the architecture model represents the first design decisions. These design decisions define the functional properties. Moreover, the architecture model is very crucial since the rest of development process will highly depend on it. In addition, it is used as a blueprint among system’s stakeholders. Consequently, evaluating the final design decision of the architecture can be vital, because any incorrect structural, functional, or non-functional properties will generate number of problems in the upcoming phases. [6]

However, before thinking about the evaluation itself, we have to think about what is to be evaluated and how it can be represented. In this context, Architecture Description Languages (ADLs) have been developed to describe system architectures, and principal architecture design decisions. They are used in context of designing software and/or hardware architectures. An ADL can comprise and represent the system level details of components, components’ interactions and configurations.

When it comes to verification, model checking can play a key role in terms of verifying some properties of the design architecture. Generally, model checking is an automatic formal verification technique, it checks if there is an existence of deadlocks, wrong interactions or inconsistency among the model states that can cause model crashes, or harm the correctness, completeness or concurrency properties of the model. [7]

There are many advantages that can be obtained by using a model check technique, the most important advantage is that model check can detect and tell why a property is not satisfied. On the other hand, and in terms of the safety argumentation, every claim must be supported by a sufficient evidence to strengthen this claim. Thus, verifying the design architecture by model checking can lead to some positive findings that lead to an acceptable level of confidence that the architecture model is consistent and complete. Subsequently, the consistency and completeness can be exploited as evidences to support safety claims in the safety case.

The work in this paper focuses on how the results obtained by the model checking of the system architectural specifications can contribute in the safety argumentation in terms of providing supporting evidences for safety claims.

The rest of the paper is organized as follows. In section 2 we present the motivation of the paper and the essential background. Section 3, describes the formal verification technique. In section 4, we present the case study. Section 5, demonstrates the safety
argumentation of the system architecture. Section 6, presents the work’s summary and some discussion about the safety argument. Finally, the paper ends by drawing an over all conclusion in section 7.

2. MOTIVATION AND BACKGROUND

This paper represents an extension to a case study that has been created in order to validate a formal verification technique [8][7]. The case study tackled a fuel level estimation system as a safety critical system from industry. The work in the case study concerned the design architecture of the system which has been analyzed and designed by the Architecture Analysis and Design Language (AADL) [9]. Then, the architecture model has been formally verified by UPPAAL [12] as a model checker. The main interest of the current work in the paper is to investigate how the results (outputs) from the model checking can be exploited as valid evidences to support the safety goals in the safety argumentation presented in section 5.

In this section we present the motivation and the background information on which we base our work. In particular, section 2.1 describes the critical systems and provides brief information about the certification challenges. Section 2.2, introduces the GSN (Goal Structuring Notation) as a means to document safety case.

2.1 Critical System

A safety critical system has plenty of definitions. In general, safety critical system is a system responsible to safe and secures its users, or at least help to ensure safety and security. Any disorder or defect of this system can dramatically cause a loss of life, severe damages, or damage of the environment. Fel! Hittar inte referenskälla. From the definition, it is clear that failures in these systems may cause dire consequences. Therefore, the significance behind the need of ensuring safety is rational, and this what makes these systems vary from non-critical systems in terms of the greater consideration before, along, and after development. There are many examples of safety critical systems in the real-life, such as, airplane’s autopilot system, medical devices, nuclear reactor systems, etc.

From quality perspectives, the main challenge of building safety critical systems is to deliver a product that can assure both safety and security. However, the increasing demand on these systems has emerged new challenges. Judging on whether the system is safe enough to be used or not is one of these challenges. The latter statement is clearly shown in some applications, where safety critical systems must be certified that they are safe before they can be put in-use.

Obtaining the certification is not an easy procedure and it demands a lot of work before the system can be stamped. What adds more challenge is the fact says: absolute safety is not attainable. Therefore, we must show “why” the system is safe enough to be deployed, where safe enough means that despite of the available risks but the system is still considered safe. In other words, we must show the reason of why the risk associated to hazard is acceptable. This emerges the need for an artifact or way by which we can demonstrate the answer to the previous “why” question, this is typically what we call Safety Case. More clearly, safety case is a document provides “A structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment” [3]. It is considered as a major deliverable to certification assessment (major deliverable to get the stamp). Therefore, being correct and understandable is crucial and very important to be taken in the consideration while developing this deliverable. Otherwise, if it is not correct, then the desired safety is not assured; and if it is not understandable then there can be little confidence in the claimed safety assurance.

Consequently, the reason (of why the risk associated to hazard is acceptable) is a part of the safety case, which is represented as an argument in which we argue on: why the system is acceptably safe? More specifically, safety argument demonstrates how safety requirements (claims) are fulfilled with the presence of the supporting (evidences), and how the available evidence supports the overall claim of acceptable safety[1], as shown in Figure 1.

Figure 1 represents the role of the safe [4]

2.2 Goal-Structuring Notation (GSN)

Claim without evidence is unfounded while evidence without claim is unexplained [4]. Thus, safety argumentation must be performed in an organized and structured manner where the claims are linked to the proper evidences.

In addition, creating a coherent set of claims where sub-claims properly support parents’ claims, claims that are unambiguous and achievable, and a well structured set towards comprehensibility and maintainability, are considerable work.

Conventionally, safety cases are written through free text but the textual representation can be seen a problematic due to the reason: 1) the writer of the safety case may fail to write well-structured and clear English text which may leads to unstructured argument. 2) The text may contain a plenty of cross-references and this may affect the flow of the main argument and makes a mess. 3) The safety case is a shared document, thus all stakeholders must share the same understanding of the safety argument, but this can be influenced by the changes and maintaining processes that may occur on the argument. [4]

These reasons can lead to unfounded or unexplained arguments. Hence, the need for more developed way, rather than, the textual representation to represent the safety argument attracted the interest of the safety researchers at University of York in the early 1990s when they introduced a graphical argumentation notation called GSN (Goal-Structuring Notation).

In GSN [6], the claims of the argument are documented as goals and items of evidence are documented in solutions. GSN allows the writers of the safety cases to structure their argumentation into flat or hierarchically nested graphs constituted of a set of nodes and a set of edges as shown in figure 2. The nodes are:
The objective of the verification criteria defined in [8] is to ensure consistency and completeness of and between the AADL flows, their refinements and their constraints through the analysis of control-flow reachability, data-flow reachability and concurrency among flows [11].

Consequently, and based on the representation of an AADL specification, the verification technique formally specified the flow among the architecture components. It specified five different relations: 1) Connection Transfer Relation, 2) Connection Property Relation, 3) Component Internal Relation, 4) Direct Component to Component Relation, and 5) Indirect Component to Component Relation.

From these AADL relations three verification sequences are derived:
1) Component Internal Transfer, 2) Direct Component to Component, and 3) Indirect Component to Component.

Upon the verification sequences, the technique defined three architecture-based verification criteria. These verification criteria will specify the requirement for a set of simulations or test cases to be adequate.

Let's assume that “S” is the set of simulations of AADL specification or a set of test cases for an implementation. Using “S” we can express:

1. **Component Internal Coverage**: requires that S covers all Component Internal Transfer Paths.
2. **Direct Component to Component Coverage**: requires that S covers all Direct Component to Component Paths.
3. **Indirect Component to Component Coverage**: requires that S covers all Indirect Component to Component Paths.

Since the technique is mainly based on AADL and UPPAAL, thus, it is important to introduce them in this section.

### 3.1 The AADL

In specialty for model-based analysis and specification of complex real-time embedded systems, the aerospace standard “AS5506” has been released by the Society of Automotive Engineers (SAE) in 2004, under the name of Architecture Analysis & Design Language (AADL). [9]
As a member of the ADLs family, AADL used to model the software and the execution platform (hardware) architectures, taking into account the performance–critical characteristics through an extendable notations, tool framework, and defined semantics. AADL architecture models are described as a component – connector structure, which means that a model of a system describes the components, components’ interfaces and the connectors among these components. Moreover, AADL can describe characteristics of elements by associating them with specific property annotations, by associating the appropriate properties for these elements, such as, timing properties, runtime properties, synchronization properties, etc.

AADL provides the capabilities to describe the mechanisms of exchange and control of data, such as, message passing, event passing, and synchronized access to shared components, thread scheduling protocols, timing requirements, remote procedure calls, etc. Dynamic reconfiguration of the runtime environment can be specified using mode and mode transitions. [9]

This language has a valuable advantage since it can be used even with limited architectural details, and allows incomplete description, for either systems already in use or new systems. Furthermore, the description of the architecture can be expressed both graphically and textually.

However, AADL lacks the implemented semantics that are needed to automate the simulation of its specification. But despite this lack, the problem has been solved by formally specifying semantics of AADL constructs using mappings to timed automata constructs in the UPPAAL language [12] (model checker).

3.2 UPPAAL

UPPAAL [12] is a joint venture toolbox developed for verification of real-time systems, by UPPsala University in Sweden and AALborg University in Denmark. UPPAAL is a model – checker and based on the theory of timed automata. It can be used to model a system as a network of timed automata, and by using the UPPAAL verifier, testers can insert Computation Tree Logic (CTL) queries to check properties of the model. A timed automaton is a finite state machine driven by clock variables that progress synchronously and evaluates to real numbers. The model is extended with bounded discrete variables that are used as any programming language variables in term of the accessibility.

The UPPAAL query language is used to verify the model requirement specifications, where the result of each query could be satisfied or not satisfied. Since UPPAAL uses a simplified version of (CTL), thus, the query language includes the path formulae that trace the paths of the model. In addition to the state formulae that describes individual states, and regardless of the model behavior, state formula can be evaluated to be true whenever it is valid. Path formulae can be classified into three properties reachability, safety and liveness.

The mappings between AADL to UPPAAL are formally defined and they are called Transformation Rules but for the sake of the space and simplicity we are not listing these rules in this paper, however, they can be found in [8].

The Verification Technique steps [8]:

1. Use the transformation rules to transform an AADL specification to an UPPAAL model upon which automated formal verification can be performed.

2. Apply the architecture-based verification criteria to the AADL specification. They define the test selection, i.e., what samples of the specification to evaluate and how they are extracted, and the coverage requirement, i.e., how many samples to evaluate. The samples generated from the criteria are sequences of component-integrations in terms of control – flows and data – flows.

3. Sequences from the previous step are transformed, in this step, to the corresponding automata paths in the UPPAAL model through a structural mapping between them.

4. The outcomes from step 3, i.e. a set of automata paths are required in this forth step to be fully simulated in UPPAAL, by using temporal logics, in order to satisfy the criteria. The verdict from the simulations reveals the consistency and completeness of the AADL specification, where a correction of the specification should be made if it is shown inconsistent or incomplete.

4. FUEL LEVEL ESTIMATION SYSTEM: THE CASE STUDY

As previously discussed, the main goal of this case study is to verify the introduced architecture–based verification technique for AADL specifications presented in the previous section. The validation procedure starts by investigating one – real life – safety critical system to model and analyze the system by using the AADL standard, and then transform the AADL model of the system to an UPPAAL model based on the transformation rules.

The main functionality of the Fuel Level Estimation System is to estimate the fuel level in the vehicle tanks and present this level by the fuel meter located on the dashboard. Additionally, the fuel level estimation system must indicate the driver that the fuel level is low if it is below a predefined level. This requires an accurate measurement by avoiding the rapid changes that may occur when the vehicle is not driven on a flat surface i.e. steep hills and tough terrain.

Why the system is safety critical?

If the system fails to indicate the real fuel level in the tank, more specifically, if the real fuel level is low while the fuel level meter presents a higher level, then the driver will be unable to determine the correct time to refuel. Consequently, there will be a possibility by which the engine will run out of the fuel required to keep it in operation. The sudden shutting down of the engine may force the truck to stop in the middle of the road and this may yield to a hazardous consequences.

The system begins with the fuel level sensor which is located in the fuel tank. This sensor sends analog signals to an ECU (Electronic Control Unit) called the *Estimator* where it converts the voltage value to a digital value using an A/D converter. The *Estimator* has a software layer called *Basic Software* and this layer is divided into two functions (B-SoftwareIn, B-SoftwareOUT) and it is the input and output presentation for the *Presenter*. For fuel level estimations system, the main functionality of the *Basic Software* is to get the converted values from the sensor (i.e. data from the A/D converter) by B-SoftwareIN and then writes it into RTDB (Real-time Database) which represents a shared memory between the *Basic Software* layer and the functions contained in the *Estimator*, as shown in figure 3.

The *Estimator* contains many functions but our considered system deals only with two of these functions, the FuelEstimation and FuelLevelWarning functions. The first function is responsible to
read the fuel level digital value from the RTDB and transforms it to the correspondent volume in percentage. The result will be used together with the maximum capacity of the tank in liters to determine the current fuel volume in percentage value. The function ends its work by writing the percentage value into the RTDB. Afterward, FuelLevelWarning reads the stored percentage value and compares it with a predefined level to decide whether the fuel level is low or not, then it writes the comparison result into the RTDB. The B-SoftwareOut will read the compared value, as well as, the percentage value (i.e. outputted from the FuelEstimation) to finally forward them to another ECU called the Presenter via the CAN bus. Thereafter, the Presenter converts the received values from digital to analog values and updates the fuel level meter, as well as triggers the low fuel level warning lamp if needed. However, in this paper we are considering the functionalities of the Estimator ECU only.

**The model Checking:**

Next, we have to transform the AADL into UPPAAL according to the transformation rules. As shown in figure 3 the Estimator system contains four processes (BasicSoftware, FuelEstimationCalculation, FuelLevelWarningCalculation, and Estimator_Other_Functions). Each process contains one thread (function), except BasicSoftware which contains two threads. The features of these five threads are specified by AADL, which shows the interfaces and the connections of these threads. According to the transformation rules, each thread will be represented as an automaton in UPPAAL in addition to the scheduler automaton. Each thread automaton has a clock (cl) to keep track of the dispatches of the thread. The dispatch edge is synchronized with the scheduler automaton to notify the dispatch based on each thread priority, execution time, and deadline. Figure 4 shows an example of the transformation, where SoftwareIN thread is considered. SoftwareIN priority, execution time, and deadline are 1ms, 1ms, and 8ms respectively. The other threads contained in the Estimator system have similar automaton template but of course they have different interfaces and properties.

**Table 1 shows the extracted sequences**

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SoftwareIN.SetSensorData → FuelEstimation.GetSensorValue_2 → SoftwareOUT.GetFinalValues → SoftwareOUT.FinalEstimationValue</td>
</tr>
<tr>
<td>3</td>
<td>SoftwareIN.SetSensorData → OtherFunctions.ReadWrite, SoftwareOUT.GetFinalValues</td>
</tr>
</tbody>
</table>

**Figure 3** Graphical AADL representation for the Estimator system

Functions in the fuel level estimation system are executed periodically every 10 ms (milliseconds) with a fixed priority scheduling. Thus, beside the data and control flows representation, AADL model represented the timing properties, as well as, the priorities of the threads (functions). Thus, each function has its priority, deadline, and execution time, and dispatch time.

Based on the steps of the verification technique above, more specifically, after applying the verification criteria three sequences have been extracted from the model as shown in the table 1.

**5. THE SAFETY ARGUMENTATION**

The safety argument for the fuel level estimation system should demonstrate the reason why the system is safe. Therefore, it should start with the system hazards and represent how these hazards are adequately mitigated. However, starting from the very high level goal is not the approach under discussion in this paper. Rather, explaining a slice of the safety argumentation can show what the idea is all about.
Figure 5 shows the selected safety argument where the high claim (G1) shows that the AADL represented all the timing and priority properties of the threads. From this claim we created two contexts to 1) explain what we mean with the properties and listed them, and 2) to indicate how we know about the properties and their source. The way that we want to support our claim is called strategy. Thus, the strategy to prove (G1) is to do (S1) which says that we have to argue over the analysis of UPPAAL. The reason behind (S1) is because we know that UPPAAL represented the AADL model after the transformation so if UPPAAL has X then AADL has it as well shown in (G2). However, we know that we transformed the AADL model into UPPAAL according to the transformation rules therefore we should argue why we trust the transformation rules (S2). Since the transformation rules were formally defined, thus we can argue over the logic of creating these rules as represented by (G4). On the other hand, (G3) represents the claim that UPPAAL represents the properties of the threads. This claim can be implemented by (G5) and (G6) where we argue over UPPAAL verifier and the created queries. (G6) is ended by a solution that indicates the assessor (the person who verifies the safety case) to check the certification pack of UPPAAL (which is supposed to give us a good confidence that UPPAAL is working as it is claimed). (G5) claims that we have sufficient UPPAAL queries to analyze the properties of the threads. Since the verification of the sequences (three sequences) and their properties have been made according to the verification coverage, thus we have to check the verification coverage as what (S4) indicates.

Again, the explained argument in figure 5 is only a slice of bigger argument therefore its end is not the actual end of the safety argument for the system.

6. SUMMARY AND CONCLUSION

In this paper, we have presented a case study of a well-defined architecture-based verification technique for AADL specifications. The technique has been innovated by a group of researchers at Mälardalen University. The goal of the technique is to evaluate the integration of components at both the specification-level and the implementation-level. The technique is driven by the architecture verification criteria. These criteria define the test objective in terms of control-flow reachability, data-flow reachability and concurrency among flows of the AADL components to ensure the completeness and consistency of the AADL model.

The system considered by the case study is the fuel level estimation system. This system is already implemented and used by a major vehicle manufacturer in Sweden. It is considered a safety-critical system because if it fails to measure the fuel in the tank correctly, then it will not indicate the actual fuel level. Thus, the fuel may run out while the driver cannot determine the proper time to refuel. The system has been analyzed and modeled using AADL. Then, we have applied the verification technique to check the completeness and consistency for its architecture model. Thereafter, we showed how the verification results can contribute in the safety argument.

The results obtained from the verification technique could give a good level of confidence that the AADL and UPPAAL models have represented the properties of threads correctly. Consequently, and by using these results, we can extend the represented argument to argue over the consistency and completeness by which we can argue over the correctness of the architecture design where safety requirements rely on.
7. ACKNOWLEDGMENTS
I would like to thank Iain Bate and Patrick Graydon for their constructive comments and the fruitful discussions on this paper.

8. REFERENCES


