Internal Report 2011-18 augustus 2011

Universiteit Leiden

Opleiding Informatica

Automated Safety Analysis for

Supporting Design of System Architectures

James Lo

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Automated Safety Analysis for Supporting Design of System Architectures

James Lo

Abstract

Nowadays an automotive contains a huge number of (software) components with its own resource hardware provided by each supplier separately. This distribution of deployment is not optimal in terms of costs, nor in terms of the number of hardware components. Therefore it is desirable to reduce and optimize hardware deployment. However, optimizing for costs solely is not possible due to factors which indirectly cause constraints. For safety-critical systems as automotives, the focus must also be put on the safety aspect. In order to find optimal solutions considering safety, a safety model is required. The safety model in this study will use a technique called fault tree analysis combined with second-order probabilities. With the use of the Monte Carlo method this will provide the safety values used for hardware configuration optimization and result in optimal configurations for hardware costs and safety.

Table of Contents

Introduction

This paper describes my bachelor project done at LIACS faculty of the University of Leiden. I have done my research in *Automated Safety Analysis for Supporting Design of System Architectures*. The title can be divided into two parts. Firstly a method for safety analysis is reviewed and selected. The second part provides the results of the safety analysis to an optimizing tool which will describe the system architecture. Because of the broad scope, this research will study the case of conceptualizing hardware architecture of subcomponents in a car. The main factors for creating a systems design are: costs and safety. This paper will continue the research of *Automated Design of Software Architectures for Embedded Systems using Evolutionary Multiobjective Optimization* [1] with now the safety factor taken into account when deriving optimal configurations for a system. Additionally, some components of a modern day automotive will be analyzed for this factor.

Taking an example from the car industry; a car nowadays has several components which need certain amounts and power of resources. Though they do not all demand the most powerful resource. Cheaper less powerful hardware can be combined, but the question is if this combination is powerful enough. An elaborated view will be discussed in the next section. Another question is if the combination is safe, based on the system failing caused by an unknown factor, to be integrated in the automotive. This topic will be handled in the chapter *Safety Model*. Furthermore the integration of the safety model is described in the chapter *Integration of the safety aspect*. The *Analysis* chapter focuses on the results from an analyzed sample model. Finally, some conclusions are presented in *Conclusion* section and in the chapter *Future work* some ideas for future work will be briefly discussed. In appendix A the code is given.

My motivation for research in this field came from the interest in creating solutions for problems related to designing a software architecture. It gives me satisfaction to contribute to a system that solves such a problem because it is not only apply to one, but several systems. This makes such a tool not only useful but also desirable.

Background and previous work

In a modern day automotive many essential and non-essential components are used, e.g. to make driving safer (essential) or more comfortable (non-essential). For each component the supplier of a component provides both software and computational unit for that particular component. This is not done by only one supplier, but an automotive nowadays will contain more than one hundred Electronic Control Units (ECU). Providing each subsystem with its own resources is called a federated system. While this method might function well, it does not provide an optimal system. Another approach and increasing trend, is moving the systems towards a method where several subsystems share its resources with other subsystems: an integrated system. One can expect that both of the mentioned methods have their own benefits. Though for car manufacturers, and of course many others, cost is an important factor. Cost reduction can be achieved by using an integrated system and by implementing such, grouping components and providing one calculation unit to them.

Using a DECOS architecture, which consists of multiple Distributed Application Subsystems (DAS), an optimal system (with help of an optimization tool) can be developed. The DAS consists of several subsystems which can be grouped by safety-critical as well as non-safety critical subsystems [3]. In the paper *An Integrated Architecture for Future Car Generations* a DAS is described as in Fig. 1.

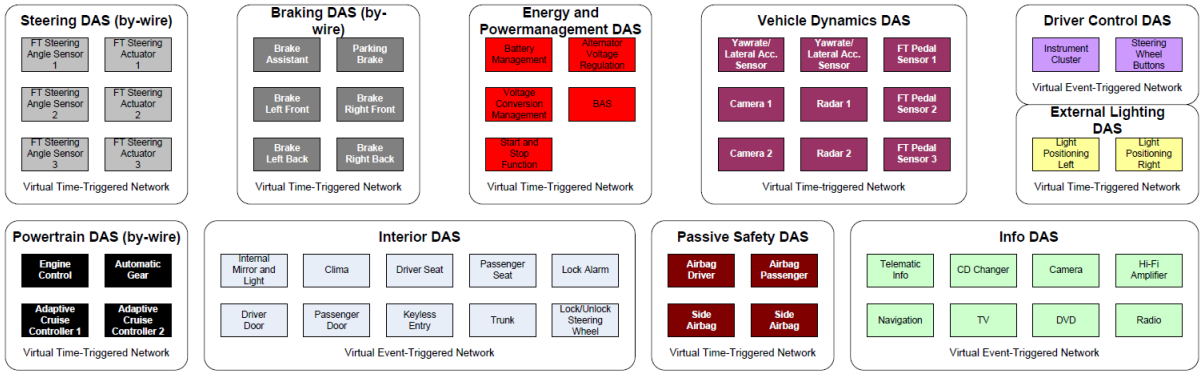


Fig. 1: Distributed Application Subsystems of an Automotive System, *An Integrated Architecture for Future Car Generations, P. Peti, R. Obermaisser, F. Tagliabo, A. Marino and S. Cherchio*

The figure above depicts groupings by close related functionalities. However, there should be mentioned that this DAS is proposed to run on a virtual network and therefore is not a complete reflection on the models in this study. On the contrary, it does describe a system containing grouped components, which do not individually have their own ECU.

Next to a calculation unit also the connection lines can reduce costs. For example the protocol Controller Area Network (CAN) has a few quality categories. Each category has a maximum of bandwidth it can pass through. So the higher the maximum bandwidth, the higher the cost of the connection will be. A video system is an example where a connection line is required to transport a large amount of data per second. Optimization for a CAN will not be discussed in this paper, but can be a point where manufacturers might want to optimize for.

As mentioned this paper is a continued research of a previous study. In the previous research a method is described where some optimal solutions for a(n) (integrated) system are calculated based on two criteria: costs and number of hardware components. These optimal solutions are found by

using an evolutionary algorithm which will calculate every possible configuration of the defined hardware. The optimal solutions are found in a graph containing a pareto front. The input values corresponding to an optimal solution are stored so a certain configuration can be chosen later on.

Safety aspect

In safety-critical systems - which are defined as where a failure in the system will lead to death or injury to people, loss or severe damage to equipment or environmental harm - the safety of the system is important. Safety can be analyzed using fault tree analysis, which was first used in the aviation industry to prove safety. Nowadays it is used in multiple engineering fields in different combinations but maintaining the core aspects.

Fault tree analysis method

The method in this project will be using fault tree analysis based on second-order probabilities. The basic principle of this method can be described in a few steps [5]:

1. Define the event to be investigated by the tree
2. Gain an understanding of the system
3. Construct the tree
4. Collect quantitative data
5. Evaluate the probability of the event chosen in step 1
6. Analyze computer output

When the quality of the software is known, a probability of failure is available to be combined in the fault tree. This probability is not based on a random failure, as software does not fail randomly, but based on a failure under certain circumstances [2]. Its structure is defined as a tree which nodes are the basic logical gates (for example: AND, OR, XOR). In the leafs of the tree we find the inports and in the root or top level of the tree the outport.

The Monte Carlo Method will be used for the values provided to the inputs and received from the output. This method must not be confused with the Monte Carlo algorithm. For each inport a randomly generated probability value, within a given error range, will be available. Then it requires a deterministic algorithm to run, which will be the fault tree. The result will be the value received from the top level. To be able to retrieve valid results, the Monte Carlo method must be run several times. This is because the values used for the inports are generated randomly and therefore will not give a representable output. The Monte Carlo method is generalized as followed:

1. Define a domain of possible inputs.
2. Generate inputs randomly from a probability distribution over the domain.
3. Perform a deterministic computation on the inputs.
4. Aggregate the results.

Here our deterministic computation is the fault tree. The set of the results can be described as a probability distribution. Depending on the inputs some returning shapes are found, which indicates some types of gates. These shapes are not random, but some typical forms will return [2, IV].

In comparison to software, hardware does fail under random conditions. Adding the probability of hardware failures to each corresponding component will now return a probability of both software and hardware failures. A hardware component will influence a subcomponent of the fault tree by multiplying as follows:

$$P_{current\ total} = P_{current\ component} * (1 + P_{hardware\ failure}) \quad (1)$$

For unrealistic high failure probabilities the output of this formula can result into a value > 1. This means that the corresponding system will definitely fail. Such failing systems will never be put in operation and therefore such cases can therefore be excluded from tests.

Other methods

Next to using fault tree analysis to prove the safety aspect some other methods could be used for proving safety as well. One will put the model through an 'ontology-based model-driven engineering' process. Using this method has the advantage that it uses the same language as the components are built in. In order to verify and validate the safety, the model needs some constraints where a Description Logic OWL reasoner and inference rules detect lacks of elements and semantically inconsistent parts [6]. This method does have the advantage of integrating it in AADL (Architecture Analysis & Design Language, explained in the next section) directly and also has the possibility to run a fault tree analysis after.

In addition to the fault tree method that is used here, some variations for more detailed safety analysis can be used. In [4] a Component Logic Model is introduced where the inputs are specified by aspects as value, correctness or veracity. These are aspects also specified in SHARD (Software Hazard Analysis and Resolution in Design) where the aspects: omission, commission, early, late and value are found. Here the early and late aspects are interesting for a later and more detailed stage of analyzing safety.

Integration of the safety model

As the core system of the project is implemented in Java using Eclipse [8] this project will continue to use this platform. With an existing program calculating and drawing a fault tree available what was left to do is connect this to the optimization system. A component or complete system is defined in an AADL model (Architecture Analysis & Design Language), where ports, subcomponents, connections and flows are known. AADL is a standardized modeling language used in engineering fields. Using a tool called OSATE [7] an AADL model can be modeled and also checked for errors, for example for: illegal connections, component duplication.

AADL Model

In this section the AADL Model presented in Fig. 2 is explained. From top to bottom from left to right the following components are defined:

- The first element, *FLOWCONN*, describes the properties which the components in the model can have. The *andFlow* and *orFlow* are the logical gates and the *Probability_LowerBound* and *Probability_UpperBound* combined define the range of for example an inport.
- The processor model, *Proc*, which is subdivided in three processors in this case. Each processor has its own cost and probability of failure. The optimizer subtracts the values for the hardware failure probability from these components.
- Next is the *Throttle* which is actually the whole system. *Throttle.impl* contains the connections between the other components. This cannot be read from the figure, but is clearer in OSATE. The components of *Throttle* consist of the *inportProcess*, *outportProcess*, *Calculator* and *SanityChecker*.
- Another property which will be used in a definition processor is *AQOSA* in which the costs of a CPU are described.
- *Calculation* and *Checker* describe the internal connections of respectively *Calculator* and *SanityChecker*.

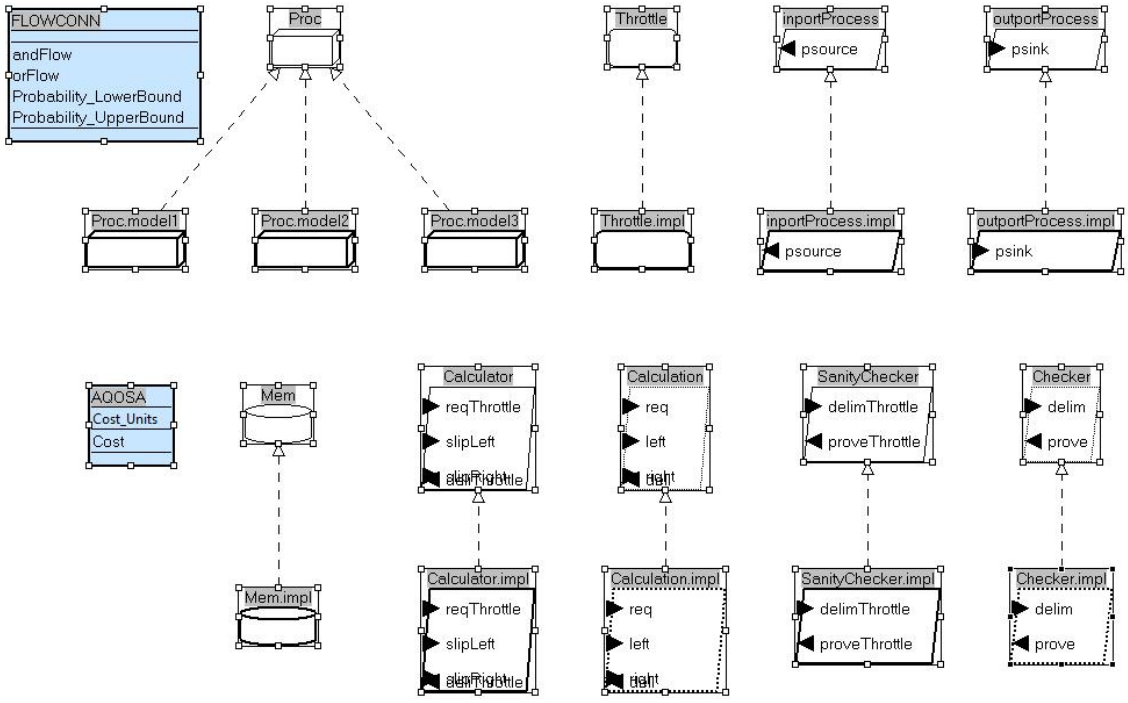


Fig. 2: AADL model of a throttle delimiter [4] (modeled in OSATE)

The AADL model has an XML file where the fault tree can be converted from. The corresponding elements between the AADL model and fault tree are:

AADL	Fault tree
dataPort	Inport (leafs)
dataPort, where direction=out	Outport (top level)
processType, processImpl, threadType, threadImpl	Subcomponents (nodes)
dataConnection	Connecting lines (edges)

Table 1: Relation of the fault tree and objects in AADL

Hardware failure probabilities are defined as a general property of a processor. Therefore there consists the possibility for different configurations in each execution. The relation of processor and subcomponent will not subtracted from the AADL model, but bound by an external file containing a list of several input ranges and CPUs. Also the inport probability range can be found in the same external file. At the start a randomly chosen probability in these ranges will be generated, which will result in a set of probabilities for all inports and number of samples. The AADL model defines threads which for the implementation is only necessary to subtract the subcomponents logical gate type and to create connections between the outport and inport of the subcomponent.

Fault tree integration

Initially a tool called ExTRAS was used to model a fault tree and analyze it. However this tool seemed incompatible with the optimization tool. This observation occurred when the hardware failure probability function was being integrated. The tool is not useless, as it is a good comparer for comparing it to the custom build convertor from the AADL file. Also it has a graphical interface where a fault tree can easily be modeled in. Converting from AADL to a fault tree will result in the tree depicted in Fig. 4 followed from Fig. 3 which describes the component.

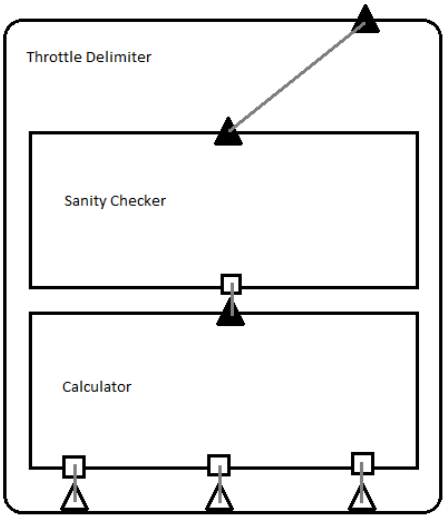


Fig. 3: Throttle delimiter and its subcomponents

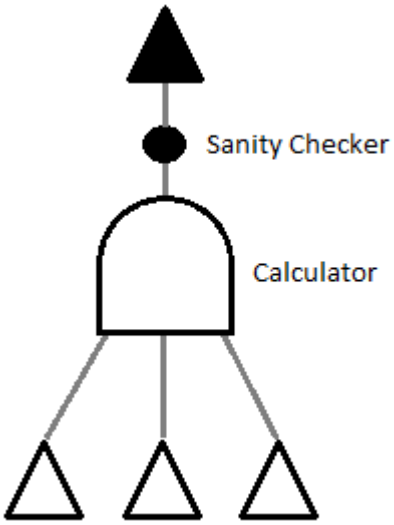


Fig. 4: Fault tree of the throttle delimiter

The fault tree is created top-down. Starting at the output it will visit every node in the tree by recursion. Three possible configurations of a node can be found: outport, subcomponent or inport. When finding an inport, its probability value is sent back to the subcomponent which called this inport. At the lowest subcomponent several inports can be found. With knowledge of the gate type the combined probability is calculated. The corresponding functions to the gate are:

Logical gate	Corresponding mathematical function
AND gate	$out = \prod_{k=in_{x+1}}^{in_y} k * (k-1) \quad k_0 = in_x$
OR gate	$out = 1 - \prod_{k=in_x}^{in_y} (1-k)$

XOR gate	$out = \sum_{i=in_x}^{in_y} (i \prod_{j=in_x}^{in_y} (1-j)) \quad j \neq i$

Table 2: Mathematical representation of the logical gates

The above functions are needed, because values provided are probabilities and therefore do not consist of a 0 or 1. This subcomponent is running on some hardware which influences the just calculated probability. To simulate this influence of the hardware, formula (1) will be used.

The value of the output will now be passed to the next subcomponent as an inport or to the outport. When it is given to the outport, this would be the final failure probability of the system. This will not be the final probability of the program, because one probability in a range cannot be generalized for the whole range. Therefore this process (Monte Carlo) will be done several times – for example 100 iterations - with each time a different set of values for both inports and hardware failures. The implementation of the fault tree in java can be found in appendix A.

Complexity

The complexity in time of the safety model depends on several values. One straight forward value is the number of samples multiplied with the number of inports that is needed to (initialization) run for one configuration: $O(numberOfSamples * numberOfInports)$. Next to this, the fault tree structure impacts the complexity. This is equal to $O(V + E + numberOfInports)$, where *V* is the number of components (processes) and *E* the connections between the components. Also the *numberOfInports* are included here, because each leaf is connected to an inport. The tree is rebuilt for every sample, and therefore resulting the total complexity to be:

$$O(numberOfSamples * numberOfInports * (V + E + numberOfInports))$$

This counts for best case and worst case, because all components are needed for the calculation and also for all samples.

Optimizing tool

After the iterations, the average of the results is taken and passed to the optimizer. Here the cost of the current configuration is known and the solution for this configuration can be mapped into a graph. The optimizer that is used is called Opt4J [9]. It will automatically plot the dominant points which form the pareto front. An example of such graph in Fig. 9 followed from a case presented in the *Analysis* chapter.

Analysis

In this chapter the fault tree analysis and results of an example will be reviewed. For the example a cruise control system will be used. In Fig. 5 the cruise control system [10] is given. First the system is depicted, then the AADL conversion to fault tree. The results of the fault tree and the graph for optimal points follow after.

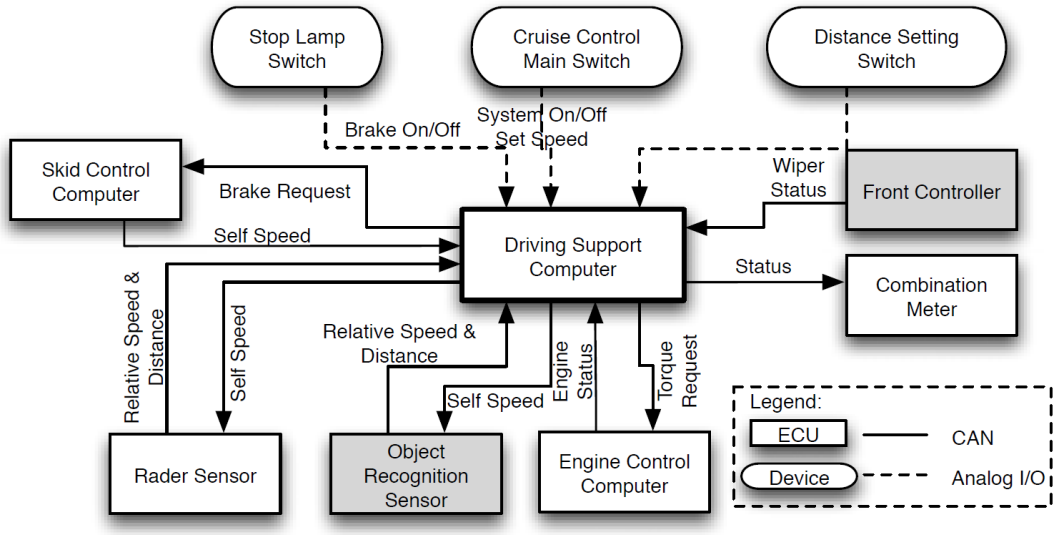
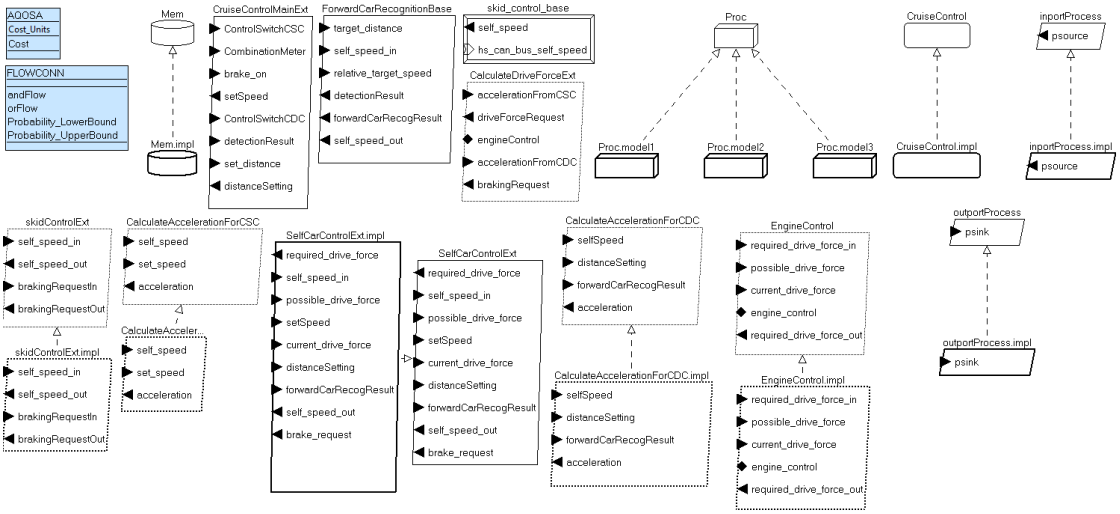


Fig. 5: Configuration of a cruise control system explained in [10]

From Fig. 5 it can be derived that there are some receiving and requesting connections from the center component. In order for the fault tree to be converted the requesting connection will be taken as an outport. This fault tree can be found in the figure below.



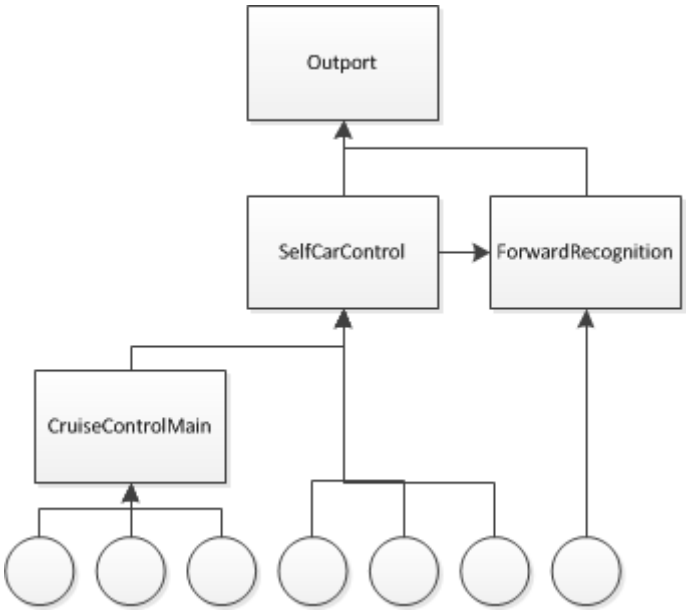


Fig. 6: AADL and fault tree diagram of a cruise control system

The value ranges, of the inports and hardware configurations, in the following tables Table 3 and Table 4 will be used for optimization. Inputs can provide faulty values; these are therefore translated to a failure probability range. From these options one will be chosen randomly and assigned to some inport or process.

Input 1	Input 2	Input 3	Input 4	Input 5	HW1	HW2	HW3
0.03 – 0.06	0.01 – 0.02	0.02 – 0.03	0.01 – 0.03	0.04 – 0.06	0.01 – 0.09	0.1 – 0.35	0.2 – 0.5

Table 3: Options for inputs and hardware with probabilities of failure

HW1	HW2	HW3
€200,-	€125,-	€75,-

Table 4: Cost of a certain hardware

The distribution of the fault tree for an example configuration is given in Table 5. In Table 4 the hardware costs are given. Cheaper hardware can have a larger chance of failure. The last three columns relate the processes CruiseControlMain (CCM), SelfCarControl (SCC), ForwardRecognition (FCR) to a certain hardware configuration.

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	CCM	SCC	FR
Input 1	Input 4	Input 3	Input 3	Input 3	Input 2	Input 3	HW1	HW3	HW1

Table 5: An example input configuration

The configuration above will result in the shape of distribution seen in Fig. 8. This distribution is found by running 10000 samples of the given configuration. The received values are the average probabilities after running it through the fault tree. This can be passed to the tool which optimizes the safety value against hardware costs in a later stage. In the cruise control system all the (logical)

gate types are generally defined with an OR gate as the system does not require information from all inputs to receive a valid output.

In general, when the input range is set to, for example 0.4 – 0.6, this will create a peak at 0.5. Having input ranges [0.3 – 0.5] and [0.5 – 0.7] results in a 'fatter' normal distribution than in the previous example. High bars on the right side of a graph are worse than high bars to the left, because lower probabilities are better.

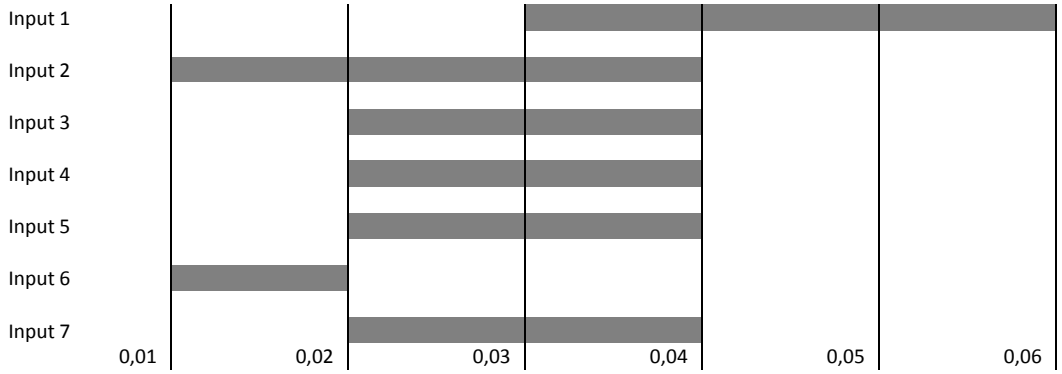


Fig. 7: Ranges of the inputs given in a visual representation

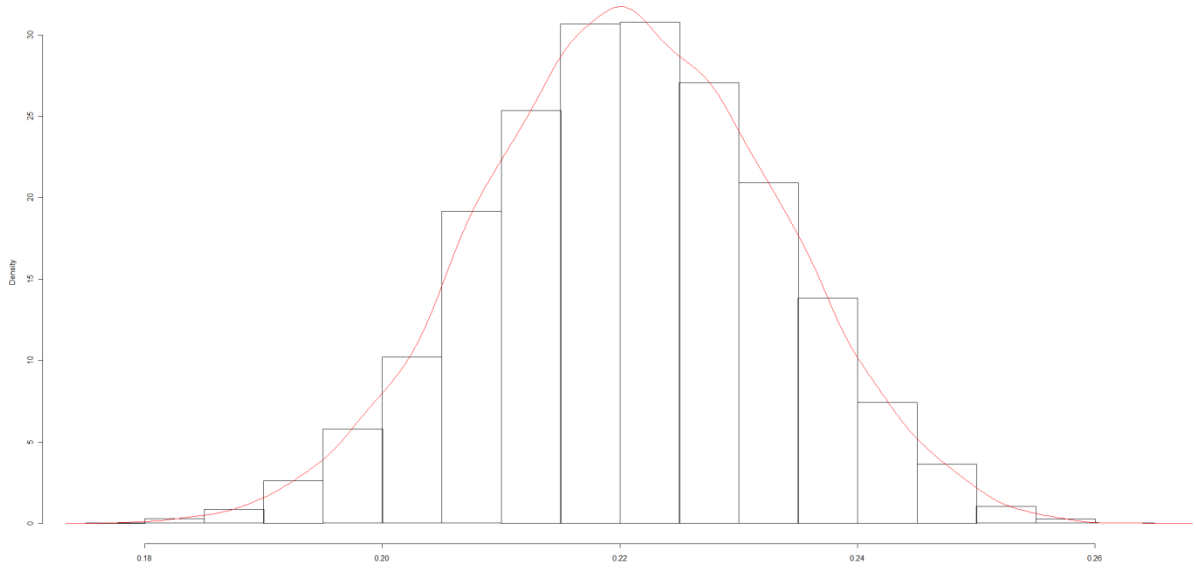


Fig. 8: Distribution of a fault tree with a certain probability configuration

The shape of the distribution is mainly caused by the used input values. In this case there is a slightly fatter peak towards the right. Looking at the input values from the above configuration one can see that it has more probability values and should shape more to the left from the 'median' (Fig. 7). However in the graph the distribution has shaped more to the right. This is due to the hardware probability failure, which (in absolute terms) strongly pulls the larger values to the right. One can see this from formula (1) in the chapter Safety aspect.

The position of the distribution is between 0.18 – 0.26. This means the probability of a system failure is 18% – 26%. Systems with such failure percentages are not appealing to use in real time systems.

However these numbers result from randomly chosen input values, hardware failure in particular, which of themselves have a relatively high chance of failure.

After running the optimizer the following pareto graph is obtained:

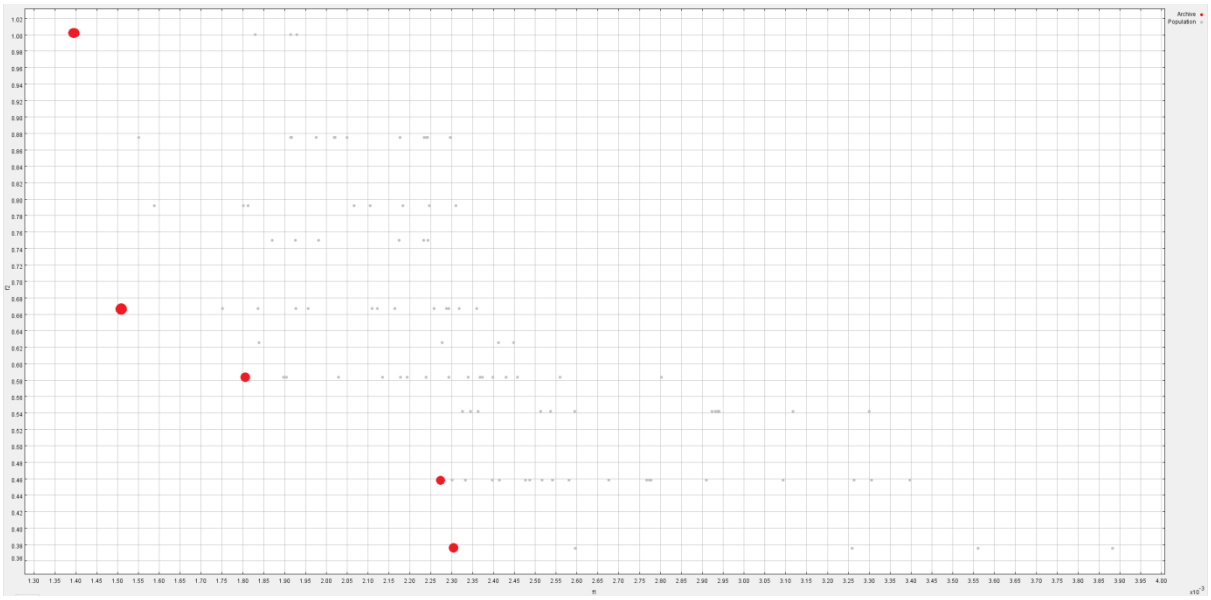


Fig. 9: Optimal solutions for cost against safety values

In Fig. 9 the red dots are the dominant points which define the pareto front when connected to each other. Grey dots are dominated by the red ones, and are therefore irrelevant to take into account. The vertical axis contains the cost values and the horizontal axis the probability values. Choosing the dot highest on the vertical axis gives a solution for the best configuration for safety, but worst for cost. This because the lower the probability number, the lower chance of a failure. This also counts the other way around; choosing the most right point of the horizontal axis will give the solution for best cost, but worst for safety.

Usually there are no solutions in a pareto graph, where one is better than the other, when there are no criteria for selection. However a remark about the bottom two solutions in Fig. 9 can be made. The top solution, 4 in Table 6, of the two has a cost factor which is greater than the bottom solution which is greater than the bottom solution 5. Because relatively the probabilities of the two do not differ much and the cost values do, one can say that solution 4 is not an interesting option to look at.

The red points are referenced in a separate output file where the configuration for that solution is recorded in. These values are given below.

No.	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	CCM	SCC	FCR	Probability	HW Cost
1	Input 2	Input 2	Input 3	Input 1	Input 3	Input 4	Input 2	HW3	HW1	HW3	0.14014	1.000
2	Input 5	Input 2	Input 4	Input 2	Input 4	Input 3	Input 2	HW3	HW1	HW3	0.15057	0.667
3	Input 4	Input 2	Input 1	Input 4	Input 2	Input 2	Input 2	HW2	HW2	HW1	0.1806	0.583
4	Input 2	Input 3	Input 2	Input 3	Input 3	Input 2	Input 1	HW1	HW1	HW2	0.22728	0.458
5	Input 2	Input 3	Input 1	Input 2	Input 2	Input 3	Input 3	HW3	HW2	HW3	0.2306	0.375

Table 6: Optimal solutions corresponding to a red dot (top to bottom) in Fig. 9

From this table one can obtain the configuration of one of the solutions in the graph. The order of the table is the order of the dots from top to bottom in the graph. The cost values are normalized. Thus where normalized cost is 1.000 in the table, it is €350,- in currencies. This can be calculated from the hardware configuration (HW3, HW1, HW3) with the prizes in Table 4. The probability values in Table 6 are not normalized, however in the graph they are. In the table there are cases where there is more than one of the same hardware type: HW2, HW2, HW1. It is possible that only two hardware types were used. So the processes CCM, SCC and FCR in solution 3 (Table 6) might have run CCM and SCC on HW2 and FCR on HW1, but also possible is CCM and SCC on two separate HW2 and FCR on HW1. The optimization tool does not show which of the two configurations it used. This however can manually be checked afterwards by looking at the HW Cost of the solution.

Conclusion

This paper discussed the integration of the safety aspect for it to be optimized with the cost factor. By converting an AADL model to a fault tree structure the safety aspect can be defined in a chance of failure of the system. The method provided the inports of the fault tree with a certain defined range of a failure probability and continued through following the edges via logical gates (nodes) until the top level is reached. These logical gates corresponded to components of the system, which are connected to a certain resource. This resource also influences the probability and will be taken into account after leaving a logical gate. Repeating this a number of times will result in the probability of failure of the entire system. This would then be passed to the optimizer, where it will be plotted against the cost factor.

The results of the cruise control system showed that the configuration of a hardware configuration influences the systems failure probability significantly. With a certain distribution of the input the hardware configuration altered the shape, by shifting it more to the right. Of the twelve possible combinations, five options formed the pareto front. One solution deviates from the front, but generally returns solutions which are distinct and therefore options which can be chosen for a specific criterion. The use of fault tree analysis to prove the safety of a system is an effective method, because the tree would be built by reading an AADL model when running the optimizer tool. Furthermore it is used in the aviation industry which is less error prone. Components can be added to the model in the future and therefore automatically added to the fault tree.

During the modeling of the example model of the cruise control, it appeared that some models received and requested from their sources (cycle). The result of this would be that the fault tree could not be converted from the model. For this problem the requests could be seen as outports, because when receiving from its sources the probability of failure from requests is included. Furthermore there could be several outports as seen in the example. These outports can be connected by adding an additional process with an appropriate gate type. The general structure of the model does not need to be altered for it to run through the optimizer. Processes with more than one path however, must be split into several equal to the number of paths that the original had.

During the analysis of the cruise control system it is good to see that the integration of different aspects (as safety) can easily be connected to the optimizer. When some other features need to be

implemented the connecting will be no problem. The optimization itself requires some time to calculate the optimal points. The source causing some significant calculation time is the fault tree. In order to get better results for the optimizer more samples (until a certain point, after that more samples will not affect the general distribution) are needed.

Future work

The current version of the tool has the possibility of optimizing systems for cost, number of hardware components and safety. To further improve the validity of the safety aspect, it is possible to add more options for the tool to analyze. This will increases the quality of the safety measure. Another small suggestion is to increase the reach of the conversion. A system can be differently modeled in AADL. Some methods might not be recognized by the tool. Though for fundamental research this is a less relevant topic.

An improvement on the implementation part is required for larger problems. This is not due to time complexity, but to the speed of read operations of the calculating system. In the current implementation the tree structure is constructed by reading several times per iteration from an XML file. As the file is on the hard drive and cannot be moved to the memory (by means of using a java function), the read operation must be reduced to one iteration and store the read structure in the memory.

Next to increasing the quality for the safety the features of the tool can be extended. There can for example be checked on reliability. This is somehow related to safety, but defines the time that the required functions of a system operate without having an error. For safety-critical systems this would therefore be interesting to analyze.

References

[1] R. Li, R. Etemaadi, M.T.M. Emmerich, M.R.V. Chaudron (2011). Automated Design of Software Architectures for Embedded Systems using Evolutionary Multiobjective Optimization.

[2] M. Föster, M. Trapp (2009). Fault tree analysis of software-controlled component systems based on second-order probabilities.

[3] P. Peti, R. Obermaisser, F. Tagliabo, A. Marino, S. Cherchio (2005). An Integrated Architecture for Future Car Generations.

[4] M. Förster, D. Schneider (2010). Flexible, any-time fault tree analysis with component logic models.

[5] T. DeLong (1970). *Master Thesis*. A Fault Tree Manual.

[6] K. Mokos, G. Meditskos, P. Katsaros, N. Bassiliades, A. Vasiliades (2010). Ontology-based Model Driven Engineering for Safety Verification.

[7] OSATE for modelling in AADL. http://www.aadl.info/aadl/currentsite/tool/osate.html

[8] Eclipse to run the program. http://www.eclipse.org/

[9] Opt4J to find optimizations. http://opt4j.sourceforge.net/

[10] S. Shiraishi (2010). An AADL-Based Approach to Variability Modeling of Automotive Control Systems.

Appendix A

```
//import java.io.File;
package edu.leiden.aqosa.safety;


//import java.io.IOException;
import java.util.Random;
import java.io.*;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
//import javax.xml.transform.Result;
//import javax.xml.transform.Source;
//import javax.xml.transform.Transformer;
//import javax.xml.transform.TransformerConfigurationException;
//import javax.xml.transform.TransformerException;
//import javax.xml.transform.TransformerFactory;
//import javax.xml.transform.TransformerFactoryConfigurationError;
//import javax.xml.transform.dom.DOMSource;
//import javax.xml.transform.stream.StreamResult;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
import java.util.Date;


//import org.w3c.dom.DOMException;
import org.w3c.dom.Document;
//import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
//import org.w3c.dom.xpath.XPathExpression;
import org.xml.sax.SAXException;



public class faultTree{

	protected double[][] generatedSamples;
	protected int numberOfSamples;
	protected int inports;
	protected double[] topProbability;
	protected double[][] hardwareFailBounds;
	protected double[][] hardwareFailCalc;
	protected String systemName;
	protected String[] hardwareName;
	protected int inportNumber;
	protected int sampleNumber;
	protected String aadlFile;

	public faultTree(int inputSamples, String aadlString) throws XPathExpressionException,
ParserConfigurationException, SAXException, IOException{
		this.aadlFile = aadlString;
```

```
		this.systemName = null;
		this.numberOfSamples = inputSamples;
		this.inports = getInports();
		this.generatedSamples = new double[this.numberOfSamples][this.inports];
		this.topProbability= new double[this.numberOfSamples];
		this.inportNumber = 0;
		this.sampleNumber = 0;
	}


	public double[] faultTreeAnalysis(double[][] inputProbabilities, String[] hardwareNames) throws
ParserConfigurationException, SAXException, IOException, XPathExpressionException {

		generateSamples(inputProbabilities);

		//GET SYSTEM NAME
		NodeList xPathQuery = getXPath("//systemType/@name");
		if (xPathQuery.getLength() > 0)
			this.systemName = xPathQuery.item(0).getNodeValue();
		else
			throw new IllegalArgumentException();

		//GET AND SET HARDWARE FAILURE PROBABILITIES
		getHardwareFailure(this.systemName, hardwareNames);

		//GET ALL PROCESSES, NUMBER OF PROCESSES IN SYSIMPL ARE ALWAYS TOTAL-INPORTS-
OUTPORT
		xPathQuery =
getXPath("//systemImpl[@name='"+this.systemName+".impl']/subcomponents/processSubcomponent/@class
ifier");
		String[] processNames = new String[xPathQuery.getLength()-this.inports-1];
		int counter = 0;
		for(int i = 0; i < xPathQuery.getLength(); ++i){
			if (xPathQuery.item(i).getNodeValue().indexOf("inportProcess") < 0 &&
xPathQuery.item(i).getNodeValue().indexOf("outportProcess") < 0){
				processNames[counter] = getSubstr(xPathQuery.item(i).getNodeValue(),
"/processImpl[@name=");
				counter++;
			}
		}

		//GET TOP LEVEL
	NodeList nodes = getXPath("//processType//dataPort[@name='psink']/@name");

	if(nodes.getLength() > 0) {
		double[] results = new double[2];
		double average = 0;
		//System.out.println("---BEGIN---");
		//Time now = new Time();
		Date today = new Date();
		StringBuilder stringBuilder = new StringBuilder();
			stringBuilder.append("out");
			stringBuilder.append(today.getTime());
			stringBuilder.append(".txt");
			File file = new File(stringBuilder.toString());
		Writer output = null;
		output = new BufferedWriter(new FileWriter(file));
```

```
                double total = 0;
                for(int i = 0; i < this.numberOfSamples; ++i){
                        average = followEdges(nodes.item(0).getNodeValue(), hardwareNames);
                        this.sampleNumber++;
                        this.inportNumber = 0;
                        System.out.println(i+": "+average);
                        output.write(Double.toString(average)+"\n");
                        average += total;
                }
                 output.close();
                System.out.println();
                //System.out.println("Average: "+average/this.numberOfSamples);
                results[0] = average/this.numberOfSamples;
                results[1] = getCost(hardwareNames);
                //System.out.println("Cost: "+ results[1]);
                //System.out.println("---END---");
                return results;
          }
          else
                throw new IllegalArgumentException();

        }

        //RECURSIVELY BUILD TREE, DFS
        private double followEdges(String dstNode, String[] hardwareNames) throws
ParserConfigurationException, SAXException, IOException, XPathExpressionException  {
                String processName = null;
                NodeList xPathQuery = getXPath("//dataPort[@name='"+dstNode+"']/../../@name");
                if(xPathQuery.getLength() > 0){//IF outportProcess
                        if(xPathQuery.item(0).getNodeValue().equals("outportProcess")){
                                processName = xPathQuery.item(0).getNodeValue();
                                xPathQuery =
getXPath("//systemImpl/connections/dataConnection[@dst='/aadlSpec[@name=" + this.systemName +
"]/processType[@name=" + processName + "]/features/dataPort[@name=" + dstNode + "']/@src");
                                if(xPathQuery.getLength() > 0){
                                        followEdges(getSubstr(xPathQuery.item(0).getNodeValue(),
"/dataPort[@name="), hardwareNames);
                                }
                        }else if(xPathQuery.item(0).getNodeValue().equals("inportProcess")){//IF
inportProcess
                                this.inportNumber++;
                                return this.generatedSamples[this.sampleNumber][this.inportNumber-1];
                        }else{//IF other processes
                                processName = xPathQuery.item(0).getNodeValue();

                                //System.out.println(processName);
                                //FLOW NAME OF PROCESS
                                String flowName = null;
                                xPathQuery =
getXPath("//processType[@name='"+processName+"']/flowSpecs/flowPathSpec/@name");
                                if(xPathQuery.getLength() > 0)
                                        flowName = xPathQuery.item(0).getNodeValue();

                             //GET INTERNAL FLOW OF PROCESS
                             NodeList flowImpl =
getXPath("//processImpl[@name='"+processName+".impl']/flows/flowPathImpl[@name='"+flowName+"']/flo
wElement/@*");
```

```
                        String[][] flowArray = new String[flowImpl.getLength()][3];
                        for(int i = flowImpl.getLength()-1; i >= 0; --i){        //READ TOP LEVEL -> LOWEST
LEVEL
                                String flowImplementation = flowImpl.item(i).getNodeValue();
                                flowArray[i][0] = null; flowArray[i][1] = null; flowArray[i][2] = null;
                                if(flowImplementation.indexOf("/flowPathSpec[@name=") > -1)
                                        flowArray[i][0] = getSubstr(flowImplementation,
"/flowPathSpec[@name=");

                                if(flowImplementation.indexOf("/threadSubcomponent[@name=") > -1){
                                        flowArray[i][1] = getSubstr(flowImplementation,
"/threadSubcomponent[@name=");
                                        //System.out.println(flowArray[i][1]);
                                }
                                if(flowImplementation.indexOf("/dataConnection[@name=") > -1)
                                        flowArray[i][2] = getSubstr(flowImplementation,
"/dataConnection[@name=");
                        }
                        //END GET FLOW

                        String[][] sinkNames = null;        //SINK NAME AND PROBABILITY
                        String[][] pathName = new String[1][2];                //PATH NAME AND PROBABILITY
//quick fix
                        for(int i = flowImpl.getLength()-1; i >= 0; --i){
                                if(flowArray[i][1] != null){
                                        xPathQuery = getXPath("//processImpl[@name='" + processName +
".impl']/subcomponents/threadSubcomponent[@name='" + flowArray[i][1] + "']/@classifier");

                                        String threadName =
getSubstr(xPathQuery.item(0).getNodeValue(), "/threadImpl[@name=");
                                        if(threadName == null)
                                                threadName =
getSubstr(xPathQuery.item(0).getNodeValue(), "/threadType[@name=")+".impl";
                                        if(xPathQuery.getLength() > 0){
                                                xPathQuery = getXPath("//threadImpl[@name='" +
threadName + "']/@compType");
                                                threadName =
getSubstr(xPathQuery.item(0).getNodeValue(), "/threadType[@name=");
                                                xPathQuery = getXPath("//threadType[@name='" +
threadName + "']/flowSpecs/flowPathSpec/@src");
                                                //SINKS
                                                NodeList xPathQuery2 =
getXPath("//threadType[@name='" + threadName + "']/flowSpecs/flowSinkSpec/@src");
                                                sinkNames = new String[xPathQuery2.getLength()][2];
                                                for (int j = 0; j < xPathQuery2.getLength(); ++j){

                                                        sinkNames[j][0] =
xPathQuery2.item(j).getNodeValue();
                                                }
                                                //END SINKS
                                                //System.out.println("dst:"+"//processImpl[@name='" +
processName + ".impl']/connections/dataConnection[@dst='" + xPathQuery.item(0).getNodeValue() +
"']/@src");
                                                xPathQuery = getXPath("//processImpl[@name='" +
processName + ".impl']/connections/dataConnection[@dst='" + xPathQuery.item(0).getNodeValue() +
"']/@src");
```

```
                                                dstNode = getSubstr(xPathQuery.item(0).getNodeValue(),
"/dataPort[@name=");
                                                pathName[0][0] = dstNode;



                                                        xPathQuery =
getXPath("//systemImpl/connections/dataConnection[@dst='/aadlSpec[@name=" + this.systemName +
"]/processType[@name=" + processName + "]/features/dataPort[@name=" + dstNode + "]']/@src");
                                                        //System.out.println(processName+":"+dstNode);

        //System.out.println(getSubstr(xPathQuery.item(0).getNodeValue(), "/dataPort[@name="));
                                                        if(xPathQuery.getLength() > 0){
                                                                pathName[0][1] =
Double.toString(followEdges(getSubstr(xPathQuery.item(0).getNodeValue(), "/dataPort[@name="),
hardwareNames));
                                                                //System.out.println(pathName[0][1]);
                                                                //this.inportNumber++;
                                                        }


                                                        for (int j = 0; j < xPathQuery2.getLength(); ++j){

                                                        xPathQuery = getXPath("//processImpl[@name='"
+ processName + ".impl']/connections/dataConnection[@dst='" + sinkNames[j][0] + "']/@src");
                                                        dstNode =
getSubstr(xPathQuery.item(0).getNodeValue(), "/dataPort[@name=");
                                                                xPathQuery =
getXPath("//systemImpl/connections/dataConnection[@dst='/aadlSpec[@name=" + this.systemName +
"]/processType[@name=" + processName + "]/features/dataPort[@name=" + dstNode + "]']/@src");
                                                                if(xPathQuery.getLength() > 0){
                                                                        sinkNames[j][1] =
Double.toString(followEdges(getSubstr(xPathQuery.item(0).getNodeValue(), "/dataPort[@name="),
hardwareNames));
                                                                        //this.inportNumber++;
                                                                }
                                                }
                                                  xPathQuery = getXPath("//processImpl[@name='" +
processName + ".impl']/flows/flowPathImpl/properties/propertyAssociation/@propertyDefinition");
                                                  String gateType = "";
                                                  if(xPathQuery.getLength() > 0)
                                                                gateType =
getSubstr(xPathQuery.item(0).getNodeValue(), "/propertyDefinition[@name=");
                                                  double sinkTemp = 1;
                                                  double sinkTotal = 0;
                                                  for (int j = 0; j < sinkNames.length; j++){
                                                        if(gateType.equals("andFlow")){
                                                                this.topProbability[this.sampleNumber] =
Double.parseDouble(pathName[0][1]) * Double.parseDouble(sinkNames[j][1]);
                                                        }else if(gateType.equals("orFlow"))
                                                                sinkTemp *= (1-
Double.parseDouble(sinkNames[j][1]));
                                                        else if(gateType.equals("xorFlow")){
                                                                for (int k = 0; k < sinkNames.length; k++){
                                                                        if(k != j)
                                                                                sinkTemp *= (1-
Double.parseDouble(sinkNames[k][1]));
```

```
                                                                }
                                                                sinkTemp *=
Double.parseDouble(sinkNames[j][1]);
                                                                sinkTotal += sinkTemp;
                                                        }
                                                }
                                                if(gateType.equals("orFlow"))
                                                        this.topProbability[this.sampleNumber] = 1-((1-
Double.parseDouble(pathName[0][1]))*sinkTemp);
                                                else if(gateType.equals("xorFlow"))
                                                        this.topProbability[this.sampleNumber] =
sinkTotal;
                                                if(sinkTotal == 0 && sinkTemp == 1)
                                                        this.topProbability[this.sampleNumber] =
Double.parseDouble(pathName[0][1]);
                                        }
                                        break;
                                }
                        }

                        //System.out.println(processName+": "+this.topProbability[this.sampleNumber]);
                        for(int i = 0; i < this.hardwareName.length;++i){
                                if(processName.equals(hardwareName[i])){
                                        this.topProbability[this.sampleNumber] *=
(1+this.hardwareFailCalc[0][i]);
                                        break;
                                }
                        }
                        //System.out.println("After HF : "+this.topProbability[this.sampleNumber]);
                        return this.topProbability[this.sampleNumber];
                       }//else
                }//if
                return this.topProbability[this.sampleNumber];
        }
        //END followEdges()



        //GET COST OF CONFIGURATION
        public double getCost(String[] hardwareNames) throws XPathExpressionException,
ParserConfigurationException, SAXException, IOException{
                Double costTemp = 0.0;
                Double max = 0.0;
                for(int i = 0; i < hardwareNames.length; ++i){
                        NodeList xPathQuery =
getXPath("//systemImpl/subcomponents/processorSubcomponent[@name='" + hardwareNames[i] +
"']/@classifier");
                        if(xPathQuery.getLength() > 0){
                                String procModel = getSubstr(xPathQuery.item(0).getNodeValue(),
"/processorImpl[@name=");
                                xPathQuery = getXPath("//processorImpl[@name='" + procModel +
"']/properties/propertyAssociation[@propertyDefinition='/aadlSpec[@name=" + this.systemName +
"]/propertySet[@name=AQOSA]/propertyDefinition[@name=Cost]']/propertyValue/@value");
                                if(xPathQuery.getLength() > 0){
                                        costTemp +=
Double.parseDouble(xPathQuery.item(0).getNodeValue());
                                }
```

```
                        }
                }
                NodeList xPathQuery =
getXPath("//processorImpl/properties/propertyAssociation[@propertyDefinition='/aadlSpec[@name=" +
this.systemName +
"]/propertySet[@name=AQOSA]/propertyDefinition[@name=Cost]']/propertyValue/@value");
                for(int i = 0; i < xPathQuery.getLength(); ++i){
                        if(Double.parseDouble(xPathQuery.item(i).getNodeValue()) > max)
                                max = Double.parseDouble(xPathQuery.item(i).getNodeValue());
                }

                return costTemp /= (max*hardwareNames.length);

        }
        //END getGost()


        private void getHardwareFailure(String systemName, String[] hardwareNames) throws
XPathExpressionException, ParserConfigurationException, SAXException, IOException{
          //GET HARDWARE BINDING
          NodeList hardFail =
getXPath("//systemImpl[@name='"+systemName+".impl']/properties//propertyAssociation/@appliesTo");
          String[][] hardBinding = new String[hardFail.getLength()][2];

          this.hardwareName = new String[hardFail.getLength()];
          this.hardwareFailBounds = new double[hardFail.getLength()][2];
          this.hardwareFailCalc = new double[1][hardFail.getLength()];
          for(int i=0; i < hardFail.getLength(); ++i){
                if(hardFail.item(i).getNodeValue().indexOf("/processSubcomponent[@name=") > -1){
                        hardBinding[i][0] = getSubstr(hardFail.item(i).getNodeValue(),
"/processSubcomponent[@name=");
                        this.hardwareName[i] = hardBinding[i][0];
                        if(hardwareNames.length > i)
                                hardBinding[i][1] = hardwareNames[i];
                }
          }

          NodeList hardFail3 =
getXPath("//systemImpl[@name='"+systemName+".impl']/properties//propertyAssociation/propertyValue/@r
eferenceElement");
          String[] hardwarenow = new String[hardFail3.getLength()];
          for(int i=0; i < hardFail3.getLength(); ++i){
                hardwarenow[i] = getSubstr(hardFail3.item(i).getNodeValue(),
"/processorSubcomponent[@name=");
          }
          for(int i=0; i < hardFail3.getLength(); ++i){
                for(int j=0; j < hardFail3.getLength(); ++j){
                        if(hardwarenow[i].equals(hardwarenow[j]) && i < j){
                                if(hardBinding[j][1] == null){
                                        hardBinding[j][1] = hardBinding[i][1];
                                }else{
                                        hardBinding[i][1] = hardBinding[j][1];
                                }
                        }
                }
          }
```

```
            for(int i = 0; i < hardFail.getLength(); ++i){
                  if(hardFail.item(i).getNodeValue().indexOf("/processSubcomponent[@name=") > -1){
                              System.out.println(hardBinding[i][1]);
                        NodeList hardFail2 =
getXPath("//systemImpl[@name='"+systemName+".impl']/subcomponents/processorSubcomponent[@name=
'"+hardBinding[i][1]+"']/@classifier");
                        if(hardFail2.getLength() > 0){
                        if(hardFail2.item(0).getNodeValue().indexOf("processorImpl[@name=") > -1){
                              String hardFailName = getSubstr(hardFail2.item(0).getNodeValue(),
"processorImpl[@name=");

                              NodeList processHardFail =
getXPath("//processorImpl[@name='"+hardFailName+"']/properties/propertyAssociation/@propertyDefinition
");
                              for(int l=0; l < processHardFail.getLength(); ++l){
                                    NodeList boundValue =
getXPath("//processorImpl[@name='"+hardFailName+"']/properties/propertyAssociation[@propertyDefinition
='"+processHardFail.item(l).getNodeValue()+"']/propertyValue/@value");
                                    for(int k=0; k < boundValue.getLength(); ++k){
                                    String bound =
getSubstr(processHardFail.item(l).getNodeValue(), "propertyDefinition[@name=");
                                    if(bound.equals("Probability_LowerBound"))
                                          this.hardwareFailBounds[i][0] =
Double.parseDouble(boundValue.item(k).getNodeValue());

                                    else if(bound.equals("Probability_UpperBound"))
                                          this.hardwareFailBounds[i][1] =
Double.parseDouble(boundValue.item(k).getNodeValue());
                                    }
                              }
                        }
                  }
                  }
            }

            //for(int i = 0; i < numberOfSamples; ++i){
            int i = 0;
                  generateHardwareFail(i);
            //}
            //END HARDWARE BINDING
          }


          private void generateHardwareFail(int k){
                  Random randomizer = new Random();
                  for (int j=0; j < this.hardwareFailBounds.length; ++j) {
                        double lowerBound = hardwareFailBounds[j][0];
                        double upperBound = hardwareFailBounds[j][1];
                        double intervalWidth = upperBound - lowerBound;
                        if (intervalWidth < 0)
                              throw new IllegalArgumentException();
                        if (intervalWidth == 0) {// lowerBound == upperBound
                              //for (int i = 0; i < this.generatedSamples.length; i++) {
                                    this.hardwareFailCalc[k][j] = lowerBound;
                              //}
                        } else {   // generate random value within interval
                              //for (int i = 0; i < this.generatedSamples.length; i++) {
```

```
                                        double nextRandomValue = randomizer.nextDouble();
                                        this.hardwareFailCalc[k][j] = (nextRandomValue * intervalWidth) +
lowerBound;
                                //}
                        }
                }
        }

        //GET NUMBER OF INPORTS
        private int getInports() throws ParserConfigurationException, SAXException, IOException,
XPathExpressionException  {
                NodeList xPathQuery =
getXPath("//systemImpl/subcomponents/processSubcomponent/@classifier");
                int counter = 0;
                for (int i = 0; i < xPathQuery.getLength(); ++i){
                if (xPathQuery.item(i).getNodeValue().indexOf("inportProcess.impl") > -1)
                        counter++;
                }
                return counter;
        }

        //GENERATE PROBABILITIES FOR THE INPORTS AND ALL SAMPLES
        private void generateSamples(double[][] inputProbabilities) throws XPathExpressionException,
ParserConfigurationException, SAXException, IOException {
                Random randomizer = new Random();
                double[][] inportBounds = new double[this.inports][2];

                //INPUT PROBABILITY IS SET BY THROTTLEEVALUATIONFUNCTION
                if(inputProbabilities.length < 1){
                        NodeList dataPortName =
getXPath("//processType[not(@name='outportProcess')]//dataPort[not(@direction)]/@name");
                        for (int j=0; j < dataPortName.getLength(); ++j) {
                                NodeList inportSrc =
getXPath("//dataPort[@name='"+dataPortName.item(j).getNodeValue()+"']/properties/propertyAssociation/@
propertyDefinition");
                                NodeList boundValue =
getXPath("//dataPort[@name='"+dataPortName.item(j).getNodeValue()+"']/properties/propertyAssociation/pr
opertyValue/@value");
                                for(int i=0; i < inportSrc.getLength(); ++i){
                                        String bound = getSubstr(inportSrc.item(i).getNodeValue(),
"propertyDefinition[@name=");
                                        if(bound.equals("Probability_LowerBound"))
                                                inportBounds[j][0] =
Double.parseDouble(boundValue.item(i).getNodeValue());
                                        else if(bound.equals("Probability_UpperBound"))
                                                inportBounds[j][1] =
Double.parseDouble(boundValue.item(i).getNodeValue());
                                }
                        }
                }else{
                        for(int i=0; i<inputProbabilities.length;++i){
                                inportBounds[i][0] = inputProbabilities[i][0];
                                inportBounds[i][1] = inputProbabilities[i][1];
                        }
                }
                for (int j=0; j < this.inports; ++j){
                        double lowerBound = inportBounds[j][0];
```

```
                        double upperBound = inportBounds[j][1];
                        double intervalWidth = upperBound - lowerBound;
                        if (intervalWidth < 0)
                                throw new IllegalArgumentException();
                        if (intervalWidth == 0) {// lowerBound == upperBound
                                for (int i = 0; i < this.generatedSamples.length; i++) {
                                        this.generatedSamples[i][j] = lowerBound;
                                }
                        } else {   // generate random value within interval
                                for (int i = 0; i < this.generatedSamples.length; i++) {
                                        double nextRandomValue = randomizer.nextDouble();
                                        this.generatedSamples[i][j] = (nextRandomValue * intervalWidth) +
lowerBound;
                                }
                        }
                }
        }

        //NON FUNDAMENTAL FUNCTION, GET SUBSTRING
        private String getSubstr(String str, String substr){
        if(str.indexOf(substr) > -1)
                return str.substring(str.indexOf(substr)+substr.length(), str.length()-1);
                return null;
        }

        //NON FUNDAMENTAL FUNCTION, GET RESULTS FOR A XPATH QUERY
        private NodeList getXPath(String query) throws ParserConfigurationException, SAXException,
IOException, XPathExpressionException  {
                DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
                domFactory.setNamespaceAware(true);
                DocumentBuilder builder = domFactory.newDocumentBuilder();
                Document doc = builder.parse(this.aadlFile);
                XPath xpath = XPathFactory.newInstance().newXPath();
                javax.xml.xpath.XPathExpression expr = xpath.compile(query);
           Object result = expr.evaluate(doc, XPathConstants.NODESET);
           return (NodeList) result;
        }

}
```