Contract number: Eurostars 6095 Safe-E

Safe Automotive soFtware architEcture – Extension (SAFE-E)

WP4

Deliverable D4.4b: Final version of plug-in for safety and multi criteria architecture modeling and benchmarking

Please note: This document is a direct input to the SAFE Project (Contract number: ITEA2 – 10039) as Deliverable D4.2.6.b2

Due date of deliverable: 31/12/2014

Actual submission date: 28/11/2014

Start date of the project: 01/09/2011 **Duration:** 40 months

Project coordinator name: Andreas Eckel

Organization name of lead contractor for this deliverable: fortiss GmbH

Editors: Maged Khalil – fortiss GmbH

Contributors: Mayank Chaudhary, Maged Khalil, Sergey Zverlov – fortiss GmbH

Reviewers: Eduard Metzker – Vector Informatik (SAFE Partner)
Sergey Zverlov – fortiss GmbH

Deliverable type: PUBLIC

Revision chart and history log

Version	Date	Reason
0.1	17.12.2012	Initialization of document
0.2	23.04.2013	Update document structure
0.3	05.09.2013	Integration of content: optimization for deployment
0.4	20.09.2013	Integration of review comments
0.9	30.09.2013	Integration of updated content and review comments
1.0	17.10.2013	Finalization of first version of document D4.4.a
1.1	01.02.2014	Initialization of final version of document D4.4.b
1.2	01.05.2014	Update of document structure
1.3	17.11.2014	Integration of content: safety constraints for optimization
1.4	21.11.2014	Update of plug-in initialization description
1.5	28.11.2014	Integration of review comments
2.0	28.11.2014	Finalization of document

1 Table of Contents

2 List of Figures

3 List of Tables

4 Executive Summary

Objective of the corresponding work task WT4.4 "*Plug-in for safety and multi criteria architecture modeling and benchmarking*" is the implementation and evaluation of conceptual results of especially work task WT3.4 *"Safety and multi-criteria architecture benchmarking"*. Therefore, this deliverable describes a first version of private prototype implementations and private plug-ins for model-based multi-criteria and safety evaluation. The implementations are based on the existing research CASE tool AUTOFOCUS3 (AF3).

The implementation described in this deliverable focuses on the functionality:

- Multi-criteria Deployment Generation, Scheduling Synthesis and Design Space Exploration

5 Introduction and Overview of the Document

This deliverable D4.4b *"Final version of plug-in for safety and multi criteria architecture modeling and benchmarking"* is included in the work package 4 "*Technology Platform"* and describes the implementations of concepts and methodologies provided by work task of work package 3 "*Model Based Development"*. Especially WT3.4 is in focus.

This document provides different selected methodologies to perform model-based assessment based on the SAFE Meta-model extensions. These assessments are facilitated on different level of abstraction and different level of granularity for safety multi-criteria benchmarking of automotive systems. Qualitative analysis regarding the impact of hardware failures on system behavior is specified. Formal consistency of the safety case is required for qualitative assessment of requirements. Additionally, a quantitative evaluation of hardware designs according to ISO 26262 is presented. Combination of the analyses allows a benchmarking of different architectures regarding safety and different criteria.

5.1 Links to SAFE and SAFE-E Work Tasks

ISO 26262 [1] was analyzed and requirements specified in work task SAFE WT2.1 *"ISO26262 Analysis"* [2] and work task SAFE WT2.3 *"Use case scenario"* [3] as well as to SAFE-E WT2.1 and WT2.2. Based on these requirements, the concept work task SAFE WT3.2.2 *"Hardware description"* [4] provides a description and a meta model for hardware modeling including failure extension, as shown in Figure 1. SAFE-E WT3.5 *"Safety and Multi Criteria Architecture Benchmarking"* [5] specifies the methodology and forms the basis for the plug-ins and implementations described in this deliverable. Additional Specifications for Safety Cases were carried out in SAFE-E WT3.4.

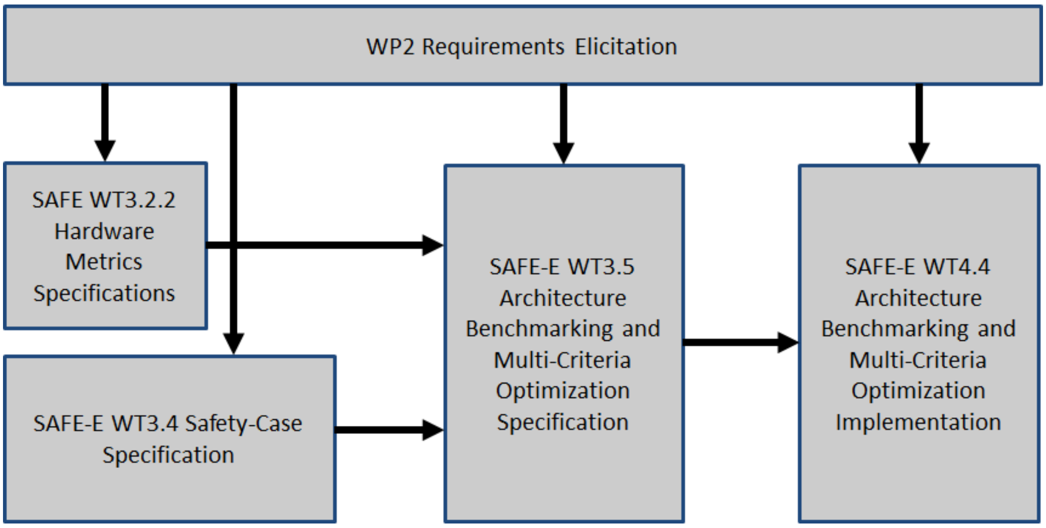

Figure 1: Overview of essential WT4.4 links to SAFE/SAFE-E Work tasks

5.1.1 Multi-criteria Design Space Exploration

The prototype implementation supports a seam-less model based approach to Multi-criteria Deployment Generation, Scheduling Synthesis and Design Space Exploration. The approach is aided by the capability to visualize the design space as well as manipulate this visualization which facilitates the understanding of the otherwise complex multi-criteria design space and thus helps the design effort for a safe product.

5.2 Structure of the Document

The next Section 5 presents the tool environment AutoFOCUS3, which was used for the implementations. An overview of AutoFOCUS3 including the relevant abstraction layers and artifacts for this deliverable is given. A detailed description of the plug-in for Multi-criteria Deployment Generation, Scheduling Synthesis and Design Space Exploration is described in Section 5, and the additional constraints developed for this project as well as their integration and interaction with the existing solution are described in detail in section 7.

6 Description of Technology Platform AutoFOCUS3

AutoFOCUS3 (AF3) [9] is a model based development research CASE tool for distributed, reactive, embedded software systems. AF3 uses models in all development phases including requirements analysis, design of the logical architecture and the hardware architecture, implementation and deployment. Furthermore, AF3 features formal analyses and synthesis methods.

AF3 and its tutorials are free for download at af3.fortiss.org.

6.1 Main Features of AF3:

Requirements Engineering (MIRA)

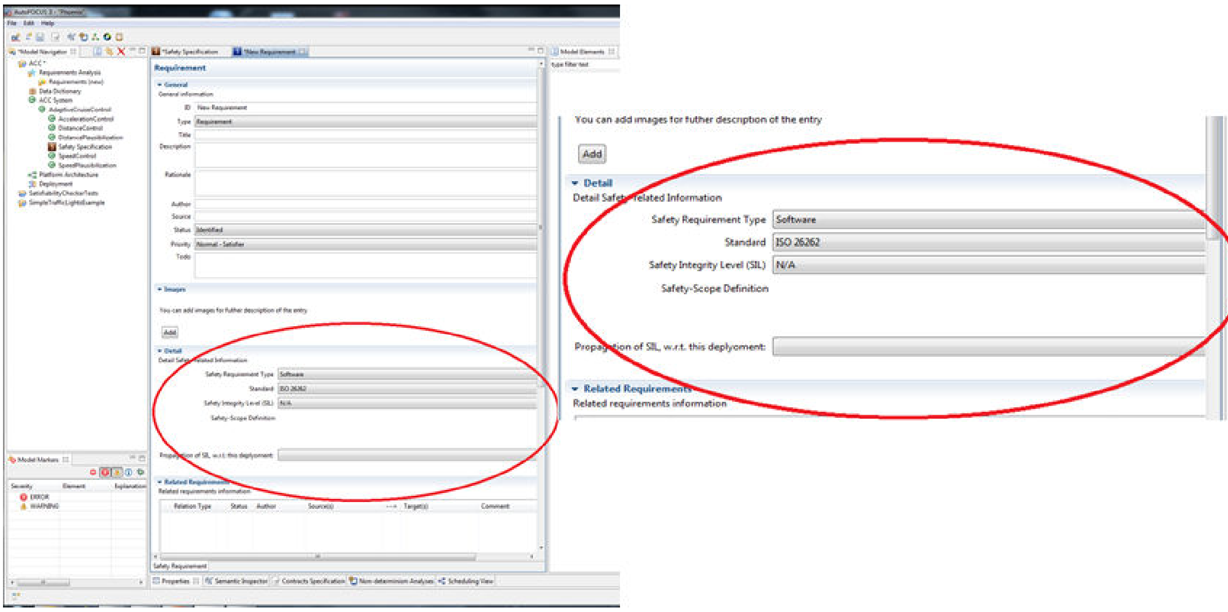

Figure 2: Safety requirements support in AF3

- Structure, analyze and validate requirements
- Specify requirements in a rich model
- Deeply integrate requirements with the architecture
- Support for safety requirements and link to a Safety-Case View

Modeling and simulation

- Advanced and deeply integrated models for architecture, behavior and platform
- Simulation and on the fly consistency checks
- System Architecture Modeling
- Data Modeling

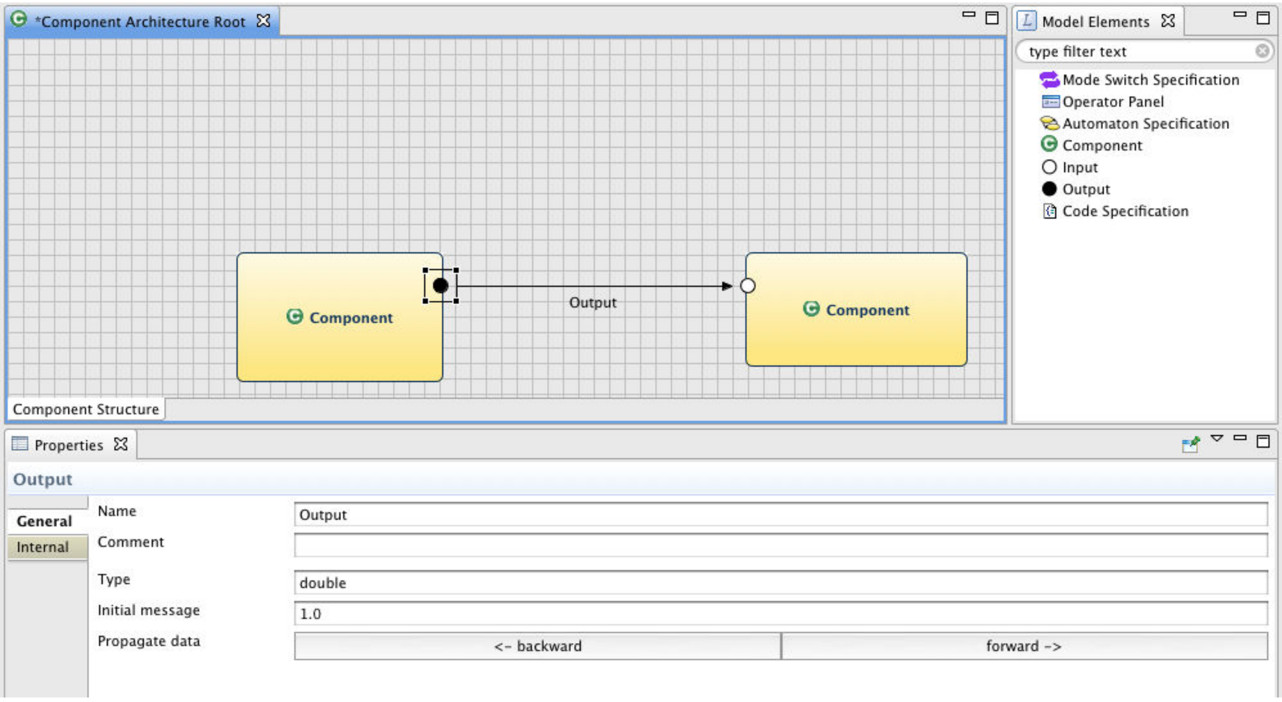

Figure 3: Component Modeling in AF3

- Behavior Modeling:
    - Code Specifications
    - State Automata
    - Mode Automata
    - Tabular Specifications
- Hardware Architecture Modeling
- Simulation
- Operator Panels
- Library of Components, Functions and Types
- Holistic Pattern Libraries

Code generation and deployment

- Flexible generation of C, Java, etc., code
- Deployment on different hardware platforms
- Generation of platform specific code

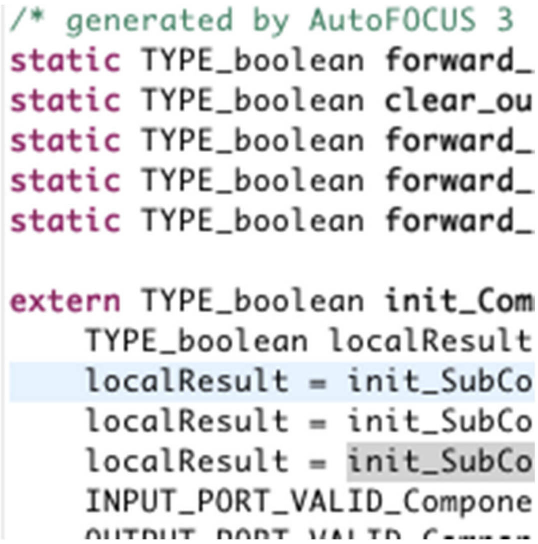

Figure 4: Code generation in AF3

<u>**Testing and formal verification**</u>

- Deep integration of the NuSMV model-checker and Yices SMT solver
- Both out-of-the-box and customizable analyses
- On the Fly Constraints Checking
- Model Checking
- Model Checking (using C-Prover)
- Non-determinism Analysis
- Model Based Testing
- Refinement Testing
- Assume/Guarantee Reasoning
- MSC Feasibility Checking

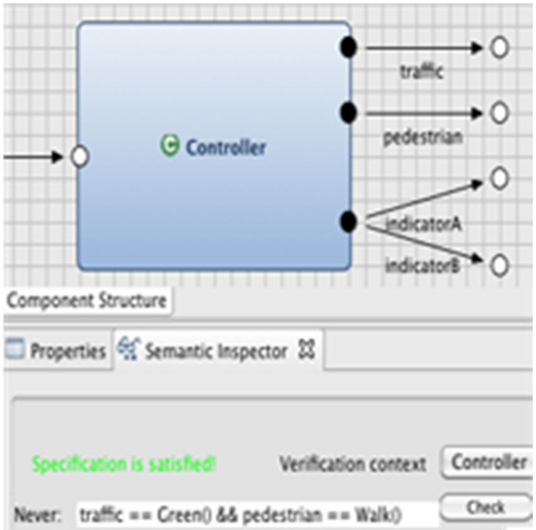

Figure 5: Testing and Verification in AF3

6.2 Installation details of SAFE plugin for Af3 developers

This section describes how the SAFE plugin can be installed in Af3. Af3 can be installed using the simple steps mentioned in the link "https://af3.fortiss.org/projects/autofocus3/wiki/Developer_Installation". The steps to start AF3 and install SAFE plugin are described below.

a) Double-click **org.fortiss.af3.phoenix.product.top**, then open **af3_phoenix.product** and select **Launch an Eclipse Application. (**Figure 6**)**
b) On the first run, you will probably get an error "**The application could not start. Would you like to view the log?**", press **No**.

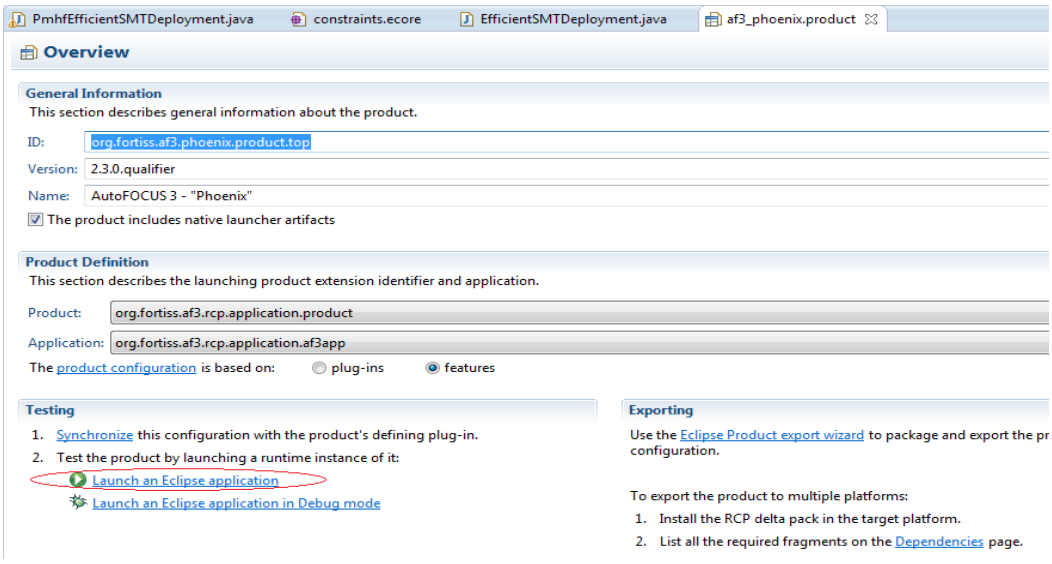

Figure 6: Installation step 1

c) To fix this, go to **Run->Run configurations...**, select **Eclipse Application-> af3_phoenix.product**, go to the **Plug-Ins** tab and check the checkbox **Workspace** in the list of plugins (if not already checked).
d) To the very right of the window (might even be hidden), press the button **Add Required Plug-Ins**. (Figure 7)

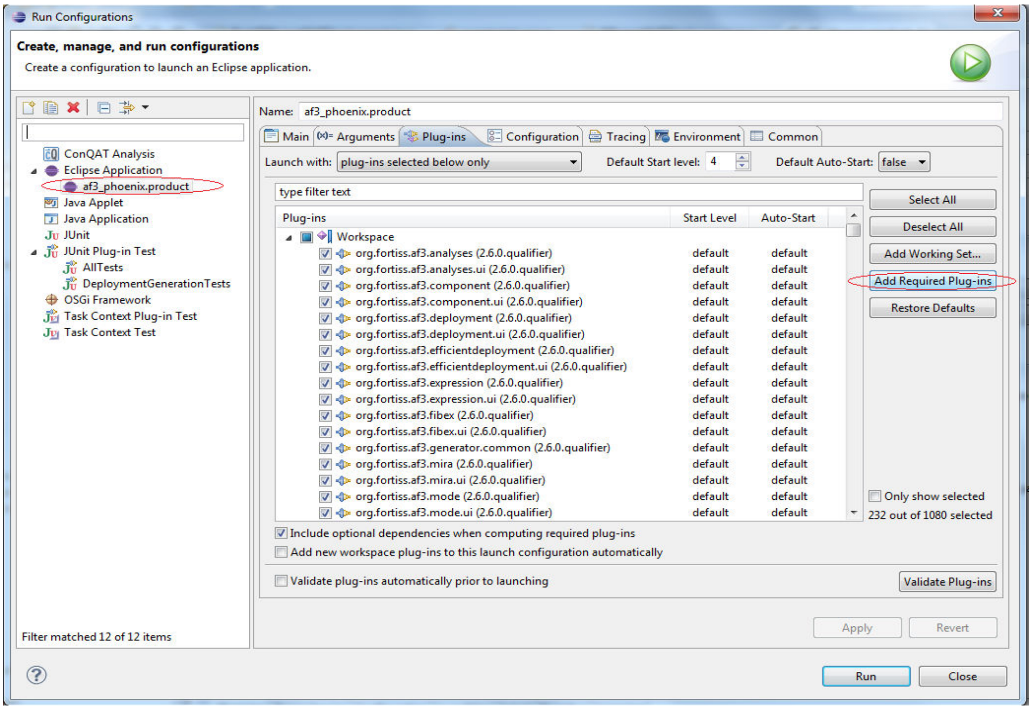

Figure 7: Installation step 2

e) If the **org.fortiss.af3.safe** and **org.fortiss.af3.safe.ui** plugins are not checked automatically, check it manually. (Figure 8). Press **Apply** and then **Run**. Now AF3 will start correctly as RCP.

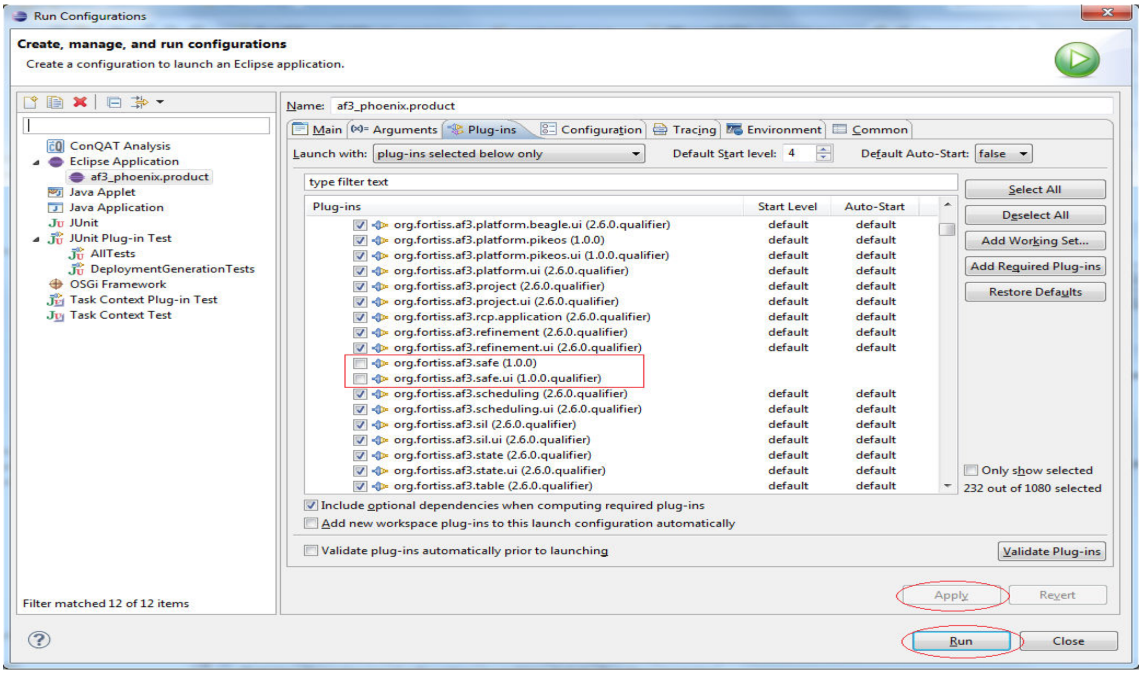

Figure 8: Installation step 3

f) There might be a case that the **org.fortiss.af3.safe** and **org.fortiss.af3.safe.ui** plugins are not on the list. Since the plugins are not added to the master branch yet, they are needed to be checked out manually. For that, go to the **SVN Repository Exploring view** and check out these two plugins from URL https://orion.fortiss.org/svn/af3, as seen in Figure 9.

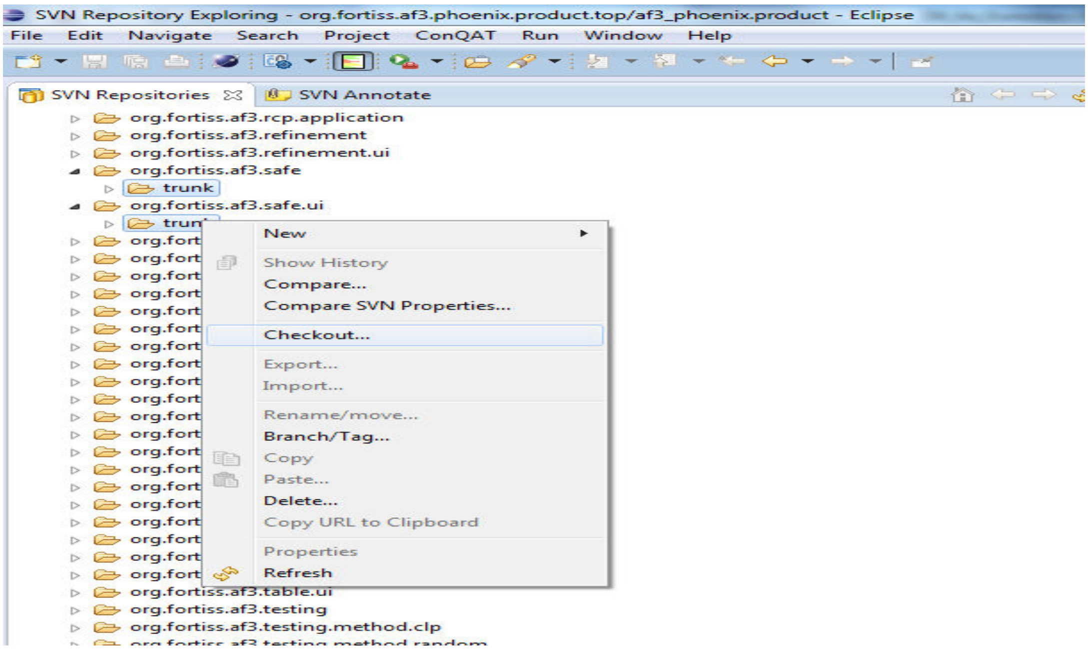

Figure 9: Installation step 4

7 Multi-criteria Architecture Optimization and Design Space Exploration

The presented methodology represents an efficient approach for generating suitable system architectures for embedded systems efficiently. The focus is on a joint generation of schedules and deployment for mixed-criticality multicore architectures using shared memory. The presented approach computes task and message schedules that are optimized with respect to a global discrete time base. As part of the solution, the approach generates an optimized assignment of tasks to computation resources (cores) concerning local memory constraints of cores and criticality constraints of tasks. This approach is integrated into the AUTOFOCUS 3 tool-chain [9], using a formally defined model of computation with explicit data-flow and discrete-time semantics to develop multi-criticality embedded systems [10].

The approach relies on a symbolic encoding scheme, based on a system model that is derived from the system architecture. A formalization describing the scheduling problem as a satisfiability problem using boolean formulas and linear arithmetic constraints, which are the tackled by a state-of-the-art satisfiability modulo theory (SMT) solver in order to compute the joint schedule and deployment for such architectures, is presented in [10].

Implementations are being carried out in the research CASE tool AutoFOCUS3 [9], presented in Section 6, and as shown in Figure 10, part of the tutorial referenced in [11], and will be presented in depth in the following subsections.

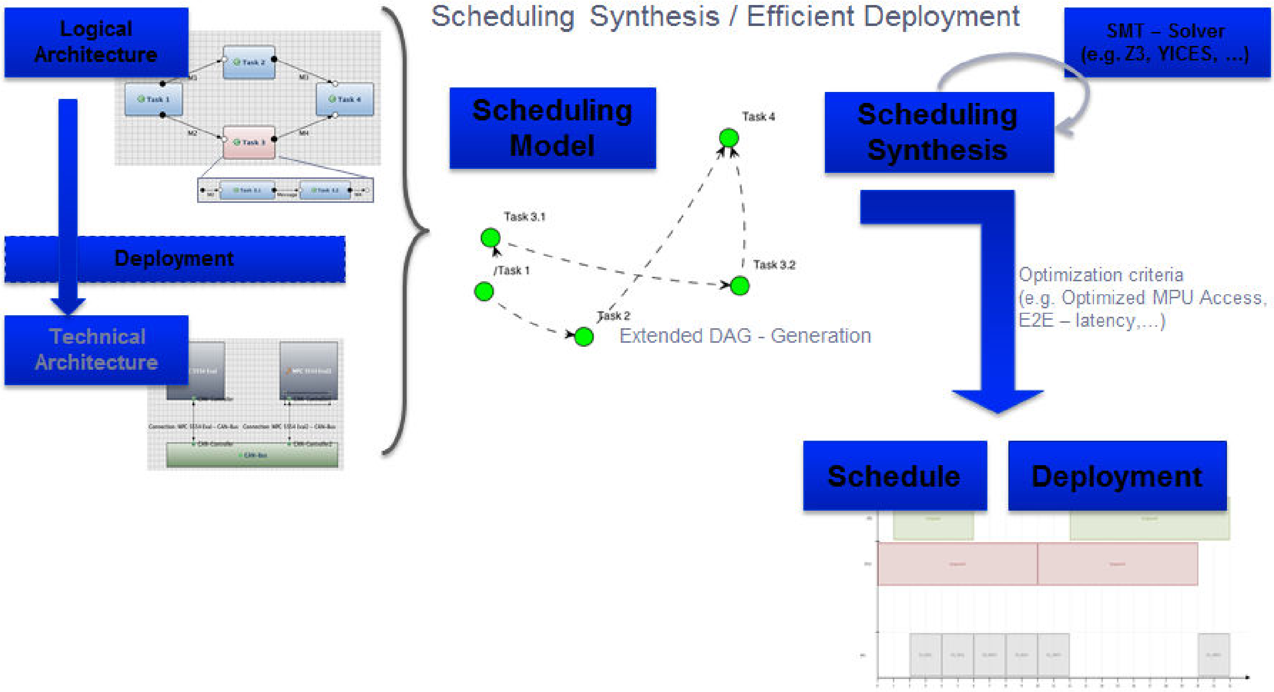

Figure 10: Deployment Synthesis in AF3 [11]

Using this approach we provide an efficient deployment for multi-criteria problems (e.g. timing, scheduling) as well as calculate (optimized) partitioning and mapping of systems according to ASIL levels in a mixed-criticality environment, and has been developed in the context of the SPES_XT Core project [12].

As shown in Figure 10, the ASIL levels, which are propagated through the component links on the logical architecture, provide one criterion and we can freely select other criteria such as execution time, energy consumption or any other resource optimization.

The deployment synthesis is based on rules definition carried out inside the solver in AF3, and is further explained in [10].

7.1 Initializing the Design Space Exploration Plug-In

AutoFOCUS3 supports an approach for supporting the system designer by generating a deployment efficiently that is the allocation of logical components to platform architecture components. This can be done according to certain criteria like timing, safety constraints and memory consumption [9].

The Design Space Exploration Plug-In can be added to any AF3 component model as seen in the following example of an Adaptive Cruise Control [9], shown in Figure 11.

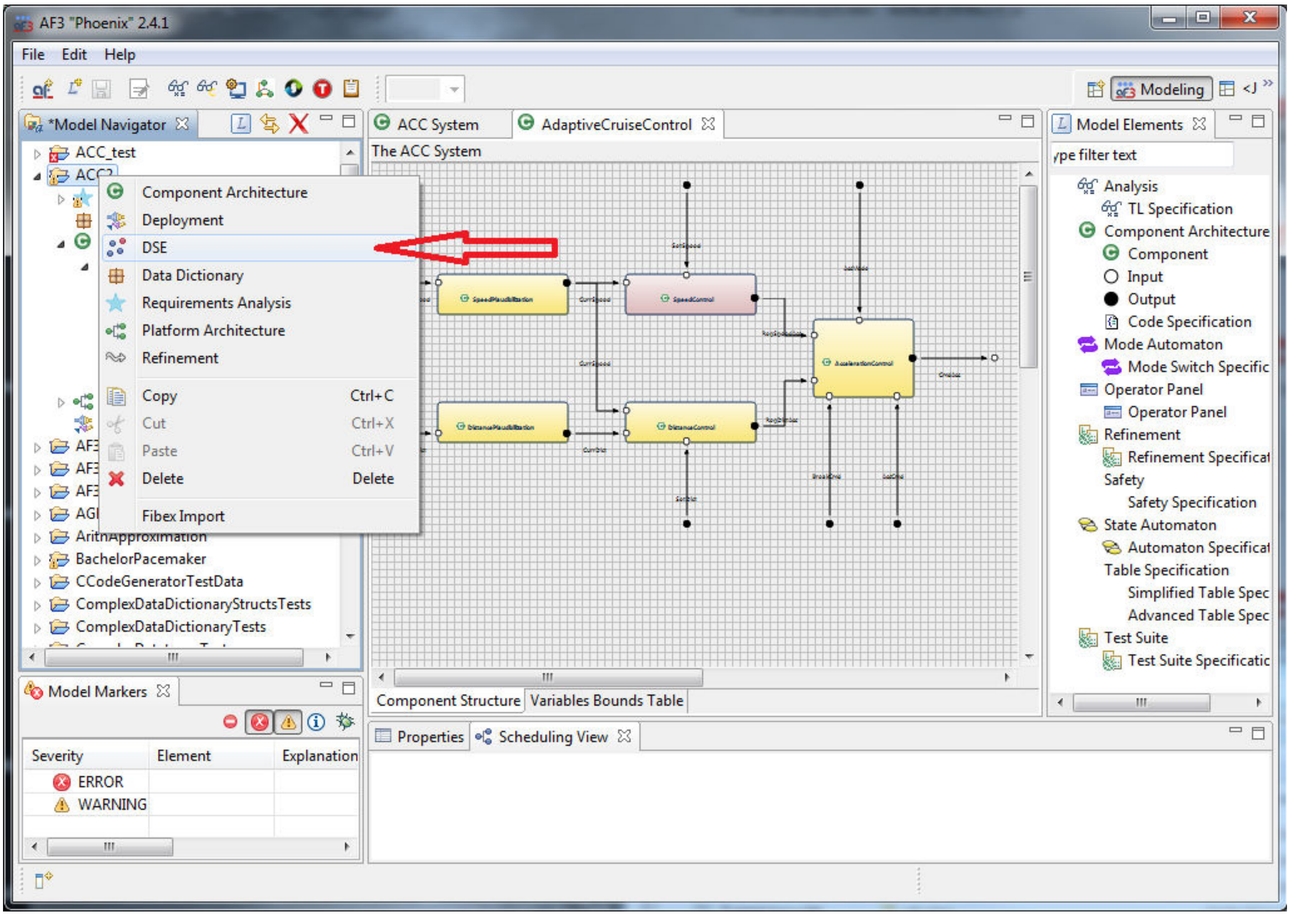

Figure 11: Design-Space-Exploration Plug-In in AF3

Selecting the Plug-In opens a user interface, seen in Figure 12, which allows for the generation of suitable deployments according to multiple optimization criteria. It is then possible to select a subset from the generated deployments and subsequently proceed to generate suitable schedules [9].

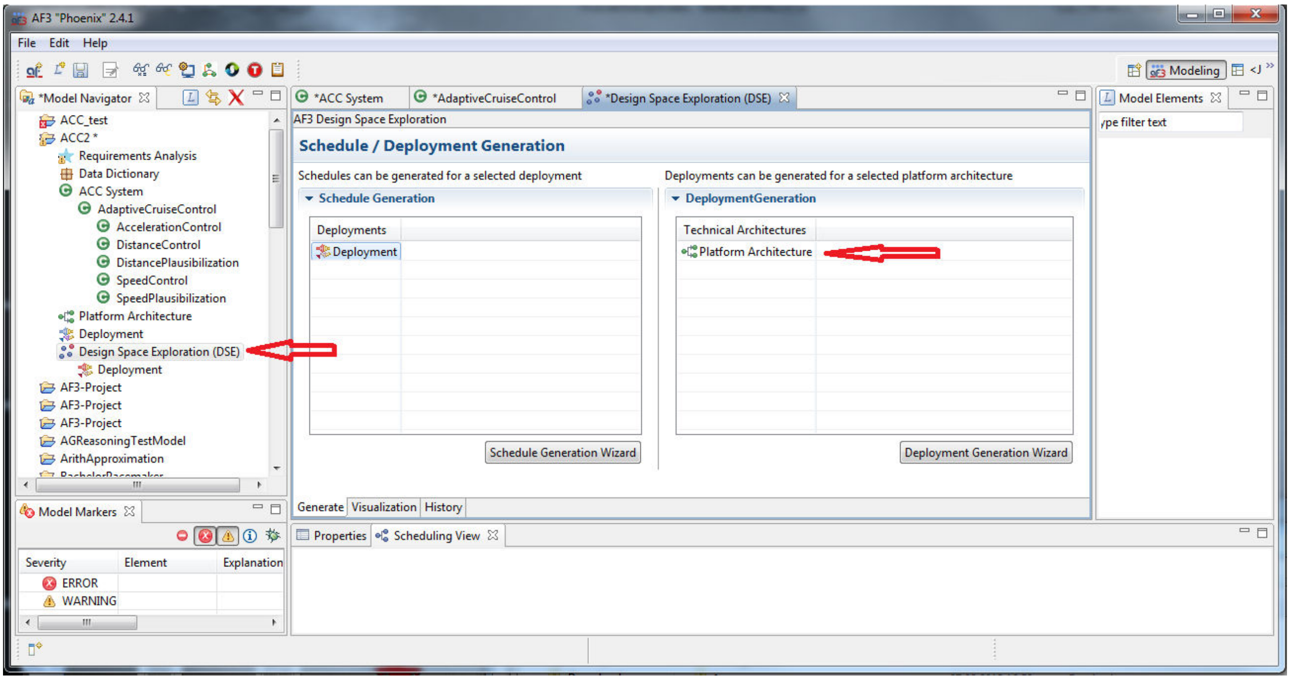

Figure 12: DSE User Interface in AF3

7.2 Multi-criteria Deployment Optimization

Starting the Deployment Generation Wizard from the DSE User Interface a window appears that automatically lists all the components in the model and allows to select the component and component level for which the deployment should be generated, as well as selecting a filter mechanism, as shown in Figure 13.This feature has been developed in the context of the SPES XT project [12].

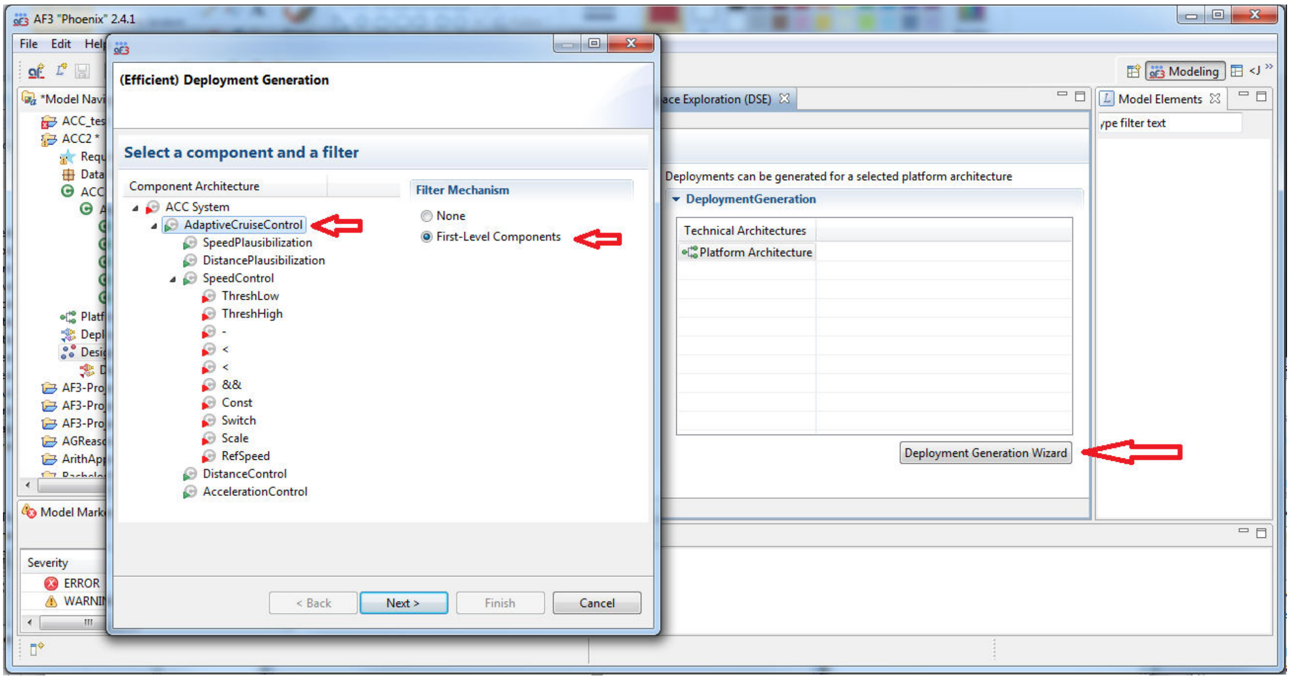

Figure 13: Component Selection for Deployment Generation in AF3

The next window allows the selection of the criteria and constraints of the design space, which will be used for the deployments generation. It is possible to use standard constraints, such as Safety Integrity Levels (SIL), as well as to employ user-defined constraints, such as Memory Consumption or any other constraint the user can express. The user then has the choice to simply generate all deployments which constitute valid solutions to the design space confined by the selected constraints, or to generate an optimized solution space, according to other criteria which can be defined, such as End-to-End (E2E) Latency or the allowable number of hardware resources (ECUs) and so on, as shown in Figure 14.

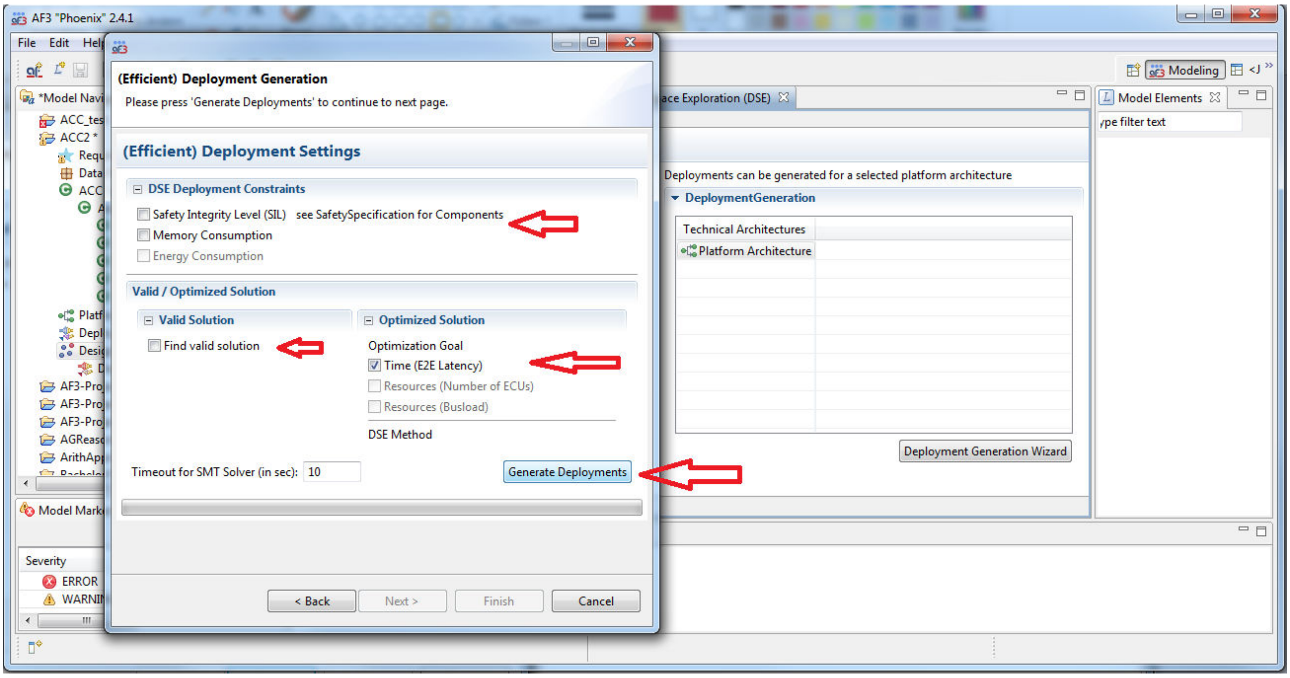

Figure 14: Selecting Design Space Constraints for Deployment Generation in AF3

The generation of the deployment leads to a list of viable configurations, constituting the design space of the possible deployments. It is then possible to select all or only a subset of these possible solutions and save them for use in the scheduling synthesis, as shown in Figure 15 [9].

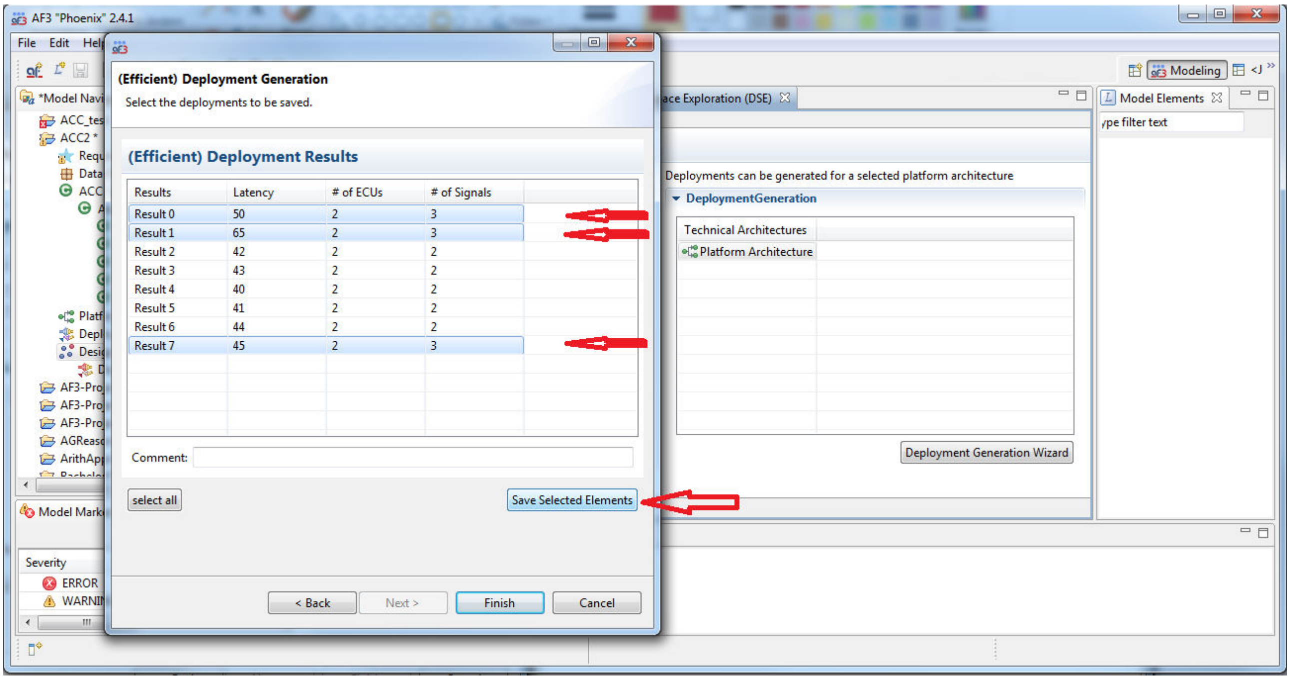

Figure 15: Selecting viable Deployment Design Solutions for Scheduling Synthesis in AF3

7.3 Scheduling Synthesis

The user can then proceed to generate the Schedule based on any given deployment and is again given the option to select the component level for performing the synthesis as well as the filter used, as shown in Figure 16. The constraints for the scheduling problem to be solved are described in detail in [10].

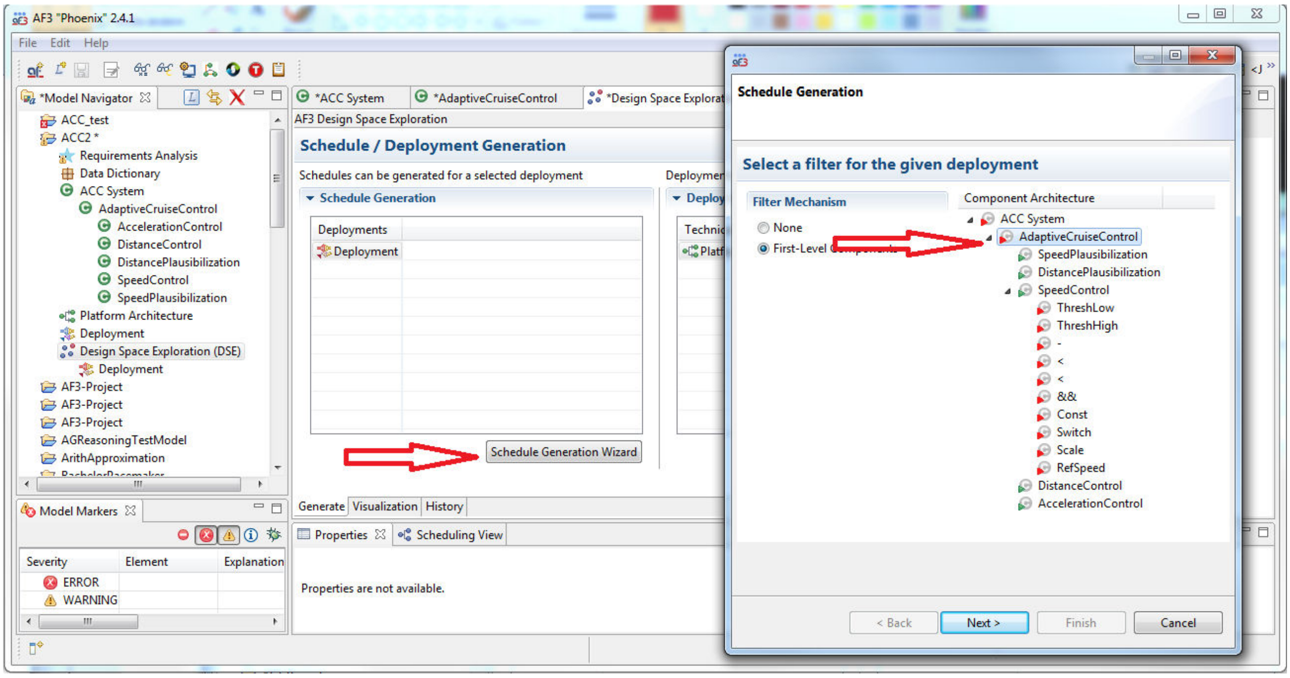

Figure 16: Component Selection for Deployment Generation in AF3

The next window allows the selection of the criteria and solver used for the design space, which will be used for the scheduling synthesis.

It is possible to use any solvers integrated into AF3, such as Z3 [13] or YICES [14]. The user then has the choice to simply generate all deployments which constitute valid solutions to the design space confined by the selected constraints, or to generate an optimized solution space, using multiple methods, as shown in Figure 17.

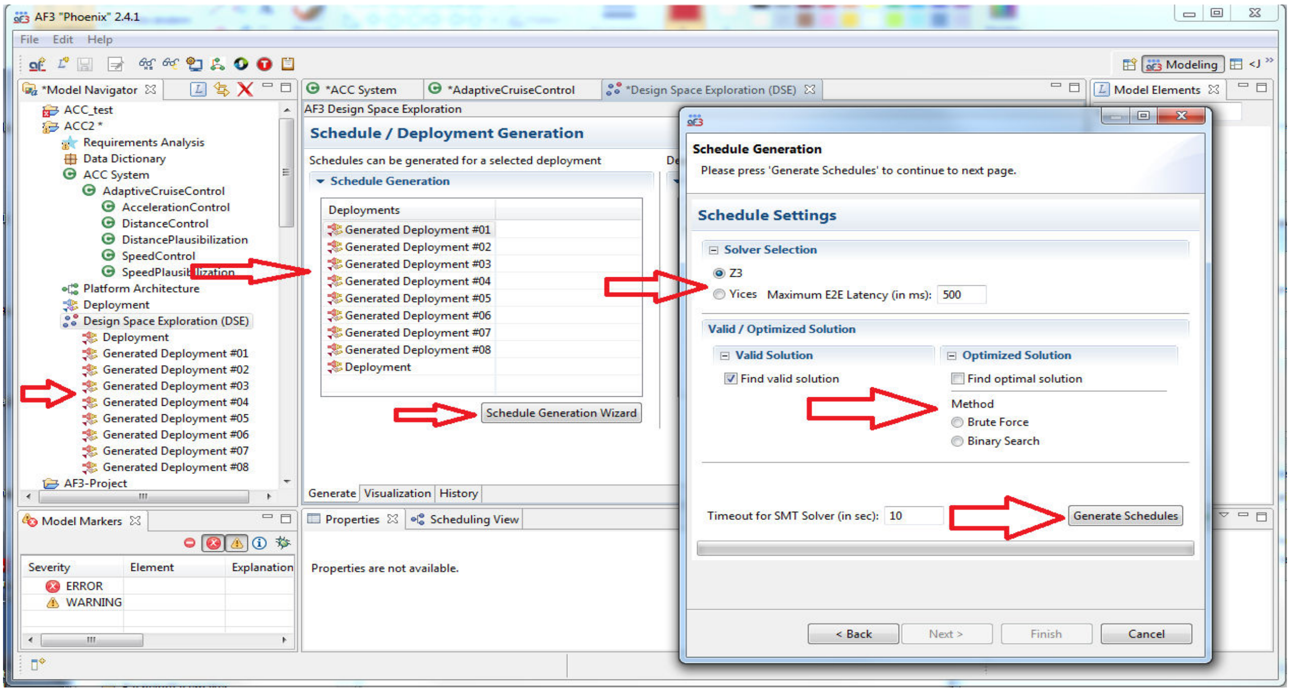

Figure 17: Selecting Design Space Constraints for Scheduling Synthesis in AF3

The generation of the schedules leads to a list of viable configurations for each deployment, constituting the design space of the possible deployments. It is then possible to select all or only a subset of these possible solutions and save them for use in the scheduling synthesis [9], as shown in Figure 18.

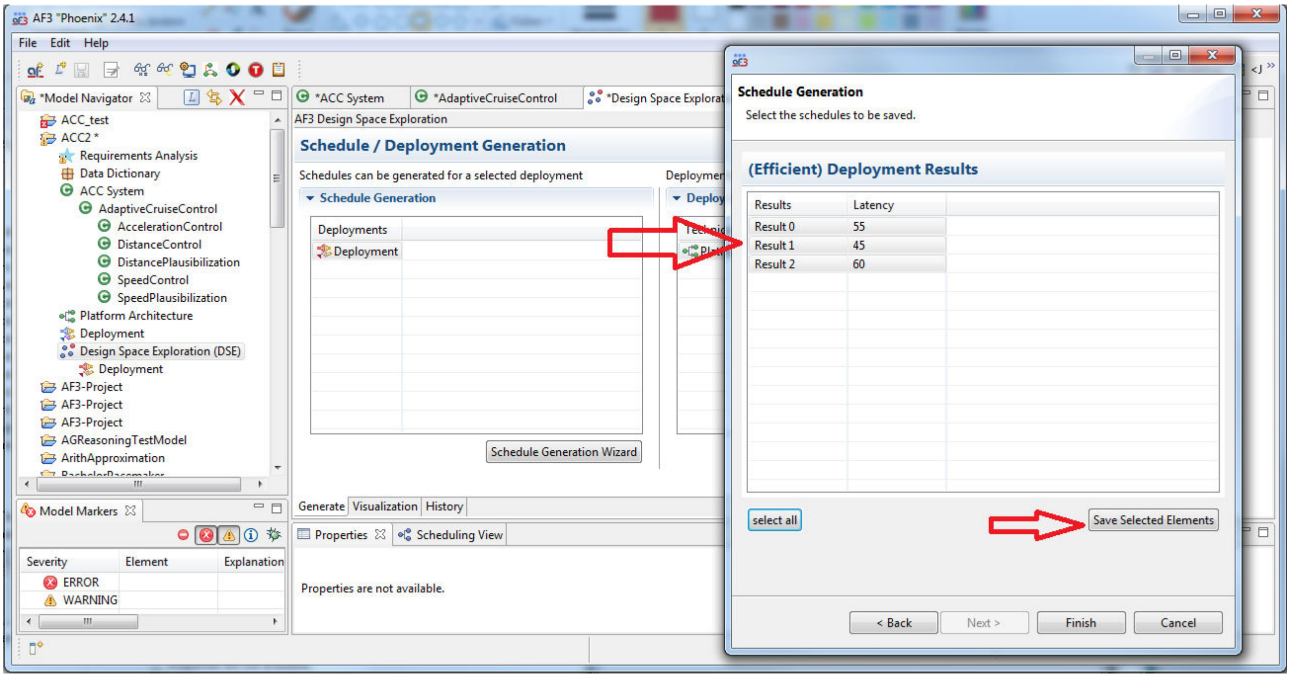

Figure 18: Selecting viable Scheduling Solutions for Design Space Exploration in AF3

7.4 Design Space Exploration

Returning to the Design Space Exploration View after having synthesized schedules for the previously generated deployments, it is now possible to visualize the optimized design space and manipulate it, using the Visualization Tab in the main window, as shown in Figure 19. This work is currently under development and has been submitted in [15]. The presented parts are a result of work carried out in the context of the ARAMiS project [16].

It is possible to assign different optimization criteria to the design space axes and visualize the representation. Slect each solution node, shown as black dots, will display the information related to that deployment design as well as visualize the corresonding schedule in the Scheduling View tab under the main modelling window, as shown in the figure.

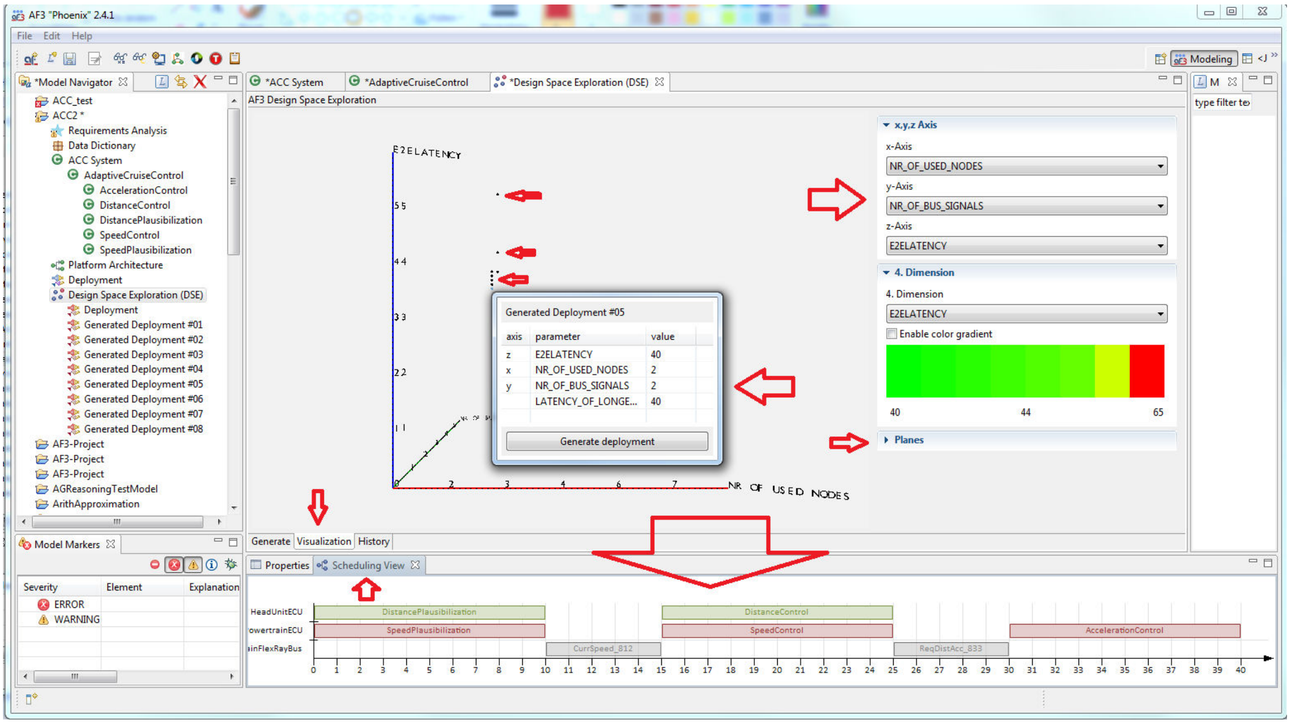

Figure 19: Visualization of Design Space Exploration in AF3

To visually manipulate the design space, it is then possible insert a fourth dimension by coloring the design solutions, e.g. according to the E2E Latency. This is done by selecting an optimization variable and then enabling the color gradient option, as shown in Figure 20. This leads to the coloring of the solution points in the design space accordingly which facilitates a visual understanding of the design space.

Next, it is possible to start solving the design space by constraining for any axis value using the "Planes" feature, where the following figure shows a plane threshold for E2E Latency for solution points with a value, e.g., above 42msec, set by manipulating the corresponding slider. The solution points not satisfying this criteria are no longer colored but grayed out, as shown in the figure, facilitate visual identification of the impact of the constraint on the design space.

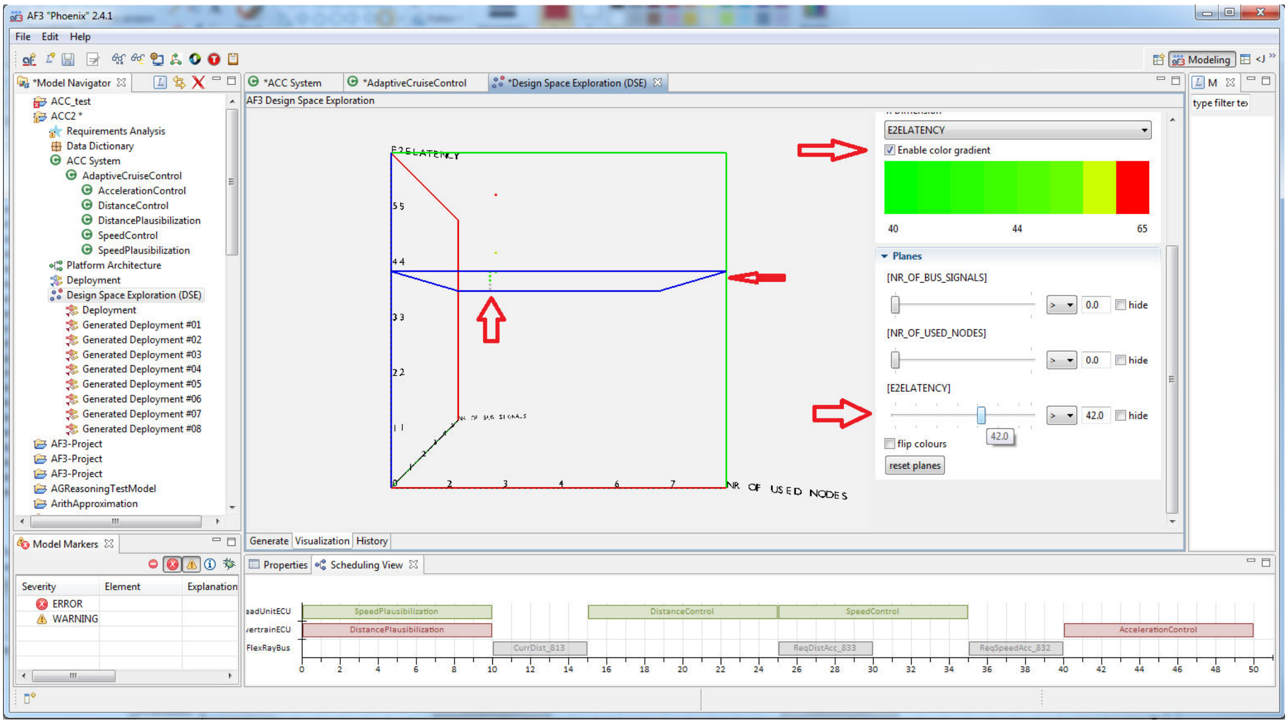

Figure 20: Graphical Manipulation of the Design Space in AF3

Finally it is possible to manually adjust the viewing angle of the design space in all directions as well as zooming in or out. Combined with solution planes for all dimensions, as shown in Figure 21, this provides a powerful visualization which helps in understanding and solving the design space leading to a multi-criteria optimization of the architecture [9][15].

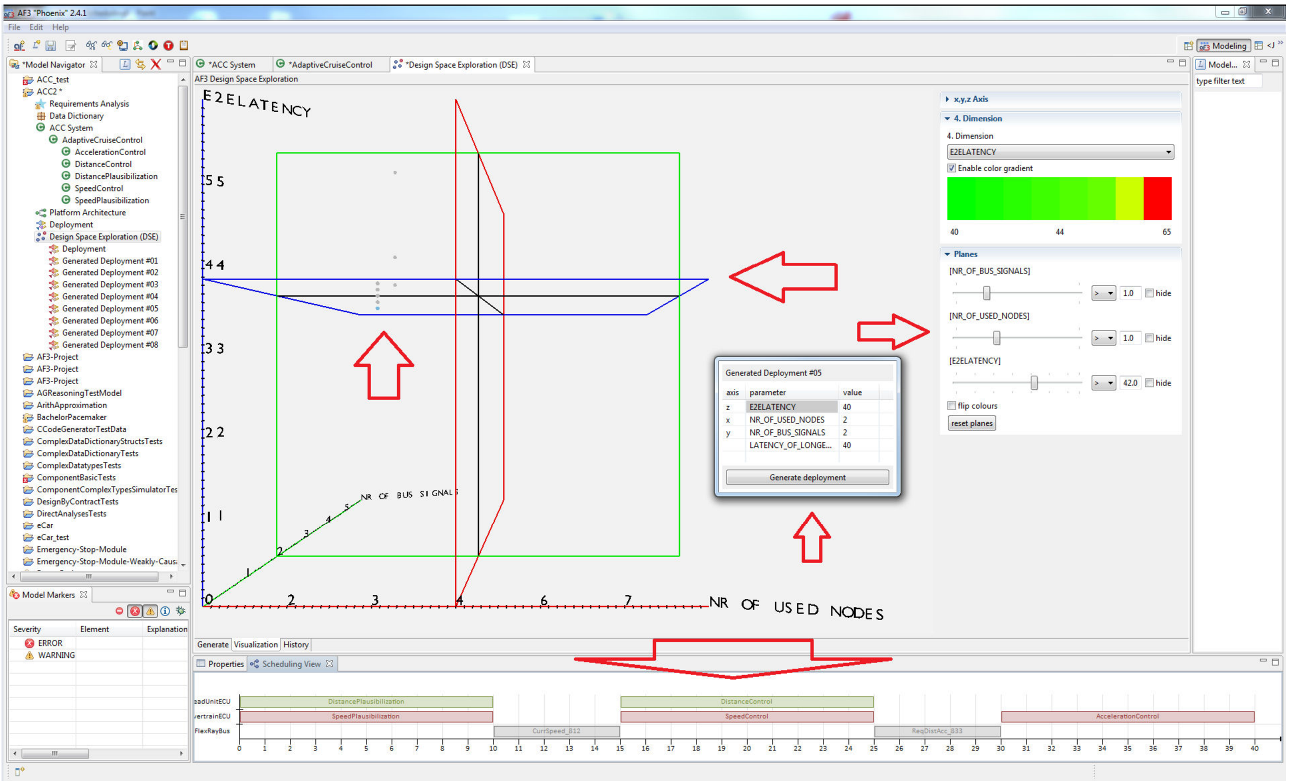

Figure 21: Multidimensional Manipulation of Design Space Visualization in AF3

8 Safety Constraints for schedule and deployment generation

In the previous chapter, we discussed a methodology representing joint generation of schedules and deployment for mixed criticality multicore architectures using shared memory and how this approach is integrated into the AUTOFOCUS3 tool-chain [9]. We now present a methodology that integrates safety metrics identified in the SAFE/SAFE-E project, such as hardware failure metrics, SIL compliance, and real time resource constraints, such as memory usage, power consumption, etc., into the DSE (Design space exploration) plugin of the AUTOFOCUS3.

AUTOFOCUS3 currently features three major abstraction layers: a requirements engineering view, a logical architecture, and a technical architecture [17]. We are concerned with the latter two in this document, specifically with the mapping of logical components onto hardware architectures, and the flip-side of that coin, i.e. the analysis of logical architectures for their suitability or adherence to different conditions and constraints. The logical architecture defines a model of the system under development from an abstract point of view (as a set of communicating components) while the technical architecture describes the execution environment of the system by means of ECU (execution control units) and buses. Prior to this work, there was no support for generating deployments (mapping logical architecture to technical architecture) in AUTOFOCUS3 that takes safety, resource and cost constraints into account.

The remaining chapter is organized as follows. In section 8.1, we talk about the motivation behind the work. Section 8.2 talks about the different safety, resource and cost constraints in detail. And finally in section 8.3 we do an analysis of results of using these constraints in the deployment generation process.

8.1 Motivation

Mapping logical components to the hardware architecture is not just a simple task of mere allocation and de-allocation, but to generate a deployment that satisfies different constraints, be it related to safety, resource or cost.

To illustrate it in a comprehensive way, let us consider different scenarios. One such scenario is the random hardware failure in hardware elements. As mentioned in SAFE-E Deliverable D3.5.b, "the hardware architecture of automotive systems needs a high reliability especially in context of random hardware failures. These failures occur unpredictably during the lifetime of electric systems due to, exemplarily, aging effects and can never be avoided" [18]. These random hardware failures can be assessed in two different ways. ISO26262 clause 8, "Evaluation of the hardware architectural metrics" describes the assessment of hardware architectures by applying hardware architectural metrics and ISO26262 clause9, "Evaluation of the safety goal violations due to random hardware failures" describes the assessment of hardware architectures in terms of residual risk of safety goal violation [1]. We presented a method that takes into account such random hardware failures while generating the deployments. For the purpose of ease of representation, we have decided to go with the "residual risk of safety goal violation" option. The evaluation of residual risk of safety goal violation can be performed using one of the alternative methods, the probabilistic metric for random hardware failure (PMHF) or the second method using failure rate classes (FRC).

The second such scenario is generating a deployment that reduces the overall power consumption or cost of the hardware architecture on which the logical component are deployed. Different SILs reflect different requirements stringencies and consequently different development costs. The allocation of safety requirements is not a simple problem of applying an allocation "algebra" as treated by most standards; it is a complex optimization problem, one of finding a strategy that minimizes cost whilst meeting safety requirements [19]. We propose a method that generates a deployment reducing the overall cost of the hardware architecture.

Other scenarios include generating deployments without violating the memory constraints of the hardware nodes and reducing the overall power consumption of hardware architecture. We will focus on each of those constraints in more detail in section 8.2.
Hence, extending deployment and schedule generation of AUTOFOCUS3 with different safety, resource and cost constraints will enable us model complex embedded systems in much more effective and efficient manner.

8.2 Safety and cost constraints

This section talks about the different safety and cost constraints that are taken into consideration while generating the deployments and schedules of mixed criticality application. We have presented a novel approach to deployment generation by using a formalization that describes the scheduling problem as a satisfiability problem using Boolean formulas and linear arithmetical constraints. We used the Z3 solver, a state-of-the art theorem prover from Microsoft Research. It can be used to check the satisfiability of logical formulas over one or more theories [13].

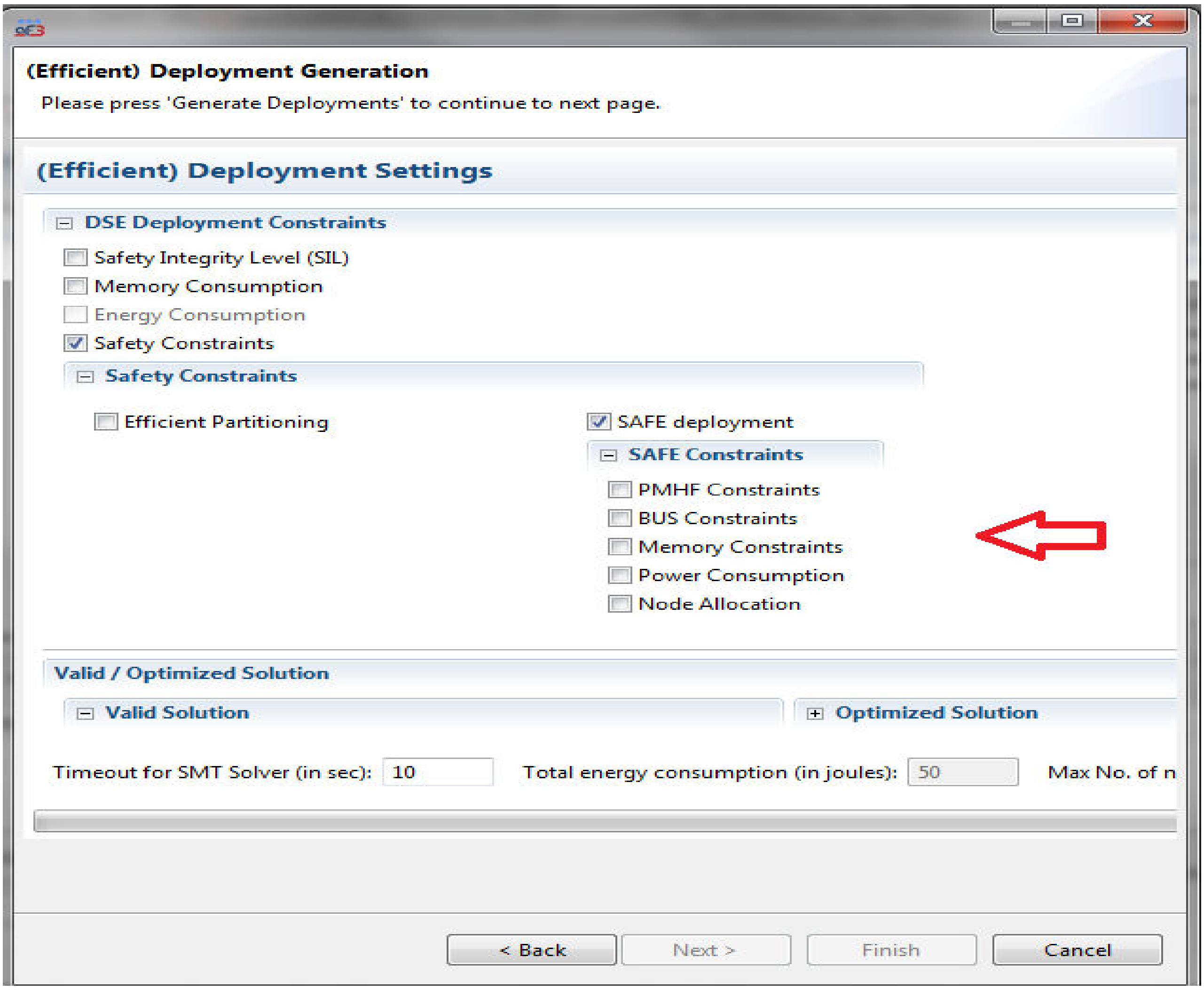

Figure 22: SAFE deployment constraint of Design Space Visualization in AF3

Figure 22 shows the new deployment constraints that are now supported. These include SIL constraint for ECU and BUS, Memory Constraints, Power Consumption and Maximum number of nodes allowed constraint. Each of these constraints is discussed in detail below.

8.2.1 PMHF Constraint (ASIL constraint for ECU)

The probabilistic metric for random hardware failure (PMHF) describes an overall probabilistic value for a top-level system failure. It is related to the probability of dangerous failure per hour (PFH) and probability of dangerous failure on demand (PFD) of IEC61508. The difference between PFH and PFD is explained in more detail in SAFE-E Deliverable D3.5.b [18]. The way a PMHF value is calculated is also explained in the deliverable. The following table shows the recommended target PMHF values for different ASIL classes.

ASIL	Random hardware failure target values (PMHF)
ASIL-D	$< 10^{-8}h^{-1}$ (*equal to* $10\ FIT$)
ASIL-C	$< 10^{-7}h^{-1}$ (*equal to* $100\ FIT$)
ASIL-B	$< 10^{-7}h^{-1}$ (*equal to* $100\ FIT$)

Table 1: ASIL and PMHF mapping

According to ISO 26262, PMHF values are represented in units of FIT (failure in time). 1FIT means 1 failure in a billion years. However, the focus here is not on the representation of PMHF values, but on how the random hardware failures can be used as deployment constraints. So for the ease of representation we have come up with our own target values. These are shown in the following table.

ASIL	Random hardware failure target values (PMHF)
ASIL - D	0.00 – 0.25
ASIL – C	0.26 – 0.50
ASIL – B	0.51 – 0.75
ASIL – A	0.76 – 0.99
QM	1.00

Table 2: Alternate ASIL and PMHF mapping used in AF3

We have created PMHF values as annotation in AUTOFOCUS3. The PMHF values are associated with the Execution Unit (ECU) and Transmission Unit (BUS). Figure 23 shows an example where Generic_ECU_1, Generic_ECU_2 and Generic_Bus have a PMHF of 0.4, 0.6, and 0.6 respectively. In other words they fall under ASIL-C, ASIL-B and ASIL-B category respectively.

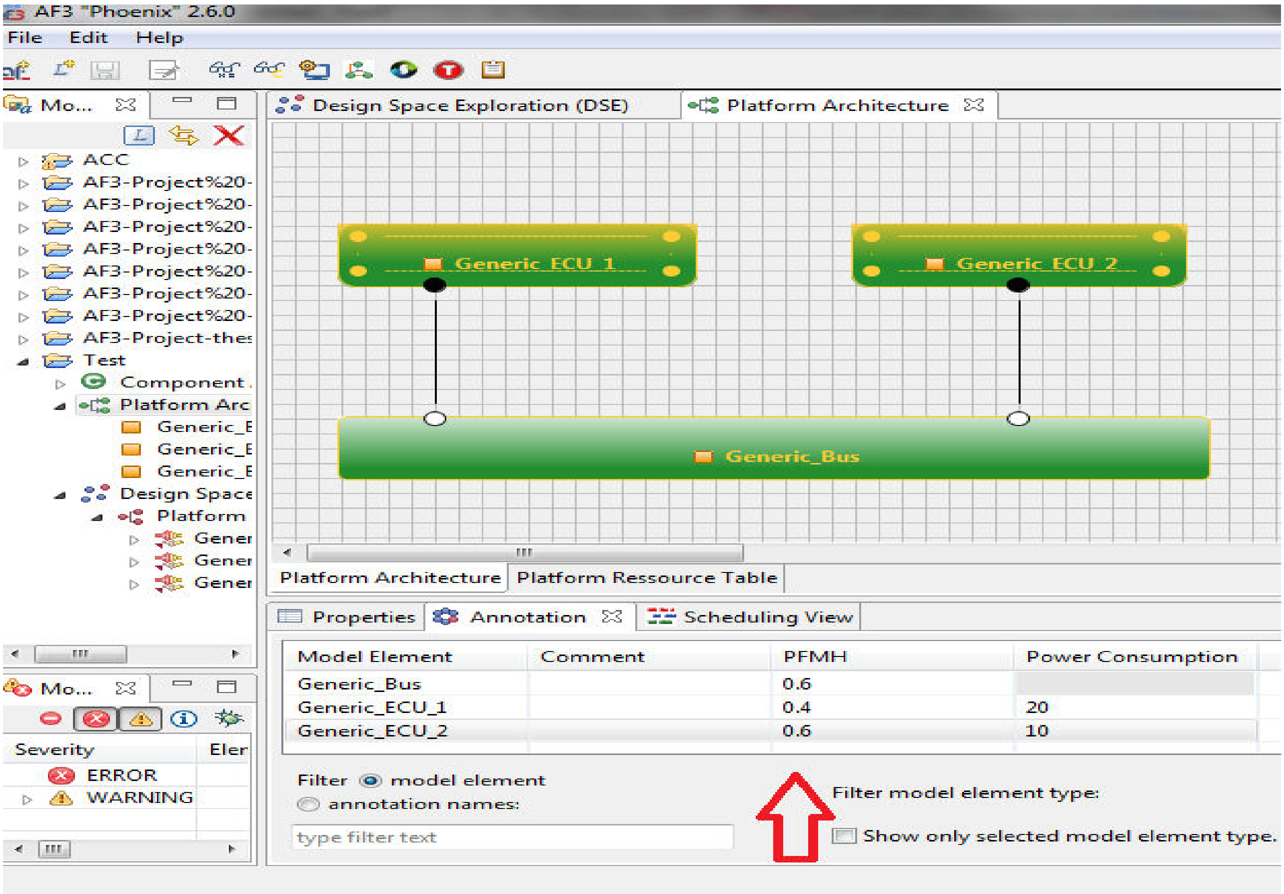

Figure 23: PMHF Annotations associated with ECUs and Bus

AUTOFOCUS3, on the other hand, allows adding a "Safety Specification" model element to each component in the component architecture. This safety specification is a list of values ranging from QM to ASIL-D. For e.g., in Figure 24 component t2, t4, t5, t6, t3 are ASIL-C, ASIL-C, ASIL-B, ASIL-A, ASIL-A compliant respectively.

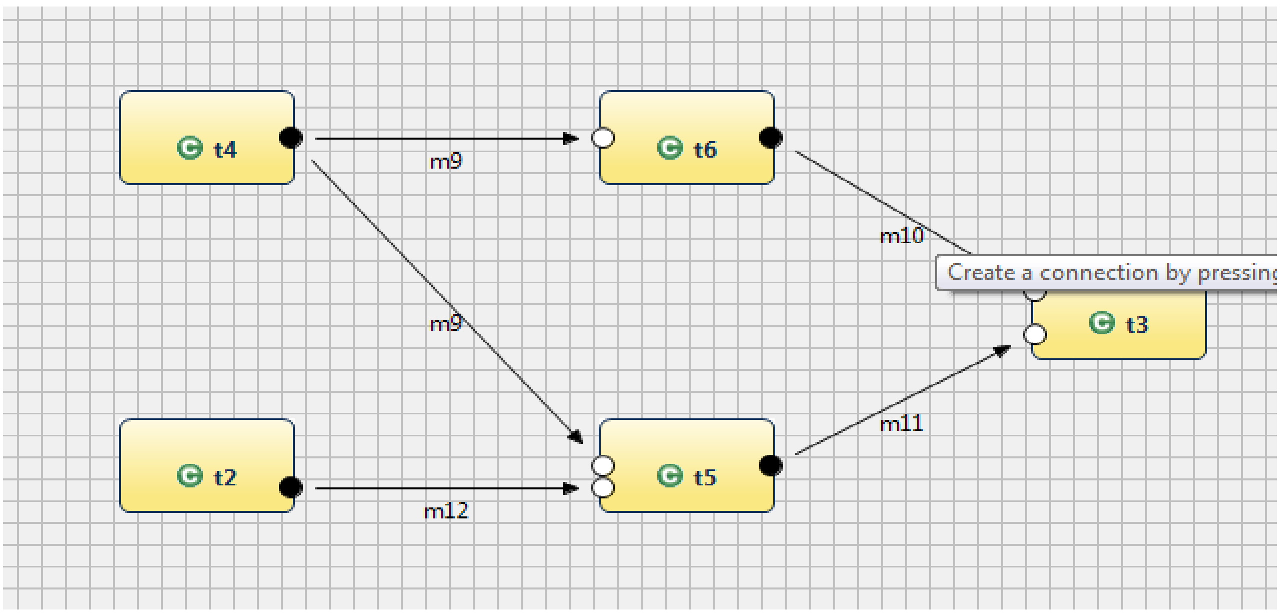

Figure 24: A SIL compliant component architecture

PMHF Constraint:

Let T be the set of tasks 'n' tasks i.e. T = {t_1, t_2, …., t_n} and N be the set of 'm' nodes i.e. N = {n_1, n_2, …., n_m}. We define a function 'ASIL' that takes as input a task or a node and gives its ASIL value as the output, i.e. ASIL: X -> Y where $X \in T \cup N$ and $Y \in$ {QM, ….., ASIL-D}.

A more stringent logical component needs to be deployed to a node with less probability of random hardware failures in order to generate a safe and reliable deployment [18]. Hence PMHF constraint ensures that a task (logical component) is deployed to a node with equal or more stringent ASIL value.

i.e. $\quad ASIL(T) \leq ASIL(N)$

For e.g. an ASIL-C compliant component can be deployed to a hardware node that is either ASIL-C or ASIL-D compliant.

Task t4: Generic_ECU_1

Task t2: Generic_ECU_1

Task t6: Generic_ECU_1, Generic_ECU_2

Task t5: Generic_ECU_1, Generic_ECU_2

Task t3: Generic_ECU_1, Generic_ECU_2

8.2.2 BUS Constraint (ASIL constraints for BUS)

In the previous section we presented a constraint that takes into account the probability of random hardware failures (PMHF) of hardware nodes while generating a deployment. However, an application consisting of multiple components deployed on different ECUs needs to communicate with each other. Since the communication goes over a bus, it makes sense to take reliability of bus into consideration as well. Similar to the ECUs, the reliability of BUS is also measured in terms of PMHF values. Table 2 is used for the mapping of PMHF to ASIL values.

BUS Constraint:

A reliable communication is ensured when a component receives a message over a bus that is at least as reliable as the receiver itself. The BUS constraint, hence, ensures that for each message, if the sending and the receiving component are deployed on the different ECUs, the ASIL of the bus is at least as stringent as the ASIL of the ECU on which receiving component is deployed.

Let 'S' be the sender and 'R' be the receiver. Let 'm' be the message sent from S to R. Let 'B' be the bus over which the message is transferred. Then:

$$ASIL(R) \leq ASIL(B) \qquad \text{if } S\_node \neq R\_node$$

This constraint needs to hold for all the messages that are exchanged between the logical components.

respectively, the following deployments are possible assuming the BUS is only ASIL-A compliant.

Task t4: Generic_ECU_1

Task t2: Generic_ECU_1

Task t6: Generic_ECU_1, Generic_ECU_2

Task t5: Generic_ECU_1

Task t3: Generic_ECU_1, Generic_ECU_2

Note the difference between the deployments generated in this section and the previous section. Since the BUS is ASIL-A compliant, message ‘m9’ from task ‘t4’ to ‘t5’ is not allowed to travel over the bus because the receiving component i.e. ‘t5’ is ASIL-B compliant. Similar case holds for message ‘m12’. Hence the BUS constraints forces tasks ‘t4’, ‘t2’, and ‘t5’ to be deployed on the same ECU i.e. Generic_ECU_1.

8.2.3 Memory Constraint

So far we have focused on safety constraints. However there might be a case where many components are deployed on the same node and the node runs out of memory. Such constraints fall under the category of resource constraints. One such resource is memory per node which we present as a new deployment constraint in DSE plugin of AF3. Figure 25 shows how an ECU can be assigned a specific amount of memory in AF3.

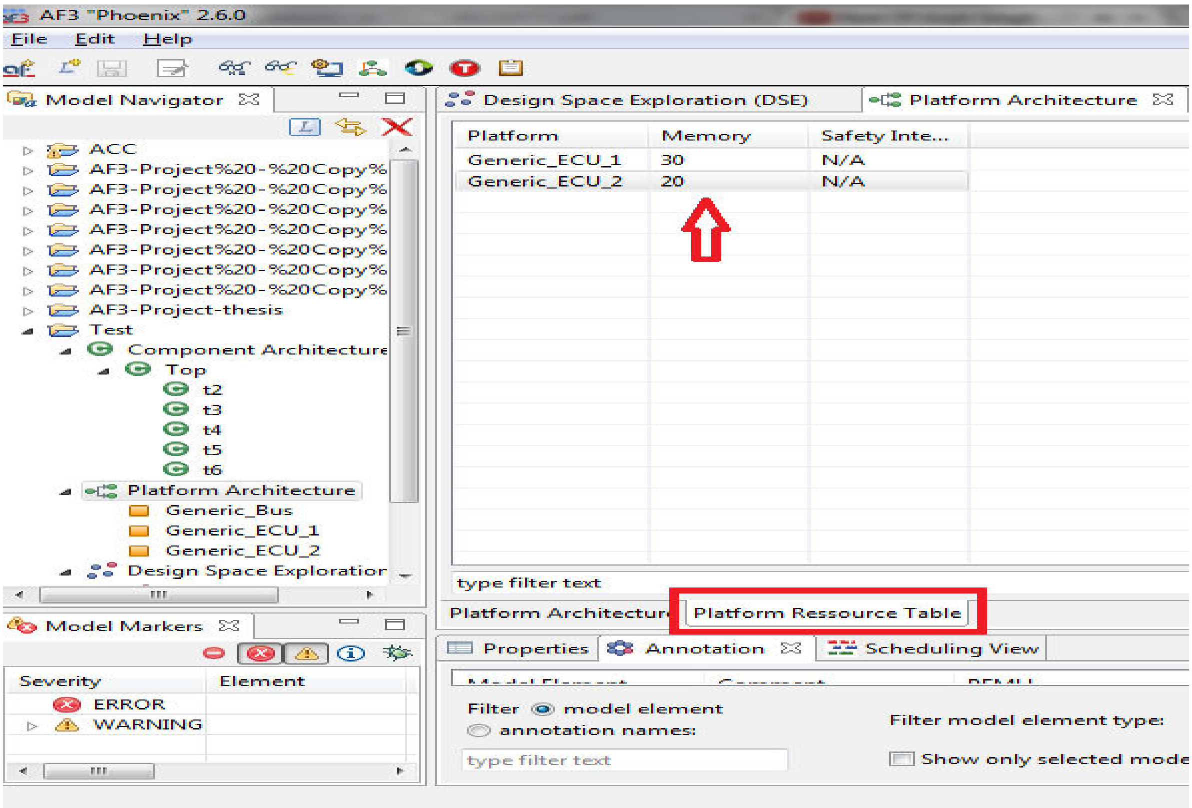

Figure 25: Memory Allocation in Platform Resource Table

Each ECU can be allocated a specific amount of memory using the Platform Resource Table in the Platform Architecture view of an AF3 project.

Figure 26 shows how each component in the component architecture can be allocated a memory by using the Component Resource Table in the Component Architecture view of an AF3 project.

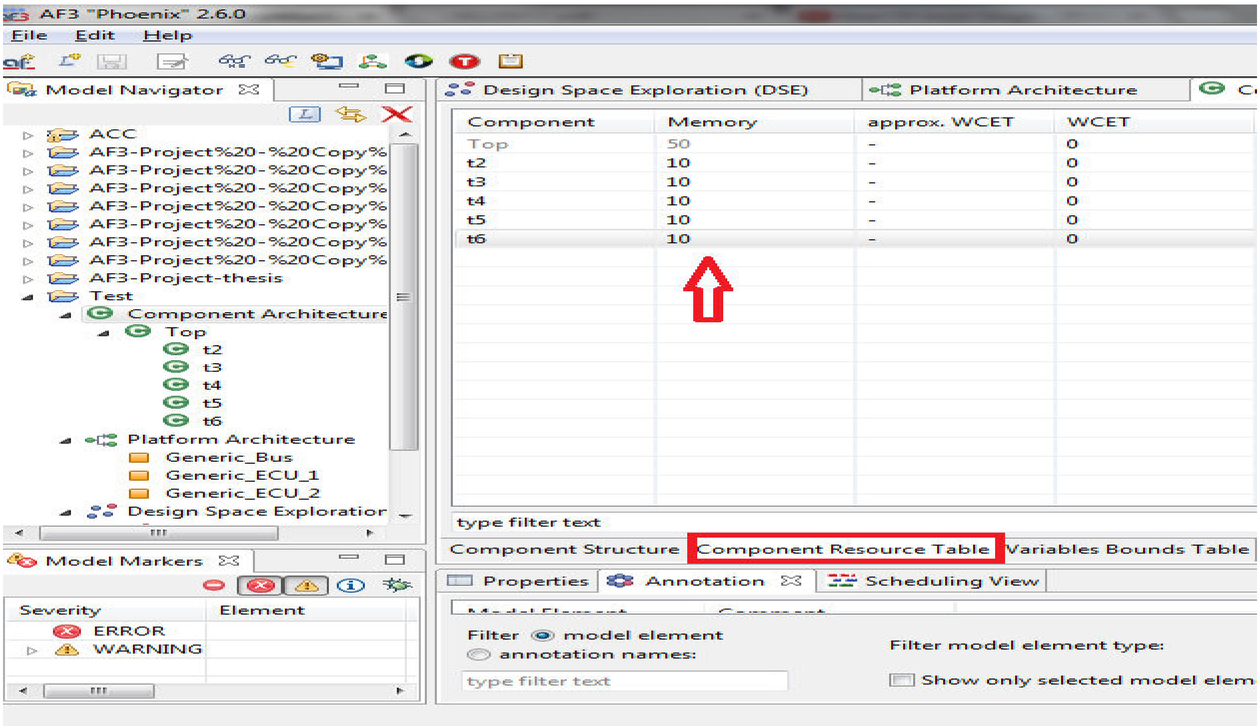

Figure 26: Memory Allocation in Component Resource Table

Memory Constraints:

Let us define two attributes for the element 'Node'. 'ram', which is the total amount of memory allocated to the ECU (platform resource table) and 'used_Memory', which is the total amount of memory used by all the tasks deployed onto the ECU. A task has an attribute 'mem', which is the total amount of memory needed by the task itself.

The purpose of memory constraint therefore is to ensure that none of the ECU's runs out of memory, which is explained by the notation below.

$$\forall n \in N : n\_used\_memory \leq n\_ram$$

$$\text{where: } n\_used\_memory = \sum_{t_i\_node=n} t_i\_mem$$

The Component Resource Table in Figure 26 shows that each component requires a memory of 10 units. The Platform Resource Table in Figure 25 shows that Generic_ECU_1 has a RAM of 30 units and Generic_ECU_2 has a RAM of 20 units. So the memory constraint ensures that a maximum of 3 logical components can be deployed on Generic_ECU_1 and a maximum of 2 logical components can be deployed on Generic_ECU_2.

8.2.4 Maximum Number of Nodes constraint

This is the second kind of resource constraint that ensures that the generated deployment do not use more number of nodes than the one given by the user. Its main purpose is to avoid the unnecessary usage of the nodes. This constraint is fed into the SMT (satisfiability modulo theorem) file and the z3 solver finds whether a deployment is feasible or not. Figure 27 shows the input field where the user can give maximum number of nodes to be used while generating the deployments.

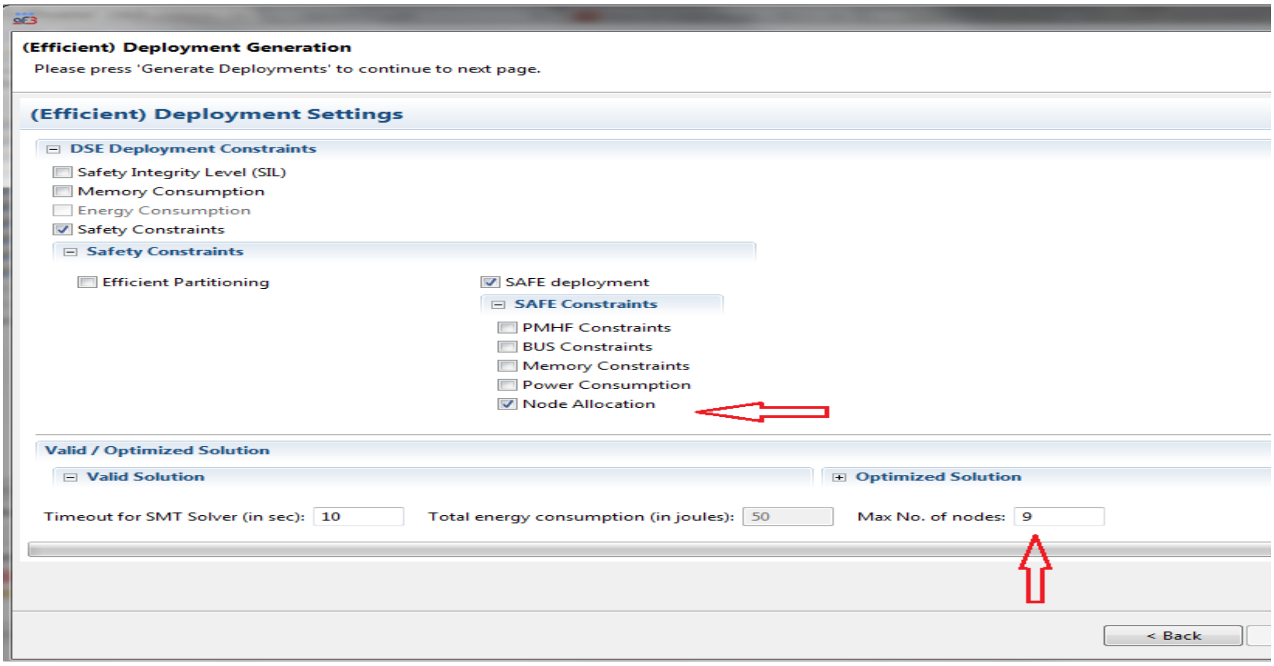

Figure 27: Maximum number of nodes constraint

Number of nodes constraint:

Let us define for each task in T a property named 'node', which defines the node onto which the task is allocated. Let there be a Boolean property for each node in N, named 'used', which indicates whether there is at least one task allocated to it. The following set of equations defines how this constraint works.

$$total\_used\_nodes \leq max\_nodes$$

Here, max_nodes is the maximum number of allowed nodes that the user gives as a constraint for deployment generation. The formula for calculating the total number of nodes is given below.

$$total\_used\_nodes = \sum_{n}^{N} n\_used$$

'Used' is a Boolean property that each node possesses, and is defined as follows.

$$n\_used = \begin{cases} 1 & \exists \text{ t} \in \text{T : t\_node} = \text{n} \\ 0 & \text{otherwise} \end{cases}$$

To define this constraint in a simple manner, we iterate over all the nodes and for each node we check if there is at least one task allocated to it. If yes, this node is categorized as a used node, otherwise, an unused node. Then we count all the used nodes and this number should be less than the number given by the user as input.

Referring to the architectures in the Figure 23 and Figure 24 and assuming that all the constraints mentioned above are taken into consideration, there can be no solution possible if the user inputs maximum number of nodes as 1. Let's understand this scenario. The following table lists the resources available per element.

Element	ASIL Value	Memory
Generic_ECU_1	ASIL-C	30
Generic_ECU_2	ASIL-B	20
BUS	ASIL-B	N/A

Table 3: List of resources available per HW node

Going with the results of the previous 3 constraints, the only possible deployment is that tasks ‘t2’, ‘t4’ and ‘t5’ are allocated to Generic_ECU_1 and tasks ‘t3’ and ‘t6’ are allocated to Generic_ECU_2. If the user wants to allocate all the tasks on a single ECU, one way is to relax the memory constraint of the ECUs. Increasing memory of Generic_ECU_2 will do no good, since tasks ‘t2’ and ‘t4’ are ASIL-C compliant and cannot be allocated to an ASIL-B node. The only option in that case is to increase memory of Generic_ECU_1 so that it can accommodate all the components.

However this example does not make much sense when we see the complexity and functionality of the modern embedded systems. But it is quite useful for elaborating how improving one constraint can degrade other deployment criteria, a concept known as Pareto-efficiency. For e.g. restricting the number of nodes might lead to an increase in end-to-end latency, due to reduction in parallelism.

8.2.5 Cost constraint

Developing a component according to a given ASIL means that a set of development and validation activities needs to be undertaken. They are translated into time, efforts and in the end costs, which vary with the specific ASIL prescribed to a component [19]. There can be many deployments possible that satisfy safety constraints and resource constraints, but in order to find the most desired and advantageous one, the problem needs to consider different cost implications.

To illustrate this fact, let us consider a simple example. A function with a safety goal (SG) that requires ASIL B can be implemented with an architecture of two components which, assuming that they fail independently, may inherit ASILs B and QM or A and A respectively. When such options exist, cost typically provides the deciding criterion. Assuming a simple cost heuristic which has a logarithmic scale (i.e. ASIL QM=0; A=10; B=100; C=1000; D=10000), the following are the possible decompositions with the second one optimizing the cost.

C1 (ASIL QM) + C2 (ASIL B): 0 + 100 = 100;

C1 (ASIL A) + C2 (ASIL A): 10 + 10 = 20;

C1 (ASIL B) + C2 (ASIL QM) = 100.

This ASIL Decomposition example directs towards the fact that how a cost heuristic can impact finding an optimal solution from multiple feasible ones. This is analogous to our case where cost is considered as a constraint while generating a deployment, respecting all the other constraints. We have divided cost constraint into two sub parts, i.e. power consumption and cost factor.

8.2.5.1 Power Consumption

Another constraint for generating a deployment is total energy consumed by the hardware architecture. We have used a very naïve approach of calculating the total energy. Each ECU in AF3 platform architecture can be annotated with a power value. This is shown in the figure below.

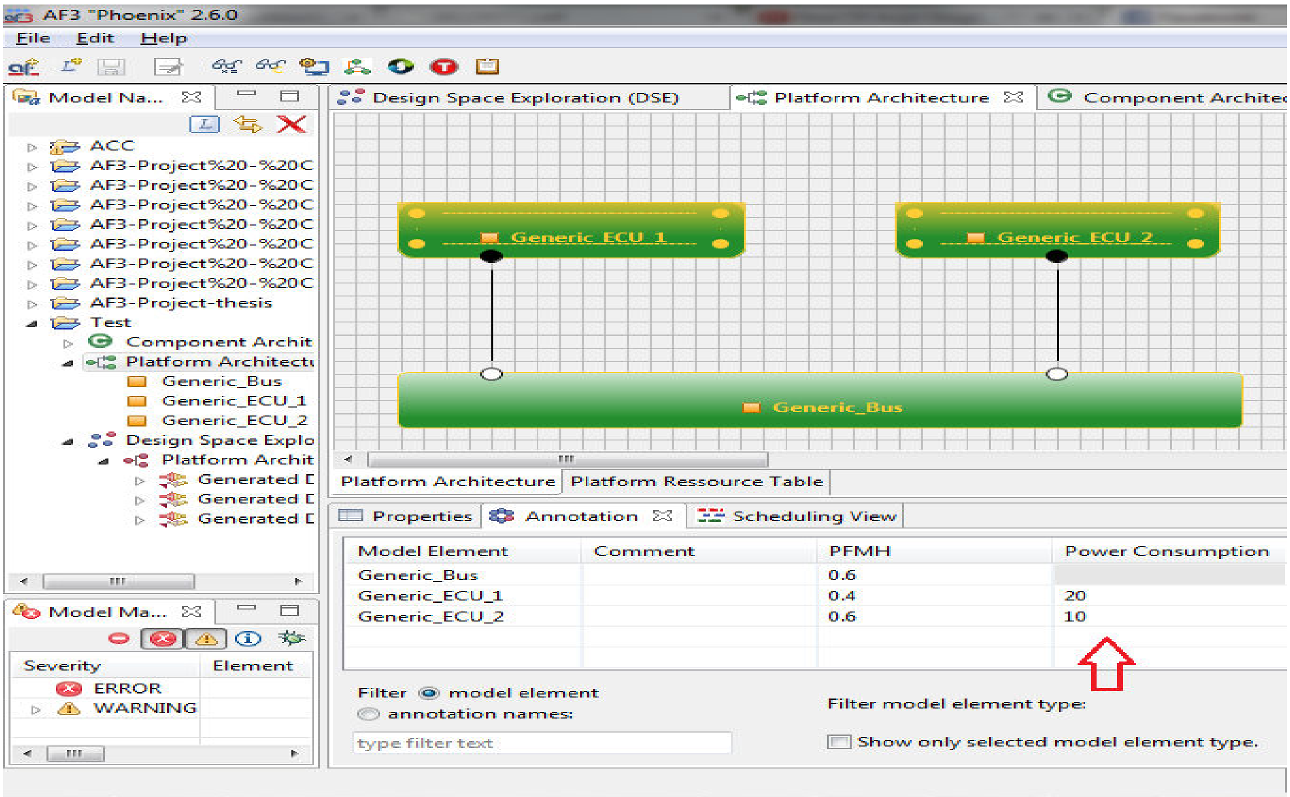

Figure 28: Power consumption annotation for ECUs in AF3

Power is defined as energy per unit time (eq. 1). 20W is analogous to 20 Joules of energy consumed per unit of time. So total energy consumed by each ECU is “Power” times “non-idle time” of that ECU (eq. 2). The non-idle time of an ECU is the sum of duration of all tasks allocated to it (eq. 3).

These individual energy consumptions are summed up to calculate the total energy consumption of hardware architecture which is then used as a constraint to generate the power efficient deployments (eq. 4).

$$Power = \frac{Energy}{Time} \tag{1}$$

$$n_{Energy\_consumed} = n_{Power} * n_{Non\_idle\_time} \tag{2}$$

$$n_{Non\_idle\_time} = \sum_{t_{node}=n} t_{duration} \tag{3}$$

$$Total\_energy\_consumed = \sum_{n}^{N} n_{Energy\_Consumed} \tag{4}$$

This constraints hence ensures that the total energy consumed by the hardware architecture (eq. 4) is less than or equal to the value given by the user at the time of deployment generation, as input in Figure 29.

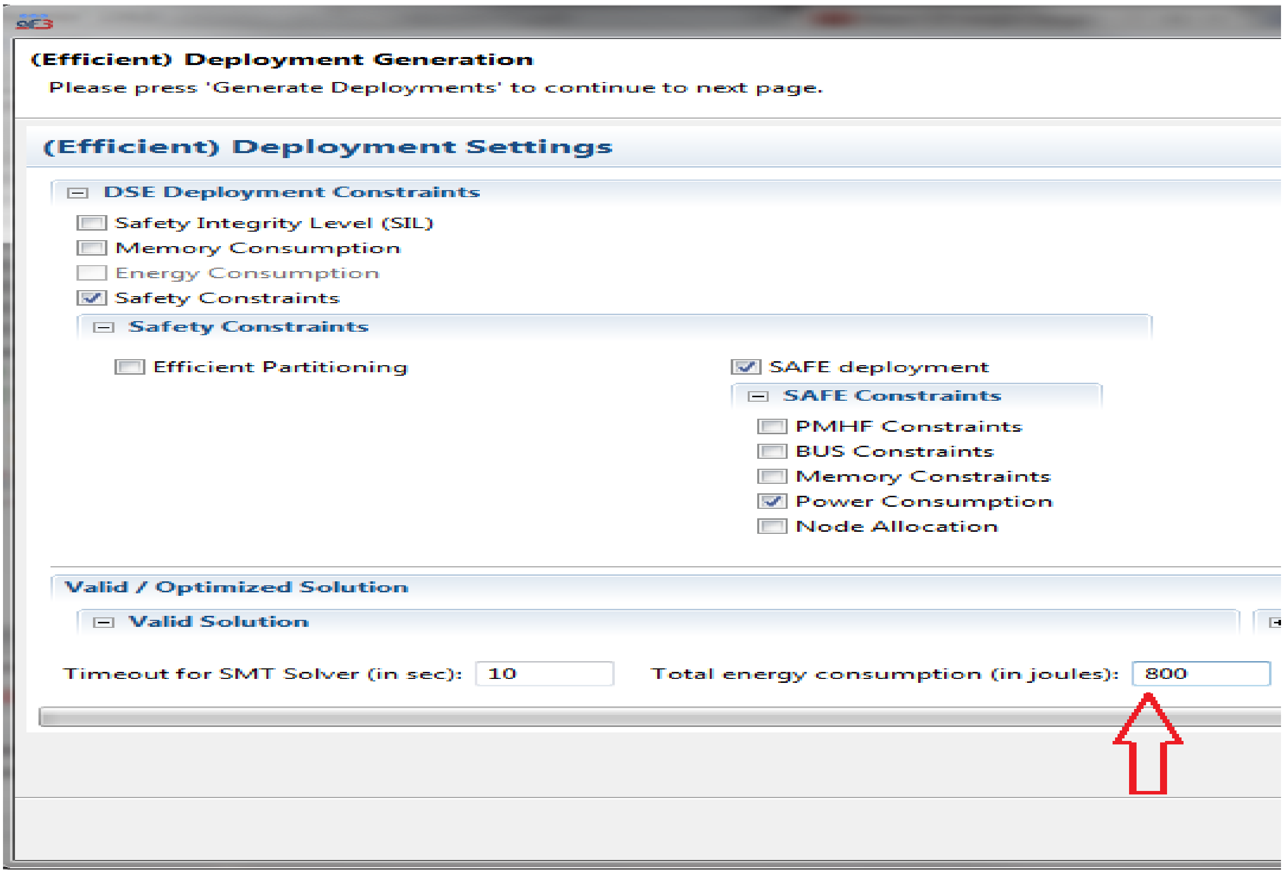

Figure 29: Maximum energy consumption as deployment constraint

Assuming each task has duration of 10 time units, and Generic_ECU_1 gets 3 tasks (t2, t4, t5) and Generic_ECU_2 gets 2 tasks (t3, t6), the total non-idle time Generic_ECU_1 and Generic_ECU_2 is 30 and 20 time units respectively. Consequently, the total energy consumed by them is 600J and 200J respectively. The total energy consumed by the hardware architecture is 800J, which satisfies the deployment constraint given by the user.

Note: the ECU does consume energy while being in the idle state, but we consider this to be negligible in comparison with the energy consumed processing tasks and thus ignore it in our calculations.

8.2.5.2 Cost factor

The second constraint in this category is the "cost factor", where each ECU is annotated with a cost value. Higher the ASIL, higher the development cost and efforts. This constraint is similar to "Maximum no. of nodes constraint" in a way, since it tries to avoid the unnecessary usage of the hardware nodes by reducing the maximum cost allowed during deployment generation. But the main difference is optimizing the overall technical architecture cost by trying to move out the hardware nodes with higher development cost, as opposed to kicking any node out of the development process. Figure 30 shows this constraint.

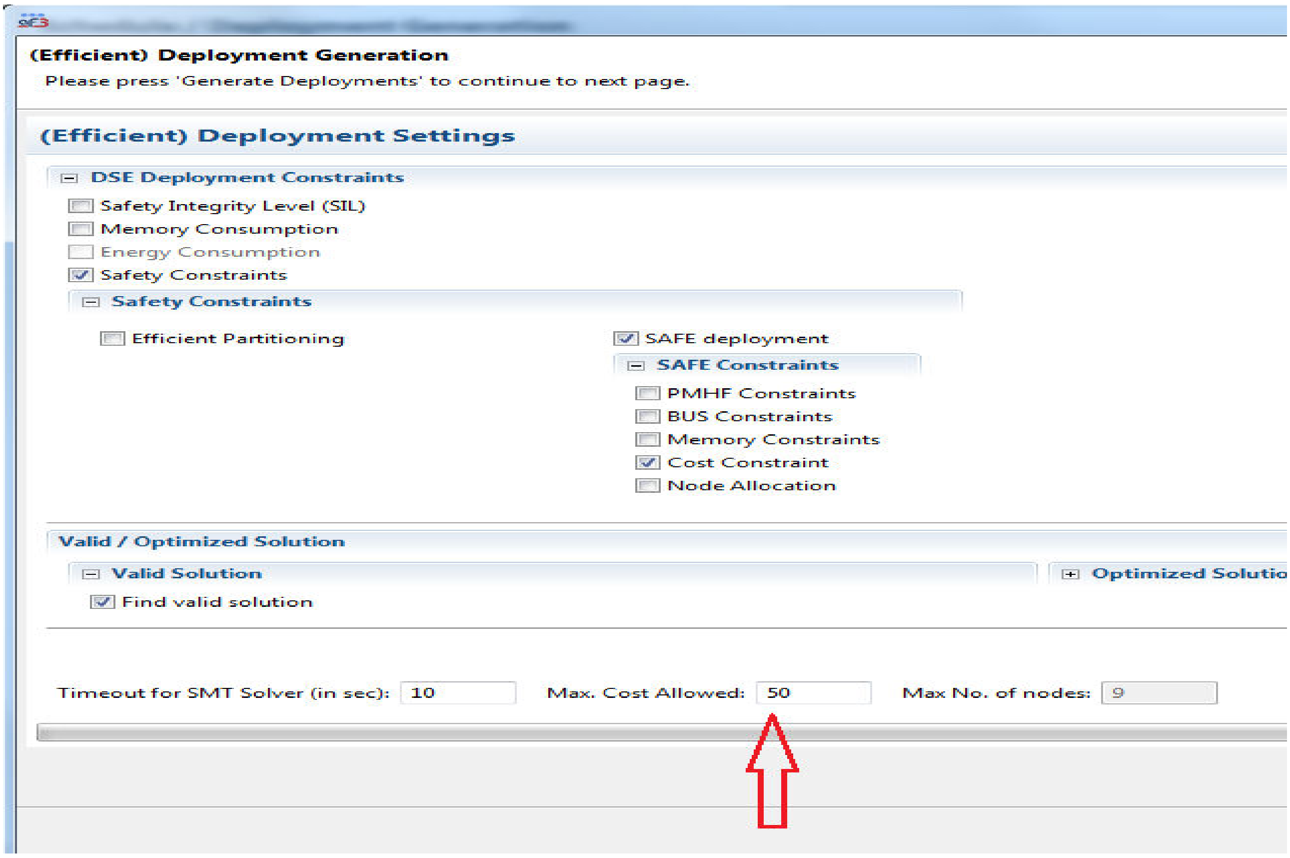

Figure 30: Maximum cost allowed as deployment constraint

Cost Constraint:

Similar to the power consumption annotation used for ECUs (Figure 28), we can annotate them with Cost Factor as well. The main purpose of cost constraint is to reduce the overall developmental cost of the technical architecture by reducing the usage of the higher ASIL nodes.

Let "used" be a Boolean property of an ECU indicating whether the ECU has been included in the deployment or not. Let "cost" be a property of an ECU that represents its developmental cost. Then, the "total_cost" is formulated as below:

$$Total\_cost = \sum_{n}^{N} n_{cost} * n_{used}$$

where

$$n_{used} = \begin{cases} 1, & \text{if } \exists t \in T : t_{node} = n \\ 0, & \text{otherwise} \end{cases}$$

8.3 Analysis of results

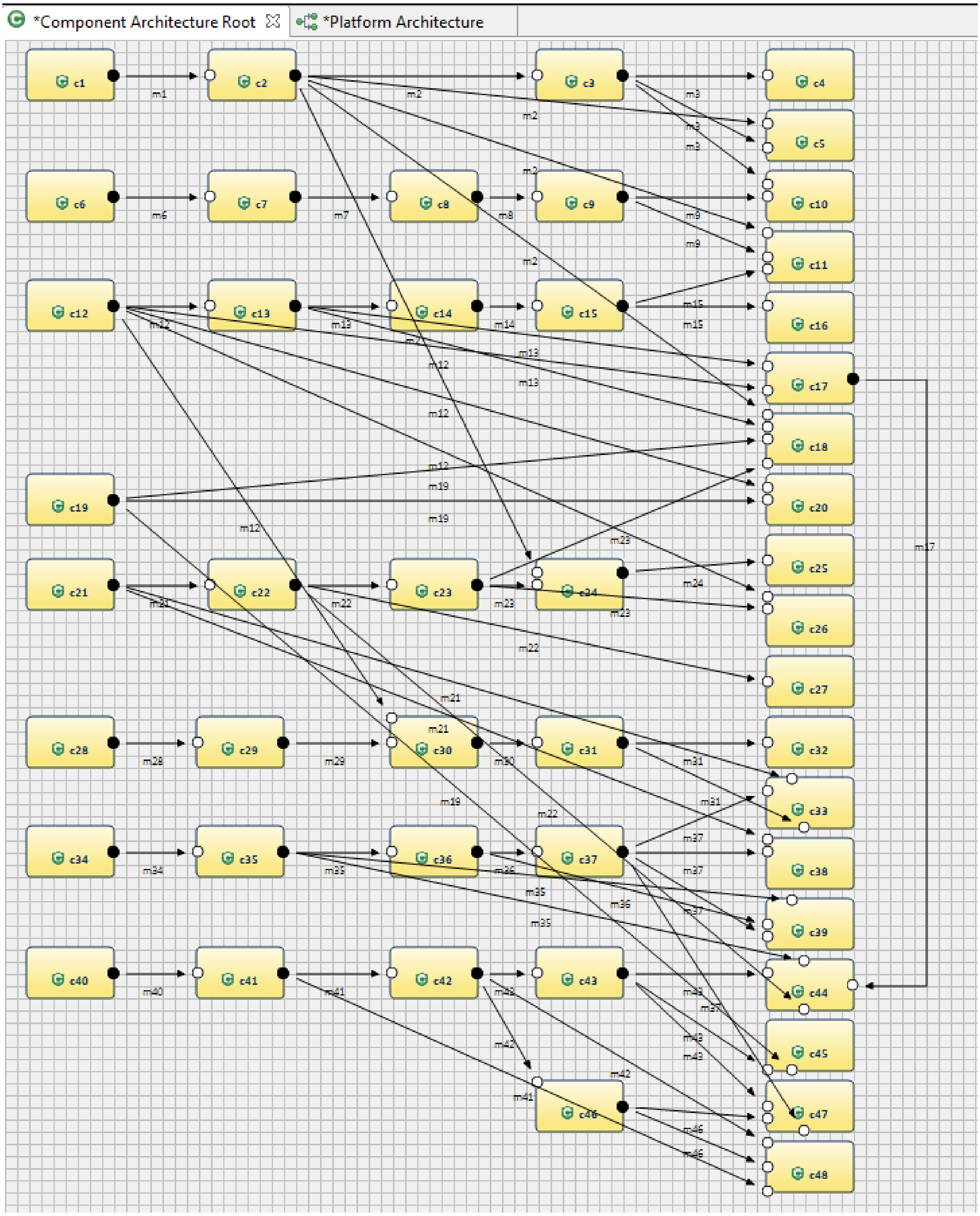

Figure 31: Used logical architecture example

So far we have seen an example that is good enough to explain how all the constraints work. Considering the complexity of modern safety critical systems, the trend is for these architectures to become systems of systems, where multiple functions are delivered by complex networked architectural topologies and where functions can share components [19]. Figure 31 shows a complex logical architecture having 48 components, each with their own safety and memory requirements. The technical architecture used to deploy these components is shown in Figure 32.

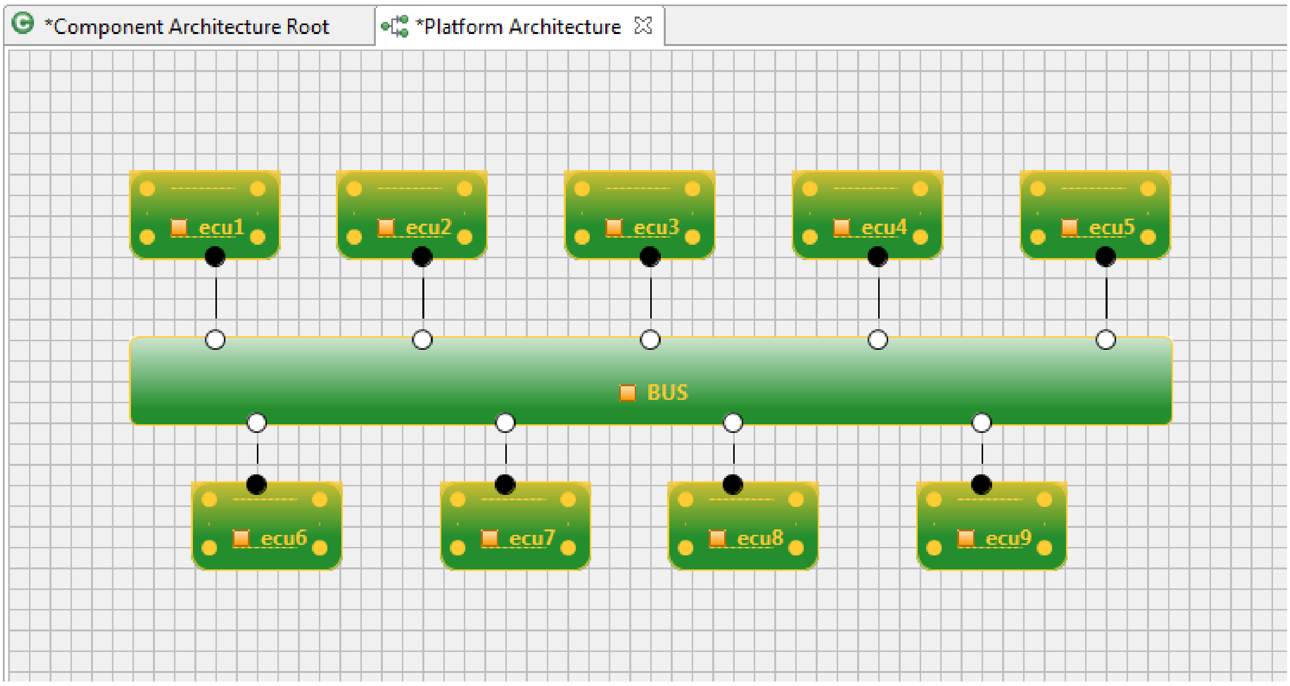

Figure 32: Technical architecture containing nodes of all possible ASIL levels

The safety requirement of all the components in the technical architecture is shown in the table below. The memory required by each component is assumed to be 10 units.

ASIL Level	Components
ASIL-D	c1, c2, c3, c4, c5
ASIL-C	c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17
ASIL-B	c18, c19, c20, c21, c22, c23, c24, c25, c26, c27
ASIL-A	c28, c29, c30, c31, c32, c33, c34, c35, c36, c37, c38, c39
QM	c40, c41, c42, c44, c45, c46, c47, c48

Table 4 ASIL allocation for components

The cost, PMHF and ASIL of each ECU is given in the table below.

Model Element	ECU1	ECU2	ECU3	ECU4	ECU5	ECU6	ECU7	ECU8	ECU9
PMHF	0.1	0.1	0.3	0.4	0.6	0.6	0.8	0.9	1.0
ASIL	D	D	C	C	B	B	A	A	QM
Cost	10	10	8	8	5	5	3	3	1

Table 5 Cost, PMHF and ASIL-Capability Attributes of HW Nodes

Each ECU has been allocated a memory of 100 units. This means, an ECU can carry at most 10 logical components, since each of them require a memory of 10 units.

In the following section we analyze the effects of reducing the overall cost of technical architecture on the generated deployment. Also, we analyze the effects of using "Maximum number of nodes" constraint in deployment generation and how it is different from the "Cost Constraint". Section 8.3.1 shows the results of using "Cost constraint" constraint and section 8.3.2 shows the results of using "Maximum number of nodes". Section 8.3.3 shows the effect of cost on E2E latency and section 8.3.4 shows the effect of optimizing for cost on Bus load (# Bus signals).

8.3.1 Cost Constraint

For analyzing the effects of Cost Constraint on the generated deployment, we assume two different cases, one where the cost is non-uniform and the other where the cost is uniform. The first case entails a staggered cost distribution reflecting real world scenarios, i.e., the fact that developing a node for higher ASIL capability means more stringent development and higher grade components, requiring more development effort and time as compared to developing a less stringent node. This translates to higher cost. The second case, where the cost is uniform, makes for an interesting thought exercise, allowing us to analyze which nodes are kicked out of the deployment first, when different ASIL capability nodes are priced the same. This highlights the difference and importance of setting a relevant cost distribution functionality to the nodes.

Reducing the maximum cost allowed will lead to reduction in number of nodes used, but which nodes are removed from the deployment generation is worth analyzing. Assuming the PMHF constraint, BUS constraint and Memory constraint hold, we now analyze the effect of cost reduction.

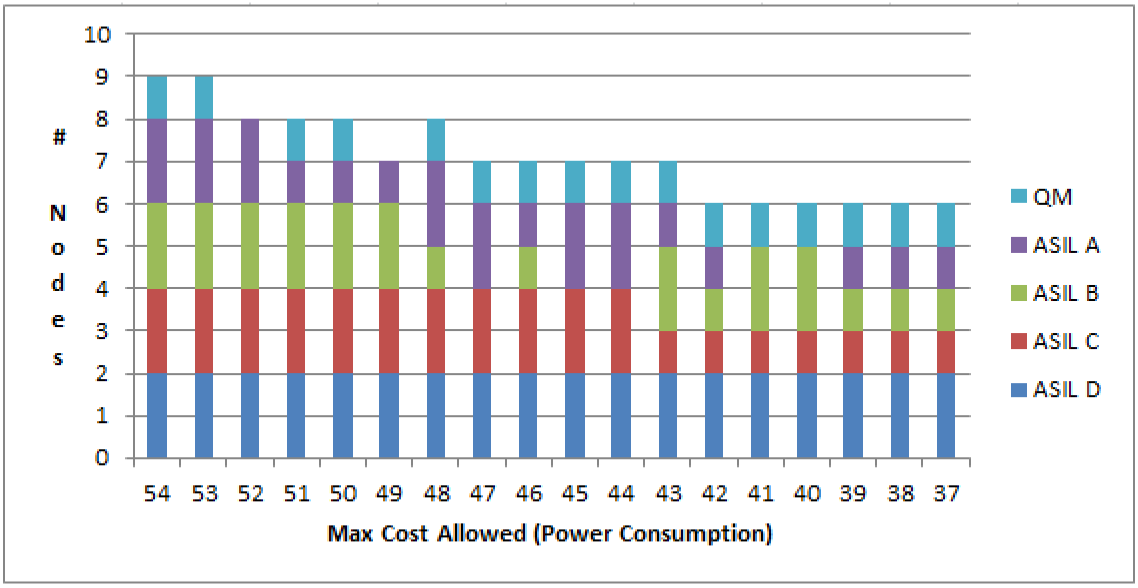

Figure 33: Effect of Cost Constraint (non-uniform cost) on deployment (ASIL-D BUS)

Figure 33 shows a plot of maximum cost allowed vs no. of nodes used in the deployment. The BUS is kept as ASIL-D compliant. By specifying the maximum cost allowed for a deployment generation, Cost Constraint forces the Z3 solver to find a deployment that does not exceed this cost. The graph shows that by reducing the max cost, we have been able to find a deployment that do not violate any of the constraints and reduces the node usage, hence optimizing the overall development cost of the hardware architecture. For e.g. for a maximum cost of 53 we have a deployment consisting of 1 QM, 2 ASIL-A, 2 ASIL-B, 2 ASIL-C and 2 ASIL-D ECUs, whereas for a maximum cost of 37, we have a deployment consisting of 1 QM, 1 ASIL-A, 1 ASIL-B, 1 ASIL-C and 2 ASIL-D ECUs.

Figure 34 illustrates the scenario where the development costs of nodes with different stringencies are assumed to be same (In our case, 6). Since the cost is uniform, the Cost constraint can kick out any node from the deployment generation process. For e.g. for maximum cost of 37, the deployment we got has 1 ASIL-A, 2 ASIL-B, 1 ASIL-C, and 2 ASIL-D nodes. But if we compare it with Figure 33, we have a deployment containing 1QM, 1 ASIL-A, 1 ASIL-B, 1 ASIL-C and 2 ASIL-D. This shows the impact of different cost heuristics in generating a deployment that optimizes the development cost and efforts. Similar analysis has been done by Azevedo et al in [19].

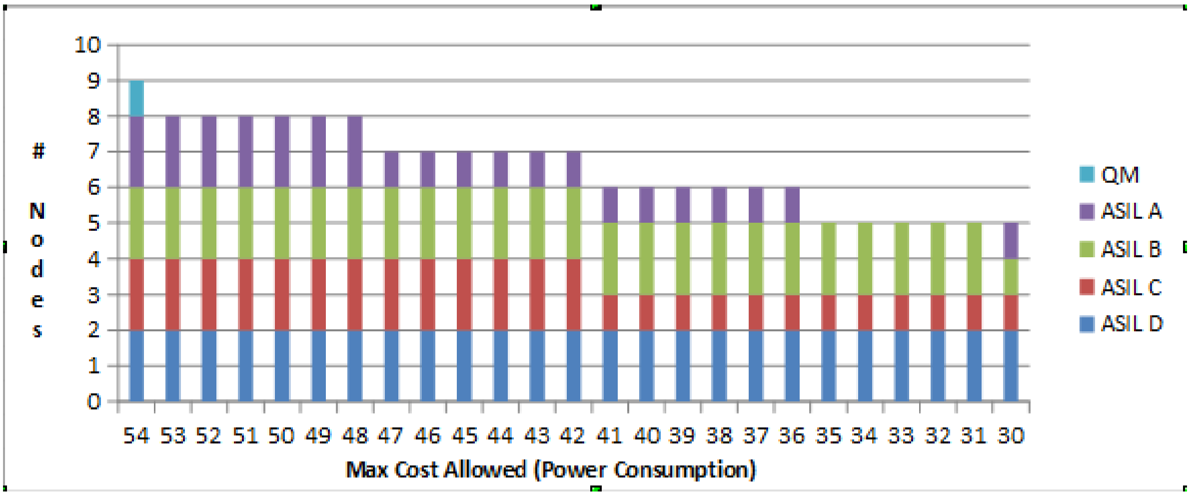

Figure 34: Effect of Cost Constraint (uniform cost) on deployment (ASIL-D BUS)

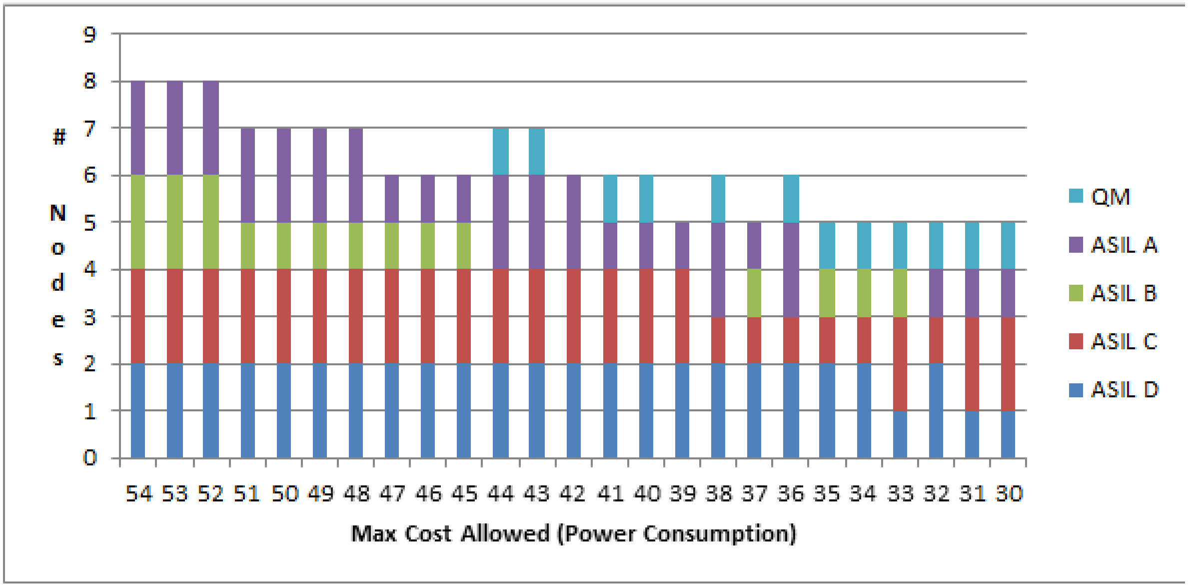

Figure 35: Effect of Cost Constraint (non-uniform cost) on deployment (ASIL-C BUS)

Figure 35 and Figure 36 show the similar analysis for ASIL-C Bus. There is no deployment possible if the Bus is kept as ASIL-B compliant or lower. This is because, an ASIL-B bus forces the component c1-c17 to be deployed on the same ECU, but the memory constraints do not allow any ECU to carry more than 10 tasks on it.

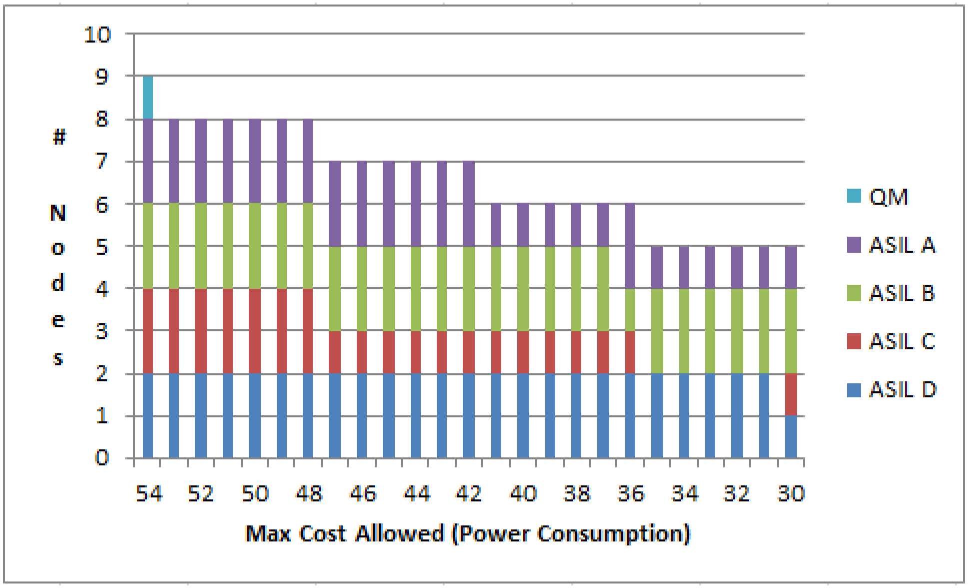

Figure 36: Effect of Cost Constraint (uniform cost) on deployment (ASIL-C BUS)

8.3.2 Number of nodes constraint

To illustrate the effect of this constraint on the deployment generation process, we have kept Maximum cost allowed to a fixed value say 54. Instead of reducing the max cost allowed, we will reduce the maximum no. of nodes in each iteration. Figure 37 shows the effect of this constraint on the generated deployment. Comparing with Figure 33, for 6 nodes, the best deployment that it can find contains 1 ASIL-A, 2 ASIL-B, 1 ASIL-C and 2 ASIL-D leading to a cost of 41, whereas the best deployment found by "cost constraint" for 6 nodes includes 1 QM, 1 ASIL-A, 1 ASIL-B, 1 ASIL-C and 2 ASIL-D, yielding a cost of 37. So "cost constraint" can find a deployment using as many nodes as used by "max no. of nodes" constraint, but with a lower technical architecture development cost.

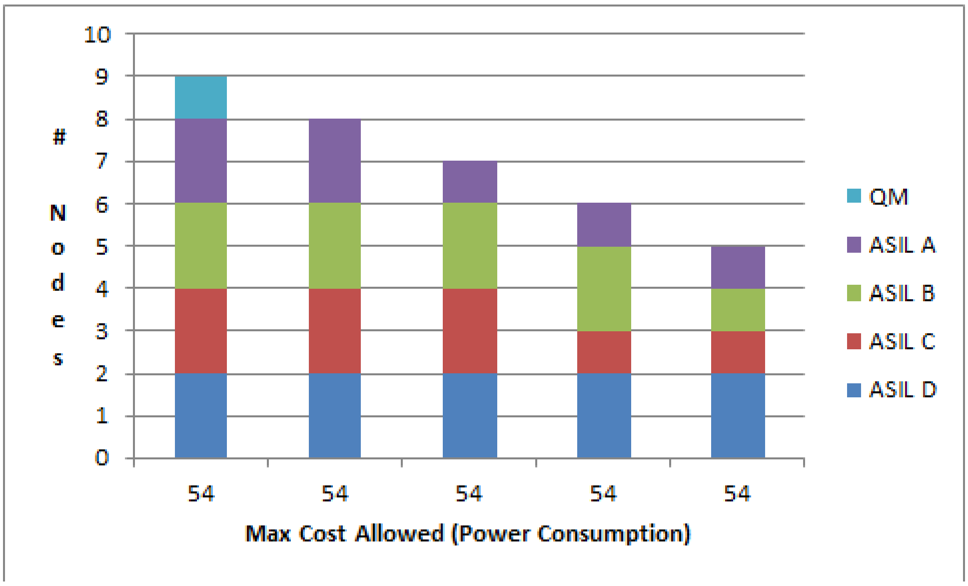

Figure 37: Number of nodes constraint (ASIL-D Bus)

Figure 38 shows the similar analysis for ASIL-C bus. Comparing it with Figure 35, the best deployment found by "maximum no. of nodes" constraint for 6 nodes includes 1 QM, 1 ASIL-A, 2

ASIL-C and 2 ASIL-D yielding a cost of 40, whereas the best deployment found by "Cost constraint" for 6 nodes includes 1 QM, 2 ASIL-A, 1 ASIL-C and 2 ASIL-D, yielding a cost of 35.

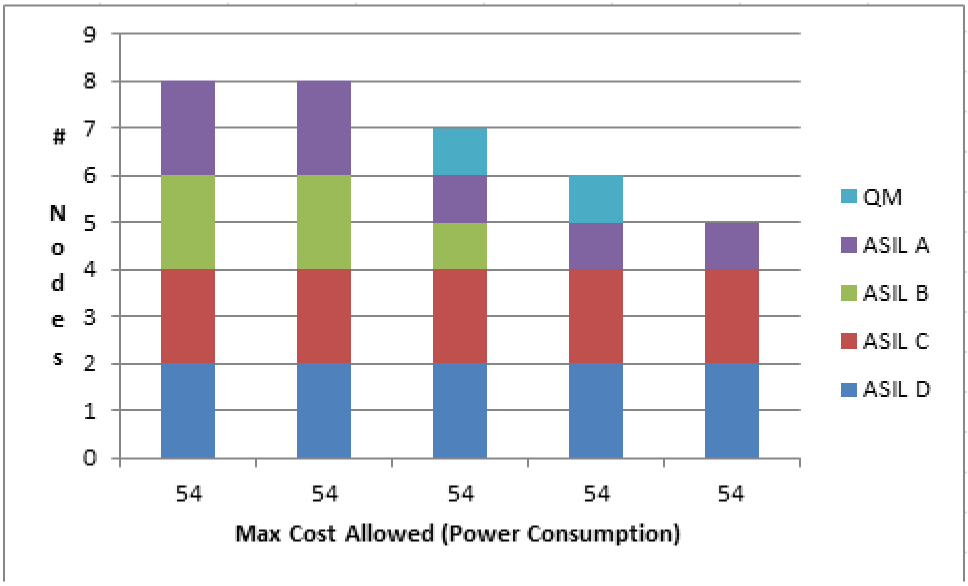

Figure 38: Number of nodes constraint (ASIL-C Bus)

This indicates how "Cost constraint" can outdo the "Number of nodes" constraint by not only finding a deployment that avoids unnecessary usage of the nodes, but also by finding a deployment that optimizes the cost by using less stringent ASIL nodes wherever necessary.

8.3.3 End to end latency vs Cost

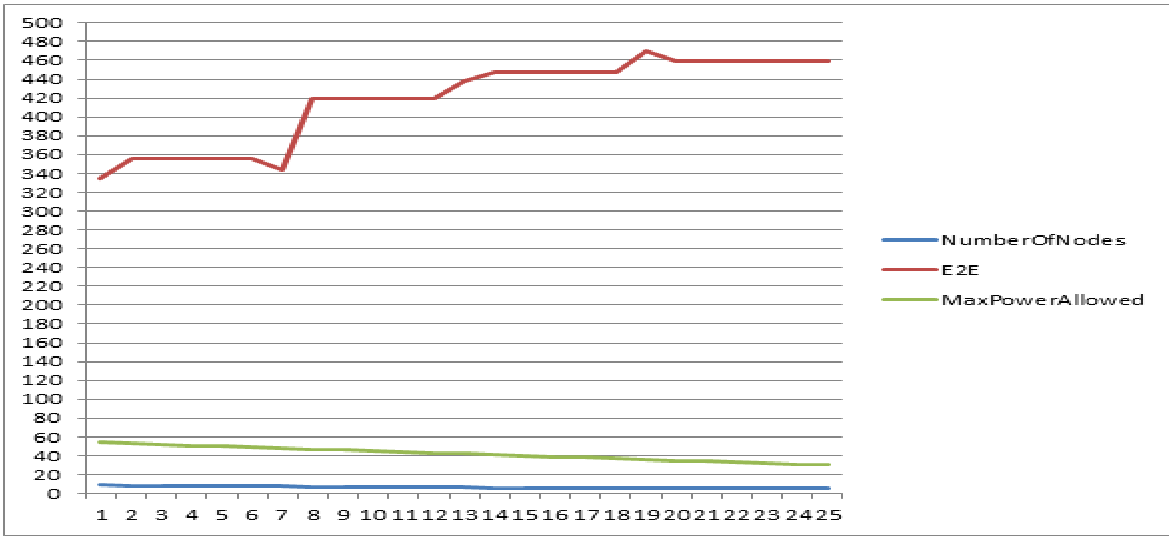

Figure 39: E2E latency vs cost

Figure 39 shows a plot between the E2E latency and the maximum cost allowed. Since the reduction in the cost, reduces the number of the nodes used in the deployment generation, this, therefore, leads to a rise in latency as there will be more tasks scheduled to run on a single ECU. In other words, the degree of parallelism to task scheduling will be reduced.

8.3.4 Bus Load vs Cost

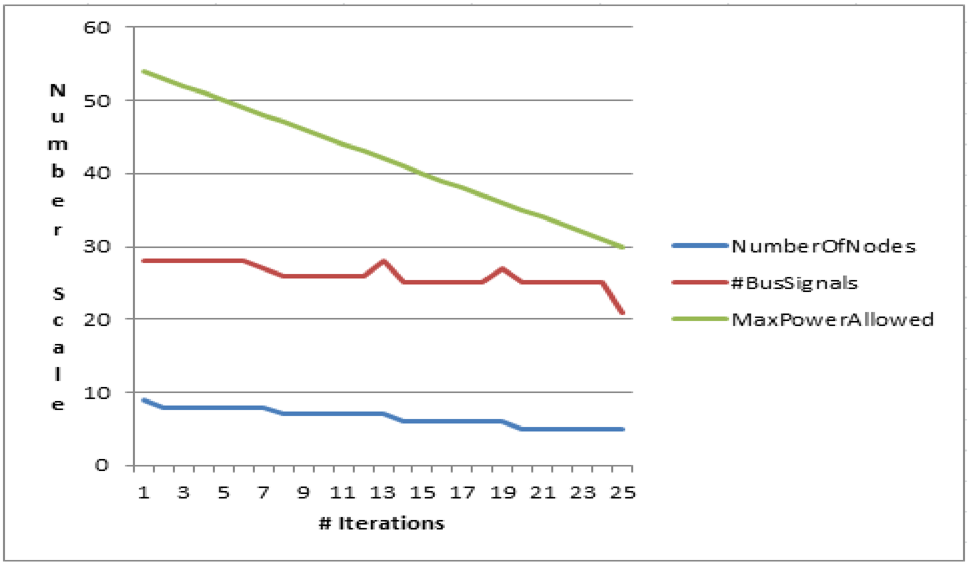

Figure 40: Bus load vs cost

Figure 40 shows a plot between the cost and the bus load. Currently in AUTOFOCUS3, the bus load is measured in terms of number of bus signals that are sent between the components over the bus. More the bus signals, more the bus load. The plot indicates that the bus load reduces with the number of nodes since more components are deployed to the same ECU, but this might not hold true always. It depends on the distribution of the components onto the nodes. If the communicating components are allocated to same ECU, the number of bus signals reduces. If the communicating components are allocated to different ECUs, the number of bus signals increases. Hence, there can be an anomaly where the communicating components are deployed to different ECUs even with the reduction in the number of nodes used.

8.4 Conclusion

The previous chapter has shown the varying results obtainable by manipulating different constraints in a multi-criteria optimization approach, as well as the interactions between the criteria, enabling optimization of constraint values by various optimization approaches.

9 Acknowledgments

This document is based on the SAFE and SAFE-E projects. SAFE is in the framework of the ITEA2, EUREKA cluster program Σ! 3674. The work has been funded by the German Ministry for Education and Research (BMBF) under the funding ID 01IS11019, and by the French Ministry of the Economy and Finance (DGCIS). SAFE-E is part of the Eurostars program, which is powered by EUREKA and the European Community. The work has been funded by the German Ministry of Education and Research (BMBF) and the Austrian research association (FFG) under the funding ID E!6095. The responsibility for the content rests with the authors.

The AF3 project is a long term ongoing tools development effort led by the fortiss department of “Software and Systems Engineering". The functionality provided currently by the tool is mainly the result of the joint work of the following persons:

- Florian Hölzl (mailto:hoelzl@fortiss.org): long-term project lead, tooling infrastructure for developing views and modular languages
- Dr. Vincent Aravantinos (mailto:aravantinos@fortiss.org): current AF3 coordinator, formal methods for model checking
- Sabine Teufl (mailto:teufl@fortiss.org): MIRA - model-based requirements engineering
- Mou Dongyue (mailto:mou@fortiss.org): MIRA - model-based requirements engineering (partly under the IMES project), model based testing, refinement modelling
- Dr. Sebastian Voss (mailto:voss@fortiss.org): design space exploration, GSN-based modelling of safety cases, ASIL allocation, and optimized (safety-oriented) deployment and schedule generation under the SPES XT, RECOMP and ARAMiS projects
- Dr. Daniel Ratiu (mailto:ratiu@fortiss.org): user friendly formal specification and verification

Many thanks also to the other members of the AF3 team.

10 References

[1] International Standards Organization, ISO 26262 Standard, Road Vehicles - Functional Safety, http://www.iso.org/, 2011.

[2] SAFE project: SAFE Deliverable D2.1.b: "Needs description to apply ISO26262 with architecture and component modeling", http://www.safe-project.eu/SAFE-Download.html, 2012.

[3] SAFE project: SAFE Deliverable D2.3.2b: "Detailed description of use case scenarios and list of requirements from industrial practice", http://www.safe-project.eu/SAFE-Download.html, 2013.

[4] SAFE project: SAFE Deliverable D3.2.2: "Proposal for extension of meta model for hardware modeling", http://www.safe-project.eu/SAFE-Download.html, 2013.

[5] SAFE project: SAFE Deliverable D3.3.3a: "Initial specification for comparison of architecture", 2013.

[6] Vector Informatik GmbH, "PREEvision User Manual Version 6.5", 2013.

[7] N. Adler, S. Otten, P. Cuenot, and K. D. Müller-Glaser, "Performing Safety Evaluation on Detailed Hardware Level according to ISO 26262," SAE Int. J. Passeng. Cars Electron. Electr. Syst., vol. 6, no. 1, pp.102–113, 2013.

[8] ATESST2 Consortium, "EAST-ADL Domain Model Specification - Deliverable D4.1.1," Rev. June 2010.

[9] AutoFOCUS3. Fortiss GmbH. 2014, af3.fortiss.org.

[10] Sebastian Voss, Bernhard Schätz, "Deployment and Scheduling Synthesis for Mixed-Critical Shared-Memory Applications". Proceedings of the 20th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS) 2013.

[11] Sebastian Voss, Antoaneta Kondeva, Daniel Ratiu, Bernhard Schätz, "Seamless Model-based Development of Embedded Systems with AF3 Phoenix". Tutorial at the 20th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS) 2013.

[12] SPES-XT Project Consortium. spes2020.informatik.tu-muenchen.de/spes_xt-home. 2013.

[13] Z3 SMT Solver, Microsoft Research, Microsoft Corportation. research.microsoft.com/en-us/um/redmond/projects/z3. 2013.

[14] YICES SMT Solver, SRI Tools, Stanford Research Institute. yices.csl.sri.com. 2013.

[15] Sebastian Voss, Johannes Eder, Florian Hölzl, Bernhard Schätz. "An integrated Design Space Exploration Approach". Submitted.

[16] Automotive, Railway and Avionics Multicore Systems – ARAMiS Project Consortium.. www.projekt-aramis.de. 2013.

[17] Florian Hölzl and Martin Feilkas. "AutoFocus 3 – A Scientific Tool Prototype for Model-Based Development of Component-Based, Reactive, Distributed Systems".

[18] SAFE-E project consortium: SAFE-E Deliverable D3.5b: "Final specification for architecture comparison", 2013.

[19] Luís Silva Azevedo, David Parker, Yiannis Papadopoulos, Martin Walker, Rui Esteves Araújo, Ioannis Sorokos, and Rui Esteves Araújo. "Exploring the Impact of Different Cost Heuristics in the Allocation of Safety Integrity Levels", 2014.