Deliverable 3.1

Specification of Safety, Security and Performance Analysis and Assessment Techniques

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737475. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Spain, France, United Kingdom, Austria, Italy, Czech Republic, Germany.

DISSEMINATION LEVEL		
X	**PU**	Public
	CO	Confidential, only for members of the consortium (including the Commission Services)

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	AQUAS
Project Full Name:	Aggregated Quality Assurance in Systems
Grant Agreement No.:	737475
Programme	ICT-1: Cyber-Physical-Systems
Instrument:	Research & innovation action
Start date of project:	01.05.2017
Duration:	36 months
Deliverable No.:	D3.1
Document name:	Specification of Safety, Security and Performance Analysis and Assessment Techniques
Work Package	WP3
Associated Task	Task 3.1
Nature [1]	R
Dissemination Level [2]	PU
Version:	1.0
Actual Submission Date:	30-04-2018
Contractual Submission Date	30-04-2018
Editor: Institution: E-mail:	Stefano Di Gennaro UNIVAQ stefano.digennaro@univaq.it

[1] **R**=Report, **DEC**= Websites, patents filling, etc., **O**=Other

[2] **PU**=Public, **CO**=Confidential, only for members of the consortium (including the Commission Services)

Change Control

Document History

Version	Date	Change History	Author(s)	Organisation(s)
0.9	17.4.2018	First submission	Stefano Di Gennaro	UNIVAQ
1.0	30.4.2018	Final version after quality review	Filip Veljković, Peter Popov and Lorenzo Strigini	TASE, CITY

Distribution List

Date	Issue	Group
17.2.2018	0.1	WP2, WP3, WP4
17.4.2018	0.9	Leaders
30.4.2018	1.0	AQUAS.ALL

Table of Contents

Executive Summary

This deliverable is a result of the work performed in Task 3.1 and its interaction with the rest of Use Cases and Work Packages. Its role towards supporting work in AQUAS are:

- Reference document to support WP2 activities on the AQUAS demonstrators, it describes methods and know-how available from AQUAS partners for analysis and assessment of safety, security and performance. For each method, section 2 gives a brief description, and section 3 then outlines examples of intended application in the AQUAS demonstrators. This inventory of techniques and application examples supports planning in WP2 and task 3.2;
- Initial step for task 3.2, dedicated to combining analyses of safety, security and performance, section 4 adds to the above documentation of individual techniques:
  (a) considerations on "interaction points", the mechanism in AQUAS which goal would be to realize these joint analyses affordably and without excessive disruption of pre-existing processes; this part shares results of initial work about the practicalities of interaction points, by a subset of project partners; and
  (b) preliminary plans for interaction points in AQUAS demonstrators, as envisaged by the demonstrator teams in view of each demonstrator's specific needs and product lifecycle.

1 Introduction

This deliverable addresses (1) methods and techniques for the assessment of safety, security and performance (as separate properties, in general, but with some methods supporting some joint analyses), and (2) initial plans for combining these techniques. The deliverable does not aspire to provide a state of the art of these quite broad areas, but rather is meant as a tool for AQUAS, including a reference list of the techniques provided to AQUAS by the project partners and sharing the initial understanding reached regarding their combinations in "interaction points".

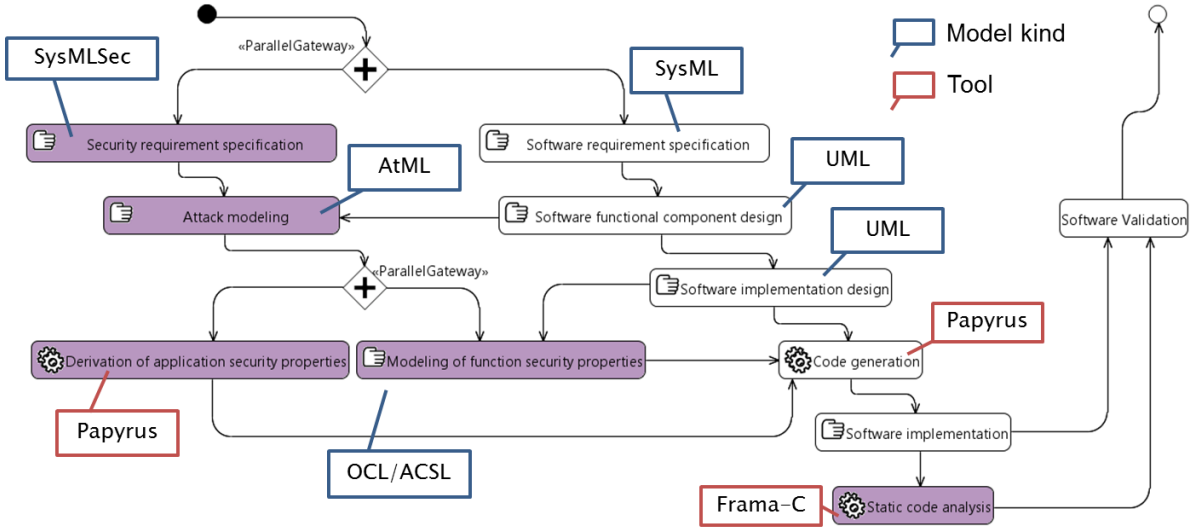
The structure of this Deliverable is as follows. Section 2 contains description of methods and techniques for the assessment of safety, security and performance. In Section 3, these analysis techniques are illustrated for each demonstrator (Use Case – UC) of the AQUAS Project. Finally, in Section 4, the interaction points in the given UCs are described.

2 Methods and techniques for the assessment of safety, security and performance

In this section, safety, security and performance metrics to be assessed are described, together with the methods and techniques for the assessment of these properties separately.

2.1 Software/security co-engineering method based on models and static code analysis [CEA]

CEA propose model-based software/security co-engineering method based on parallel processes, model kinds, and transformation tools. The goal is to separate concerns so software development can run in parallel to security engineering. The figure below shows the co-engineering method.



The purple steps are related to security requirements and properties specification and verification. The white steps are classic software development steps.

The following sections explain the two major processes in the method, i.e. software development and security engineering. We finish by focusing on the methods we use for static code analysis for security properties verification.

2.1.1 Software development process

The classic software development process is composed of the following steps of the method:

- Software requirement specification: The specification of software requirements is done with the SysML::Requirement profile. This profile may be extended for specific domains or project, i.e. the <<Requirement>> stereotype may be extended with domain-specific or project-specific properties.
- Software functional component design: The design of software functional components is done with UML component-based models enhanced with state machine based behaviours. The component-based model typically contains components inter-connected through ports and connectors. Each component specifies its interfaces required/provided through its ports.

Functional components are described with UML composite structure diagrams, while their behaviours are described with UML state machine diagrams.

- Software implementation design: The software functional components model is refined (automatically or manually) to a software implementation model. In this model, the functional components are represented as classes, while references and interaction classes are designed for connections between component ports. Methods are also designed to implement the previously state-machine specified behaviours. The implementation model may also contain annotations for specific programming languages.
- Code generation: The code generation step consists in generating code in a programming language from the software implementation model.
- Software implementation: The generated code may be incomplete. This step completes the code, by writing the missing method bodies for example.
- Software validation: This step concerns software testing, which is out of the scope of the method.

The next section presents the parallel security engineering process of the co-engineering method.

2.1.2 Security engineering process

The security engineering process is composed of the following steps:

- Security requirement specification: The specification of security requirements for the software is done with the SysMLSec::Requirements profile. This profile extends the original SysML::Requirements profile for security specific requirements, e.g. it proposes several new security requirements stereotypes that extend <<Requirement>>.
- Attack modelling: The security requirements are refined with interaction model, represented in UML sequence diagrams. The language to describe the models is Attack Modelling Language (AtML). Nominal, attack, and prevention scenarios can be described in such models.
- Modelling of function security properties: The nominal, attack, and prevention scenario models can be further refined with security properties expressed in the the ANSI/ISO C Specification Language (ACSL). ACSL is presented in the next section on static code analysis. The security properties are specified as pre/post-conditions of functions of the software implementation model.
- Derivation of application security properties: Global security properties for the whole application can be inferred from the AtML models. These properties are expressed in Relational-ACSL specifications, which is ACLS focusing on function call partial orders rather than local function pre/post-conditions. The specifications are input in the software implementation model.
- Code generation: The code generation step consists in generating code in a programming language from the software implementation model. It also generates the ACSL-expressed specifications in the software implementation model.
- Static code analysis: The implemented code is analysed with several analysis methods and tools of Frama-C, presented in the next section.

The next section gives more information on the static code analysis methods.

2.1.3 Static code analysis

The ANSI/ISO C Specification Language (ACSL) allows specifying formally the properties of a C program, in order to be able to formally verify that the implementation respects these properties. This verification is done via analysis methods that are able to take into account ACSL annotations attached to the C code. Verification is performed with the Frama-C tool (outside of the scope of this deliverable).

Within AQUAS, CEA propose to use deductive verification. Based on weakest precondition computation techniques, it provides proof for the correctness of a C-function against its ACSL specification. The proofs are modular: a function is proved independently from its calling context, and only from its actual code and the specification of the other functions. For each ACSL code annotation, the deductive verification generates a bundle of proof obligations, i.e. first-order logic formulas that entail their correctness. These proof obligations can be submitted to the Alt-Ergo automated theorem prover or the Coq proof assistant and many other provers via the Why platform.

CEA also propose to use value analysis. This method computes an over-approximation of the possible values for each variable of the program, using the theory of abstract interpretation. The analysis is automatic, although the user may guide it through several means, and the method handles a wide spectrum of C constructs. The results of the value analysis can be exploited in several ways.

CEA propose to use such formal static code analysis method to verify security properties of a program.

2.2 Art2kitekt [ITI]

Art2kitekt (A2K) is a software tool-suite developed (and also under continuous development) that is used for modeling embedded critical systems. It provides various analysis and simulation techniques over those models in order to refine them and to guide and assist the engineer during the design process. A2K makes use of different user profiles for offering to the engineer the model template and needed features for modelling and analysing the system. When the designed system has finally been validated with the different analysis algorithms, a target platform can be addressed to automatically generate the high level code to manage the flows and tasks modelled before.

Different hardware execution platforms can readily be redefined and available for the engineer to configure with the corresponding parameters of the specific hardware to be used. The engineer is thus able to rapidly customise the hardware platform in terms of CPUs, buses, memories, devices, etc. Different configurations can be quickly explored to see their effects.

Application software can be grouped into different interconnected subsystems. These are further decomposed into sets of tasks with dependencies and messaging among them, as well as properties and constraints: worst execution time, flow periods, deadlines. Also shared resources and various communications protocols can be used by the tasks.

The engineer is able to specify the particular kind of system analysis she wants to perform. It is also possible to interact with the tool to fix any issues that makes the system unfeasible. The implementation of these analyses is fast enough to be execute “on the fly”, so the user can easily test several possible options and see the result immediately. This is called Design Space Exploration.

A target platform and O/S can be addressed to automatically generate the high level code to manage the flows and tasks modelled before.

User interaction with A2K is via a web based browser graphical interface, which communicates with back end servers which provide the analysis algorithms.

The A2K tool currently provides the following functionality:

Compositional Response Time Analysis – Calculates the theoretical upper bounds for the response times of a set of tasks & flows running on a multiple, distributed, heterogeneous processors with message passing by various means such as shared memory, busses, networks and so forth.

Task Allocation – Provides various scheduling algorithms to allocate tasks to processors to satisfy different optimisation criteria such as processor load and power consumption.

Simulation – simulates the running of a set of tasks and flows running on multicore processors and provides chronograms of the task interferences and response times.

Sensitivity Analysis – provides algorithms for evaluating the sensitivity of task response times to variations in other system parameters.

Code Generation – generates template C/C++ code skeletons for implementation.

Execution Tracing – can log the execution of tasks running on real hardware and provide summary statistics of performance such as task execution times.

Requirements and quality management – Links a set of high level requirements to the outputs of analysis, simulation, or real world measurements in a target system.

Test Plan management – manages a database of test cases and test results as well as linking these to associated requirements. Can interface with real or simulated hardware systems and can provide automatic generation of test case parameters, collection of measurements, and assessment of results.

2.3 Experimental data collection and associated analyses for safety, security and performance assessment [City]

In the research community, it is widely accepted that empirical measurements provide the essential link between empirical observations and rigorous statements of quantitative relationships.

Similarly to natural sciences, experimental data collection in software engineering (e.g. for embedded, safety-critical systems, and cyber security more recently), is a cornerstone for creating new theories and then validating them to offer provable explanation about various man-made and natural phenomena (Schneider, 2012).

Thus, this assessment method is rather general – it is not a typical *method* for safety, security and performance assessment. Instead, it can, and should, be used as a complement to several others discussed in this section: the results of empirical studies are used to inform quantitative modelling for dependability, security and performance assessment, e.g. to validate assumption and establish accuracy of stochastic, state-based models. However, exploratory analyses based solely on empirical data are frequently performed, too.

As such, this technique reinforces the much-needed focus, for researchers and practitioners, on real evidence and rigorous, scientific reasoning.

Empirical data collection and analyses are a general-purpose “technique”, and as such, no specific requirements by standards related to safety, security and performance are directly applicable.

This sub-section presents some examples of the method: empirical data collection and analyses for safety, security and performance conducted in the Centre for Software Reliability (CSR), City, University of London. We provide a brief explanation of the given experimental analysis method, the

related case study with a summary of respective results, and list the main measures of interest in each. These measures are not meant to be definitive for a particular case study. We, instead, give examples of what researchers and practitioners might decide to be the focus of their studies.

A long-standing interest of CSR has been the use of "diversity" in computer-based systems, and especially safety-critical systems, and its effects on system dependability, including security. This approach can be summarised as doing things differently, in two or more ways, to protect against the failures of individual components or the whole system. The examples of empirical studies presented here have all been concerned with use of this form of diversity and assessment of its effectiveness. However, many parts of the experimental approaches explained here are applicable to other types of analyses, too.

2.3.1 Examples of empirical studies for reliability/safety analysis – Diversity Seeking Decisions (DSDs)

Diversity Seeking Decisions (DSDs) are practices, used or recommended, for increasing the degree of diversity between redundant implementations of software-based systems, with the aim of improving system reliability/safety (Littlewood and Strigini, 2000).

A rationale behind DSDs is a reduction of the likelihood of similar faults in the constituent components making up a given software-based system. However, DSDs effectiveness is measured by the actual failure diversity between the used components.

Experimental evidence of the effectiveness of individual DSDs, e.g. in reducing system's probability of failure on demand (PFD), and their limits, is scarce. This is even more acute for the effectiveness of combined DSDs. Thus, our aim has been to contribute to the ongoing CSR research directed at informing decision makers how great an advantage could be expected from the use of DSDs, in absolute and/or comparative terms (Popov et al., 2012), (Popov et al., 2014).

For the analyses referred to in this document, we used computer programs submitted to *Online Judge* programming contest (http://uva.onlinejudge.org/). This online resource allows any member of the public to enter the contest and submit *programs* that solve various types of computing *problems* such as: string operations, sorting, arithmetic and algebra, number theory, etc.

Our focus, initially, has been on the reliability of a 1-out-of-2 (1oo2) system – a system made up of two components, which only fails if both components fail.

A DSD amounted to forming all pairs of programs from programs that differ with respect to a certain *attribute*. In this study, we considered *programming language* or *program structure* as examples of such attributes.

For one of the "problems" analysed, we performed exhaustive testing of the whole demand space. This is usually infeasible, or impossible, in practice (e.g. when the demand space covers a subset of real numbers). For other "problems", the respective demand space has been sampled in a bounded and contiguous manner over a subset of it – it is an example of simple bounded exhaustive testing. Clearly, other approaches, which allow for further coverage of the demand space are possible, e.g. one could choose a suite of demands drawn from the whole demand space according to some distribution (for production software systems, this distribution is meant to be representative of the operational profile, i.e. a quantitative characterization of how a system will be used).

We conducted exploratory statistical analyses in two strands:

- "Analysis on average" - the measure of choice was *system reliability improvement ratio*, R = $E(PFD_A)$ / $E(PFD_{AB})$. This analysis explored the potential for reliability improvement in terms of the expectation of PFD of a random diversely-redundant systems and a random single component,

i.e., by computing the ratio between the average PFD of 1-out-of-2 systems and the average PFD of the constituent software programs. This analysis compared the mean PFD of pairs of programs built by a certain set of DSDs (including no DSDs), against the mean PFD of individual programs. We wanted to investigate the potential reliability gains of 1-out-of-2 system for a range of different marginal PFD values of single programs.

- "Analysis of distributions", which computed the PFDs of a *specific pair* of programs and the *specific individual programs* (components) used to form the pair. We analysed the statistical *distribution of the gain* from using software fault tolerance in comparison with the better or the worse of the two programs used to form the pair.

The studies revealed a number of *interesting observations*, some of which are summarised below.

Based on the results of the analysis "on average", we obtained "proof of existence" of cases in which "more diversity" is beneficial, i.e. a *combination of two DSDs leads to better reliability than when no DSD is used, or when a single DSD is used*. For a particular combination of DSDs, this claim was indeed true. We observed, however, counter-examples where using either of these two DSDs on their own on average leads to a more reliable system than applying both DSDs simultaneously.
The analysis of distributions demonstrated some patterns that would quite plausibly occur in many other cases of simple software: i) PFD of the single programs and of the 1oo2 pairs appear in discrete distributions; and the distributions do not resemble any standard unimodal distribution; ii) much of the gain from diversity comes from the fact that a sizable fraction of the programs are fault free, and thus a much greater fraction of the pairs are. However, we also showed that the probability of building a perfect pair out of *faulty components* may become significant - in one of the studies we recorded a frequency of more than 40%. This is an encouraging result, as it shows a likelihood of an extreme gain from software diversity – a *perfect* system.
We do *not* propose our results as predictions of what will happen when applying a specific DSD in an industrial development for, say, safety critical applications. Rather, they are a starting point. First, this kind of empirical research shows patterns of behaviour that may inspire new analyses and insight; secondly, they demonstrate, more strongly than theoretical considerations, that common-sense claims for DSDs may be misplaced.

2.3.2 Examples of empirical studies for security analysis – Diverse AntiVirus Software

All systems need to be sufficiently dependable and secure in delivering the service that is required of them. There are various ways in which this can be achieved in practice: use of various validation and verification techniques in the software construction phases, issuance of patches and service releases for the product in operation, as well as the use of software fault/intrusion tolerance techniques, etc. Software fault-tolerance via use of diverse products – *diverse redundancy* – has been traditionally regarded as expensive. However, the wide proliferation of off-the-shelf software for various application domains has made software diversity an affordable protection against both malicious and accidental failures.

Importantly, there has been a lack of sufficient empirical studies to evaluate the effectiveness of diversity for detection of malware. Thus, we have quantified the possible gains in malware detection from using more than one diverse AntiVirus (AV) engine (Gashi et al., 2009), (Bishop et al., 2011), an essential cyber-security control tool.

For this purpose, we used real-world data, namely the information provided by a distributed honeypot deployment, SGNET (Leita and Dacier, 2008). In one of the studies, we analysed 1599 malware samples collected by the SGNET distributed honeypot deployment over a period of 178 days. Using these malware samples, we studied the evolution of the detection capability of the signature-based component of 32 different AV products and investigated the impact of diversity on such a capability.

Exploiting the implemented submission policy in SGNET, we have considered for each sample the submissions performed on the 30 days succeeding its download. Therefore, the input to our analysis was a series of triplets associating an AV product, a certain malware sample and the identifier of the submission day with respect to the download date $\{AV_i, Malware_j, Day_k\}$.

We considered several types of diverse setups ranging from *simple detection setups* (where a malware is deemed to have been detected as soon as one of the AV products raises an alarm for it) to *majority voting setups* (where a malware is deemed to have been detected only if a majority of AV products in a given diverse configuration raise an alarm for that malware).

We also analysed the dataset in the time dimension to quantify the extent to which the use of diverse AV engines reduces the "at risk time" of a system.

In this study, all demands were confirmed malware, and thus we could not perform analysis about false positives.

A sample of the outputs of this research is as follows:

- No AV product achieved 100% detection rate;
- The detection failures were both due to an incomplete signature databases at the time in which the samples were first submitted for inspection, but also due to *regressions* in the ability to detect previously known samples – this is a consequence, possibly, of deletion of some signatures;
- Considerable improvements in detection rates can be gained from employing diverse AVs, e.g. no single AV product detected all the malware in our study, but almost 25% of all the diverse pairs, and over 50% of all triplets in 1-out-of-N configurations successfully detected all the malware;
- Significant potential gains in reducing the "at risk time" of a system from employing diverse AVs: even in cases where AVs fail to detect a malware, there is diversity in the time it takes different vendors to successfully define a signature to detect a malware.

Similarly to the DSD studies (see 2.3.1), these results are not readily transferable to other AV setups (e.g. due to specifics of malware investigated), but show extent of possible gains when diverse products are used.

2.3.3 Examples of empirical studies for performance analysis – Diverse Database Replication (DivSQL)

Numerous database replication schemes are built on the crash failure assumption where majority of failures are self-evident. The study in (Gashi et al., 2007) convincingly refuted this common assumption showing that many of the faults cause systematic *non-crash* failures; most notably incorrect results. This was the first study of its kind where publicly available fault reports of a category of off-the-shelf software - Database Management System (DBMSs) – were used for assessment of potential reliability gain. Similar results were obtained in a subsequent study (Vandiver et al., 2007).

Consequently, the existing database replication solutions, which typically use the same DBMS products and guard against crash failures, are not sufficiently effective fault-tolerant mechanisms. The study in (Gashi et al., 2007) showed that a only a very small proportion of the tested faults caused identical, non-detectable failures in two diverse DBMS products; there were no such failures affecting more than two. Thus, using diverse database servers, in a parallel redundant configuration, is a suitable way of protecting against non-crash failures.

We have devised and implemented a novel middleware-based database replication protocol, DivRep, and deployed it with diverse database servers into a replicated database system (DivSQL) for improved fault tolerance.

We performed extensive experimental studies of performance implications when diverse database servers are used for replication. We obtain our experimental results on a *real* implementation that evaluates the feasibility of the middleware-based solution in a *real environment*.

The chosen application is the leading standardised industry benchmark for performance evaluation of DBMSs – Transaction Processing Council's Benchmark C (TPC-C). We collected detailed logs of our measures of interest: database transaction response times and throughput[3], SQL operation response times, database transaction abort counts etc. These have been obtained under varying load – we have deployed a range of simultaneous TPC-C clients against different DBMS setups.

The reliability gain – obtained via using diverse DBMS products – comes with inherent performance penalty. Therefore, we provided a baseline experimental evaluation of such performance cost: we compared DivSQL performance to that of the single DBMSs. We also performed a comparison of diverse and non-diverse database servers when our replication solution is optimised for either maximum dependability or maximum performance. Some of the findings of the experimental studies are as follows:

- Systematic differences between performances of diverse servers exist, i.e. one of the servers exhibits better performance on particular (sets of) SQL operations. This observation suggests that performance improvement might be observed when a pair of diverse database servers is used: if the execution of SQL operations is distributed in the way that each server executes only the portions of operations it is faster on, the diverse pair performance will be better than the performance of individual servers (Gashi et al., 2004).
- Under a workload with a single modifying client and multiple clients that execute a read-only workload, a significant performance gain can be obtained with a pair of diverse database servers when compared to non-diverse pairs or single server configurations (Stankovic and Popov, 2006).

Apart from the presented case studies, further information about empirical data collection, measurement and analyses for safety, security and performance assessment conducted by the CSR from City, University of London can be found on the following webpages: www.csr.city.ac.uk /diversity and www.csr.city.ac.uk/security.

2.4 Design and Development of Mixed–Criticality System Applications [UNIVAQ]

UNIVAQ will contribute to the specification of Safety–Security–Performance (SSP) models and properties, to the choice of quantitative metrics for SSP, and to the selection of the appropriate methods to evaluate them. In particular, considering the criticality levels associated to system tasks and industrial standards related safety assessment processes (i.e. *Development Assurance Level*, DAL, in aeronautics and avionics domains; *Safety Integrity Level*, SIL, for electrical controls; *Automotive Safety Integrity Level*, ASIL, for road vehicles standard), the mixed–criticality systems general problem related to the assurance level of space and temporal isolation will be analyzed with focus on different platform architectures (single/multi/many-processor systems, dedicated HW systems). Related to specific architectures, possible HW/SW solutions and technologies to guarantees isolation and safety–security performance among criticality levels will be considered.

[3] A particular throughput metric – number of New-Order transactions per minute (tpmC) – is the principal measure of performance in TPC-C.

In fact, the successful integration of various critical tasks depends on isolation mechanisms that allow creating multiple partitions with a strict temporal and spatial separation. According to this approach, subsystems with different levels of criticality can be positioned into different partitions, which can be verified and validated in isolation:

- Temporal isolation ensures no interferences in the time domain to access shared resource;
- Spatial isolation protects memory space so that it cannot be accessed by not authorized tasks or functions.

In case of single-processor embedded systems is crucial to ensure temporal isolation between tasks. A single–processor platform can be viewed as a system with TDMA, in which resources are assigned to each task in different time instants. In the case of multi/many–processor systems, the hypothesis of sequential execution is no longer valid, since different applications run in parallel on different processors. Applications must compete to access shared resources, using different communication architectures. In such a context, it is possible to consider different separation techniques related to final implementation.

Considering direct HW implementation, temporal isolation is guaranteed by HW scheduling solution meanwhile spatial isolation is guaranteed by allocating tasks on dedicated components (HW ad hoc, FPGA etc.). On single core solutions, scheduling policy with OS/RTOS/HVP shall ensure correct tasks execution while MMU, MPU and/or HVP partitioning techniques provide data coherence. When considering multi/many–core architecture, the scenario is more complicated and the use of advanced communication technologies and scheduling policies is needed.

In such a context, UNIVAQ effort will be applied in the joint analysis of performance and safety. In particular, UNIVAQ will

- Adapt an existing “system–level HW/SW co–design methodology for performance and reliability” by integrating it with other model-based approaches;
- Provide support to fill the possible gap between too much abstract model-based methodologies and real HW/SW development tool–chains suitable for the target platform and able to satisfy relevant standards;
- Define a design space exploration (DSE) methodology targeted to multi-core mixed-criticality systems that exploit hypervisor technologies. UNIVAQ will focus on PikeOS to apply and validate the proposed methodology;
- Compare different hypervisor technologies (e.g. PikeOS vs Xtratum on quad LEON3/LEON4, etc.) and mixed–critical NoC technologies by means of benchmarking activities.

Results related to the previous listed topics will be applied to two use cases of the AQUAS Project.

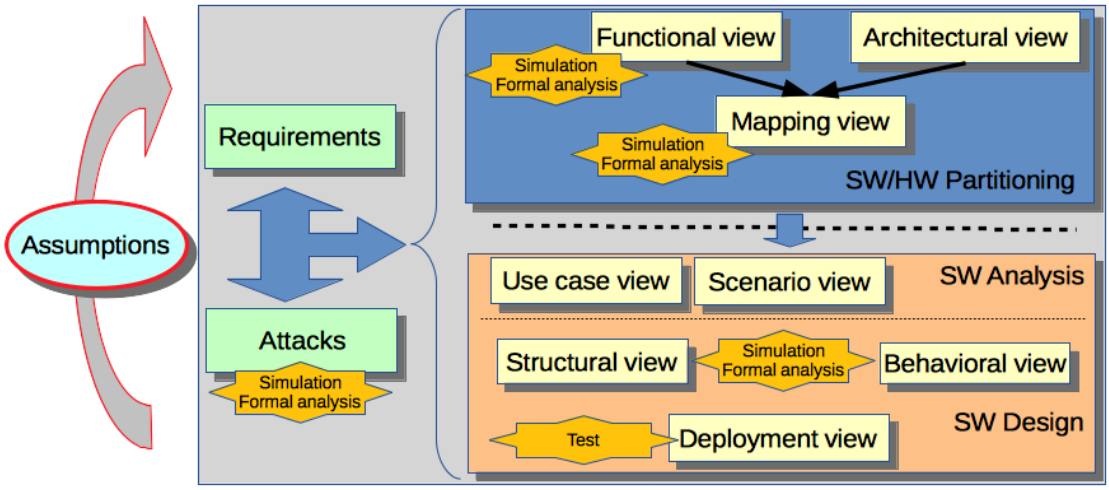
2.5 Security Problems of Fault Detection and Isolation [UNIVAQ]

UNIVAQ will develop interdisciplinary techniques across machine learning and control theory for addressing security related problems of fault detection and isolation in the Clearsy use case. In particular, according to technical details of the use-case that will be provided by Clearsy, we envision to exploit algorithms on Regression Trees and Gaussian Processes adapted with classical residual generation and fault diagnosis techniques in control theory.

2.6 SysML–Sec [MTTP]

SysML-Sec (sysml-sec.telecom-paristech.fr) [10] is an environment to design safe and Secure embedded systems with an extended version of the SysML language. SysML-Sec targets both the software and hardware components of these systems with two main modeling phases: HW/SW partitioning and embedded software design. SysML-Sec is fully supported with the free and open-source toolkit TTool (ttool.telecom-paristech.).

The methodology of SysML-Sec is as follows (see the figure below).



- *Requirement capture*. Requirements are captured with SysML requirements diagrams. Requirements can be tagged as functional, as security requirements (confidentiality, authenticity, integrity, etc.), or as performance.
- *Attack capture.* Enhanced attack trees can be captured using customized SysML parametric diagrams. Requirements and attacks can be related to each other, so as to easier trace if objectives have been set up to counter attacks. Usual attack trees can connect attacks of different levels using or/and operators. Our attack trees offer also sequence, before and after operators. Last but not least, our attack tree diagrams are formally defined: this makes it possible to perform reachability analysis on e.g. root attacks. Thus, when an attack is disabled thanks to a given countermeasure, it is possible to compute if the root attack can still be performed, or not.
- *Hardware/Software Partitioning.* The partitioning stage consists of identifying a good repartition of functions between hardware and software implementations. This stage has three sub-stages:
  - In the first one, the application is defined in a functional way using SysML block diagrams (for the definitions of functions) and with SysML state machine diagrams (for the behavior of functions). Safety and security properties can be explicitly expressed in this functional view, and formally verified using model-to-uppaal transformation. TTool hides this transformation, and backtrace verification results directly on the SysML diagrams.
  - The second stage consists in capturing the candidate architectures using UML deployment diagrams. Hardware nodes are highly abstracted. They consists in processing nodes (CPU and its Operating System/middleware, hardware accelerators, DMAs), communication nodes (buses, bridges, NoCs) and storage nodes (memories). These nodes can be

parametrized with a set of functional (e.g. pipeline size for the processor) and performance parameters (e.g. cache miss rate for the processor).

- The last stage consists of mapping functions (and their communications to the candidate architectures. Then, fast simulation techniques and formal verifier help determining the relevance of the selected mapping with regards to safety, security and performance properties. Usually, there is an iteration there between attack trees, requirements and mapping. Indeed, a given mapping could resolve natively security properties – for example, an attacker cannot spy on a bus, and so confidentiality on that bus is intrinsic, thus making some attacks not possible. This stage also includes two additional features. First, it is possible to perform design space explorations automatically, that is searching for an optimal mapping that respects all safety and security properties while offering the "best" performance. Second, once a mapping has been selected, TTool can automatically generate all the security mechanisms – including hardware accelerators if asked for by the design – in order to fulfil all security properties (security by design).

- *Software design.* In this last stage, the usual V-cycle for developing software can be performed with the support of usual SysML diagrams: software analysis (use case, scenarios, activities), software design (with block and state machine diagrams), and finally software deployments. Again, the particularity in TTool is the ability to perform formal verification from SysML diagrams. Safety verifications can be performed either with internal provers, or with UPPAAL. Security verifications rely on model-to-proverif transformations. Performance evaluation can be performed with code generation from models. Generated code can be executed in SystemC cycle-accurate-bit-accurate simulator.

Last but not least, models are performed from assumptions. Assumptions are commonly ignored in modelling processes. Nonetheless, TTool offer a Modelling Assumption Diagram in which assumptions on the environment and on models can be explicitly captured. Assumptions can also concern properties of the system e.g. assumptions on attackers capabilities. Finally, when refining SysML-Sec models, assumption diagrams have explicit operators to trace assumptions according to the modelling version (first model, refinement #1, etc.).

2.7 Support for Real Time Software Development using CHESS [INT]

CHESS [CHESS], [CHESS2] provides a model-driven, component-based methodology and tool support for the development of high-integrity software systems for different domains.

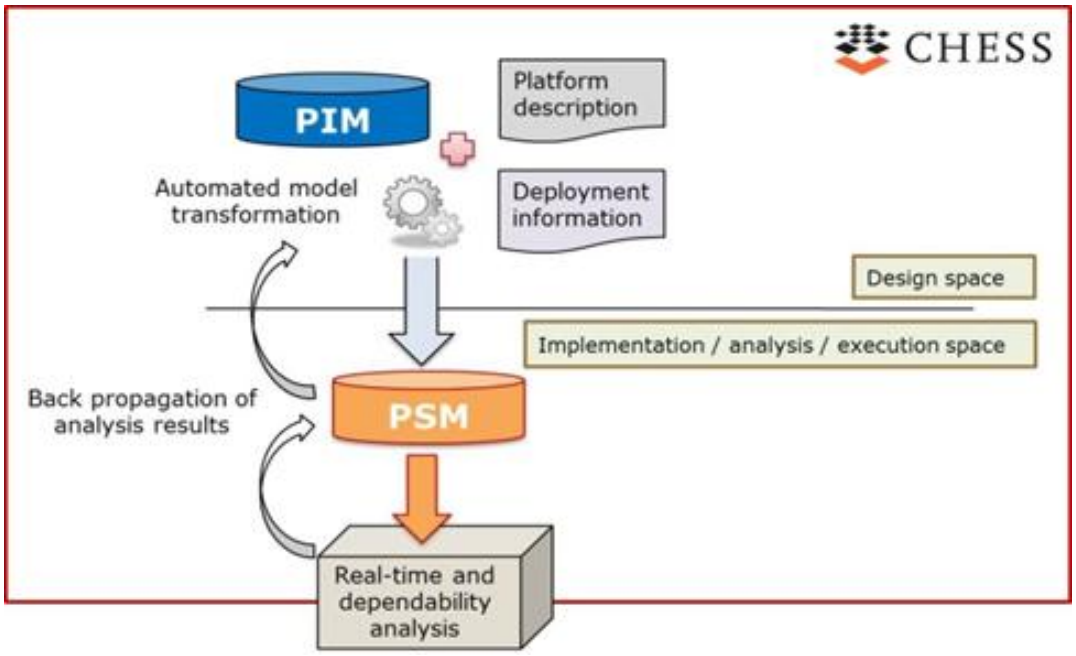
CHESS defines a dedicated MARTE, UML and SysML profile [CHESSML] to address solutions to problems of property-preserving component assembly in particular for real-time embedded systems, and supports the description, verification, and preservation of real-time properties (like sporadic/periodic activation patterns, worst case execution time, deadline) of software components at the level of component design down to the execution level. CHESS is particularly suited for space systems and industrial domains.

The CHESS methodology and supporting toolset provides a disciplined approach to mastering complexity by enforcing separation of concerns between the functional and the extra-functional (e. g. real-time) dimensions, based on the adoption of a Component Model. In the CHESS approach components at design level encompass functional concerns only, in particular, they are devoid of any constructs pertaining to tasking and specific computational model concerns. The declarative specification of non-functional attributes of components (for instance the real time activation pattern of a given provided operation) is used in CHESS for the automated generation of the

container, to be regarded as a component wrapper responsible for the realization of the non-functional attributes declared for the component to be wrapped. Components (and containers) in CHESS are also independent of communication concerns, which are handled by dedicated connectors.

The CHESS component specification, as defined at user level in terms of functional behavior and non-functional property specification, is therefore completely platform-independent: it represents the Platform Independent Model or PIM. Containers and connectors, instead, are platform specific and form the Platform Specific Model or PSM.

The overall CHESS development process is sketched in the following figure.



The overall CHESS development process

The automatic generation of the PSM from the PIM is achieved in CHESS thanks to the application of the CHESS component model and to the implementation of model-to-model transformations. Model-to-code transformation engines derive then correct-by-construction implementation from the platform-specific-model to the specific target execution platform.

For schedulability analysis, the PSM includes the schedulability analysis model (SAM), the analysis is executed by using the open source tool MAST, a scheduling and timing analysis tool developed and maintained by the Universidad de Cantabria [MAST].

The CHESS toolset [CHESS-Polarsys] is part of the Polarsys Eclipse Industry working Group initiative and provides an integrated framework to support the modeller through the whole development process, following the CHESS methodology, from the definition of requirements, to the modelling of the system's architecture, down to the software design and its deployment to hardware components.

It also offers support for real-time analysis as well as code generation functionality to automatically generate the infrastructure code needed to implement the non-functional properties defined in the model.

CHESS tooling extends Papyrus editor to properly support the CHESS methodology, in particular allowing working with different views on the model including requirements, system, components, deployment and analysis view.

The CHESS tool environment is composed by:

- A MARTE, UML and SysML profile (CHESSML);
- An extension to the Papyrus UML graphical editor that supports the notion of design views;
- A model validator that assesses the well-formedness of the model before model transformations can be undertaken;
- A set of model to model and model to text transformations, the former for the purpose of model-based analysis, and the latter for code generation toward multiple language targets (currently Ada only is baselined).

Real-time analysis is performed in the CHESS toolset thanks to its integration with an extension to the MAST engine [MAST], making it possible to perform schedulability analysis and end-to-end response time analysis for single core, distributed and multi-core architectures.

2.8 Support for Contract-based Design using CHESS [INT]

Contract-based reasoning [CONTR1], [CONTR2], [CONTR3], [CONTR4] is also supported by the CHESS toolset. CHESS provides a profile [CHESSML] allowing contract-based design and dedicated model transformations enabling seamless integration with external tools for the verification of contracts specification.

Component properties are formalized in terms of contracts, composed of assumption and a guarantee models as formal properties, where the assumptions are a constraints on the component's environment or usage, and guarantees are properties that must be satisfied by the component - provided that the environment satisfies the assumptions.

Contracts can be used at all stages of system design, from early requirements capture, to any level of system architecture and detailed design, involving software and hardware, and used to formalize constraints and assumptions for any system element at any level of system decomposition, as well as conditions for correctness for their integration.

The CHESS extended methodology introduces stepwise refinement, where the de-composition of a component is accompanied by the decomposition of its contracts, as a central activity in the development process. Stepwise refinement is subject to formal verification and is a key point in the overall verification process as in [FoReVer].

Support for modelling of contracts and for stepwise refinement is provided in the CHESS toolset. Formal verification of the contract refinement is performed by integration with OCRA (Othello Contracts Refinement Analysis) [OCRA] by Fondazione Bruno Kessler.

This extended methodology can be exploited at its best if a library of standard qualified components with associated contracts is available. In the top-down modelling process, a library of components represents a bottom-up driver to ensure convergence to a feasible solution based on the reuse of possibly certified components.

CHESS is currently the subject of extension and adaptation in the context of the AMASS ECSEL project. The goal of AMASS[4] is to create an open tool platform, ecosystem, and self-sustainable community for assurance and certification of Cyber-Physical Systems for different domains of interest. In particular, the project will investigate how the usage of CHESS, that is, its contract-based component model, verification and code generation features, can enable architecture-driven assurance support. For this goal, the CHESS tool is currently under improvement for what regards the language support for contract-based design and the related editor capabilities, e.g. by providing new views dedicated to the contracts modelling. Moreover new analysis (still based on top of formal tools, like OCRA) will be made available to validate the component-based architecture with respect to the components contracts. Examples of such analysis are the validation of components assembly by checking the associated contracts (i.e. by checking the compatibility between contracts assumptions and guarantees of the assembled components) and the validation of components contracts with respect to the components behaviour, the latter provided through state machine diagrams. In addition, automatic support for safety analysis based upon contracts specification will be also provided: further information about this is provided in section 2.9.

2.9 Support for Safety and Dependability using CHESS [INT]

Dependability is a further dimension of interest to CHESS, in the exploration of extra-functional concerns.

CHESS addresses the description and verification of system and component dependability properties (like fault, error, failures and failures propagations) through a dedicated UML profile for dependability [CHESSML] and seamless integration with embedded and external dependability analysis tools.

The CHESS profile for dependability is used to enrich functional models of the system with information regarding its behaviour with respect to faults and failures, thus allowing properties like reliability, availability, and safety to be documented and analysed.

Currently CHESS is integrated with tools enabling Failure Logic Analysis (FLA) and State Based Analysis (SBA).

The CHESS extension for FLA [CHESS FLA], [CHESS FLA2] allows users (system architects and dependability engineers) to decorate architectural models (specified using the CHESSML extended with the dependability profile) with dependability-related information, execute Failure Logic Analysis (FLA) techniques, and get the results back-propagated onto the original model. FLA is a compositional technique to qualitatively assess the dependability of systems, building on top of Failure Propagation Transform Logic (FPTC) and partially combining and automatizing traditional safety analysis techniques (i.e., FMEA and FTA).

FLA may be exploited to support also security analysis, Intecs plans efforts in AQUAS to add dependability threats as failure types.

State-based stochastic dependability analysis starting from a high-level description of the system architecture in a UML-based language. With the term state-based analysis we refer to model-based dependability evaluation using state-based stochastic methods (e.g., see [22]). In such methods, a stochastic model of the system is constructed, describing its states and the possible transitions

[4] https://www.amass-ecsel.eu/

between them. State-based stochastic models are used to evaluate different kind of properties, including performance, reliability, and availability.

The CHESS "State-based analysis" plugin (CHESS-SBA) [CHESS-SBA] allows users to perform quantitative dependability analysis on models specified using CHESS-ML and enriched with quantitative (i.e., probabilistic/stochastic) dependability information, including failure and repair distribution of components, propagations delays and probabilities, and some fault-tolerance and maintenance concepts.

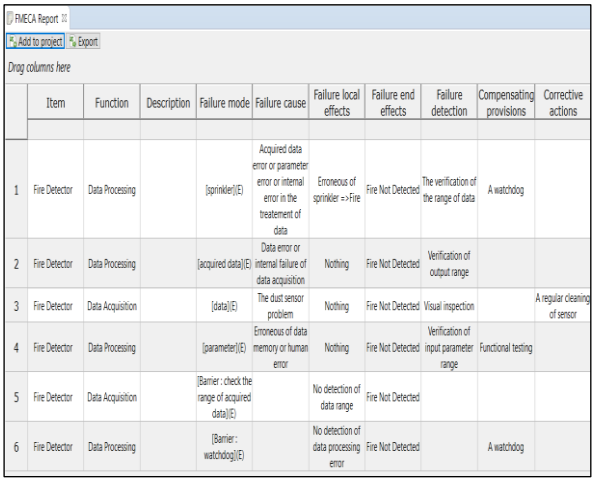
Such information is added to the functional model in two main ways: i) with a set of stereotypes that allow simple attributes to be attached to software and hardware components [CHESS-SBA2], or ii) with an "error model", i.e., a particular kind of State Machine diagram in which a more detailed failure behaviour of a component can be specified.

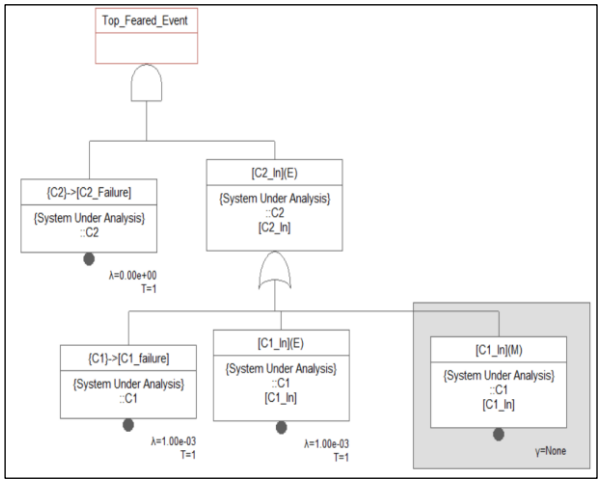
Such enriched model is transformed to a stochastic state-based model, which is then analysed to evaluate the degree of satisfaction of system-level dependability attributes, in the form of probabilistic metrics. As opposed to combinatorial models like Fault-Trees, state-based methods like Stochastic Petri Nets (SPNs) are able to take into account complex dependencies between events.

In addition to the aforementioned dependability profile and analysis, in CHESS the information about the components contracts can also be used to enable safety analysis (the so called contract-based safety analysis [CONTR5], ongoing in the context of the AMASS project). This analysis is based upon the identification of the component failures as the failure of its implementation in satisfying the contract. When the component is composite, its failure can be caused by the failure of one or more subcomponents and/or the failure of the environment in satisfying the assumption. This analysis produces a fault tree in which each intermediate event represents the failure of a component or its environment; the top-level event is the failure of the system component, while the basic events are the failures of the leaf components and the failure of the system environment.

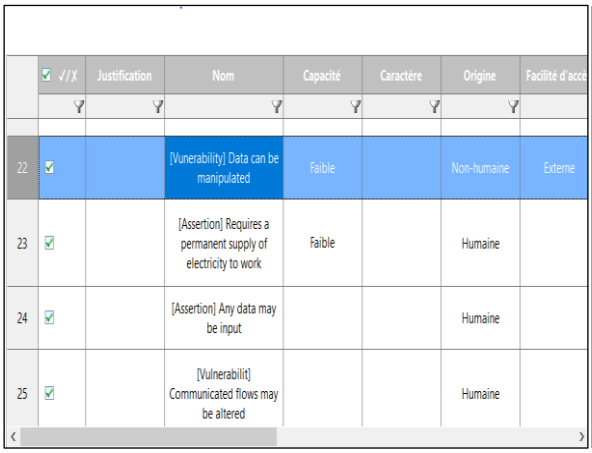
2.10 Safety and Security co-engineering with Safety Architect and Cyber Architect [A4T]

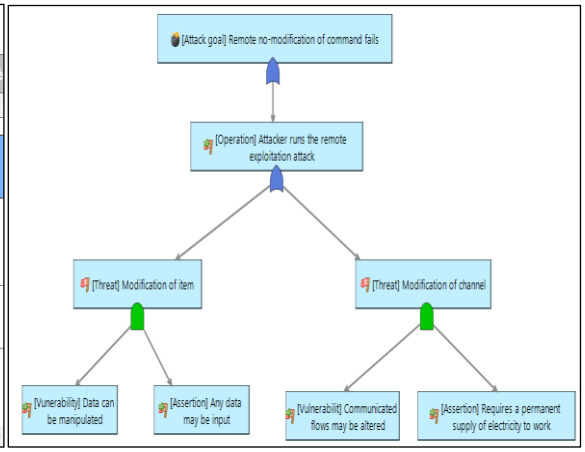
Background on Safety Architect (SA). SA is initially dedicated to perform classical FMECA and FTA by generating Fault Tree from system model (Capella, Papyrus, Rapsody and System Architect models) [Safety Architect]. Thanks to the MERgE project the classical FT can be enriched with malicious events, which can be caused by an attacker [MERgE_A4T]. Examples of FMEA and FT extended with a malicious event are shown in following figures.



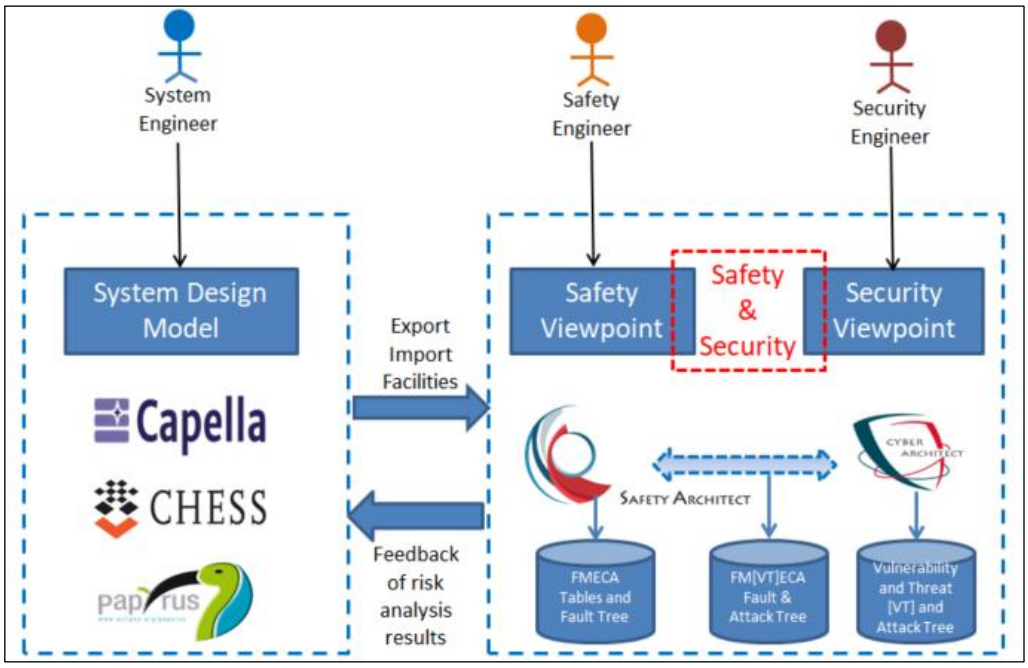


Background on Cyber Architect (CA). CA is a security analysis tool based on the EBIOS method (Expression of Need and Identification of Security Objectives) used to assess and treat risks [EBIOS]. The tool implements the five modules of EBIOS method (Module 1 - Study of the context, Module 2 - Study of the feared events, - Module 3 Study of threat scenarios - Module 4 study of the risks, Module 5 -Study of the control). For example, the objective of module 2 is to systematically identify generic scenarios and feared events that need to be avoided within the study's boundaries. From this systematic analysis, threat sources table and attack tree can be automatically generated in the tool as illustrated bellows.





Proposed Methodology. ALL4TEC proposes to extend [Clarity_A4T, MERgE_A4T] methodologies to go towards an enhanced system, safety and security co-engineering method. Concretely, we propose to use of partner's tools based on UML/SysML language, such as Capella, Papyrus or Chess, for system modelling and to exploit the bridge between these tools and All4TEC tools (Safety Architect and Cyber Architect) for Safety and Security co-analysis.



Firstly, the seamless interoperability between tools allows decoupling the system architecture model from safety & security views, so each engineer (System, Safety or Security Engineer) can solely focus on his concerns, with dedicated tools and terminology. For example, Safety Engineer could use an interface between Capella and Safety Architect to generate system or component level fault trees or FMECA tables. Secondly, the development of a specific-concern viewpoint in a tool dedicated to another concerns allows co-engineering between these concerns. For instance, with the Security viewpoint in Safety Architect, Safety engineer can use the results of security analysis realised in Cyber Architect (e.g., Attack Tree) to generate a merged Fault tree and Attack Tree for assessing the influence of security-relevant events on safety top event.

The objective of this methodology is to define Safety or Security requirements with respect of performance. Consequently, the approach can exploit the performance functionalities of dedicated tools to analysis the impact of safety and security requirements onto performance issues. For example, system design models enriched in Safety Architect or Cyber Architect with Safety or Security requirements can be imported in these dedicated tools (for example, CHESS supports End-to-End Response Time Analysis, which can be used for the performance of components interactions). Performance analysis with respect of Safety or Security requirements could lead to the definition or detailed technical Safety or Security requirements, which can in turn be exported back to Safety Architect or Cyber Architect.

2.11 Security aspects of software development in a product life cycle [TrustPort]

Industrial systems as well as ICT systems are vulnerable from different points of view in cyber and physical security matters, for example an internal and external communication protocols, a third party embedded component security or externally accessible physical ports.

To prevent attacks on the industrial systems it's necessary to define and follow, in its whole life-cycle, all security rules and recommendations defined by norms and the best practises.

Trustport's aim is to define general set of security requirements with respect of safety and performance, define threat model for the UCs and create a simple tool for implementing the security requirements in a life-cycle of selected UC's – Secure Software Development Life Cycle tool (SSDLC)- and create a CVSS system, scoring severities of potential threats with the aim to suggest it's mitigation. SSDLC approach brings several security aspects into development life cycle and provides:

- General security controls for information systems for effective risk management;
- Flexible catalogue of security controls to meet the security threats, requirements and technologies;
- Involving security activities into the whole product lifecycle (creating security requirements, verification of analysis and design from security point of view, developer guidelines, code reviews, verification of security requirements, security assessment);
- Based on generally accepted standards (NIST, OWASP, Microsoft SDL) to treat the various attack vectors;
- Measurement metrics for security control effectiveness.

SSDLC tool will help to create also security, hardening, testing, and validation reporting guidelines for selected UCs.

Part of the process will be definition of common vector of attacks with help of previously gained experience with building tools (Automated Intrusion Prevention System - AIPS) for detection of intrusions, based on network traffic analysis. This approach was tailored for general use on any data (NetFlow, network data, general files, logs and configuration).

The aim of Automated Intrusion Prevention System (AIPS) systems is the detection of network attacks using behavioural characteristics of network communication. In [Barabas 2012], [Barabas 2013] we proposed an idea of framework architecture that would be used for detection of various network threats. The papers presented the novel AIPS which uses honeypot systems for the detection of new attacks and the automatic generation of behavioural signatures based on network flow metrics. The detection method is based on extraction of partial communication and creation of behavioural signature from the network flow by previously defined set of network metrics. We have successfully experimented with the architecture of the AIPS system and we defined 167 metrics divided into five categories according to their nature. These metrics are used to describe properties of detected attack not upon the fingerprint of common signature, but based on its behaviour. Metrics are formally specified and extraction of them can be generally realized for each data flow. The specification includes statistic, dynamic, localization and especially behavioural properties of network communication. For the learning phase of classification, we used simulated and captured set of buffer overflow attacks and new attacks extracted from shadow honeypots deployed in protected network.

Defined network metrics were created from general network communication definition and by applying fundamental mathematical rules. Afterwards, these metrics were tailored for TCP/IP communication with together 657 numerical values that were used as input metrics for reduction algorithms, e.g. Principal Component Analysis. Extracting the metrics information from network traffic we assembled detection engines based on machine-learning techniques (e.g. SVN, Decision Trees or Naïve Bayes Classifier). We were able to achieve 99,76% precision of detecting known attacks.

Based on metrics definition, this method can be further applied for defined states of protocols, communication, system configurations, logs, etc. As a one of deliverables, we will use these methods to create metrics for detection of defined known attacks on network and system level.

2.12 System assurance analysis method [Tecnalia]

According to OMG SACM (Object Management Group, 2017), specification:

Systems Assurance is the process of building clear, comprehensive, and defensible arguments regarding the safety and security properties of systems. The vital element of Systems Assurance is that it makes clear and well-defined claims about the safety and security of systems.

In this context, Opencert (Eclipse, 2017) will be used as a tool support for system assurance analysis during AQUAS project. This tool has been used in several domains such as automotive (Larrucea et al., 2017). Figure 1 and Table 1 provide an overview of the Opencert functionalities which should be extended and adapted to AQUAS scenarios. From a methodological point of view in AQUAS project we are going to be focused on Assurance Case Specification, Evidence management and Compliance Management.

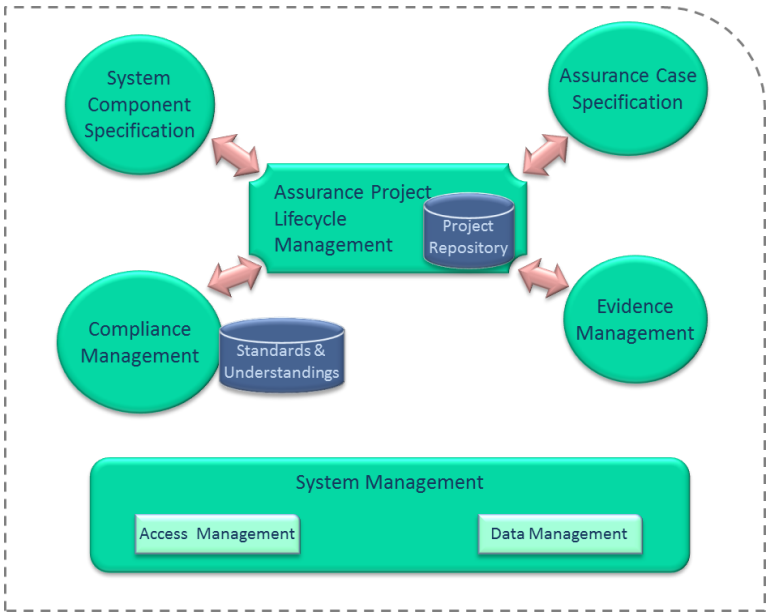


Figure 1: Opencert functionalities overview for system assurance

Functionality Group	Description
System Component Specification	This group manages System architecture specification by decomposing a system into components. It also includes mechanisms to support compositional assurance, contract based approaches, and architectural patterns management.
Assurance Case Specification	This group manages argumentation information in a modular fashion. It also includes mechanisms to support compositional assurance and assurance patterns management.
Evidence management	This module manages the full lifecycle of evidence artefacts and evidence chains. This includes evidence traceability management and impact analysis.

Compliance Management	Functionality related to the management (edition, search, transfer, etc.) of process and standards' information as well as of any other information derived from them, such as interpretations about intents and mapping between processes and standards. This functional group maintains a knowledge database about "standards & processes".
Assurance Project Lifecycle Management	This functionality factorizes aspects such as the creation of locally assurance projects. This module manages a "project repository".
Access Management	This is an infrastructure functional module. It includes generic functionality for security, permissions, and profiles.
Data Management	This is an infrastructure functional module. It includes generic functionality for data storage, visualization, and reporting.

Table 1. Summary of functional groups

2.12.1 Assurance cases specification

Assurance cases are used in safety critical applications to ensure that a system's functionalities are analysed appropriately. Each assurance case is composed of arguments and evidences that are used in order to demonstrate that a product or process is safe. Assurance cases are a structured form of an argument that specifies convincing justification that a system is adequately dependable for a given application in a given environment. Assurance cases are modelled as connections between claims and their evidence. Ideally, they should provide for every possible instance in regards to dependability of a system, and give proof through justifications and evidence that a system is indeed dependable (safe, secure, reliable, and the like).

Typically, assurance cases are used to describe and analyse these kinds of systems (Ayoub et al. 2012), especially during the certification process (Dodd and Habli 2012; Hawkins et al. 2013). Goal structuring notation (GSN) (Spriggs 2012) is used to graphically describe safety cases (Piètre-Cambacédès and Bouissou, 2013; Redmill and Anderson, 2008; Zeng et al., 2012). Risks are a fundamental concept (Piètre-Cambacédès and Bouissou, 2013) and they should be managed jointly with assurance case elements, especially arguments and claims. There is a wide range of critical applications where software plays a relevant and key role, and where one needs to analyze safety conditions. Avionics (Hawkins et al., 2013) and automotive (Törner and Öhman, 2008) are some of these critical applications where assurance cases are carried out. But there are other domains where one might need to potentially apply assurance cases in order to prove that a concrete situation is safe.

It is unlikely to be feasible (or desirable) to fully automate and formalise an assurance case because arguments are inexact, and dependability is not a property that can be fully quantified. Therefore, human review and judgement is ultimately required to assess whether a system is "acceptably dependable", and this stems from an understanding of the argument and compatibility of evidence.

Figure 2 represents a use case for system assurance where assurance cases are defined, arguments and evidence are identified, and finally processes are generated and argumentations are monitored.

2.12.2 Structured Assurance Case Metamodel

The SACM, released in February 2013 (Object Management Group, 2017), is a metamodel for assurance cases widely used in safety critical systems. The main purpose of this metamodel is to

make clearer claims and reasoning between claims. Additionally, an assurance case is structured with arguments and evidences.

Assurance cases have an important role when developing safety critical applications because they are considered the main proof for considering an artefact safe. In addition, assurance cases have a clear role in certification processes (Wassyng et al., 2011). Figure 5 provides an overview of the basic elements in an assurance case that inherit from a model element: argumentation is a textual reasoning about a fact, and EvidenceContainer represents the elements supporting an argumentation. This assurance case used to contain a set of argumentations and evidences validating safety goals. This figure represents a chunk of the SACM (Object Management Group, 2017) at a high level. All these elements constitute the aforementioned main proofs.

GSN is a notation for describing assurance cases. SACM and GSN are aligned at the Object Management Group's technical groups. Basically, SACM provides a foundational background to GSN notation. Goal, strategy, and solution concepts are some of the most relevant elements in GSN (Kelly and Weaver, 2004). With these elements one can argue that a goal can be satisfied if there is an evidence or a set of evidences supporting an argumentation. Obviously, a specific strategy is used in a specific context. Industrial experiences on using safety cases and GSN have been already reported in (P. Chinneck et al., n.d.), and GSN has been evaluated to be used also in software safety certification (Hawkins et al. 2013).

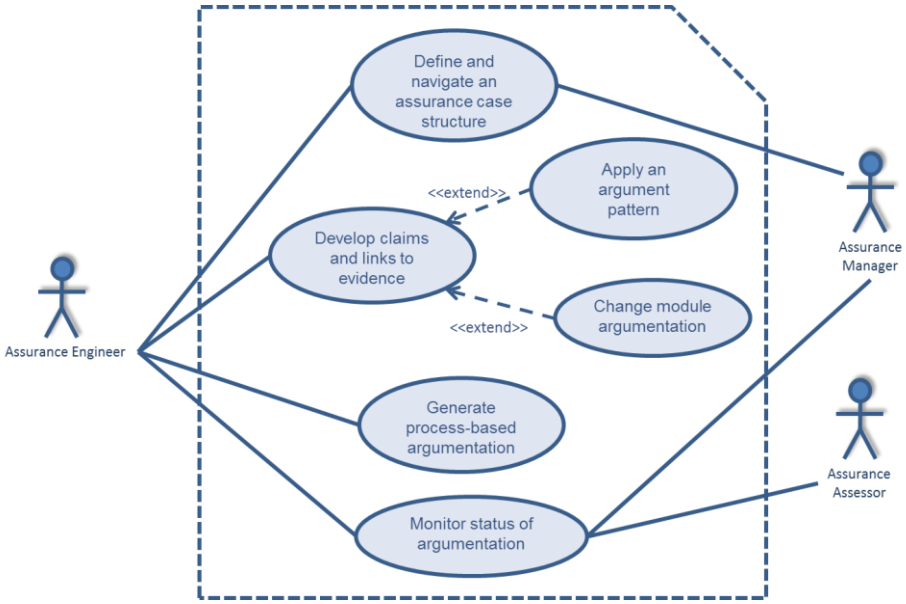


Figure 2: Use Cases for "Assurance Case Specification" module

2.12.3 Evidence Management

The use cases for the Evidence Management model deal with those basic aspects related to the specification of the information of the actual artefacts that can be or are used as assurance evidence in an assurance project. These artefacts are called "Managed Artefact". Managed artefacts can have specific properties (e.g. the result of a test case), be stored in an external tool (e.g. DOORS for a requirement), and be related between them (e.g. the test case that validates a requirement). When a managed artefact changes, the change could affect other managed artefacts and thus a change impact analysis is usually necessary. All these aspects are managed through a managed-artefact lifecycle, which corresponds to the changes to a managed artefact (e.g. creation and modifications)

and can include evaluations (e.g. about the completeness of a document). Process information about the execution of an assurance process can also be associated to managed artefacts, such as the technique used for the creation of a managed artefact or the participant (i.e. person) that is the owner of a system specification. Figure 3 and Figure 4 depict the main activities to be carried out during the evidence management.

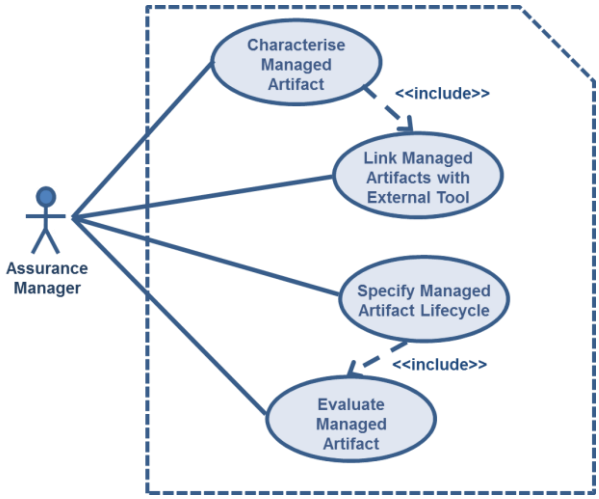


Figure 3: Use Cases for "Evidence Management" module (1)

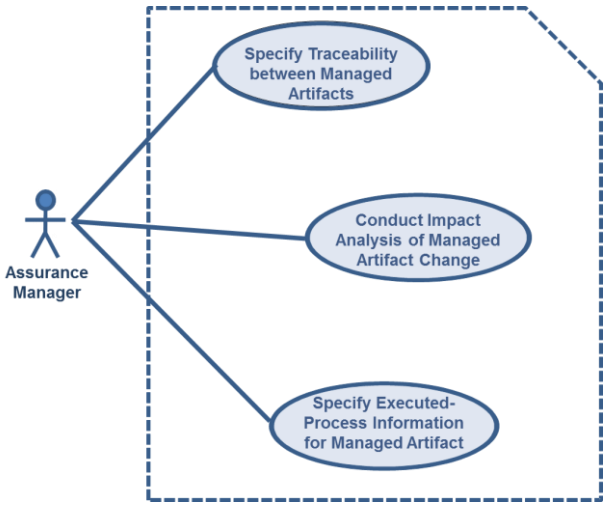


Figure 4: Use Cases for "Evidence Management" module (2)

2.12.4 Compliance Management

The uses cases for the Compliance Management model capture information about how information of standards is captured, managed and monitored in the context of an assurance project. The tool should provide to the Assurance Manager mechanisms to model all the information that is contained in a standard, which includes the life-cycle defined on it, requirements and recommendations. Additionally, this building block makes possible the management of the assurance project, which implies the modelling of the baseline and the mappings for compliance and equivalence. The monitoring of the assurance project will be provided by the third functionality of this building block. For this prototype, the monitoring will be performed by the generation of a compliance report. Figure 5 identifies the main activities to be performed for compliance management.

2.13 Tooled Approach for Performance Evaluation Before Implementation [TRT]

Efficiently exploiting the hardware resources of parallel platforms to reach significant performance becomes a real challenge when facing heterogeneity of underlying accelerators (e.g. the Zynq Ultrascale+ MPSoC includes different parts: GPP, GPU, FPGA), memory/interconnection hierarchy, diversity of related programming languages, etc.

Implementing a computing-intensive application on such parallel machines requires choices to map given functions on some given hardware resources, then adding communications between them and scheduling them in a proper way in order to get a valid and efficient generated parallel code.

This choice problem is known as design space exploration. Usually it is manually done according to the user experience. It means that a few mapping choices are experimented depending on the granted time budget.

That is the reason why a tooled-up approach is required now. It enables exploring new mapping choices before implementation that is an expensive step, especially with some complex accelerators to be programmed (e.g. FPGA).

The approach consists of:

1. Choosing a relevant mapping;
2. Extracting the related scheduling activities (both computations and communications);
3. Using them to feed an architecture simulator that matches the hardware platform (assumption: the most characterising points are faithfully modelled) and behaves as such;
4. Analysing the underlying traces (Gantt chart) for real-time performance purpose;
5. Iterating on step 1 till an acceptable performance is reached;
6. Implementing on the actual parallel platform using the last results in terms of mapping / scheduling.

It saves time and money because implementation complexity and related debug are taken into account only once.

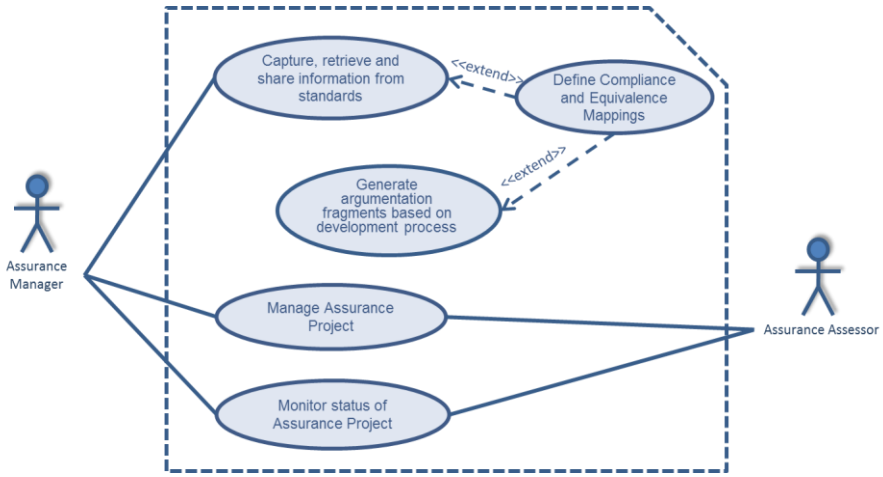


Figure 5: Use Cases for “Compliance Management” module

2.14 Performance Analysis Through Design Architecture [TRT]

Tempo Verifier is tool developed at TRT since 2015 and its aiming to verify timing performance of design architecture through models. The Tempo Verifier uses Time4Sys Models as input models.

Some definitions are given hereinafter.

• Execution Time: Execution Time is the time taken by the CPU for executing a sequence of code. The time taken by interruptions is not taken into account. In practice, it is difficult to evaluate execution times because they depend on several factors:

- the processor architecture, its raw performances such as clock frequency, bus frequency, pipelines, cache memory.
- the compiler used, the compiler options
- perturbations induced by other tasks (time for context change, effect on cache memory)
- the compiler used, the compiler options
- the code functional entries (such as data size)

Execution times can be estimated, theoretically calculated or measured.

• Response Time/Latency: Response Time (RT) is time elapsed between the activation and the termination of a functional chain. It depends on the execution time of the different processing involved in the chain, increased with the time taken by interruptions. Response Time is often used for tasks, and latency for chain of tasks.

An iterative approach. To get benefits from the Tempo Verifier methodology, its approach has to be iterative.

- We start from hypothesizes formalized in a model.
- We obtain results with analysis or simulation
- Results are compared with execution traces, when available
- When bottlenecks appears, we can try different mappings or priority assignment

Modelling activity. This activity aims at building a performance model of the system; conform to the standard UML MARTE that holds enough information to allow timing analysis on it. This model is obtained by extracting the needed information from the design model of the system under study. It is composed with 2 kinds of information:

- First, structural information that correspond to physical parts of the system, such as Computing Resource (typically processing unit), and software resources statically allocated on it.

  Scheduling information (scheduling policy, scheduling parameters) is specified on these structural elements.

- Then, behavioral information done through the representation of a particular scenario of execution. These scenarios are expressed in the form of an oriented graph of steps.

  Timing information, such as duration of execution or activation schemas, is specified on the elements of this graph.

  This modelling activity is done in the Time4Sys Framework. Time4Sys model is the input of the TEMP Verifier verification tool.

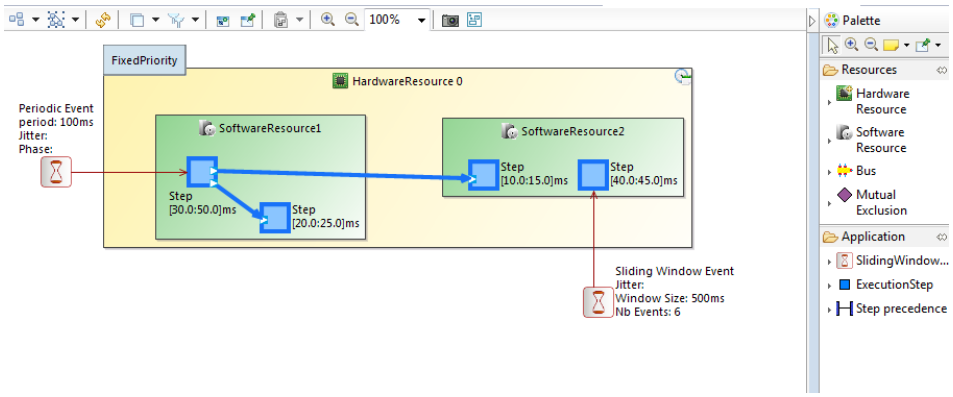


Figure 6: Time4Sys Model

Analysis activity. The analysis activity aims at computing Worst Case Latency. It is achieved with a dedicated tool. First, an adaptation of the model for this tool is done by applying some transformations: the tool creates a new equivalent model but compatible with analysis techniques.

When analysis is too pessimistic or fails, simulation can be a good alternative but offers no guaranty. The analysis is become too pessimistic when it is impossible get a mathematical formula which compute the exact worst case-latency, in such case, the analysis computer a over-estimation of the latency which guarantee anyway the real worst case latency will be below the value computed.

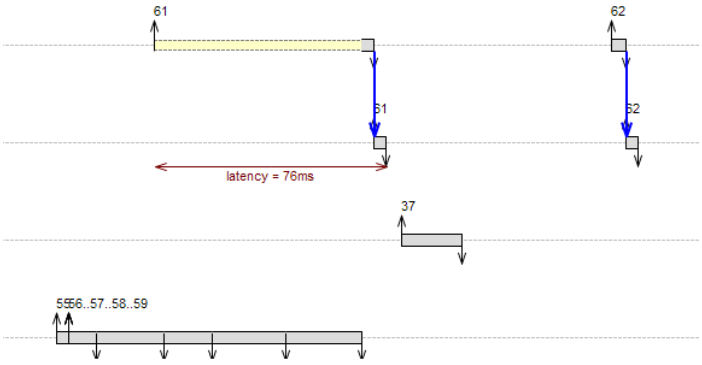


Figure 7: Gantt charts produced after analysis or simulation

Simulation activity. Use to evaluate distribution of the response time (best, worst, mean and more...). It is based on generated random values. It only gives an idea but does not guaranty a critical situation will never happen.

*Results interpretation/Correction activity***.** Results from the analysis/simulation tool have to be converted backward to the modelling tool in order to be interpretable within the original context.

2.15 Tracking the interdependence of Safety, Security and Performance [Magillem]

Magillem Design Services is well established in the semiconductor industry: indeed, more than 80% of tier one SoC designers are our clients. We can mention among others: Samsung, Qualcomm, Intel, ST Microelectronics, Texas Instruments, etc.

Magillem Platform Assembly (MPA) is based on the IP-XACT/IEEE1685 standard which is an XML schema. Magillem has used its expertise in XML to build a powerful Content Assembly Platform (MCP) for advanced publishing and multi-channel delivery of valuable technical, legal, etc., content. Note that it includes Microsoft Word and Excel.

Moreover, Magillem is taking advantage of its XML mastery to link the specification of design elements captured in IP-XACT to a set of XML fragments of documentation content: whenever a design element changes, the user can easily control the propagation of the update to the documentation contents associated with it. The link is bidirectional and therefore offers even greater benefits. It is for instance possible to aggregate the design data with any external product information.

Systems become an XML-based coherent set of hardware, software descriptions and documentation… More precisely, for each fragment of data, metadata defining them are created: these last ones are then linkable to each other and give the possibility to monitor the data.

Magillem is able to propose its solution built on accumulated experience and expertise: **ISDD©**. This tool addresses the co-engineering aspect of ever more complex projects: hence, multidisciplinary teams, even if they are spread out in various sites and using different tools, address the same objects. Impacts generated by any modification are propagated and each concerned actor is automatically notified and informed.

Linking these elements will already provide many benefits:

- Changes will be propagated throughout the whole project;
- The modification of a high-level requirement (or the replacement of an obsolete component, etc.) is detected and impacts are calculated on all linked elements: this may start a domino effect;
- Moreover, all stakeholders having always access to the up-to-date documentation, discrepancies are limited.

Magillem will study the opportunity to create new connectors to third-party tools to reinforce the workflow.

2.16 Dynamic Analysis of Multi-threaded Applications [BUT]

Concurrent, or multi-threaded, programming has become highly popular. New technologies such as multi-core processors have become widely available and cheap enough to be used in common computers. However, as concurrent programming is far more demanding, its increased use leads to a significantly increased number of bugs that appear in commercial software due to errors in synchronization of its concurrent threads. This stimulates a more intense research in the field of detecting and removing such bugs. One of the most common approaches used for uncovering bugs in software is in general testing. However, finding good test cases that will exercise a large enough portion of the behaviour of programs is not easy. That is why, various approaches to automated test

generation are sought, including the use of various advanced search algorithms, searching through the state space of possible test settings, yielding the so called search-based testing. The problem of uncovering bugs in concurrency by testing is in particular difficult since 1) the bugs may manifest only under very special scheduling circumstances, and 2) the root cause of bug in concurrency mostly lies in low-level data manipulation. One of the ways to cope with computation interleaving in a program is to use techniques suitably influencing scheduling, such as noise injection, and then, of course, use repeated test execution [BUT-Atomrace]. To cope with data manipulation, one needs a tool that monitors the execution of a program, usage of synchronisation primitives, and data accesses in the given execution. However, monitoring the execution of a program can be quite challenging and programmers might spend more time writing the monitoring code than by writing the analysis code itself. ANaConDA [BUT-Anaconda] provided by BUT is a framework for adaptable native-code concurrency-focused dynamic analysis built on top of Intel PIN instrumentation tool. The goal of the framework is to simplify the creation of dynamic analysers for analysing multi-threaded C/C++ programs on the binary level. The framework provides a monitoring layer offering notification about important events, such as thread synchronisation or memory accesses, so that developers of dynamic analysers can focus solely on writing the analysis code. In addition, the framework also supports noise injection techniques to increase the number of interleaving witnessed in testing runs and hence to increase chances to find concurrency-related errors.

2.17 Static Analysis of Pointer Manipulating Programs [BUT]

Dynamically allocated memory space and complex pointer manipulations are a frequent cause of nasty errors that are easy to cause but often difficult to discover. They can sometimes manifest only after a long period by a sudden memory protection error or by a shortage of memory. Static analysis is one way how such errors may be detected. However, dealing in a precise way with complex unbounded memory structures is rather difficult. Among the most promising current approaches to this kind of analysis, one can find approaches based on various kinds of logics, automata, or graphs. One of the most efficient analysers for low-level pointer manipulating programs is Predator [BUT-Predator] based on the so-called symbolic shape graphs.

2.18 Static and Dynamic Performance Analysis of Programs [BUT]

Static program analyses targeted at automated derivation of various program complexity measures (resource bounds) is considered to belong among the most demanding kinds of program analyses, which are, however, more and more needed: e.g., due to a pressure on not wasting time or energy by inefficient computations. This area of program analysis is still under very active research. However, many useful kinds of resource bounds analyses and tools based on them have already been proposed. Among such tools, one of the most light-weight, scalable, and practically applicable approaches is the approach implemented in the Loopus analyser [BUT-Loopus] based on difference bounds constraints, identification of program variables representing local bounds of loops, and using transition and variable bounds defining how often and how much the variables can be increased. Loopus was originally designed primarily for programs with integer variables at the Vienna University of Technology and recently extended in collaboration with BUT to handle also pointer manipulating programs in the Ranger tool [BUT-Ranger].

Program performance can, of course, by analysed dynamically too as, e.g., in the Perun framework [BUT-Perun]. Perun provides a generic framework for collecting various performance measures (running time, memory consumption, etc.) for various input data. The obtained measures can be

subsequently analysed by means of statistical methods, allowing one to, e.g., derive performance models of the analysed programs. Moreover, Perun allows one to collect and store performance measures for program versions stored in the git versioning system and to automatically detect and analyse performance-related regressions.

2.19 Efficient Methodologies for the Development of Formally Verified System Software [HSRM]

HSRM is doing research on efficient methodologies for developing formally verified system software. This will result in a foundation of proven common algorithms and a set of best practices for the development and verification of future system software. The methodologies are exemplified by applying them to the development of a formally verified microkernel.

For a better understanding of the meaning of formal verification as opposed to testing both terms shall be defined.

- Formal verification of software denotes the mathematical proof showing that the software satisfies the given specification. Thus, testing an implementation is expendable.
- Testing of software means the systematic checking the response of the software to different stimuli in order to validate the specified requirements. This implies that as long as not all possible cases have been tested, there remains a chance of not detecting existing programming errors. Even for medium complex systems it is practically impossible to enumerate and execute all possible test cases.

Tools can assist in developing mathematical proofs for software. These tools can be classified as either automated or interactive provers. Automated provers attempt to automatically prove that the source code complies with the given specification. If the prover is not able to prove a certain logical expression, then the proof simply fails, whereas an interactive prover prompts the user instead at this point. The user is then requested to prove the assumption manually and let the prover continue afterwards. FRAMA-C [Frama] and SPARK [Spark 2014] are examples of programming and verification environments for C or (a subset of) Ada respectively including automated provers like cvc4 [Stanford] or alt-ergo [Alt-ego]. Typical examples for interactive provers are coq [Coq] or provers based on the Isabelle/HOL [Cl.clam] framework. A typical verification environment consists of a set of provers cooperating via a common intermediate representation based on languages like ML [Why3].

One goal of methodologies to be developed by HSRM is to not be bound to a specific toolset, but to be able to combine different tools and methods in a common framework using and extending the SPARK environment to be able to handle non-functional properties especially specifying and verifying timing constraints.

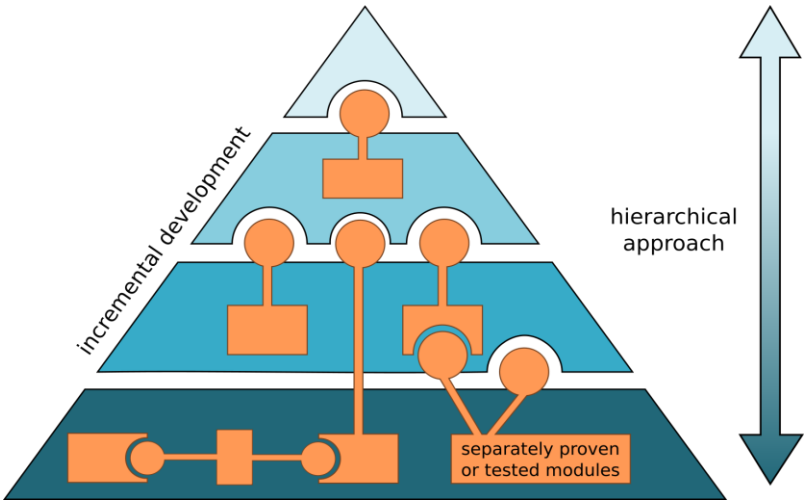
SPARK 2014 is based on a subset of Ada 2012 and is developed by AdaCore [Adacore]. It enables the developer to write software in SPARK along with specification and annotations as part of the source code. The specification is established by phrasing verification contracts as logical expressions.

The SPARK tools transform the contracts and source code into an intermediate ML dialect (WhyML [Why3.Iri]) which then can be passed to an SMT (Satisfiability Modulo Theories [Barrett]) solver to attempt the proof. This makes SPARK a viable tool to write verification contracts and run them automatically through different SMT solvers. Its application requires knowledge and experience in formal verification, in order to be able to phrase the right verification conditions and to write provable source code.

The goal of the methodologies to be developed by HSRM is to provide a common concept for specifying verification conditions and writing provable source code for system software. Furthermore, proven commonly used algorithms for system software will be provided as a library together with corresponding proven lemmas for future projects.

The research topics are the following:

- Improve development and verification efficiency by applying a hierarchical architectural approach;
- Combine proof and test approaches into a common methodology;
- Specify and verifying timing properties.



HSRM uses a hierarchical approach through a layered architecture of the system software. The functionality of each layer is grouped into modules, which are proven separately. Each layer can use all proven modules of the lower layers and extends those layers in terms of functionality by adding additional modules, thus implementing an incremental development approach.
By applying this hierarchical approach to the design of a microkernel, the following layers will be distinguished:

1. Single Tasking System
2. Multi-Tasking / Multi Core System
3. Memory Protection
4. Virtual Memory

Each layer adds additional functionality (and complexity) satisfying additional requirements. The resulting prototypes of layers 2 – 4 are useful operating system kernels on their own right. The microkernel that is developed as an example for the new methods serves as an internal demonstrator for the application of the developed methods.

Modules that cannot be proven (or have not been proven yet) will be thoroughly tested by using the verification conditions for runtime tests, thus combining both methods of formal verification and testing. Unproven, but tested modules can be used if acceptable and can later be replaced by a proven version if such a version becomes available.

Besides the functional proofs described above, the methodology will also enable the developer to integrate timing specifications. To this end, the methodology will make use of WCET analysis tools such as the tools provided by partner AbsInt and temporal proof techniques. HSRM will analyse the integration of standard timing specification methodologies and evaluation methods such as MARTE.

2.20 Probabilistic State–Based Models for Safety, Security and Performance Evaluation [City]

The use of stochastic state based models for evaluation has been used for decades. Many international standards recommend the use of such models. For instance IEC 61508 provides and extensive coverage and examples of how Markov models and Stochastic Petri nets can be used in safety analysis. Queuing system are an example of applying state-based probabilistic models for model based performance evaluation. The use of probabilistic state-based models for security assessment has been gaining popularity in the recent years, too. A recent authoritative survey of the important publications and techniques used for solving state-based models (analytic, numerical and via Monte-Carlo simulation) can be found in a survey by Nicol, Sanders and Trivedi[5].

City have used extensively some of the well-known probabilistic state-based formalisms such as Stochastic Activity Networks (SAN) and semi-Markov models to model resilience of cyber-physical systems (RESS'2017[6]), performance evaluation (QEST'2015[7]) and the dependencies between safety and security (SAFECOMP'2015[8], ISSRE'2017[9], EDCC'2016[10]).

A probabilistic state-based model is defined by a state-machine (a set of states whose semantic is defined for a specific application context) and transitions between the states are governed by probability distribution laws, e.g. probabilities or distribution of sojourn times in a particular state. Some formalisms introduce additional mechanisms to augment the transitions as needed by the application context, e.g.:

- by introducing tokens (in the case of Stochastic Petri Nets (SPS) and Stochastic Activity Networks (SAN) which enable transitions from a particular state/place). A transition will only take place from states holding tokens;
- gates, expressions which control the marking of the places (i.e. the number of tokens in places) in SPS/SAN, etc.

[5] D. M. Nicol, W. H. Sanders, and K. S. Trivedi, Model-Based Evaluation: From Dependability to Security, IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, January-March 2004, pp. 48-65.

[6] Bloomfield, R. E., Popov, P. T., Salako, K., Stankovic, V. & Wright, D. (2017). Preliminary Interdependency Analysis: An Approach to Support Critical Infrastructure Risk Assessment. Reliability Engineering and System Safety, doi: 10.1016/j.ress.2017.05.030

[7] Popov, P. T., Salako, K. & Stankovic, V. (2015). Stochastic modelling for performance evaluation of database replication protocols. *Lecture Notes in Computer Science*, 9259(9259), pp. 21-37. doi: 10.1007/978-3-319-22264-6_2 ISSN 0302-9743

[8] Popov, P. T. (2015). *Stochastic Modelling of Safety and Security of the e-Motor, an ASIL-D Device*. Paper presented at the 34th International Conference on Computer Safety, Reliability, and Security, SAFECOMP 2015, 23-09-2015 - 25-09-2015, Delft University of Technology, Netherlands.

[9] Popov, P. T. (2017). Models of Reliability of Fault-Tolerant Software Under Cyber-Attacks. doi: 10.1109/ISSRE.2017.23 ISSN 2332-6549

[10] Gashi, I., Povyakalo, A. A. & Strigini, L. (2016). *Diversity, Safety and Security in Embedded Systems: modelling adversary effort and supply chain risks*. Paper presented at the Proceedings of the 12th European Dependable Computing Conference, 5th - 9th September 2016, Gothenburg, Sweden.

- additional grouping mechanisms necessary to deal with model complexity, e.g. REP/JOIN formalisms in SAN, which allow for replication of anonymous “atomic” models (REP) and of named “atomic” models (JOIN). The replication is further supported by shared places and activities (shared between several models).

A typical study with the probabilistic state-based models will include a definition of “reward(s)”, e.g. measures (metrics) of interest. One can undertake studies in which various statistics (e.g. mean and variance) can be computed for the chosen reward. The entire probability distribution of the chosen reward can be computed, too, which characterises the uncertainty associated with the value of the reward.

The stochastic state-based models are generic in nature, which makes them suitable for studies focussed on safety, security and performance. The techniques are well supported by a number of tools developed in academia but used by industry, too. For instance SHARPE (by Prof. Kishor Trivedi at Duke University) is a popular choice in academia. It supports semi-Markov modelling and SPS, but was also used by NASA and Boeing in several projects (after demonstration that it is compliant with tool requirements by these organisations, despite not being designed to meet these requirements). Mobius (by the perform group), started as a PhD project by its creator Prof. Bill Sanders, now with the University of Urbana-Champaign, has now gained reputation in academia and industry.

The state-based models have been used for decades for combined analysis, too, e.g. for performance analysis of system subject to degradation (known as performability, i.e. performance under partial failure and repair). In these circumstances, some of the states represent fully operational system, some other states – a partially degraded system (e.g. due to outages). This approach has been recently adapted to modelling “green computing”, where acceptable performance at minimal power is sought. In these circumstances some of the states represent the operation under full power, while other states – the operation under reduced power.

Probabilistic state-based models clearly lend themselves quite naturally for combined analysis of safety, security, and performance, which are the focus of the AQUAS project. In our own work (at City), we have already applied the state-based models for modelling safety and security. A couple of examples are summarised below:

- In SAFECOMP’2015 we studied the trade-off between safety and security of an E-motor, an ASIL-D device with 2-channel software. In this device a cyber-attack (e.g. via the car infotainment system) can compromise the safe-state defined for the device. If such a compromise occurs, an accidental failure (due to either a hardware or a software failure) will lead to unsafe failure of the device. An acceptably low probability of unsafe operation can be achieved by either hardening the E-motor device (i.e. reduce the likelihood of compromising the safe-state) or reducing the time the E-motor device can remain in a compromised safe-state. A compromised state can be restored (“cleansed”) by a suitably designed mechanism (known as “proactive recovery”). Even if the safe state is compromised by increasing the frequency of proactive recovery one will reduce the duration of the compromised safe state and hence the risk that the device will be unable to move to a safe state should a failure occur. Our analysis, in which malicious activities compete in time with proactive recovery, allows one to establish an acceptable trade-off between improved intrusion detection/prevention and proactive recovery. This form of analysis is widely applicable to, including most of the AQUAS demonstrators.
- In ISSRE’2017 we developed a model of the effect of a successful cyber-attack on software reliability of an on-demand 2-channel industrial control software. The view taken in this work is that successful cyber-attacks may reduce software reliability. In the extreme case a successful cyber-attacks on industrial software may lead to an immediate failure, but there is empirical evidence that the most sophisticated attacks are such that they do not cause immediate failures.

Instead, they decrease software reliability without causing an immediate failure. With this model, we conducted a study of the effectiveness of two different maintenance policies – cleansing (also called "proactive recovery", whereby a software channel is periodically "cleansed" by reinstalling from a clean copy) and patching. We also looked at how deferred patching (e.g. patching is delayed until software is detected with certainty to have been compromised) compares with immediate patching (i.e. is applied as soon as a patch becomes available). The study was conducted at a high level of abstraction, similar to what is typically done in the early safety analysis, which makes it applicable to a broad range of specific case–studies, including most of AQUAS demonstrators.

- While the previous two examples demonstrate the usefulness of probabilistic state-based models in combined analysis for safety/reliability and security, in an earlier study we used a sophisticated SAN model to evaluate the performance of a database replication protocol under unknown load. The objective of the study was to build a credible model of a replication protocol with diverse DBMSs (i.e. from different vendors). The model of the replication protocol implements faithfully the logic of the replication protocol and of the specific benchmark (TPC-C was used in the study, referred to in section 2.4.3) and will take as input the probability distributions of the responses by the different replicas (DB server products) to different SQL statement types. These response times are collected only once – for a very light load of the server (a single client/connection). The model can then be configured to execute a different load (a large number of concurrent connections which will be served in a round-robin fashion) and even a different operational profile (i.e. a different mix of transactions). Using a model, solved via Monte – Carlo simulation, one can obtain very accurate estimates of the performance of the replication protocol under the chosen load and profile. The mean transaction response time measured experimentally (by deploying the replication with real replicas and middleware implementing the replication protocol) differs negligibly from the mean response time obtained with the probabilistic model (the recorded difference was <1%), which indicates the usefulness of the approach. Again, what has been done in [QEST'2015] seems applicable to some of the AQUAS demonstrators. We are planning to apply this style of modelling to the ATM use case.

The reader will have noticed that state-based models presented here are referred in some of the other sections, too. We are particularly keen to explore to what extent the support for state-based models provided in CHESS (via the CHESS "dependability plug-in") allows one to undertake meaningful and useful combined model based-analysis (for safety, security and performance). It seems clear that the current provisions for dependability modelling in CHESS by a dedicated plug-in are suitable to build models suitable to study some dependability attributes (e.g. reliability, availability) by modelling the probabilities of failure propagation and the propagation delays between the components. The existing provisions, however, may turn out to be insufficient to capture various stochastic dependences between the stochastic transitions between the states of a system model and hence be inadequate for a detailed combined analysis of safety (safe states), security (e.g. integrity) and performance (e.g. when the system is in a compromised state). We would like to study and, if necessary, together with the colleagues from INTECS extend the provisions of the dependability plug-in for more elaborate combined analysis.

2.21 Analysis of Diversity and Defence–in–Depth in Reliability, Safety and Security [City]

2.21.1 Background and kinds of analysis

Diversity and defence in depth are design principles recommended in most safety-critical and security-critical systems. Given that a system contains protective redundancy, e.g., more than one subsystem able to achieve a required function, or to prevent certain undesired events, this redundancy could be made useless by the presence of similar flaws in the redundant subsystems. Diversity between these subsystems mitigates this risk.

Analysis methods for diversity address questions about:

1. deciding, when designing a system, *how* to use diversity (what will be effective?). This involves two kinds of decisions,
    a. *architecture* decisions [Strigini, 2005]: what degrees of redundancy and diversity will be used, and how will the redundant and/or diverse items be connected? For instance, given a safety function to perform, will there be two or three redundant "lanes" ("channels", "trains", etc.) to perform it? Will they be all different, or divided in groups of identical components? Will they each be implemented with further redundancy and/or diversity inside? How will their outputs be "adjudicated" in case of disagreement – For instance, will any one channel have authority to perform a safety shutdown, or will there be some kind of voting (to reduce the frequency of spurious shutdowns? Will this adjudication be implicit in the architecture chosen (e.g. parallel safety shutdown systems, each with full authority, or firewalls laid out in successive layers, both implicitly implement 1-out-of-N adjudication) or implemented in algorithms, and where will these algorithms be executed?
    b. *diversity-seeking* decisions: *in what respect* shall the "diverse" items be different? Standards and guidelines abound with recommendations of this kind, from using different programming languages to using radically different principles of operation in the diverse subsystems. We have available as bases for analysis:
        i. in deterministic terms, a detailed mapping of how various diversity-seeking decisions address various possible causes of common failures [Littlewood 2000];
        ii. in probabilistic terms, criteria sufficient for deciding *in some cases* which one between two ways of organising the development, verification or procurement of "diverse" subsystems is *likely* to yield better results (e.g. [Salako, 2014], [Popov, 2014], [Gashi, 2016]);
2. how to *assess* a completed system that uses diversity? We use diversity to improve dependability (security, safety, etc.). But can we estimate how much diversity, as used in a specific system, improved those attributes that it is meant to improve? This assessment can be divided [SeSaMo 2013] into two kinds of analysis of the specific form of redundancy and diversity used in the system:
    a. assessing whether it, *if functioning as intended, i.e. without common failures*, would indeed prevent system failure due the common causes against which it is intended to defend. For instance, if we used two identical sensors to feed data into two different noise-filtering algorithms, this analysis would tell us that the design is correct for defence against common design faults in the filter algorithm, but is not correct for defence against design faults in the sensor type used. This is a form of verification that

the designers answered correctly the questions in (1a) and (1b.i) above. It does not consider whether these two specific implementation of the filtering algorithms happen to have defects that can cause common failures in this system, and how likely these failures would be;

b. quantitatively assessing how effective it actually is, that is, what is the probability of common failure of the diverse components in the system as built; or, how much better the system is with diversity than it would be without. This form of question is very hard to answer, just as it is for any other software reliability improvement method. One can *experimentally* assess the probability of failure of a complete system, by operating it, at costs that vary from very affordable to unfeasible ([Littlewood 1993, Littlewood 2011]). But for assessing *without this operational testing* (or other hard evidence, not much easier to gain) how much we *gain* by using diversity (or any other dependability-improving method), there is no easy and universal method. Assuming *failure independence* between diverse subsystems is demonstrably wrong; general experience shows that diversity can give impressive reliability gains, but there is no simple rule to determine the gain in a specific system. On the other hand, we have produced forms of probabilistic reasoning that integrate results of operational testing with system-specific quantitative knowledge, to derive system reliability (e.g., [Zhao, 2017, Littlewood 2013, Bishop, 2014]; some are unpublished). For instance, these methods can use as inputs partial statistical descriptions (e.g. confidence bounds on reliability, probability of being fault-free), for the individual diverse channels together with results of operational testing.

We at City have contributed a major part of research on diversity, with results relevant to all the questions above. This includes both experimental studies (actual effect of diversity on reliability, performance, safety or security) and probabilistic modelling results (www.csr.city.ac.uk/diversity/).

A more extensive discussion of combined security and safety issues with diversity is in sections 2.19 “Redundancy and diversity”, 3.6 “Modelling of redundancy and diversity”, and 4.4 of [SeSaMo 2013]

2.21.2 Applications to security and trade-offs

Diversity has been studied extensively for its value for reliability and safety, starting with a period of intense experimental work in the 1980s. For security applications, this initial phase of intense empirical data collection started about ten years ago, and probabilistic analysis has been very limited.

In the SeSaMo project, we modelled the trade-offs between aspects of security and dependability given by various forms of redundancy/diversity (diversity of keys or of cryptosystems) in encryption of communication [Gashi, 2016]. We have now an extended version of this set of probabilistic models which allows comparison of architectural solutions for a more general set of architectures and of threats. With this approach we can (i) identify in which cases trade-offs exist between e.g. the integrity and the confidentiality aspects of security, and support architectural choices in view of these trade-offs; (ii) take into account uncertainty about the presence of hidden vulnerabilities known to potential adversaries, and the fact that their likelihood increases over time. We expect to apply this style of modelling to the industrial drive use case; and others as may be useful.

2.21.3 Summary for application in AQUAS

In summary, for application in AQUAS, the following kinds of capabilities are available:

- analysis of empirical data about efficacy of diversity observed in operation/testing/experiment, of the system considered or of subsystems thereof in previous applications. Given the empirical

data (measures of operation time or demands, numbers of failures/successes of each diverse component/subsystem), these analyses produce quantitative assessment for the system reliability. Limited forms of prediction are also possible given documentation of faults, rather than failures;

- qualitative analysis of system design (architecture and any “diversity seeking decisions”) to check or compare alternatives for match between the design decisions and the assumed failure causes (faults, security threats) against which diversity is deployed.
- quantitative, probabilistic analyses at design level as in the previous subsection, for steering decisions about trade-offs. These can be integrated with state-based modelling so as to consider whole-system effects (e.g. on the controlled physical system) of decisions about specific reliability mechanisms and security controls;
- assessing and specifying adjudication methods.

2.22 Astrée and RuleChecker [AbsInt]

Astrée (https://www.absint.com/astree/) is a sound static program analyser that has been developed by ENS and is licensed by AbsInt for industrialization. Addition of new features is done in cooperation with ENS and Sorbonne University (formerly known as UPMC). Astrée was designed to prove the absence of runtime errors and further critical program defects, including array index out of bounds, invalid pointer dereferences, int/float division by 0, arithmetic overflows and wrap-arounds, floating point overflows and invalid operations (Inf and NaN), and uninitialized variables. The tool is based on abstract interpretation, a provably correct formal method, and does not require the program under analysis to be instrumented, executed, or stimulated by test cases. The tool can be used on handwritten code, automatically generated code, or any combination thereof. It can be integrated into continuous verification frameworks such as Jenkins (a Jenkins plugin exists).

Astrée contains the seamlessly integrated RuleChecker that checks code for compliance with MISRA, CWE, ISO/IEC, and SEI CERT C coding rules. The tool can also check for various code metrics such as comment density or cyclomatic complexity. Custom extensions for user-defined coding guidelines are available on request. Using RuleChecker in conjunction with the sound semantic analyses offered by Astrée guarantees zero false negatives and minimizes false positives on semantical rules. No standalone MISRA checker can offer this, and no testing environment can match the full data and path coverage provided by the static analysis.

Astrée and RuleChecker can be qualified according to DO-178B/C, ISO 26262, IEC 61508, EN-50128, the FDA Principles of Software Validation, and other safety standards. The qualification process can be automated to a large extent using Qualification Support Kits (QSK). Each QSK contains two major parts that depend on each other: a Report Package and a Test Package. The Report Package consists of an Operational Requirements Report and a Verification Test Plan. The Requirements Report lists all functional requirements of the tool to be qualified. For each requirement, the Verification Test Plan describes one or several test cases that should give confidence about the correct functional behavior of the tool. The Test Package contains all test cases listed in the Verification Test Plan. The framework provides scripts to execute all test cases and evaluate the results. Qualification Support Kits can be easily extended with additional test cases to also cover specific operational environments. In addition to QSKs, Qualification Software Life Cycle Data (QSLCD) reports are available that document AbsInt’s development processes. The reports describe the entire development process of Astrée and RuleChecker including all verification and quality assurance activities.

Astrée consists of two parts:

1. The Astrée client, for setting up an analysis and viewing the results. The client offers both a GUI and a batch mode for easy automation and integration.
2. The analysis server, which carries out the actual analysis (or several analyses as separate processes).

Both parts may run on the same machine at once. In production, however, the server typically runs on a powerful remote host, while clients are run by the individual developers and managers on their PCs or other devices.

Input

Astrée works on preprocessed C code. If desired, a built-in preprocessor can be used to obtain preprocessed code. The code is then parsed and translated into an intermediate representation on which the runtime error analysis is performed.

For each analysis, Astrée needs an entry point — typically a function of particular interest, or simply `main`. Astrée will then analyze all portions of the code that can be reached by non-interrupted sequential program execution from that entry point.

It is possible to provide an analysis wrapper — e.g. to model reactive system behavior — in a dedicated C file associated with the analysis.

Astrée can also be configured with different ABI (application binary interface) settings.

Lastly, Astrée accepts formal analysis directives that provide external information to the analyzer, e.g., about the environment, or to steer the analysis precision. The directives are specified in the dedicated, human-readable Astrée Annotation Language (AAL), so that the source code does not have to be modified. The locations to which the directives refer are specified over the program structure and are robust with respect to line numbers.

Output

The most important result of the analysis is a list of alarms, i.e. of potential runtime errors. Each error is reported together with its type and the source code location where it occurs. If Astrée can prove that an alarm will always occur in a specific context, it is classified as a definite runtime error. In addition, various kinds of statistics are compiled. Interactive tables, graphs and charts let you quickly see which code areas are most prone to which kinds of errors.

Report files can be generated for documentation and certification purposes. The entire analysis project can be saved as well, including all files, settings, annotations and comments.

The analyzer also provides coverage information showing unanalyzed code statements. In absence of definite runtime errors, code reported as unanalyzed is definitely unreachable.

Lastly, Astrée can be used to check for functional program properties by a static assertion mechanism. If Astrée does not report the assertion to be violated, the asserted C expression has been proven correct.

Astrée will always stop with an error if indispensable data is missing or if source files cannot be correctly parsed and translated.

Handling the errors

Each reported error can be interactively explored, commented on, or fixed right away in the built-in C source code editor. Possible false alarms can be marked as such using AAL annotations, so that they no longer occur on subsequent analysis runs. Alternatively, you can tweak the analysis settings or increase analysis precision for selected code parts. After that, you can run the analysis once again and examine the improved results.

These steps are repeated as needed until all alarms have been dealt with and no errors are reported anymore. At that point, the absence of errors in the code has been formally proven.

3 Application of the Analysis Techniques to the AQUAS Demonstrators

In the following, it is described the application of the techniques, illustrated in the previous section, to the five use cases proposed in the AQUAS Project.

3.1 Air Traffic Management

3.1.1 Data Distribution Service [City]

We plan to use our expertise in conducting empirical, and possibly large-scale, data collection and analyses (see Section 0) for this use case. For example, together with other project partners (e.g. IntegraSys), we plan to conduct performance evaluation of the chosen Data Distribution Service (DDS)[11] middleware – OpenSplice (http://www.prismtech.com/vortex/vortex-opensplice). In the absence of standardised benchmarks for performance evaluation of DDS implementations, an alternative is to use the performance tests available from the vendor http://download.prismtech.com/docs/Vortex/html/ospl/EvaluationGuide/tests.html. In addition, we will consider using the performance benchmarks available from competitor products; one such is available for the commercial implementation of DDS from RTI: https://www.rti.com/products/dds/benchmarks

We will base our evaluation on standard timing/performance metrics, such as response time and throughput. A complete set of performance indicators is yet to be defined by the partners involved, however. We will use our experience in database server performance evaluation for this purpose (see 2.3.3). For example, we will conduct performance evaluation under different *load* profile, e.g. number of simultaneous UAV (Unmanned Aerial Vehicles) considered in the ATM use case.

We will investigate possible use of tools available in AQUAS for this performance evaluation, e.g. TimingProfiler (https://www.absint.com/timingprofiler/) offered by AbsInt.

In addition, we plan to investigate publicly available bug repositories for OpenSplice middleware, such as http://opensplice.org/bugzilla3/ in order to inform our probabilistic, state-based modelling for security assessment. We plan to collect security-related bugs to, for example, aid our Adversary model development, in which successful security compromises will be modelled as having negative consequences on the middleware: these can range from performance deterioration to middleware crashes.

We will apply state-based probabilistic modelling to the ATM use cases, which will include two phases:

Phase 1. ATM in “trusted environment”. A model of the entire use case will be developed, which includes the model drones (UVAs), the ground services and the middleware (implementing the OMG DDS specification). An essential part of the work would be the load model which will target to represent accurately the response time delays as the load on the system (i.e. the number of UVAs interacting with the system simultaneously) varies. The load model will be parameterised using the

[11] The Data Distribution Service (DDS) is an Object Management Group (OMG) standard for scalable, real-time, dependable, high-performance and interoperable data exchanges based on a publish–subscribe pattern.

results from empirical lab measurements on prototypes of real hardware/software to be used in the demonstrator. The load model will also take into account the effect of the number of subscribers on the delivery times of messages generated by the publishers.

Phase 2. ATM in adverse environment. Here the model derived in Phase 1 will be extended by adding models of Adversaries, e.g. malicious agents who might be interested to disrupt the operation of the ATM. The types of adversaries will be decided based on security analysis of the ATM system and the threats which can lead to non-negligible disruptions. As a proof of concept we started looking at 2 types of Adversaries: i) adding new fictitious topics to the DDS, which will consume resources (memory and CPU time) and may lead to non-negligible overheads (additional delays); ii) adding fictitious subscribers to legitimate topics. This type of malicious activity may seem less extreme than i), but may still lead to non-negligible additional delays.

The level of abstraction in the model will be chosen carefully to allow us to see the specifics of the chosen middleware (DDS), without making the model solution prohibitively expensive.

Additional aspects that we might look at while working on is demonstrator are:

- An Adversary generates additional flow of messages (e.g. exploiting vulnerabilities with authentication)
- An Adversary may compromises the middleware in away, which slows its operation down.

The outcome of the modelling work will provide an insight with a number of areas:

- finding out the limitations of the deployed off-the-shelf software, e.g. of the Open-SPLICE middleware. This is an important consideration. Should Open-SPLICE turn out to be too slow to handle the anticipated number of UVAs using the service simultaneously with the envisage hardware deployment, either a more powerful hardware platform may be needed or a different implementation of the DDS specification may be used instead.
- Even if the system performs well in trusted environment, it may turn out not to be so under realistic cyber-attacks. The model will allow us to have cost-effective study of the effect of cyber-attacks on system performance. Should some of the attacks lead to unacceptable deterioration of performance, the analysis may recommend:
    - Adding security control should be put in place (to reduce the likelihood of success of the attacks). Then the effectiveness of these additional controls will be assessed using a new model, in which the additional controls and their benefits (i.e. reduction of attack success rate) are modelled explicitly.
    - A different implementation of the DDS specification will be used instead. Selecting an alternative may benefit from applying the models (Phase 1 and Phase 2) to all candidates of DDS. In this case, each candidate implementation may require a different parameterisation, which in turn will require separate measurements with each of the candidates.

We will define a set of “measurement” tasks, which will need to be completed in order to parameterise the model of a specific system deployment. The measurements will help with each of the phases:

- For Phase 1, a series of direct measurements of the response times under different loads, SLA guarantees, etc. will be conducted. This work is conceptually similar to the work we have done in the past with complex off-the-shelf software, e.g. RDBMS.
- For Phase 2, vulnerability analysis should be undertaken (either using the records of known vulnerabilities) or identifying vulnerabilities using other means (e.g. via pen-testing). Once a vulnerability is identified which can lead to non-negligible impact on performance, the impact

should be established with sufficient confidence (e.g. looking at WCET or some of the meaningful measure). The measurement results will be used to parameterise the Adversary.

We also intend to look at the issue of model transformation (e.g. between SysML/UML and probabilistic state-based models). At least one of the tools available in AQUAS, the CHESS tool offer a such a transformation (including language support) We intend, together with the colleagues from INTECS to find out to what extend the SAN models that we are going to develop for the ATM use case can be derived using the CHESS capabilities, i.e. the CHESSS dependability plug-in. If the current capabilities of the plug-in turn out to be insufficient, then we would like to extend them so that the SAN models we develop for the ATM case are fully supported, including modelling support for cyber-attacks.

3.1.2 Design Methodologies and Benchmarking for Hypervisor Technologies [UNIVAQ]

The methodology developed by UNIVAQ involves different aspects related to system modeling, co-design activities, and final implementation, considering performance analysis under mixed-criticality constraints. The task-level application model, to be used as input, will be provided by UCs, and UNIVAQ plan regards the enhancement of an existing framework and related prototypal tools (HEPSYCODE: HW/SW Co-Design of Heterogeneous Parallel Dedicated Systems).

UNIVAQ methodology takes into account application models using Concurrent Sequential Processes (CSP) Model of Computation (MoC) as input model, and considering also Model-to-Model transformation to directly manages and extract as-much-as-possible system functionalities and requirements from different AQUAS tools.

The main activities related to the methodology are related to metrics evaluation and estimations, supported by external performance analysis tools provided by different partners, to collect system features that will be exploited during the Design Space Exploration (DSE) step in order to analyze “Pareto optimum” or “weak Pareto optimum” solutions in term of different allocations and binding with respect to the application model and target HW architectures.

After that, the DSE approach will be extended to consider safety requirements in order to drive and restrict the design space. The methodology will introduce only safety-related constraints into the design flow, other requirements (e.g. security, power consumptions, fault tolerance issues etc.) will be considered in future works.

To increase reliability and to guarantee compliance with the constraints, schedulability and performance analysis (using interaction among different partners) will verify and validate the different solutions found by DSE, and final testing and implementation steps will be made to perform and refine evaluation and estimation activities.

Connected to demonstrator platform selection, UNIVAQ will provide its expertise on LEON3 multi-core architectures and Linux/OpenMP embedded solutions, to evaluate and compare performances respect to different processors and HW architecture technologies.

Finally, to bridge the gap between design methodologies and Hypervisors (HPVs) modeling, in order to manage also hierarchical scheduling policies, isolation functionalities and other HPV features, UNIVAQ will extend its benchmarking activities related to PikeOS, to identify possible behavioral anomalies and system vulnerabilities in terms of safety and performance constraints (e.g. spatial/timing isolation, scheduling overheads, communication bottlenecks etc.).

3.1.3 System design and analysis with CHESS [Intecs]

The CHESS Methodology, as introduced in Sections 2.8, 2.9 and 2.10, is supported by the CHESS toolset that is based on Papyrus and the Eclipse framework, available as open source under the Polarsys Eclipse project.

CHESS supports modeling of all phases of development: from the definition of requirements, to the modelling of the system functional, logical and physical architecture, down to the software design and its deployment to hardware components.

CHESS offers schedulability and dependability analysis functionalities: models are decorated with timing and dependability properties that are the inputs for the analysis, according to the analysis results that are back-propagated onto the model itself, the modeller can perform some tuning on the model in order to satisfy real time and dependability requirements.

Contracts can be used at all stages of system design, from early requirements capture, to any level of system architecture and detailed design, involving software and hardware, and used to formalize co-engineering constraints and assumptions.

Intecs work plan regards the application of the CHESS methodology and toolset to support the definition of requirements, to the modelling of the system functional, logical and physical architecture and dependability analysis of the ATM use case.

The analyses enabled by CHESS may be used to validate the safety and security requirements identified for the system. Safety and security requirements are supposed to be derived with dedicated risk analysis, by the adoption of common practices and from the recommendation of safety and security standards.

3.1.4 Efficient Methodologies for the Development of Formally Verified System Software [HSRM]

The methodologies developed by HSRM can be used for a more efficient development process of formally verified system software. Also, the verified microkernel can be used as an alternative operating system for comparison and evaluation.

For a better understanding of the meaning of formal verification, as opposed to testing, both terms shall be defined:

- Formal verification of software denotes the mathematical proof showing that the software satisfies its given specification. Thus, testing a formally verified implementation is expendable.
- Testing of software refers to systematically checking the response of the software to different stimuli in order to validate the specified requirements. This implies that as long as not all possible cases have been tested, there remains the possibility of not detecting existing programming errors.

Even for medium complex systems, exhaustive testing (i.e. to perform and execute all possible test cases) is practically infeasible.

Older safety standards in avionics such as DO-178B [HSRM-11] have employed solely testing because formal verification was not practically feasible for any realistic piece of software. However, the new standard DO-178C has superseded its predecessor. This new standard explicitly considers formal verification as superior to testing and recommends its use for the highest criticality levels.

Tools can assist in developing mathematical proofs for software. These tools can be classified as either automated or interactive provers.

- Automated provers attempt to automatically prove that an implementation given as source code complies to the given specification. If the prover is not able to prove a certain logical expression, then the proof simply fails.
- An interactive prover prompts the user instead at this point. The user is then requested to prove the assumption manually and let the prover continue afterwards.

SPARK includes automated provers like cvc4 [HSRM-3] or alt-ergo [HSRM-4]. Typical interactive provers such as coq [HSRM-5] or provers based on the Isabelle/HOL [HSRM-6] framework can be integrated as well.

The Ada language has a long track record for high reliability software development, especially in aeronautics and SPARK has been and is being used in avionics applications both in the ground segment (see: [HSRM-12])as well as in aircraft (see: [HSRM-13]).

Besides functional proofs, the methodology will also enable the developer to integrate timing specifications. These can be used to formally prove the schedulability of a program system. To this end, the methodology will make use of WCET analysis tools such as those provided by partner AbsInt and further temporal proof techniques. HSRM will analyse the integration of standard timing specification methodologies and evaluation methods such as MARTE.

To demonstrate applicability of the approach, HSRM develops as an example a formally verified microkernel [HSRM-14]. This microkernel -although implemented in the SPARK subset of Ada- can be equipped with multiple language bindings, enabling its use also by applications written in languages other than Ada.

Furthermore, HSRM will contribute sDDS [HSRM-15], a data distribution middleware designed for resource constrained systems such as wireless sensor nodes and small embedded devices. In this context, HSRM will collaborate with partners BUT and ISYS in:

- identifying any modifications and/or extensions to sDDS needed to increase its applicability to the ATM use case;
- providing support in implementing and evaluating such modifications;
- providing support in applying BUT's ANaConDA tool to generate concurrency related tests for sDDS.

3.1.5 Dynamic analysis of concurrency [BUT]

Methodologies developed by VeriFIT research group from BUT are intended to support dynamic analysis of concurrent software. ANaConDA [BUT-Anaconda] is a framework that simplifies the creation of dynamic analysers for analysing multi-threaded C/C++ programs on the binary level. The Java Race Detector & Healer is a prototype for a run-time detection and healing of data races and atomicity violations in concurrent Java programs. SearchBestie [BUT-SearchBestie] (Search-Based Testing Environment) is a generic infrastructure that is designed to provide environment for experimenting with applying search techniques in the field of program testing (e.g. to find optimal settings of injected noise to increase efficiency of ANaConDa and Race Detector & Healer). In particular for Air Traffic Management use case, ANaConDA and SearchBestie will primarily focus on concurrency issues in multi-threaded environment of Data Distribution Service (DDS). Furthermore, BUT can focus also on concurrent bugs in implementation of the application itself and on the

implementation of PikeOS operating system. Dynamic analysers can be used to measure coverage of tested source code artefacts. Noise injection technique can refine test case scenarios to better emulate execution of system under test in a real environment. In case a problem concerning a specific part of the code will be identified at an interaction point, monitoring and noise injection targeted at that particular part of the code will be applied in order to make the analysis as efficient as possible.

Apart from the primary focus on dynamic concurrency analysis, static or dynamic analysis of memory-related or performance-related problems will be applied on the code of the use case whenever appropriate. For that, the Predator, Loopus, Ranger, and/or Perun tools will be used.

3.1.6 Security analysis, standards compliance [TrustPort]

Within this use case, TrustPort will contribute by specifying security requirements for used components and communication protocols, SWIM profiles and security segments. Task define but not limited to:

- Including the specific requirements of SWIM co-related standardizations, recommendations and best practices from cyber-security significant documents such as NIST SP 800-53r4 or ISO/IEC 27001:2013.
- SWIM Segment security analysis – Segment 1 Architecture security view (Application layer security, network layer security and support service security with standardized interface) and Segment 2 NAS Environment (Cyber threats, commercial software security leaks analysis, general threat identification, Enterprise Security & IT Service security recommendations, NAS Enterprise Security Gateway tests, COTS-Based Application Layers Security analysis).
- SWIM Infrastructure security and identity management analysis, requirements definition
- Threat Model – risk analysis and specification of threats and impact on operation, security, safety and performance and countermeasures.
- Security audit of infrastructure, processes, security mechanisms used in ATM UC.
- Penetration tests – penetration tests of ATM infrastructure to measure security level and compliment with specified security requirements and best-practices.
- SWIM Web Services (WS) Security Specification analysis and identification of the security controls for Web Services in the SWIM environment.
- The general security requirements specification – defined from security standards, norms and best-practices applicable during service operation - access control (public key infrastructure), message security and transport security (PKIX509, WS-Security, DDS Security, ...), data security, etc.

Mapping of the security controls to integration of 3-rd party modules and provide specific and verifiable requirements for each security control.

3.1.7 Static Astree Analysis of PikeOS Kernel [SYSGO]

SYSGO wants to apply static analysis techniques to the PikeOS kernel in order to find bugs and to reveal tautological conditional checks that, when removed, would improve performance. For this, two methods are used: (1) Astree, and (2) LLVM. We apply the Astree analyser in UC1 and the LLVM analyser in UC5.

PikeOS is an embedded real-time operating system and hypervisor. Its kernel is, like many other OS kernels, written in C plus some occasional use of Assembly language for parts that are highly architecture specific or outside of what can be expressed in C. The code structure is unlike a normal application, i.e., there is no single entry nor single exit of the program, but the OS is structured more

like a multi-threaded callback library with multiple entries, notably system calls for services provided to applications, plus interrupts and exception handlers for control, maintenance and scheduling. The code has many non-linear properties because it is pre-emptible, but this structure is abstracted away in special function calls that do thread switching. Except for such functions, the code can be analyised sequentially from entry to exit. I.e., each entry-exit can be analysed like a separate piece of application. However, these pieces work together on a common memory, so the analyser needs to consider them in parallel. This plus the assembly special function and gcc extensions that are used, are the main problem for applying a COTS Static Analyser to PikeOS.

The method to tackle this is to provide to Astree a 'meta main' entry, which is essentially a virtual additional single entry point that invokes all the possible entry points into the PikeOS kernel. Furthermore, the special code (assembly, gcc extensions) are stubbed out into special function provided by configuration. In total, this enables us to apply Astree.

The static analysis method Astree can now basically analyse the PikeOS code. Next steps are to apply new analyses so that special code like tautological comparisons can be found.

Example:

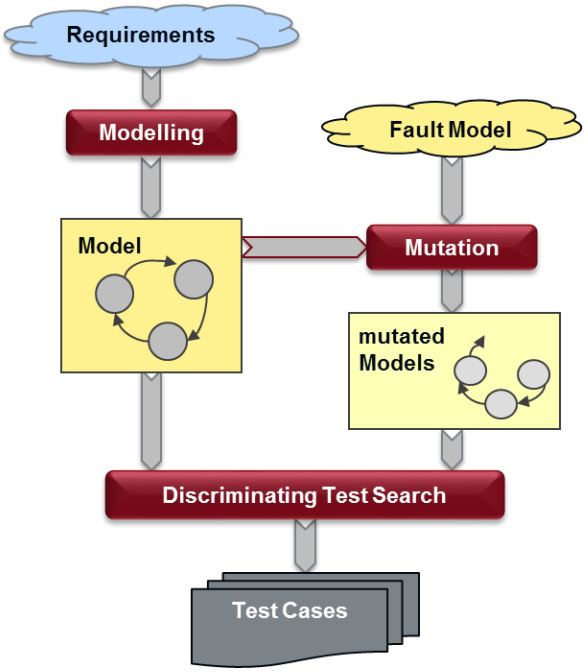
```
if (p == NULL) { return P4_E_INVAL; }

...more code, possibly function calls...

if (p == NULL) { return P4_E_INVAL; }
```

In this example, the second check is redundant (provided that the value of 'p' has not changed and that there are no other control flow entries introducing different values of 'p'). The analyser could identify this so that we can eliminate the second conditional.

Since this is a static analyser, it is proven that 'p != NULL', so the transformation is safe. We are thinking about the possibility to add an annotation, however, instead of the conditional that we 'assume that p != NULL' at the point of the second conditional so that after code changes, in case another possible value of p is possible to be encountered, a re-run of the analysis tool could point out that our assumption might have become wrong. An alternative for 'assume' (a static check) would be to insert an 'assert' (a debug runtime check) so that such changes could at least be found during testing.

3.1.8 Model Based Testing for Multiple Concerns [AIT]

AIT will apply its tool model-mutation based test case generation called MoMuT, to the ATM use case. The MoMuT family of tools belongs to the so-called black-box testing tools as it derives test cases from models of the system under test (SUT) and does not rely on, e.g., the source code of the SUT. In difference to most other test case generation tools, MoMuT's original approach was fault-based testing: the tools generate test cases that are guaranteed to detect models that contain certain user-selectable, seeded faults.



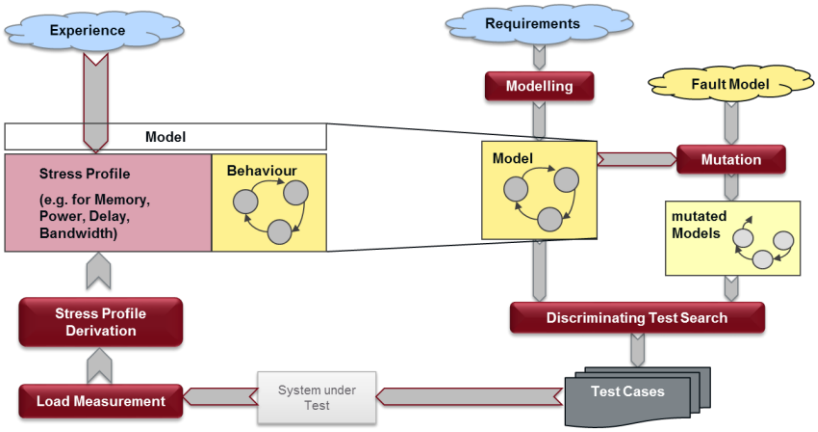
With the fault-based, functional based test case generation, we can generate tests:

- with good (functional) requirements coverage
- to test also functional safety features
- that can be used to safe-guard against faults introduced by code generation (if used)
- in early phases of model development. The generated efficient (and thereby short/few/ compact) tests can be reviewed, implicitly reviewing the model they are based on.

The test case generator can also be used to localize faults. If a generated test fails on the system under test, MoMuT can highlight which model elements and thereby which related requirements are implemented wrongly. For long regression tests, a short test for debugging that fails because of the same implementation problem can be provided automatically.

By now and going forward within the project, this is extended towards testing non-functional properties from the same model - performance, robustness and safety requirements.

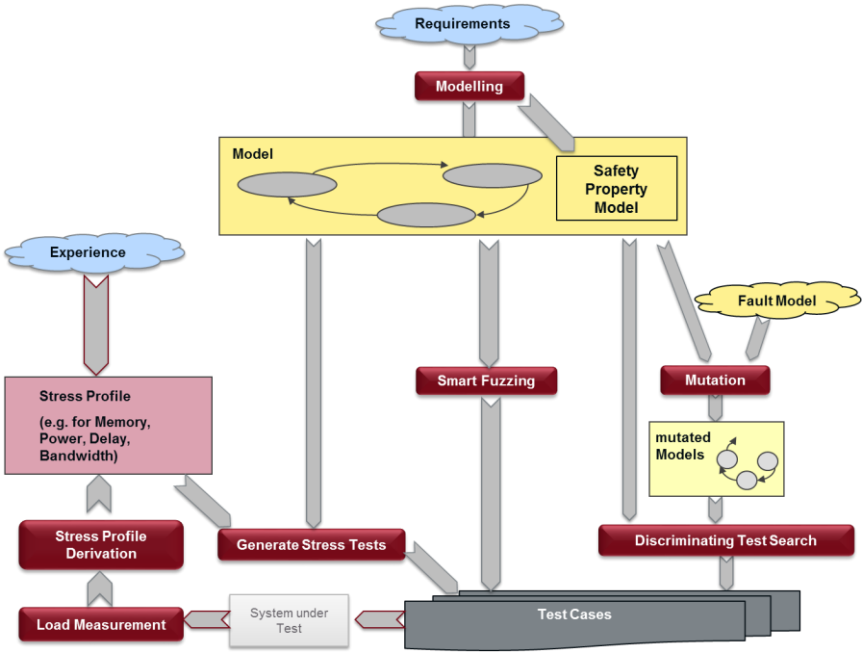
For performance testing, the generated functional tests are run on the system and a stress profile is learned. Using this profile, a new test suite can be generated, that attempts to maximize the stress



on the system.

To test robustness, we apply smart fuzzing and re-use the model that was prepared and used for generating the tests for functional requirements. Smart fuzzing combines fuzzing, i.e. exposing the system under test to many unexpected inputs, with systematic system exploration. The latter becomes possible by using a grammar for the “protocol” or in our case a behavior model. This ensures that robustness tests are generated for as many as possible different system states.

If safety properties are included in the model, they can be checked during test case generation, thereby ensuring that the safety properties hold on the model. If the test environment allows the model to be simulated in parallel, the safety properties can be also checked on the system under test while running detailed (generated) tests of all kinds (functional requirements coverage, performance, robustness).



These new features aiming at multi-concern testing are improved and partially developed a new within the project.

Which of the features will be applied within the ATM use case (and to which part of the use case), is not yet fully defined.

3.1.9 Proving the Absence of Runtime Errors and Rule Violations [AbsInt]

AbsInt will help in assessing the safety and security of the system. To this end, AbsInt will use its tools Astrée and RuleChecker to analyse the operating system (SYSGO's PikeOS kernel) statically for finding violations of certain safety and security principles, or proving the absence of such violations by means of automatically found program properties. Such formally proven properties can be used as arguments for safety and security in the certification process.

RuleChecker will be run in combination with Astrée. This allows to use sound semantic knowledge obtained by Astrée's runtime error analysis also for rule checking.

In order to assess the safety of PikeOS, RuleChecker is used to detect or exclude violations of MISRA-C:2004 and MISRA-C:2012. Security properties are assessed by using RuleChecker to detect or exclude violations of CERT, CWE, and "Secure C" (ISO/IEC TS 17961) rules. RuleChecker will for all

checks be run in combination with Astrée in order to exploit Astrée's sound semantic analysis of C code to further improve the quality of the rule checking results.

Furthermore, Astrée will be used to statically detect invariant assertions and invariant branching conditions. Statically decidable branching conditions can be exploited to increase system performance; while detecting invariant assertions yields further insights regarding the system's safety and security.

In a final step, Astrée will be used to statically analyse the locking behaviour of PikeOS. To this end, Astrée will be extended by a suitable abstraction and analysis model for the employed locking mechanisms of PikeOS.

Assessing safety and security properties of the operating system using a sound static analyser aims at significantly alleviating certification processes. Using a sound analyser provides comprehensive assurance of the properties analysed by the tool. Hence, assessing critical safety and security aspects using a sound tool will produce strong arguments to be used in certification processes.

3.2 Medical Devices

3.2.1 Security Analysis through Model-Based Formal Static Code Analysis [CEA]

For the medical devices use-case, CEA proposes to conduct security analysis through a model-based formal static code analysis method. The goal is to cover the development lifecycle of the medical use-case from requirements to verification, while also bridging the gap between our low-level static code analysis method and higher level models produced by other partner tools.

The modelling tasks will be done with the Papyrus modeller, for which the SysML-Sec and AtML languages must be implemented. The static code analysis will be done with Frama-C plugins. Papyrus code and annotations generators, to be developed, will bridge the gap between models and analysable code.

We shall focus on security requirements and properties of the use-case, although functional aspects can also be analysed with such an approach. The following steps of the analysis method can be applied to the medical devices use-case:

1. Modelling in SysML-Sec of security requirements of the medical devices. Here the requirements are still very textual and un-formalized.
2. Modelling in SysML of the system architecture abstracting the medical devices high-level architecture, including hardware elements.
3. Modelling in UML of the software architecture abstracting the medical applications that run on the medical devices. In particular we shall focus on classes, functions, and behaviours of functions represented with state-machines.
4. Refinement of the textual security requirements as attack/nominal scenarios in the system architecture. The refined attack/nominal scenarios can be used to infer application-wide properties (relational properties) in the software architecture.
5. The user may also directly specify local pre/post-conditions of the functions in the software architecture as refinements of the textual security requirements.
6. Every successive refinement of a requirement is traced to the original requirement thanks to SysML traceability mechanisms.

7. From the software architecture model, generation of C/C++ code with function pre/post-conditions as ACSL annotations and application-wide properties as relational ACSL annotations.
8. Verification of the generated code, with ACSL annotations, through Frama-C plugins.

Concerning static code analysis itself, for the medical application, the results of value verification will only be used to verify the absence of undefined behaviours (including runtime errors) of the C language in the software of the medical device.

The value verification is able to emit alarms for C operations that could lead to an undefined behaviour. If no alarm is emitted for an operation in the source code, then this operation is guaranteed not to cause a runtime error (its behaviour is well defined in the sense of the norm of C).

Since the value verification does not omit any alarms, it can give some certification credits even if the use of such verification methods is not mandatory by the norm ISO 62304.

3.2.2 Test and Quality Management [ITI]

ITI's A2K tool will be extended and used to provide a requirements management, quality management, and system testing functionality to the UC2 medical devices system (RGB's topcuff). It will manage a database of test cases and test results, as well as automatically generating system parameters for the different test plans, measuring the results of these test cases from the system under test, collating, analysing, verifying, and reporting the results. This will be accomplished by interfacing the A2K tool with either simulated or real hardware and software of the topcuff system (hardware in the loop (HIL)).

The system architecture for this is still under discussion and development. However, it will probably involve A2K in the following interaction point scenarios:

1. How to create specific test plans and how these relate to system requirements.
2. How particular test plans will relate to specific system parameter inputs, how to generate these input signals, and how to securely provide (connect) these stimuli to the system under test (real or simulated).
3. How to collect the system output signals and how to determine the test results (pass/fail).
4. How and what data should be stored regarding a particular test case scenario so that it may be recalled at a later date, perhaps for verification or auditing.
5. How to provide "Design Space Exploration" (DSE), that is how we may investigate different system architectures and how these relate to different performance measures.

How the test case results relate to metrics regarding safety, performance and security.

3.2.3 Applying Medini Analyze to the Medical Domain [AMT]

Domain Independence and Medical Device Domain Profile. The origin of Medini Analyze is in the automotive industry, hence many of the features and analysis methods stem from the automotive safety standard ISO 26262. One example, that is quite prominent for users of the tool, is the attribution of SysML model elements with the ASIL (Automotive Safety Integrity Level). Nevertheless, with ISO 26262 being derived from the generic and general IEC 61508, many of these items are very similar to features and methods required by sibling standards, that are also derived from IEC 61508, like IEC 60601 for Medical Devices.

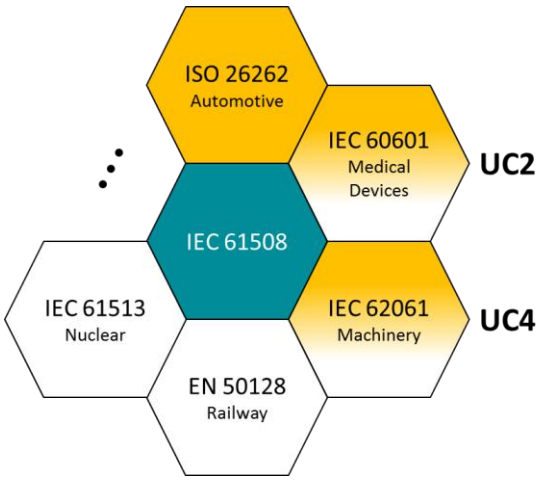


Figure: Generic safety standard IEC 61508 and its derivatives

In a first step, AMT plans to provide a domain-independent version of Medini analyze and in a second step, provide an initial domain-profile for Medical Devices, that can then be grown to a full-featured profile throughout the project.

Transferring Methods from ISO 26262 to Medical Devices. Medini analyze provides a rich feature set for the analysis of automotive functional safety including HAZOP; HARA, FMEA, FMEDA, SFF, SPF, FTA. AMT plans to investigate relevant standards for the medical devices domain for the applicability of these methods to medical devices and will provide adaptations required, to support this use case with these analysis methods.

Security Analysis Methods. Based on Medini analyze, AMT has developed a prototype for cyber security analysis in the SESAMO project. It features methods like graphical attack tree modelling, TARA (with Evita risk graph) and security objective and requirements management. This prototype shall be applied to UC2 and enhanced according to demands, e.g. providing a customizable risk graph for the TARA.

Integration with Requirements Management Tools. AMT plans to enhance the integration capabilities with further requirements tools, namely DOORS NG, which is used by RGB, the provider for this use case.

3.2.4 Support for Risk Management with Safety Architect and Cyber Architect [ALL4TEC]

The methodology proposed in Section 2.11 is supported by a toolset: partner tools based on UML/SysML languages and ALL4TEC tools (Safety Architect and Cyber Architect), which provide respectively supports for safety and security risk management.

As mentioned in deliverable D2.1.2, Section 4.1, it would be desirable to develop a safety risk management tool, which could provide support to prevent /reduce harm, in such a way that given a risk cause, we could include a risk control measure to avoid a hazardous situation. Risk management is recommended even mandatory by standards, such as EN ISO 14971:2012. Medical devices - Application of risk management to medical devices.

As a consequence, we plan to develop a tool-chain to support safety and security risk co-engineering inside product life cycles (Requirement and modeling phases) and across these two phases. For example, if system requirements or models are created in partner tools based on UML/SysML model (Capella, CHESS and Papyrus, …), the resulting UML/SysML models could be imported in Safety Architect or Cyber Architect for Safety risk analysis or Security risk analysis.

Then, as explained in Section 2.11, the development of a specific security viewpoint in a Safety Architect could allow co-engineering between Safety and Security. Indeed Safety engineer could use the results of security analysis realised in Cyber Architect (e.g., threats analysis or attack tree) to generate a merged safety artefacts (e.g., failure mode and effects) and security artefacts (e.g., vulnerabilities and threats) for assessing the influence of security-relevant events on safety top event.

Finally, based on the analysis results, the system architecture solution could be confirmed or rejected with the need for re-design or enhancements across product life cycles.

3.2.5 Modelling and simulation of a patient [BUT]

BUT will implement a complex tool for modelling and simulation of a patient and his response to anaesthesia during surgery. The tool will be applicable in two modes: (1) as a complex framework simulating the patient under anaesthesia (monitoring, drug injection and control mechanism) and (2) as a component in the loop of RGB medical device and patient model. The simulation tool will allow us to develop safety and performance of the system under various situations. The safety means ability to achieve and stabilise the intended patient's body conditions (blood pressure, neuromuscular relaxation, etc.) while the performance focuses the minimal amount of drugs injected.

BUT can also contribute in several areas of static and dynamic code analysis: namely, concurrency-related, memory-related, and/or resource bounds analysis, using the above mentioned ANaConDA, Predator, Loopus, Ranger, and/or Perun tools.

3.2.6 Engineering Methods and Medical Standards [Tecnalia]

Tecnalia will analyse RGB's Current Engineering Methods and the required medical standards in order to identify interaction points among security, safety and performance especially during the product lifecycle. The main regulatory aspects are related to the functional safety standard IEC 60601 which is referenced in the European Medical Devices Directive 93/42/EEC of 14 June 1993, revised 2007/47/EC. In fact, we analyze:

- UNE-EN 60601 safety norm as the main requirements during the development stage as part of the patient safety requirements.
- The development process shall consider requirements of applicable safety standards (IEC 61508, DIN EN 62304, DIN EN 60601-1/ 2).
- We aim to analyze ISO 11073 standard for Medical Devices interoperability.
- If available we will analyses part of the source code with Sonarqube in order to identify vulnerabilities. This code evaluation is part of the assurance cases.

3.2.7 Implications of human factors aspects for safety and security [City]

For the Medical use case, the main applications of City's methods are expected to concern the human factors aspects of safety and security, specifically for scenarios of operator interventions: effects of deviations of operators' actions from those required by design.

We expect to proceed from a classification of the operator interaction with the device into (a) routine operation, in which human actions are the normal setting up, monitoring, and ending of a session of use of the device, and (b) exception situations, other operator interventions, required by conditions of the patient or the medical device, or decided by the operator although unneeded. A preliminary probabilistic model would indicate the relative weights of risks in the two kinds of situations, and thus the requirements on correct intervention in the exception situation and the acceptable frequency of these situations.

Then, a qualitative analysis of the exception scenarios would help to identify and mitigate possible risks. The outputs towards the development process would be thus (a) a contribution to hazard identification and analysis, with respect primarily to safety concerns (and secondarily to security concerns, as inappropriate operator interventions that may happen accidentally may also be the goal of hostile action); and downstream of these analyses, (b) help in defining the envelope of operation within which the closed loop control algorithms controlling the infusion pump must be demonstrated to be robust, and the requirements for operator intervention; and (c) help in tuning of (c1) the user interface design and of (c2) the alarm threshold settings to reduce human interaction-related risks , especially in exception scenarios; and finally (d) contributing inputs to specifying human factor testing that will take place after the AQUAS part of the lifecycle.

3.2.8 Security Analysis, Standards Compliance [TrustPort]

TrustPort will provide a security requirement specifications of medical UC architecture. Specially focus on confidentiality and integrity of messages exchanged between components of proposed architecture. This will be done theoretically and validated by assessment (audit, penetration test) in theoretical and practical stages of the project. We will provide security requirements based on existing standards and best-practices for implementation, follow these requirements through product life-cycle and force their compliment in all stages. We will focus on following topics based on ISO/IEC 27034, ISO/IEC 11073-00103 security requirements compliance – confidentiality, Integrity, availability, nonrepudiation. The tasks are but not limited to:

- Analysis of security aspects of proposed architecture based on ISO standards and best practice.
- Analysis and establishment of security requirements based on the recommendations of FDA, FTC, HITRUST and NIST organizations.
- Creating risk analysis results from security perspective. Considering the most common cyber-security threats. Based on the vulnerability recommendations and best practice.
- Adding security perspective into system specification with respect to confidentiality, integrity and availability of data stored in the intelligent infusion controller also based on the new cyber security regulations of European Commission.
- Design of the communication protocols between devices in BP Control (according the schematic).

3.3 Rail Carriage Mechanisms

3.3.1 Security analysis through formal static code analysis [CEA]

For the railways application, the deductive verification will be used for verifying the conformity of the C-functions (issued from the code generator of the Atelier B) with their specification expressed in B0 refinement language. To reach that objective, the translation of B0 specification to ACSL specification has also to be studied.

3.3.2 Machine Learning and Control Theory [UNIVAQ]

UNIVAQ plans to develop interdisciplinary techniques across machine learning and control theory for addressing security related problems of fault detection and isolation in the ClearSy use-case. In particular, according to technical details of the use-case that will be provided by ClearSy, we envision to exploit algorithms on Regression Trees and Gaussian Processes adapted with classical residual generation and fault diagnosis techniques in control theory.

3.3.3 Security and Safety Analysis [MTTP]

This use case is proposed by ClearSy. Currently, they design their systems using the B Atelier. They mostly capture safety properties that they verify during each refinement stage.

SysML-Sec could be used for two aspects:

- *Pure evaluation of security aspects during interaction points.* SysML-Sec could be used jointly with the B Atelier, with a focus on security aspects. During each refinement – a refinement would be an interaction point -, the B model would be imported by TTool, and would be checked against a list of security requirements using the SysML-to-Proverif model transformation. Thus, the work would mostly consists in developing an import function, and in defining and capturing security requirements. Yet, there, the impact of security issues onto safety would be difficult to evaluate.
- *Joint use of Atelier B and TTool for evaluating safety, security and performance properties.* There, we would study how the partitioning phase of TTool could be used to capture the whole system model – thus including the B model – at a quite high level of abstraction - in order to capture safety, security and performance aspects during the system partitioning phase. Then, once a satisfactory HW /SW partitioning has been found, we would generate from TTool a first B model. Whenever the B model would be refined, we could use what we proposed during the first bullet in order to assess again the security properties.

3.3.4 Safety, Security and Performance Analysis through Dynamic Virtual Testing [SISW]

Siemens PLM will investigate the use of system time-domain simulation combining mechatronics and controllers as a mean to assess system safety, security and performance.

The assessment method should follow several steps:

- Modelling of the physical system of interest
- Coupling and integration of the controllers with the physical system

- Preliminary validation of the dynamic behaviour of the system
- Modelling the attacks and failures
- Modelling the feared hazards detection at physical level (affecting the safety of the users)
- Automated Runs of large variants of scenarios to detect the occurrence of feared hazards
- For the detected events, use simulations data (sensors, buses, etc.) as requirements for controller's performances, safety and robustness improvement.

These methods investigations, if demonstrating their relevancy, should contribute in improving real systems security, safety and performance, reducing unexpected operation incidents and accidents.

3.4 Industrial Drive

3.4.1 Enhancing Medini Analyze for Application in Machinery [AMT]

Domain Independence and Machinery Domain Profile. The origin of Medini Analyze is in the automotive industry, hence many of the features and analysis methods stem from the automotive safety standard ISO 26262. One example, that is quite prominent for users of the tool, is the attribution of SysML model elements with the ASIL (Automotive Safety Integrity Level). Nevertheless, with ISO 26262 being derived from the generic and general IEC 61508, many of these items are very similar to features and methods required by sibling standards, that are also derived from IEC 61508, like IEC 62061 for Machinery.

In a first step, AMT plans to provide a domain-independent version of Medini Analyze and in a second step, provide an initial domain-profile for Machinery, that can then be grown to a full-featured profile throughout the project.

Application of FMEA to Security. Medini Analyze provides a mature feature for FMEA (Failure Mode and Effect Analysis). In the SESAMO project, it has been investigated, how this method could be applied to cyber security analysis. With AIT, one of the project partners in UC4 has proposed a similar method for applying FMEA in combination with vulnerabilities analysis, the FMVEA.

AMT plans to investigate the differences between SESAMO's Security FMEA and AIT's FMVEA and provide an implementation of a merged approach.

Integration with INTECS CHESS and other Partners' Tools. AMT plans to enhance the integration capabilities with other analysis tools. Medini Analyze has a rich set of integration adapters for requirements (DOORS, PTC Integrity, Jama), system models (Rhapsody, Enterprise Architect, Scade Architect) and other safety analysis tools (MSR FMEA). To become a central tool for the management of all non-functional requirements and quality attributes, AMT plans to continue the integration with INTECS CHESS and provide interfaces to incorporate analysis data for timing and analysis.

Component	Asset	Classification		IAS Violation
		Sensitivity / Criticality / Value for organization	Impact on safety?	
DDR RAM	*Motor speed set value*	*Critical*	*yes*	*Injected malign values*

Cause	IAS Violation Probability	IAS Violation Detectability	Effects		
			on safety	Reference	on other assets
Unprotected information storage	*P*	*D*	*Motor applies unintended torque*	*FMEA effect #n*	*Economic, image*

IAS Violation Risk	Countermeasures (CM)	CM Cross-effects on safety	CM implementation costs
R	*Encryption/Decryption Building Block for information storage*	*++/--*	*CMC*

Security FMEA sheet from the SESAMO project

3.4.2 ISDD [Magillem]

ISDD© can be viewed has the basic foundation of projects: it doesn't aim to replace current tools, rather to enrich the technical environment with new features.

For example, for this use case:

- Medini is the reference tool concerning safety and security;
- CHESS manages the timing analysis;
- In Magillem MPA, for each key component, we could set parameters linked to the performances of the system;
- ISDD© is responsible for the links between all data coming from the different tools.

A probable scenario is the modification of a high-level requirement: security needs to be strengthened, a new control is added. Medini content is updated and impacts are detected on the timing analysis: new metrics are then integrated in MPA. Elements related are then highlighted: experts have all data to analyse and balance SSP.

3.4.3 Adaptive Safety, Security and performance virtual testing [SISW]

Siemens PLM will investigate the use of system time-domain simulation combining mechatronics and virtualized controllers as a mean to assess drive controllers safety, security and performance for multiple commissioning configurations.

The assessment method should follow several steps:

- Modelling of the inverters and drives
- Coupling and integration of the controllers with the inverters and sensors
- Preliminary validation of the dynamic behaviour of the system
- Modelling the physical failures cases
- Modelling the feared hazards detection at physical level

- Automated Runs of large variants of scenarios and configurations to assess the efficient calibration, operation and fault detection by the controller.
- For the unexpected behaviours, use simulations data (sensors, buses, etc.) as requirements for controller's performances, safety and robustness improvement.

These methods investigations, if demonstrating their relevancy, should contribute in improving drive controllers security, safety and performance, reducing unexpected commissioning and operation incidents or accidents.

3.4.4 Safety-Security Analysis and Certification Support [AIT]

AIT technologies support two essential processes in the Industrial Drive use case: Safety-Security Analysis and Certification Support[12].

Safety and Security Analysis

A semi-quantitative Safety- and Security Co-Analysis is supported by the FMVEA (Failure Modes, Vulnerabilities and Effects Analysis) Method [Schm2014a], which was developed in the context of the ARROWHEAD project [Arrow2013] and extends the established Failure Mode and Effect Analysis with security related threat modes.

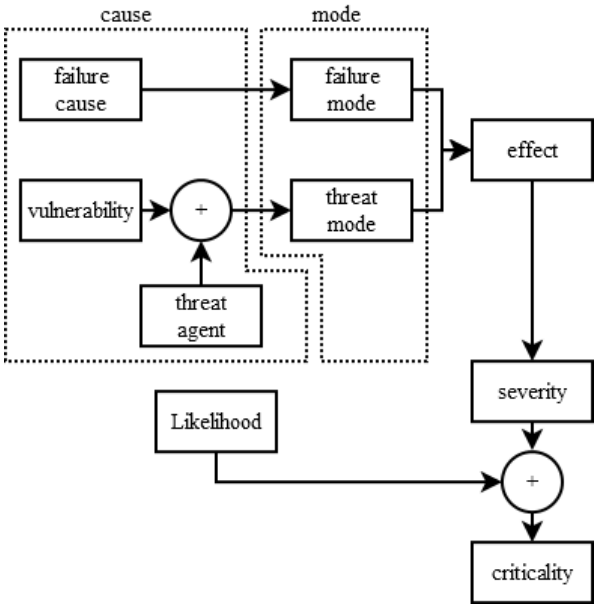


Figure 8: Depiction of the relation of cause and effect model for failures and threats

Figure 8 gives an overview of the cause and effect model for the Failure Modes, Vulnerabilities and Effects Analysis. The failure part consists, as before, of failure cause, failure mode, and effect.

[12] In derogation from the case study description in the FPP, the originally planned use of the tool DTFSim (Data Time Flow Simulator) has been dropped because in the context of the actual case study, which has been changed since the FPP, this kind of a time flow simulation would not allow relevant additional results. Moreover, skipping this simulation technology is no essential loss w.r.t. the range of assurance methodologies applied in the Industrial Drive case study, because system time-domain simulation is applied anyway by use case partner Siemens PLM (see further above).

Security related parts are added here, including vulnerability, threat agent, threat mode and effect. Depending on the level of analysis a vulnerability can be an architectural weakness or a known software vulnerability. Compared to safety, security requires not only a weakness but also an element, which is exploiting this weakness. This can be a software or a human attacker. Different threat modelling concepts can be used for the identification of threat modes such as CIA (confidentiality, integrity, availability), summarizing security properties an attack could exploit, or also STRIDE. Based on the severity of the effect, measured in terms of financial damage, loss of confidentiality or privacy and operational or safety impact and the likelihood of the failure or threat the criticality is measured. In the likelihood context, the system properties and attacker properties should be investigated. More information about FMVEA applications can be found in [Schm2014b] and [Schm2015].

In the Industrial Drive case study, FMVEA is used for assessing the safety and security properties of the use case. According to AQUAS methodology, the results of the FMVEA can be compared to results of another security analysis method, e.g. provided by Trustport, in an interaction point.

Certification Support
The tool WEFACT (Workflow Engine For Analysis, Certification and Test) is result of predecessor projects. Its goal is to support the complete engineering lifecycle of safety and or security relevant systems based on pre-defined processes. To achieve this goal, each project in WEFACT contains Requirements, Processes and Workflow Tools.

Requirements are defined as the entities needed to achieve the objective of the project. Requirements can be structured in different levels, where a top-level Requirement can be seen as the sum of its sublevel Requirements. Once all sublevel Requirements are fulfilled, the top-level Requirements *enters the state of completion***.** A Requirement can hold a connection to predefined processes. If all processes are executed successfully, the Requirement's status changes to "fulfilled".

Processes describe the steps that need to be conducted. The principal for the structure of Processes conforms to the structure of the Requirements mentioned earlier. Top-level Processes consist of sublevel Processes and the top-level Process reaches the status Successful once all Subprocesses have been executed without errors. Each Process can be linked to one or more Requirements. Moreover, a Workflow Tools can be associated to a certain Process. This way the Process becomes an executable which uses existing input and produces new output. This output can serve as input for subsequent Processes.

A **Workflow Tool** represents an application or component that can be addressed via URL. By defining Workflow Tools inside WEFACT, these applications and components can be directly invoked. Solely type for the Workflow Tool, the path to the corresponding executable and some input arguments need to be specified.

Figure 9 shows a screenshot of WEFACT:

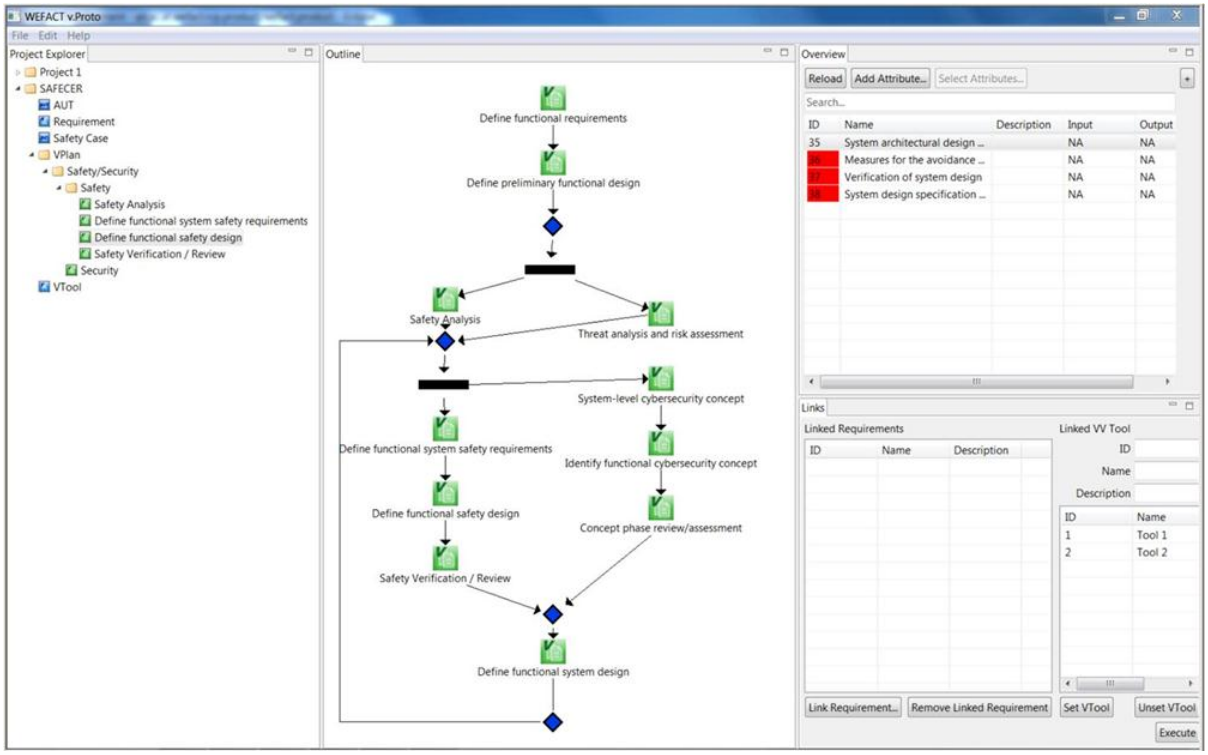


Figure 9: WEFACT

The project explorer to the left allows selecting a project and displaying the necessary process steps, in the middle, the respective process model is depicted. The example shows a process for separate safety and security analyses and how they can be combined in an interaction point for defining the system design. To the right, attributes can be set and Workflow Tools for performing the individual activities in the process can be assigned.

In AQUAS in particular, WEFACT is applied as a workflow engine which controls the certification processes in conformance to selected parts of IEC 62443 and IEC 61508. WEFACT shall be used to combine the invocations of the tools of the different partners that eventually constitute the complex product lifecycle while tracking the progress and success of the assurance processes.

3.4.5 Applying the CHESS Methodology [Intecs]

The CHESS Methodology, as introduced in Sections 2,8, 2.9 and 2.10, is supported by an toolset, based on Papyrus and the Eclipse framework, available as open source under the Polarsys Eclipse project.

The CHESS Methodology supports modelling of all phases of development: from the definition of requirements, to the modelling of the system functional, logical and physical architecture, down to the software design and its deployment to hardware components.

CHESS offers schedulability and dependability analysis functionalities: models are decorated with timing and dependability properties that are the inputs for the analysis, according to the analysis results that are back-propagated onto the model itself, the modeller can perform some tuning on the model in order to satisfy real time and dependability requirements.

Contracts can be used at all stages of system design, from early requirements capture, to any level of system architecture and detailed design, involving software and hardware, and used to formalize co-engineering constraints and assumptions.

Modelling in connection with Medini Analyze. For the application of CHESS to the Industrial Drive Use Case, CHESS and Medini Analyze support will be integrated. The approach is derived from the SESAMO project results.

System models, defined and detailed in Medini Analyze, associated with safety/security goals and requirements may be imported into the CHESS environment in the Requirements and System Views.

The modeller can refine the architecture, design, model and develop the software components in the CHESS Component View and associate elements in the software model to system level architectural elements and requirements.

Following the CHESS methodology, the modeler is able to focus separately on the software functional and extra-functional matters: the functional model enriched in CHESS with the specification of extra functional attributes such as real time (WCET, deadlines, occurrence kind etc.) and dependability properties (failure rate, propagation rate etc.).

Specific support is available in CHESS for the modelling of ASIL according to the ISO IS 26262 standard (and more in general for the modelling of criticalities and criticality levels, e.g. according to the needs f the domain currently addressed) and automatic verification on the model of the correct inheritance and decomposition of ASILs in the System and Requirements View (Figure 10).

Dependability, schedulability and end-to-end response time analysis can then be carried out in CHESS on the software model. If necessary some iterations can be carried out until real time and dependability constraints are satisfied.

Dependability, schedulability and end-to-end response time analysis will lead to the definition of more detailed technical requirements, which can in turn be exported back to Medini Analyze.

Once the analysis results are satisfactory, the software model and analysis results could be imported back from CHESS into Medini Analyze.

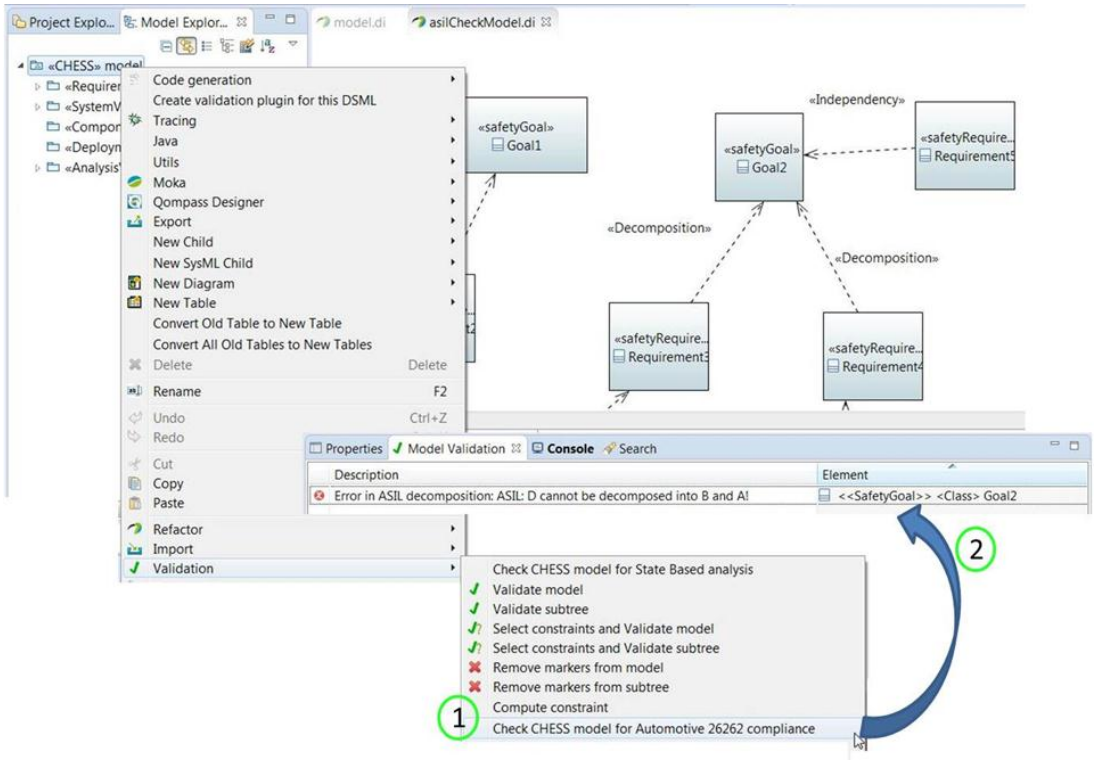


Figure 10: ASIL Decomposition Verification in CHESS

Schedulability Analysis. CHESS offers a domain-independent support to model real time properties and implements a chain of transformations that allows applying schedulability analysis on top of the CHESS model.

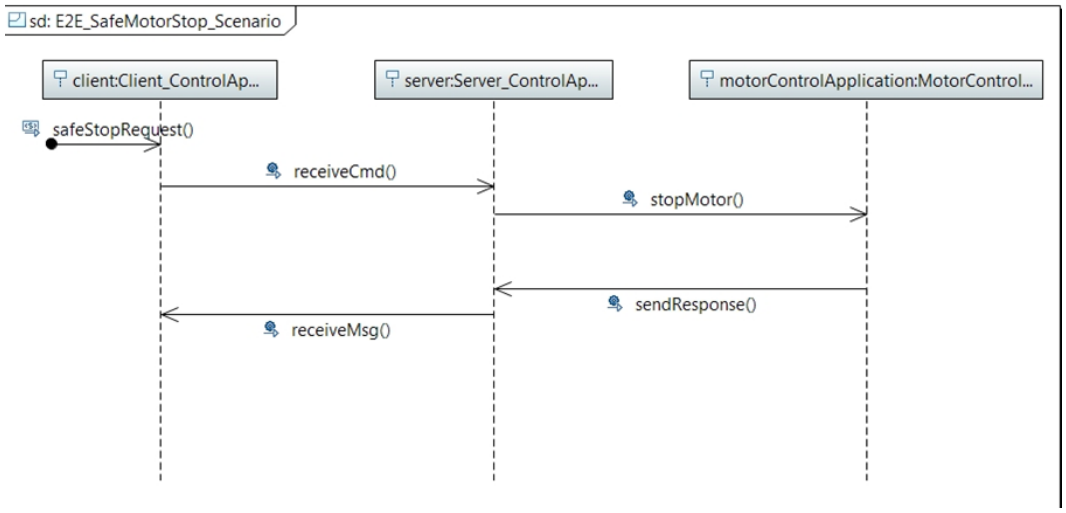
Schedulability analysis can be executed in order to statically prove that the deadlines specified for the components' operations can be actually met.

Schedulability analysis is executed by the MAST tool (Modelling and Analysis Suite for Real-Time Applications) [MAST]. MAST provides different kinds of scheduling analysis algorithms and methods. CHESS is seamlessly integrated with MAST in order to run different types of analysis and report the results back in the CHESS model.

End-to-end Response Time Analysis. End-to-end response time analysis may be used to take into account not only the performance of atomic actions, but the whole sequence of calls and operations necessary for fulfilling a complete task.

The process for the end-to-end response time analysis follows the MARTE modelling approach and allows modelling and analysing different end-to-end scenarios.

In order to allow end-to-end response time analysis, we introduced the specification of inter-component interactions in the CHESS methodology. Inter-component interactions are modelled by means of UML Sequence Diagrams in the Functional View (see Figure 12): the necessary support for such diagrams and their usage in the context of timing analysis was added in the CHESS editor.



Modelling inter-component interactions with a Sequence diagram

In order to perform end-to-end response time analysis, the user has to model the end-to-end call scenario using a Sequence Diagram in the Functional View (see Figure 12), and to define the end-to-end timing requirements that must be met. End-to-end response time analysis results are finally propagated back into the model.

Dependability Analysis. CHESS provides a cross-domain dependability analysis based on a State-based technique that allows to quantitatively evaluate the dependability attributes of the system. The term "state-based" refers to the fact that such techniques provide a representation of the system that is based on its possible states, and on the possible transitions between them. The analysis is performed on a state-based stochastic model of the system (e.g., Continuous-Time Markov Chains – CTMC). Dependability analysis is executed in CHESS by the DEEM tool [DEEM].

State-based analysis can be used to evaluate different dependability and performance metrics.

Within CHESS the following measures of interest can be evaluated:

- Reliability
  - instantaneous: the probability that the system continuously remains in a "healthy" state from time 0 up to time *t*.
- Availability
  - instantaneous: the probability that the system is in a "healthy" state at time *t*;
  - interval of time (averaged): the fraction of time that the system is in a "healthy" state in a given interval.

Other support. Security aspects could be managed in CHESS, by extension of the CHESS profile with suitable stereotypes that could be envisaged to model specific security aspects and properties (e.g. integrity and authentication).

Integration with SysMLSec by MTTP could be supported.

The CHESS FLA method can be applied to identify at software level to construct FMEA/FMECA and FTA for dependability analysis.

3.4.6 Probabilistic analysis of effects, trade-offs and synergies between Safety, Security, Performance [City]

The expected applications of our methods to this use case are mostly in the area of probabilistic modelling to explore the effects of the system design, and specifically of the safety and security mechanisms, in the presence of various accidental faults and intentional attacks, on system-level safety, security and performance qualities. So, this work will mostly contribute to analysing conflicts and synergies between SA/SE/PE objectives both for the high-level design – to select acceptable trade-offs and set requirements on subsystems – and at later stages – to check that refinement of, and any deviations from – these initial requirements do not violate the system-level requirements, and if necessary to select new trade-offs.

Which threats and design aspects should be analysed first is to be agreed in discussion with SAG. Likely initial targets are safety-relevant attacks on the commands from and inputs to the drive, and on the settings of the drive, especially the definition of fail-safe states.

We expect to integrate in the analysis our advances in modelling the combined effects of both the decay over time of protection offered by encryption-based mechanisms and the diversity employed for both security and safety (extending the analysis mentioned in section 2.21.2 "Applications to security and trade-offs", in 2.21 "Analysis of Diversity and Defence–in–Depth in Reliability, Safety and Security").

3.4.7 Safety, Security and Performance Evaluation with TTool [MTTP]

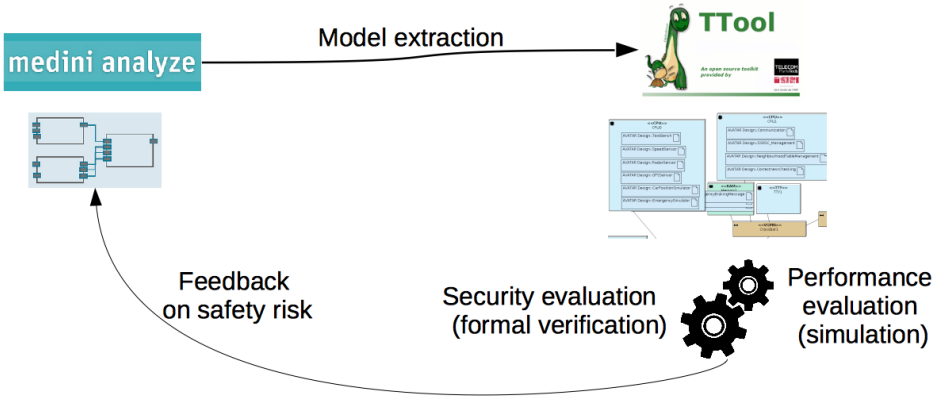
Siemens has performed a model of the industrial drive system using the Medini tool. Basically, Medini facilitates safety analysis based on the traceability between different modelling elements: requirements, faults, functions and hardware elements, with a focus on the probability of occurrence of failures and the risk. Traceability matrices then help deciding how to address the corresponding risk.

Our approach, SysML-Sec and TTool, offers safety, security and performance analysis. The obvious complementary aspects are on the security and performance analysis.

To evaluate the efficient complementary use of the two frameworks (Medini, TTool), we propose to proceed as follows:

- Manually model parts of the Medini model within TTool: requirements diagram, functional view and architecture/mapping view.
- Enhance the model with more refined attack trees and security requirements
- Perform security and performance analysis from the mapping models, and figure out which feedbacks could be provided to the Medini model-based. An example could be to better estimate the security risk by e.g. estimating the complexity of attack traces, and then to inject this risk into the Medini model. Another example would be to use the performance estimation of TTool (e.g., bus/processor load) to refine the safety risk defined in Medini.

- Propose a way to automate the process: model import/export, and how to fit the joint use of Medini/TTool within the AQUAS method, e.g. interaction points.



3.4.8 Security standards application [TrustPort]

In Use Case Industrial drive, TrustPort is going to define security system requirements and measures for individual components in the demonstrator system and define standard requirements and activities for the Industrial Product Life Cycle. The tasks are but not limited to:

- Management of security requirements of standards (ISO/IEC 62443, NIST, BSI, ISO/IEC 19790:2012, FIPS140-2 standards, and co-related documents) and best-practices;
- Preparing the uniform security requirements for using it in the partners' simulation and modelling tools.
- Addressing and finding the relationship between security and safety/performance such as synergies between security and performance will be evaluated for particular security requirements or level for the computational effort (delay, memory consumption, computation and CTU time, CPU cycles, hardware fault tolerance, …).
- Defining and managing functional and non-functional requirements on architecture (HW, operating system and application level) from the information and process security;
- Providing threat model and risk analysis of proposed architecture based on the known general vulnerabilities and TP best practice;
- Performing security assessment and penetration tests of use-case prototype in interaction with other partners by using their provided simulations;
- Ensure highest level of security with regard of environment and architecture and check with compliance.

3.5 Space Multicore Architectures

3.5.1 Schedulability, Timing Analysis, and Execution Monitoring [ITI]

The A2K tool will primarily be used in the space Use Case as a task timing and schedulability analyser, by both theoretical computation and by simulation. We will also provide a sensitivity analysis facility.

This will examine how task response times are related to variations in the other system parameters. For example, we can provide upper bounds on task WCETs that will make the system become non-schedulable. We can also identify what are the most critical tasks for system schedulability.

A2K takes as inputs (a) a Platform Model, (b) an Application Model, and (c) a Deployment Model.

The Platform model is a high level specification of the system hardware in terms of processors, memories, busses, ports and so forth. Included in the Platform Model are parameters such as bus latencies, processor speeds, memory access times etc.

The Application Model is a high level specification in terms of the system software in terms of flows, tasks, their dependencies, their messaging, and parameters such as WCETs, periods, priorities, message sizes, etc.

The Deployment Model is a specification of how The Application Model is mapped onto the Platform Model. This would include, for example, which tasks run on what processors and at what priorities, as well as how the tasks communicate data between themselves over busses or other communications links.

Within the project we will need to consider and develop adaptors to import the above data from the tools used by other partners (and vice versa).

The functions that A2K will provide in the Space UC context will be: task response times, task schedulability, and how these relate to operational mode changes in the system. Also of particular interest will be evaluation of the sensitivity of the task response times to changes in the system parameters and operational modes.

A2K will also be extended to provide code skeletons for different operating systems (e.g. PikeOS, RTEMS).

We are aware that other partners in the project will also provide (or use) tools, which perform task timing analysis. It will be interesting to compare the output results (e.g. task response times) provided by these various tools in different scenarios.

We also need to address what to do with the outputs of A2K. For example, once a response time or sensitivity analysis has been performed, how do these results relate to system specifications and requirements?

3.5.2 Introduce Hypervisor Technology into Design Space Exploration [UNIVAQ]

UNIVAQ HW/SW Co-Design methodology (called HEPSYCODE: System-Level Methodology for HW/SW **Co**-**De**sign of **He**terogeneous **P**arallel Dedicated **Sy**stems) has been adapted for performance and reliability, by integrating it with other model-based approaches and then by applying the resulting one to the development of the use cases.

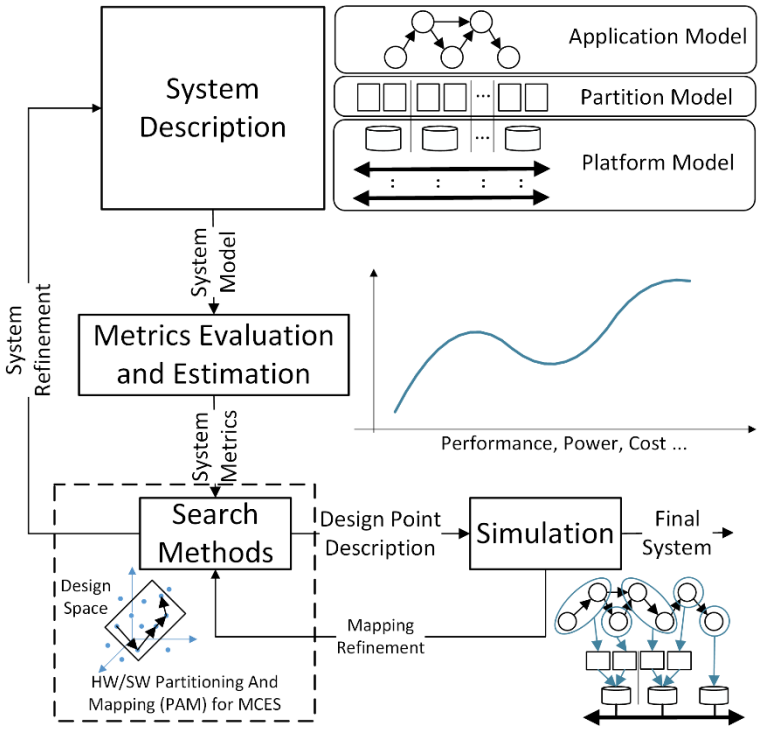
In particular, the design flow is divided into different activities, each of which at different level of abstraction and at different granularity.

In the *System Description* step, the UC application model is taken as input, and the different models related to “Platform Model” and “Partition Model”, that consider Hypervisor (HPV) SW partitions and related processors has been considered.

Then, the “*Metrics Evaluation and Estimation*” step involves the use of different performance analysis tools in order to collect system parameters and metrics useful for *Design Space Exploration* (DSE). These metrics are related to different system constraints (e.g. cost, workload, power

consumption etc.) and used as input parameters for a multi-objective optimization function, injecting safety requirements as one of the constraints.

Finally, the *Design space Exploration* step provides two iterative activities: *Search Methods*, involving evolutionary heuristic approach to find Pareto frontiers and sub-optimal solutions for orthogonal objectives, and *Simulation*, that starting from Design Point Descriptions, in terms of tasks, allocation/mapping and HW platform, provide analysis respect to different critical issues (in terms of timing execution, tasks schedulability, safety requirements matching etc.).



Using this design-flow, UNIVAQ wants to provide support to fill the possible gap between too much abstract model-based methodologies and the real HW/SW development tool-chains suitable for the target platform and able to satisfy relevant standards.

The final performance-based design space exploration step will be extended using performance analysis needed to achieve and refine search steps, using in the right manner safety and performance attributes to better drive the search of solutions suitable for input requirements. To enhance the design space exploration methodology, performance parameters will be taken from different partners activities (involved in performance, schedulability analysis and timing simulation), introducing FPGA accelerators into the proposed solution, and modeling HPV in a correct way.

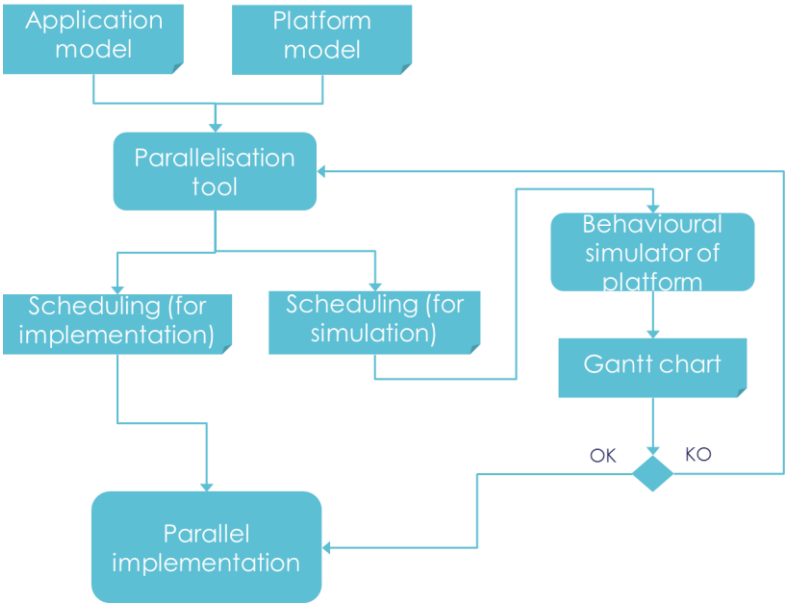
Finally, UNIVAQ will apply the proposed methodology to support PikeOS exploitation in Leon3-based architectures, since UNIVAQ have experience with Leon3 and Leon4 boards. With respect to technology solutions, UNIVAQ will investigate HW and SW architecture in multicore Leon3 scenario.

3.5.3 Tooled approach for performance evaluation before implementation [TRT]

The methodology described in the previous section can be depicted as follows for the space use case:

1. An application model is extracted from a functional code to get the parallelisable functions to be mapped and the related parallel parameters (e.g. loop nests, data structures);

2. The architecture model abstracts the physical hardware (may be imported from a MARTE modelling);
3. The user selects a mapping / scheduling choice through a parallelisation tool;
4. The underlying computation and communication activities are generated to produce stimuli for the architecture simulator of the space platform;
5. The behaviour of the architecture simulator is shown on a trace viewer; the user analyses the related Gantt chart, decides to improve the results by coming back to step 3 or is satisfied with the current results and decides to carry on with the next step;
6. The implementation step starts on the basis of the simulated results.



Note: TAS-E platform includes both a GPP and a FPGA. Considering FPGA adds complexity because such kind of accelerator requires that data are ready to be processed at a given time (accurate cycle), otherwise they are lost. Moreover the place and route / FPGA synthesis usually lasts hours; so such kind of expensive step has to be avoided. Hence scheduling is crucial to get an efficient implementation.

3.5.4 Performance analysis through design architecture [TRT]

The methodology described in the previous section can be depicted as follows for the space use case.

Modelling. The space software application will model in Time4sys to capture the real-time behaviour of the application and its interactions with the hardware platform (LEON3) and the operating system (RTEMS).

Execution Time profiling. Since the space application is running on multicore, we need to capture the execution time profile of each task which composes the application. To achieve this, we will stress each task in various conditions to capture its execution time profile.

Multicore interference Modelling. As the application is running on multicore, tasks are slowdown to due interference s between cores. Typically, when 2 tasks are running on different core they

interferes each other since shared resources blocking each other's. So, a new activity will be created in the methodology to model the LEON3 interferences according the space application.

Software design architecture analysis. Finally, based on the space design model, based on the executing time profiles of different tasks and the interference model of the LEON3, we will compute the worst case scheduling which could appear in the application and deduce the worst case latency for each task of the application. Then, from these results, we will compare to the timing performance requirements (Response Time allocated to this application) and validate if the performance architecture of the software is correct or not.

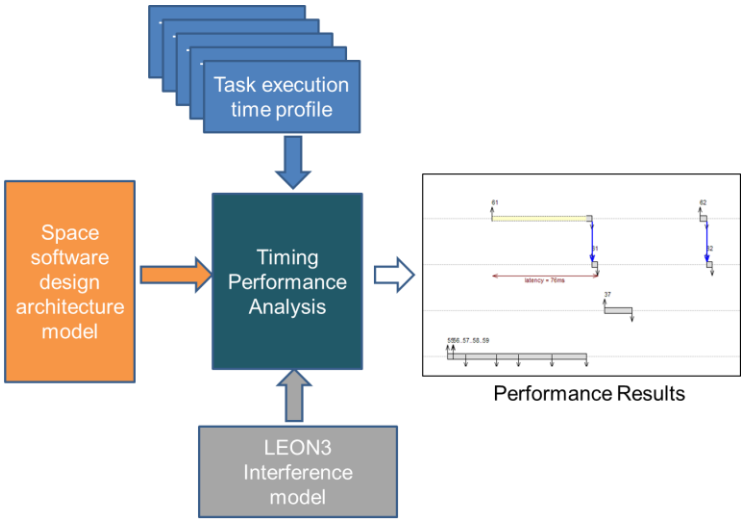


Figure 10: Global view of the methodology for performance analysis for LEON3

Co-engineering Safety & Performance. Based on the results of the performance methodology, and according to the ECSS-E- ST40C the safety engineering will compare the various performance requirements excepted by the safety cases. If for all safety cases the performance requirements are met, so the design architecture of the space software is correct. Since the performance scheduling analysis of the methodology is based on an analytical model, the safety engineer can use the proof deliver by the analysis tool to argue the correctness of the real-time behaviour of the application. Else, the software architect have to modify it according errors detect in aim to satisfy all performance requirements.

3.5.5 Analysis of performance and complexity [BUT]

Within the use case, BUT will mainly aim at checking various aspects of the program code of the use case by means of its static and dynamic code analysers. A particular stress will be put on analysing the computational complexity of the code due to its tight relation to the energy consumption, which is quite critical in the given area. For that, the above described Loopus, Ranger, and/or Perun tools will be applied. Moreover, an application of the Perun tool for automatically detecting performance-related regressions among different code versions will be considered. Finally, static and dynamic analysis of memory-related or concurrency-related issues will be applied wherever needed. In all cases, we will aim at making the analyses as modular as possible and targeted mainly at new/modified parts of the code and/or parts of the code found problematic at interaction points.

Further, having long-term experience with hardware-accelerated solutions, BUT will aim at performance attributes of computationally demanding subsystems. BUT will consider various tasks of

developed architecture to be accelerated using FPGA technology. If investigated that an accelerated solution would provide better performance attributes (such as less expensive computational time and/or power), BUT will provide hardware-accelerated models of considered subsystem.

3.5.6 Applying CHESS [Intecs]

System Level Modelling. The System and Software Functional Requirements Techniques (SSFRT) was an ESA study for the application of model based engineering technologies to support the space system-software co-engineering development processes, from mission level requirements to software implementation through model refinements. It aimed at making the software constraints present from the system analysis and avoiding any rupture during the development process [SSFRT].

The study resulted in the definition of a specific MBSE methodology to support the Space System Engineering (MBSSE) [MBSSE] that addresses the early phases of the system life cycle, in particular the ECSS Phases 0, A, and B, from the early definition of mission needs to the elaboration of a feasible, preliminary system definition.

European Cooperation for Space Standardization (ECSS) provides standards for engineering, management and qualification of a space system, more specifically, ECSS-E-ST-10C for system engineering. MBSSE was conceived with reference to the ECSS system engineering standard, and the technical processes and the principles defined in the ISO/IEC 15288 and the INCOSE System Engineering Handbook, and based upon a model-centric definition of the system using SysML.

During the whole modelling process the system is designed in terms of architectural components formally described with their well-defined interfaces and related properties. Such components are considered as black boxes until they are decomposed into their lower level components. System level entities are designed initially at a higher level of abstraction and then hierarchically decomposed applying a “divide et impera” approach in order to be able to focus separately on the different composing parts of the system. This way a deeper understanding of the parts can be achieved, allowing to add details to the parts, refining their models, and thus implementing an incremental process.

The whole development process envisaged in the CHESS extended methodology runs through different conceptual models: from the higher level functional architecture and logical architecture - with functions identified in the functional architecture, allocated to the entities of the logical architecture - down to the lower level physical architecture - with specification of software and hardware entities and their deployment, i.e. allocation of software to hardware.

A major concern for Space projects is the weakness of the link between the system and the software, due also to the fact that the professionals involved in these two activities have different profiles and background. Moreover the software schedule is becoming shorter and shorter, so it is extremely important to be fully aware of implications between system and software level requirements. Often software level decisions have an impact also on the higher system level, requiring an iteration loop: it is frequent that the system’s decomposition itself must be adjusted based on the envisaged software organization. For these reasons it is very important to provide efficient support for the system and software co-engineering activities.

Passing from one architectural level to the next requires a change in perspective and often implies also a change in the involved responsibility and professional profiles. Particular emphasis is dedicated in the CHESS approach to ensure adequate support to the system-software co-engineering phase, keeping consistency between system level entities and requirements on one side and the corresponding software and hardware level entities on the other side. When system decomposition

has been performed to a suitable extent, leaf elements from the higher-level architecture are mapped onto elements in the lower level architecture, and in particular links between system level entities and the corresponding software and hardware components are explicitly specified in the model. This is a valid support to ensure coherence and continuity in the development process, allowing full traceability of design decisions across different design phases and thus enabling precise evaluation of the impact of changes in different phases.

ECSS Standard for product assurance has dedicated documents for dependability and safety i.e., ECSS-Q-ST-30C and ECSS-Q-ST-40C respectively. The former puts an emphasis on the reliability attribute of the dependability, where the latter targets safety explicitly. Both documents require the reliability and safety analysis at all the stages of product design and development. The standards suggest various analysis methods both top-down and bottom-up approaches, where FMEA/FMECA and FTA are two of the methods.

The CHESS FLA method can be applied to identify the causes of top event and to construct FMEA/FMECA and FTA for reliability and safety analysis.

Software Level Modelling. The CHESS methodology support the modeller through the whole software development process, from the definition of requirements, to the modelling of the software design and its deployment to hardware components. It also offers support for real-time and dependability analysis as well as code generation functionality to automatically generate the infrastructure code (currently Ada) needed to implement

An extension of MAST has been implemented in the context of the CONCERTO ARTEMIS project to allow analysis of software allocated on multi-core architectures [CONCERTO].

The ECSS standards include specifically the ECSS-E-ST-40C for software engineering and ECSS-Q-ST-80C for software product assurance. ECSS-Q-ST-80C defines different criticality levels for a system based on the severity of consequences of the failure.

CHESS id derived from experience achieved for Space on board software development and is compliant with the ECSS standards. The basic principles of CHESS for component-based development are adopted by SAVOIR-FAIRE, a working group organized by ESA with representative participation from across all the European Space Industry to elaborate the concepts for an On-Board Software Reference Architecture [OBSW-RA].

3.5.7 Estimating Worst-Case Execution Times [AbsInt]

AbsInt will contribute to the use case with its TimingProfiler tool. The tool is meant to produce estimates for the worst-case execution times of non-interrupted tasks. Such estimates are needed to perform a schedulability analysis that verifies that all tasks can meet their deadlines.

TimingProfiler takes executable code as its input and performs an abstract static simulation of each task on a generic processor implementing the target instruction set, e.g., Leon 3 or PowerPC. It does not need to be stimulated with concrete inputs: by default, it takes all potential inputs into account - but it can also be restricted to specific execution scenarios, if desired.

TimingProfiler explores all potential execution paths of the task and determines a worst-case path with regard to execution time. User interaction is minimized; information not statically available is filled in by specialized heuristics. Thus, TimingProfiler trades some precision against ease of use, speed, and reduced resource consumption. Regarding the software structure, TimingProfiler makes potentially optimistic assumptions about loop bounds and recursion limits. All assumptions can be overruled by the users to provide more precise scenarios for profiling their use cases.

TimingProfiler gives detailed information about the execution time and time-critical paths, shows call and control flow graphs, and displays all relevant information about the executable. The resulting execution time is an estimate of the worst-case execution behaviour, not a worst-case execution time guarantee. Still, the computed timing estimation has to be realistic, which means that the analyser has to start at a fully linked binary executable in order to take the compiler effects, the instruction set characteristics, and the influence of a realistic cache and pipeline architecture into account. In early development stages the application often is incomplete and not executable as a whole. While precluding timing measurements, for TimingProfiler this is no problem since developers can provide a wrapper code, embed the existing code into the wrapper, and link everything together to an executable file.

Because the analysis is purely static and does not require access to physical hardware, it can be easily integrated in the development process and can be applied in continuous test and integration frameworks. AbsInt already offers a plugin for automatic integration of TimingProfiler in Jenkins, an open-source automation server continuous integration and continuous delivery (https://jenkins.io/). Using this plugin, developers can automatically analyse the worst-case execution times of their Jenkins builds, automatically mark a build as failed depending on the analysis results according to self-defined criteria such as violated expectations or specific errors, view a compact summary of the analysis results and failed items in the Jenkins build output, access detailed analysis results via the Jenkins web interface, and archive report files directly to the Jenkins workspace.

In previous projects, TimingProfiler has already been coupled with model-based code generators such as SCADE from Esterel and TargetLink from dSPACE and with scheduling tools such as SymTA/S from Symtavision. In AQUAS, they could be integrated with art2kitekt (a2k) from ITI, which is both a code generator and a scheduling tool. We refer to these partner tools as system-level tools because they hold a global view of the system whereas TimingProfiler operates on the level of single tasks. The benefit of integrating a system-level tool with TimingProfiler is that the system-level tool may request timing analysis results from TimingProfiler and display them in its user interface for inspection or use them in the scheduling algorithm.

3.5.8 Safety, Security or Performance [Magillem]

For the UC5, Magillem proposes to start linking these elements:

- Requirements managed in SUMO, which is based on Microsoft Excel, should be imported in ISDD©;
- Specifications should also be imported in ISDD©;
- A high-level representation of the FPGA in MPA including for each component, key parameters such as frequency, available memory space, etc. These parameters must be metrics related to or used to either assess Safety, Security or Performance. For example, the CPU frequency affects the software execution speed and thus the performances;
- Test plan and test cases.

Initially, static metrics come from datasheets and characterize the components. They will be included in the design for each relevant component and linked to the specifications, test cases and test results.

Then, if a modification occurs anywhere in the project, for every potential impact, experts are notified: they must evaluate them and choose to propagate them or not. For example, if a high-level requirement is modified and becomes stricter, a domino effect may occur: in turn, specifications,

design and test cases may be impacted. Then, if a component must be replaced, our solution will bring key elements at the forefront so that experts can make evidence-based decisions.

At the same time, ISDD© should be supplied with the results of simulations or actual measurements: in this way, if these metrics show a result that does not meet safety, security and/or performance objectives, experts are alerted to assess impacts and align the rest of the project. Being alerted in real-time of all potential impacts ensures to make evidence-based decisions and correctly balances SSP aspects.

3.5.9 Static Analysis of PikeOS Kernel using LLVM [SYSGO]

SYSGO wants to analyse the PikeOS kernel using LLVM as a static analyser. This is part of work on a bigger picture together with UC1, where static analyses are applied using Absints Astree analyser.

The problem with applying LLVM to the PikeOS kernel is similar to what needs to be done for Astree. The details have been described in the UC1 contribution above, in this deliverable. Also, one goal is to find tautologies, and this has also been described above.

One additional aspect to be analysed with LLVM is to find lock bugs. Since PikeOS is an SMP operating system, data races cannot be avoided by raising the thread priority alone, but instead, spin locks are used to protect against other CPUs, too. Locking/Unlocking may sometimes be distributed across multiple functions, so a human analysis of the correctness is error prone, despite our best efforts to use the simples possible structure. The simplest structure is often difficult still, because the corner cases an OS has to deal with may be inherently difficult.

A typical example of locking is to access threads or tasks or CPUs. Considering threads, typically, there are two possibilities: the current one and any other one (possibly also the current one). The current thread pointer is always valid, because it is currently executing. Any other thread may not be valid and requires locking to be accessed. Consequently, there are two ways to lock a thread for operation: manually with a 'lock' function for the current thread, and an atomic 'find&lock' function for other threads:

```
/* current thread: */
thr1 = sched_current();
thread_lock(thr1);
...
thread_unlock(thr2);


/* other thread: */
thr2 = thread_find_and_lock(thread_id);
...
thread_unlock(thr2);
```

There are various things that may go wrong and that we want to identify.

Nested lock: requires a special function to lock both threads, because locking them in arbitrary order may be a dead-lock. E.g. the following is a bug:

```
thr1 = sched_current();
```

```
thread_lock(thr1);
thr2 = thread_find_and_lock(thread_id); /* potential dead-lock! */
...
unlock(thr2);
unlock(thr1);
```

In this scenario, a special 'lock_two' function needs to be used – there is no easy way to lock the two threads manually in a correct way.

Another problem is escaping locks, i.e., control flow where a lock may or may not be locked. We want to avoid this:

```
thr1 = thread_find_and_lock(thread_id);
if (...) {
        thread_unlock(thr1);
}
/* bug: thr1 could be locked or not here */
```

The current state of our analysis with LLVM can already track lock pairs and handle several special cases where identification of whether something is locked or not is not so easy. E.g., the function 'thread_find_and_lock' can actually return a NULL pointer if the thread_id is not found, so the following code would be correct, but is a bit hard to track to see that there is no 'thread_unlock' missing in the 'if()' branch:

```
thr1 = thread_find_and_lock(thread_id);
if (thr1 == NULL) {
        return P4_E_NOENT;  /* no 'lock escape' here because thr1 is NULL here */
}
...
thread_unlock(thr1);
```

3.5.10 Efficient Methodologies for the Development of Formally Verified System Software [HSRM]

HSRM contributes efficient methodologies for developing formally verified system software. This will result in a foundation of proven common algorithms and a set of best practices for the development and verification of future system software. Furthermore HSRM develops as an example a formally verified microkernel based on these methodologies.

For a better understanding of the meaning of formal verification, as opposed to testing, both terms shall be defined.

- Formal verification of software denotes the mathematical proof showing that the software satisfies its given specification. Thus, testing a formally verified implementation is expendable.
- Testing of software refers to systematically checking the response of the software to different stimuli in order to validate the specified requirements. This implies that as long as not all

possible cases have been tested, there remains the possibility of not detecting existing programming errors.

Even for medium complex systems it is practically impossible to perform and execute all possible test cases.

Older safety standards such as DO-178B [HSRM-11] have employed solely testing because formal verification was not practically feasible for any realistic piece of software. Similarly, the current ESA standard ECSS-E-ST-40C [HSRM-12] adopts this approach. However in aeronautics, the new standard DO-178C has superseded its predecessor. This new standard explicitly considers formal verification as superior to testing and recommends its use for the highest criticality levels. It is expected that a similar trend will happen in the space domain, especially considering that space applications are typically less complex and are thus more amenable to this technology.

Tools can assist in developing mathematical proofs for software. These tools can be classified as either automated or interactive provers.

- Automated provers attempt to automatically prove that an implementation given as source code complies with the given specification. If the prover is not able to prove a certain logical expression, then the proof simply fails.
- An interactive prover prompts the user instead at this point. The user is then requested to prove the assumption manually and let the prover continue afterwards.

FRAMA-C [HSRM-1] and SPARK [HSRM-2] are examples of programming and verification environments for C or (a subset of) Ada respectively. Both include automated provers like cvc4 [HSRM-3] or alt-ergo [HSRM-4]. Typical examples for interactive provers are coq [HSRM-5] or provers based on the Isabelle/HOL [HSRM-6] framework. A typical verification environment consists of a set of provers cooperating via a common intermediate program representation based on languages like ML [HSRM-7].

One goal of the methodologies to be developed by HSRM is to not be bound to a specific prover, but to be able to combine different tools and methods in a common framework, using and extending the SPARK environment to be able to handle non-functional properties, especially specifying and verifying timing constraints.

SPARK 2014 is based on a subset of the Ada 2012 programming language. It is developed by AdaCore [HSRM-8]. It enables the developer to write software in SPARK along with specification and annotations as integral part of the source code. Thus, the specification is established by phrasing verification contracts as logical expressions. The SPARK tools transform these contracts and the program code into an intermediate ML dialect (WhyML [HSRM-9]) which then can be passed to an SMT solver to attempt the proof. This makes SPARK a viable tool for writing verification contracts and running them automatically through different SMT solvers. Its application requires knowledge and experience in formal verification, in order to be able to phrase the right verification conditions and to write provable source code.

The goal of the methodologies to be developed by HSRM is to provide a common concept for specifying verification conditions and writing provable source code for system software. HSRM uses a hierarchical approach through a layered architecture of the system software. The functionality of each layer is grouped into modules which are proven separately. Each layer can use all proven modules of the lower layers and extends those layers in terms of functionality by adding additional modules, thus implementing an incremental development approach.

Furthermore, several algorithms which are commonly used in system software shall be proven and the proofs shall be provided as a library together with corresponding lemmas for future projects.

Modules that cannot be proven (or have not been proven yet) will be thoroughly tested by using the verification conditions generated by the tool for runtime tests, thus combining both methods of formal verification and testing. Unproven, but tested modules can be used if acceptable and can later be replaced by a proven version if such a version becomes available. Besides the functional proofs described above, the methodology will also enable the developer to integrate timing specifications. These can be used to formally prove the schedulability of a program system. To this end, the methodology will make use of WCET analysis tools such as those provided by partner AbsInt and further temporal proof techniques. HSRM will analyse the integration of standard timing specification methodologies and evaluation methods such as MARTE [HSRM-16].

3.5.11 Support for FMEA/FMECA and FTA [All4Tec]

As recalled in deliverable D2.1.3, the ECSS standards for dependability (ECSS-Q-ST-30C) and safety (ECSS-Q-ST-30C) require dependability (reliability, availability and maintainability) and safety analysis at all the phases of product life-cycle. The standards suggest various analysis methods both bottom-up methods (e.g., FMEA/FMECA) and to-down methods (e.g., FTA).
The methodology proposed in Section 2.10 is supported by toolset with seamless interoperability between system engineering tools (e.g., Papyrus/CHESS or Capella) and safety/security analysis tools (e.g., Safety Architect / Cyber Architect). Thus, each engineer (System, Safety or Security Engineer) can solely focus on his concerns, with dedicated tools and terminology. For example, Safety Architect provides support for FMEA/FMECA and FTA.
In terms of co-engineering, a specific-concern viewpoint in a tool dedicated to another concerns will allow co-engineering between these concerns. For instance, thanks to the Security viewpoint in Safety Architect, Safety Engineer can use the results of security analysis realised in Cyber Architect (e.g., Vulnerabilities and Threats) to generate a merged Failure Mode, [Vulnerabilities, threat] and Effects Analysis for assessing the influence of security-relevant events on safety top event. In the same way, system/software architecture models enriched in Safety Architect or Cyber Architect with safety or security requirements can be imported in performance dedicated tools for performance analysis. This co-engineering approach is an iterative work because performance analysis with respect of safety or security requirements could lead to the definition of detailed and technical safety/security requirements, which can in turn be exported back to safety or security analysis tools.

3.5.12 Assurance Case Development [Tecnalia]

The main contribution is to analyse the use case and to develop an assurance case in order to satisfy the following requirements:

- ECSS-E-HB-40A: this handbook provides the main requirements for implementing the ECSS-E-ST-40C. Basically it indicates the product lifecycle that it will be used along the development of a use case. Therefore, all these requirements should be included in our UC5 and on the assurance cases.
- ECSS-E-ST-40C: this document identifies the expected outputs. We analyse which outputs are related to security, safety and performance aspects that must be included in the assurance cases.
- ECSS-Q-ST-40 “is a source itself of safety-related requirements for software” as stated by the standard.
- ECSS-Q-ST-80C: more precisely we want to generate evidences for analysing clause 6.3.4.6 a. The supplier defines measurements, criteria and tools to ensure that the software code meets the quality requirements which are going to be specified in Sonarqube, and this code evaluation is used within the assurance cases.

As result we will obtain a set of assurance cases supported by OpenCert tool. These assurance cases indicate which requirements must be meet in order to satisfy with these standards. We identify which co–engineering aspects and lifecycle aspects are intertwined and which interaction points can be identified on this scenario.

4 Interaction Points

This chapter first recalls the motivations that led to AQUAS's focus on interaction points and the general concept of "interaction point"; and develops thoughts on how to realise the concept of interaction points in AQUAS, including outlines of expected interaction points in some of the AQUAS use cases.

4.1 The Need for Interaction Points

AQUAS is dedicated to exploring how interaction points can be applied successfully through applying them in the demonstrators of WP2 and report the lessons learned this way; so, this text is meant to recall the purpose of interaction points, highlight general considerations likely to affect all the demonstrators (e.g. the requirements on the analysis tools) and foster constructive debate within each WP2 demonstrator team and the project as a whole; it is not meant to prescribe how each demonstrator should implement the concept in detail, since we expect the project to learn from the diverse experience of tailoring the application to the needs of the demonstrators.

The AQUAS project is motivated by the problems found by industry in combining in a cost-effective way the tasks of ensuring satisfaction of various non-functional requirements (where "ensuring" means "achieving and demonstrating"). Most such problems have been reported with the task of ensuring both safety and security in embedded systems. Companies with established processes for ensuring safety would import processes for ensuring security as well, but problems may arise because on the one hand, the two need to be considered together (e.g. because security violations affect safety, and because design trade-offs may arise between these two sets of goals), but on the other hand, they are the preserves of different technical cultures with their own languages, habitual assumptions in their analyses, etc. It is sometimes said, deprecatingly, that these specialists of different cultures work in "silos", with information flowing vertically within a specialism but not across specialisms. As the SAE J3061 Cybersecurity Guidebook has noted: “A tightly integrated process for Cybersecurity and safety has the advantage of a common resource set, thus, requiring fewer additional resources. However, since both activities require different technical expertise and both activities are resource intensive, it may not be feasible to expect a single team of experts to have the skills to perform both Cybersecurity and safety tasks simultaneously.” It is for this reason that the Guidebook, while recognizing the advantages of the ideal integrated process, makes provisions for non-integrated safety and security processes that communicate in more or less well-defined ways – what in AQUAS are known as interaction points.

We call "interaction point" both an activity and the point in a product life cycle (PLC) at which it occurs. The activity is "interaction" in that (a) experts in the various aspects of the system and its properties interact., e.g. security and safety experts; (b) their analyses are combined in some way, that may be anywhere in the range from informal discussion and mutual critique to using mathematical models to assess various measures of interest for alternative design options, or even a single, summary measure to be optimised (e.g., probability of an undesired event); (c) the need for changes or decisions may be recognised that require an integrated view, e.g. because of inevitable trade-offs between desirable properties, and these trade-offs are discussed between the various experts to produce recommendations/decisions.

4.2 The Planning and Scheduling of Interaction Points: Fault Tolerance View on a Process of Co-Engineering for Safety, Security and Performance

An important question is when these interactions should take place, to be cost-effective for a given project in a given company.

One viewpoint is that the lifecycle model used by the developers should identify from the beginning when interaction points will be needed. These "statically scheduled" interaction points are so scheduled as to achieve a reasonable trade-off between

- The cost of too many interactions for those "lucky" projects that never have conflicts or resulting rework (for these projects, all interactions may be counted in hindsight as unnecessary costs) and
- The cost of too few interactions for the "unlucky" projects, in which conflicts between requirements and unsatisfactory design trade-off are recognised late, requiring expensive rework or causing project failures. For these projects, frequent interaction points would save money by reducing rework.

That is, these pre-planned interaction points are scheduled to be frequent enough that any necessary rework identified will not be too expensive, but also rare enough not to be an excessive overhead cost. This view postulates that such a "happy medium" can be found; the possibility that it cannot, that is, that the best trade-off is still too expensive, adds support to the alternatives that we discuss further down.

Pre-planned, statically scheduled interaction points are akin to scheduled maintenance of equipment: they happen at predictable times, their cost is factored into the total cost from the beginning, and they are frequent enough to avoid nasty surprises. However, a regime of scheduled maintenance does not necessarily avoid ALL surprises and there is a need to have a design that can deal with failures occurring between maintenance points.

To continue this analogy, we can think of the product lifecycle process as having components (viz the design refinement cycle, the safety lifecycle, the security lifecycle) that may fail; it needs to tolerate these failures. Scheduled tests and diagnostics (statically scheduled interaction points) are a valuable defence. However, if components of a system fail during operation, the system typically needs: means for *failure detection*; means for *diagnosis*; means for *repair* or *reconfiguration, recovery and restart*. In the case of the co-engineered lifecycle, examples of failures and their detection mechanisms might be the following:

- Initial requirements from a client are found to be in conflict during the implementation phase (for instance encryption of data for a particular security standard takes too much time to meet a performance requirement). This may trigger interaction points in the current phase of the PLC, and/or in previous phases (that is, undoing some refinement activity for some system part, going back to change and re-analyse a higher-level design, so as to make the satisfaction of the requirements feasible; or even going back to renegotiate these requirements).

- Inadequate performance may lead to a safety related issue. For example, a machine vision component in an automated system may turn out to be insufficiently robust to adequately recognize a sufficiently large set of risky scenarios and may need to be upgraded for performance. The introduction of new, redundant mechanisms to deliver the needed performance might open up a new attack surface that was previously unanalysed.

- in the process of refining an aspect of design, the design team discovers that they violated some 'contract' established at a previous stage of refinement (e.g., they agreed to implement a certain

message encryption as a security control in less than a certain fraction of the main control loop period of a system; but they discover that when implemented it takes longer)

- the safety specialists realise that they may have missed out something important in communicating their proposed architecture to the security team; so, the analysis by the latter that gave the 'all clear' to the architecture may be wrong

- independently of an on-going development effort (or, alternatively, after deployment), a new vulnerability has been discovered in a component or algorithm. The security team wishes therefore to introduce new controls, which might violate some assumption made by the other teams (e.g. about timing, or about possibility of communication between two components, or authority given to a component) on the basis of the currently specified controls.

In all these cases, the "detection" amounts to somebody becoming aware of something potentially being wrong. Triggering an interaction point (possibly delayed, just as responsive maintenance can be delayed) then serves to perform *diagnosis*: to decide whether something is indeed wrong, possibly through intermediate steps of more extensive analysis. The interaction point may in turn trigger more extensive analyses (e.g., if our trust that a deadline would not be violated was built simply on extensive statistics of the delays observed in off-line testing, it may trigger another similar round of offline testing), just for the purpose of reaching a diagnosis, and then possibly some rework/redesign, again possibly requiring new analyses on the redesigned system. Analyses of the results of the rework/redesign would be subjected to another interaction point, to check that indeed the problem is resolved.

The detection mechanisms may be part of the current tasks of the various teams; or one may want (with the help of SESAMO and AQUAS experience) to prescribe some of them. The combination of statically scheduled interaction points and dynamically scheduled ones might prove more cost-effective than a more frequent series of statically scheduled ones. Another argument is that if potential problems are discovered, having as the only possible response "let's wait until the next interaction point" may be not only expensive money-wise but also in terms of project ethos.

One approach being considered in the automotive domain is to try to base the amount of interaction needed on a unifying concept of “risk”. While both safety and security are both firmly grounded in the concept of risk-based development, finding a mutually compatible definition that works for both sets of established practices has been a challenge. This challenge was addressed in the SESAMO project with partial success, and is being addressed once again in the development of new domain standards such as in the automotive sector. If such a compatible definition is found, then the amount of interaction could be determined by the relative “Risk Integrity Levels”: low risk projects would need less interaction because the consequences of missed interaction opportunities would not be severe, and vice versa. Note that this would also lead to the need for a compatible definition of risk for the performance dimension, which is not at all obvious (Risk of not achieving required performance levels? Risk of inadequate performance leading to safety or security violations? Risk of inadequate performance leading to low sales of the product?).

The risk-based approach to managing interaction points is an *a priori* approach: the risk levels are determined first, and then the interaction points follow from that scheme. But in AQUAS, we want to be able to experiment with both statically *and* dynamically determined interaction points, with more degrees of freedom than would be allowed by the purely risk-based approach outlined above.

In AQUAS we expect that demonstrators will initially plan static interaction points, but may study the option of dynamic ones, possibly find a need for triggering some interaction points dynamically, and report about experience with them.

4.3 System Design vs. Safety/Security/Performance Analyses

A system is designed to deliver specific functionality, i.e. to meet a specific set of functional requirements. Following a chosen PLC the system models undergo a set of refinements from more abstract forms (e.g. concept and requirements) to more specific forms (e.g. full implementation and deployment). The evolution of the system through the PLC is captured by models (the system documentation), chosen by the developers. In AQUAS the system models of most of the demonstrators will be based on the OMG SysML/UML formalisms. A significant part of it may be created directly from these models, including by e.g. automatic code generation. Should the system be changed (e.g. fixing faults/vulnerabilities in development or post-deployment), the system model will be modified too, so that the "real system" and the model of it are kept consistent throughout the phases of the PLC. It is with reference to this model or design of the system that the non-functional properties (e.g. safety, security and performance of interest in the AQUAS project) can be analysed, and analyses can flag problems and trigger recommendations for changes. There may be issues of incompleteness of this documentation, to which we will return later.

Assurance about the required non-functional properties of the designed system is achieved by dedicated *methods of analysis* (e.g. Safety, Security, Performance – SSP analysis), focused on assessing whether the system has the required non-functional properties or not. The SSP analysis, in full or in part (i.e. addressing only some of the SSP aspects) may be applied at any phase of the chosen PLC. The analysis is bound to operate with the details available in the particular phase of the PLC: it will be more abstract at the early phases of the PLC, using extensively assumptions about what the system will look like and how its components are likely to behave (e.g. in terms of reliability, performance, etc.), from which a judgement will be drawn about the likelihood that the *system properties* of interest meet their respective requirements. At the later stages of the PLC, some of the assumptions made earlier may be refined (i.e. changed to what becomes known via direct measurements).

Each one of the various methods used for analysing security, safety and/or performance relies on its *own models*. In some cases, these models coincide with parts of the design documentation: e.g., some verification methods are applied directly to source code or to state machine diagrams used in specifications. But for many SSP analyses, the models they need rely on formalisms that are *very different* from SysML/UML (or whatever else is used to document the designed product). E.g., performance modelling might use Petri nets or queuing networks. The models for SSP analysis are created either manually, or derived from the model of the designed system semi-automatically (e.g. CHESS supports "dependability modelling", MTTP tools support different types of modelling, etc.) or automatically.

The important point here is that whenever an SSP analysis is needed, a model suitable for it must be extracted or derived from the model of the designed system, available at that particular point in time. Two further important points are worth making here:

- Some methods of analysis (and their respective models) may not be applicable at all before the system model has matured enough. For example, the methods of analysis by AbsInt and BUT proposed for the ATM demonstrator rely on the existence of either the source code or of an executable of a piece of software.
- Some analyses may ignore some details of the designed system even if such details are available. For instance, if one uses a probabilistic state-based model such as Stochastic Petri Nets (SPN) one may be unable to benefit fully from having the full source code of the designed product. Instead, with SPN one will still need aggregated parameters characterising the designed product, e.g. probability of failure on demand, distribution of the response

time, etc. An analysis based on SPN may be used at different phases of a PLC but with a different precision of the aggregated modelling parameters: at early stages the parameter values may be derived from plausible assumptions; at later stages, the same parameters may be assigned values derived from direct measurements (e.g. from testing).

- The design models or design documentation are normally incomplete descriptions: they are meant to be sufficient guidance for implementing an operational system (e.g. for procuring and soldering hardware components and burning ROMs); but they may not include details that are important for verification of some properties. For instance, designers may specify the type of a microcontroller or memory chip to use in the system; to facilitate verification, appropriate data sheets for these products can be considered parts of the design documentation (even if not explicitly included); so can the detailed API specification of the operating system used. But implementation details inside these components may have major effects on non-functional properties[13]. So, analyses for security, safety etc. may require adding extensive "annexes" to system design documentation (and in particular may reveal the need for further annexes, as they proceed). In what follows we will leave this aspect implicit, and talk of the system design (or documentation or models) as though they already had the annexes required for these analyses. In the AQUAS exploration of interaction points it will be useful to report both to what extent the models used in design documentation need such additions, and how the process can ensure that they are requested and added as needed.

4.4 Different Forms of Analysis

We informally introduced the term “SSP analysis” without spelling out that in fact this abbreviation refers to a range of different analysis methods. Safety analysis (the first S in “SSP”) is a very mature discipline. The methods of analysis are defined in well-established and used international standards, e.g. generic standard IEC61508, sector standards like ISO26262 or standards on specific methods of analysis: EN60812 on FME(C)A, EN61025 on FTA, IEC61882 on HAZOP, etc. Often, the standards are supplemented by guidelines on how to apply specific techniques in practice. Similarly, techniques for security analysis (the second “S” in SSP) are defined in various standards, e.g. the generic ISO 27000 family if standards or more specific industrial standards, e.g. IEC 62443. These standards, which focus primarily on safety or on security, may acknowledge the impact of the other concern: e.g. IEC 61508 explicitly states that all hazards, including those due to malicious activity, must be recognised and managed.

Finally, there is an emerging trend of guidelines/standards of “SS(P) co-engineering”, addressing various aspects of dealing with more than one concern. An example is the recent SAE J3061, in which various ideas are put forward of alignment of the development processes of products in which safety and security are important.

When *combined analysis* is discussed, these considerations seem essential.

1. We undertake the combined analysis in order to overcome the limitations of the analysis methods focused on a single non-functional property (*unidimensional* analysis). Non-

[13] E.g., chip mask changes may have undocumented performance implications, or add/remove design faults; the much publicised "Spectre' and "Meltdown" vulnerabilities result from vendor-controlled chip design details that a system designer would typically ignore; and the new security/performance trade-offs required by the fixes for these vulnerabilities were arranged by vendors with limited communication to users.

functional properties may be inter*dependent variables* and the combined analysis deals with this dependence comprehensively, without unjustifiable assumptions that the dependence will take a particular form.

2. Different non-functional properties may be *in conflict*, e.g. improving security may only be possible at the expense of safety or performance, etc. Recognition of this specific form of dependence leads to the need for a rational *trade-off analysis*. Applying two unidimensional methods of analysis (e.g. of the integrity of the stored definition of the safe state that an embedded device must reach in case of failure, and of device safety which relies on the existence of that information) does not necessarily mean that a trade-off analysis is conducted. The trade-off analysis may rely on the results from the unidimensional analyses, but requires an *additional model* in which the event of an attack corrupting the information, and of the corrupted information causing harmful events, are both represented.

4.5 Interaction Points for Co-Engineering: Work and Information Flow

In AQUAS, we apply the term "co-engineering" to refer to the need to address in depth more than one of the three SSP qualities in building a dependable system.

Clearly, there can be different ways of implementing co-engineering. In system engineering, the safety and security engineering processes (performance is rarely a separate engineering process) address their respective concerns. The coordination between these processes typically depends on the organisation, on its adopted PLC, on which one of safety and security will take priority, etc. There may be *integrated phases* in the engineering process, where an activity is cooperatively conducted, for example static code analysis for checking compliance with MISRA C coding guide for safety and security, or we can have a *supportive activity* or information flow from one side to the other, e.g. safety supports the security analysis with information about hazards. Two simple cases are:

- Co-engineering adds a *new process step* as interaction (interaction points between the engineering processes – system engineering, safety and security engineering). Interaction itself can be implemented differently – from sessions of reviews and discussions between experts of system, safety and security engineering, to more formalised interaction of combined analysis supported by tools.
- Co-engineering adds an information flow for interaction. Here we also have a potential for optimization if the information flow was either missing or would be generated twice.

More complicated interaction patterns may occur, for example there can also be an iterative process between safety and security, but these two simple base cases should cover most interactions which are easy to identify and should bring rather immediate benefits. Especially the second case could be added to existing processes without much change.

The "interaction points" are the points in time when *combined analysis* is undertaken, regarding two or more of the SSP, non-functional properties. This may range from a mere communication of information from the system engineering team to some of the specialist teams or between the latter. The activities undertaken in interaction points will vary, but examples include:

- Review of documents, validation of the assumptions made in different models.
- Trade-off analysis between the system properties of interest with a view of checking whether the set targets for system SSP properties, established at earlier PLC phases, are still feasible given the more refined documentation (design) reached for the system. Should some of the targets be found infeasible, new sets of feasible targets for SSP properties will be established,

at system level or for subsystems (e.g. reallocating timing between software components, or reliability targets, or changing the "contracts" between subsystems about which properties of its behaviour one of them must "guarantee".

- The output of an interaction point will be:
    - a proposal for a change of the system design, in case the targets set at previous phases are revised, which in turn may require alteration of the system architecture (including modification of the mechanisms for safety and/or security);
    - system SSP targets are updated. These may be the same as those produced at previous phases or revised, in case the old targets have become infeasible.

Each interaction point is associated with a phase of the PLC (not necessarily fixed in the calendar), when some form of combined analysis is undertaken.

Clearly, the interaction points require an *information flow* (exchange of documents to review) between the system, safety and security engineering teams, and possibly direct communication (review meetings, discussions) between the teams. More specifically the communication will include the following:

- Information about the system (i.e. the current system model, e.g. SysML/UML) will acquire an increasing level of detail as the development progresses. As discussed above, each of the teams undertaking SSP analyses will need to derive a model, suitable for their specific analysis, from the system model.
- The results of the analysis will be communicated back to the system engineering team, which in turn may require design changes, resulting in a modification of the system model.
- An interaction point's outputs will include:
    - confirmation that the SSP targets are feasible; or
    - a trade-off analysis producing a new set of system SSP targets. In this case the IP may produce recommendations to the system engineering team for changes of the system design (architecture) by adding/removing/changing functional components (including safety mechanisms, security controls or policies or performance improving mechanisms) so that an acceptable trade-off can be achieved.

4.6 Tool Support

As noted elsewhere in this chapter, interaction points occur within the context of a number of questions:

- *Why* an interaction point would be needed (e.g. a potential conflict may arise)
- *When* an interaction point should take place (e.g. statically or dynamically determined)
- *What* will take place during the interaction point (e.g. joint examination of a design artefact, trade-off analysis of conflicting design decisions)
- *How* it will take place (e.g. manual observation and discussion, automated tool support, semi-automated tool support)

As challenging as the first two questions are, it is equally challenging to address the second two questions. That is, when an interaction point does occur, there must be a viable set of artefacts (at whatever level of abstraction or lifecycle phase) available.

- *What*. If, for example, two competing architectural designs have been created with entirely different and incompatible formalisms, the situation is not viable. The practical difficulties of reconciling them in the concrete project will be too onerous.
- *How*. Even if two artefacts have been created with the same formalism (e.g. SysML), there may be a lack of adequate tools to support the needed analyses (e.g. tools for worst-case performance analysis). More critically, even if the tools are individually available, they may not be able to interact due to poor planning of the overall toolchain framework.

Efficiency of interaction is also an important factor here. People might limit themselves to simpler analyses if it is too time/effort-consuming to do deeper analyses, such as the combined analyses for SSP. Inadequate tool interoperability and inconsistencies of modelling formalisms can severely hamper efficiency, but they can be addressed through emerging interoperability standards. In the end, tool interoperability and judicious automation will improve not only the economics of the work, but also the quality of the result.

In sum, the need for efficiency implies that it is critical to address effectively the What and the How in the above list. Some previous experience has been gathered in organizations which practice so-called "concurrent engineering." One example is the European Space Agency, which has run a Concurrent Design Facility for many years, in which engineers from multiple disciplines come together in a form of interaction points in order to work out compatibility issues in the design of space missions. For example, it may be necessary to coordinate requirements for weight, dimensions, and power supply among different disciplines. The When and Why are well understood by now. The What and How take place through a large set of spreadsheets that the different disciplines have worked out to be more or less compatible and exchangeable through the experience gained over the years.

The spreadsheet approach works fairly well in the Concurrent Design Facility because it is almost exclusively concerned with the early feasibility and concept-oriented phases of the mission. AQUAS must cover the entire life cycle, and therefore a more systematic approach is necessary. This could be offered, for example, by the coordinated viewpoints of SysML implemented in the CHESS tool, together with integration principles already experimented in earlier projects such as SESAMO (e.g. CHESS integration with the *Medini Analyse* tool of partner AMT). Such combination of analysis methods and of tools will be explored in the continuation of AQUAS WP3 and WP4.

4.7 "Likely" Interaction Points for the Use Cases

The approach taken in this deliverable to identifying the interaction points likely to occur in different use cases has been to allow the UC teams to concentrate on the real needs of the UC and the specifics of the PLC adopted in the development of the specific UC. This approach is consistent with the proposal.

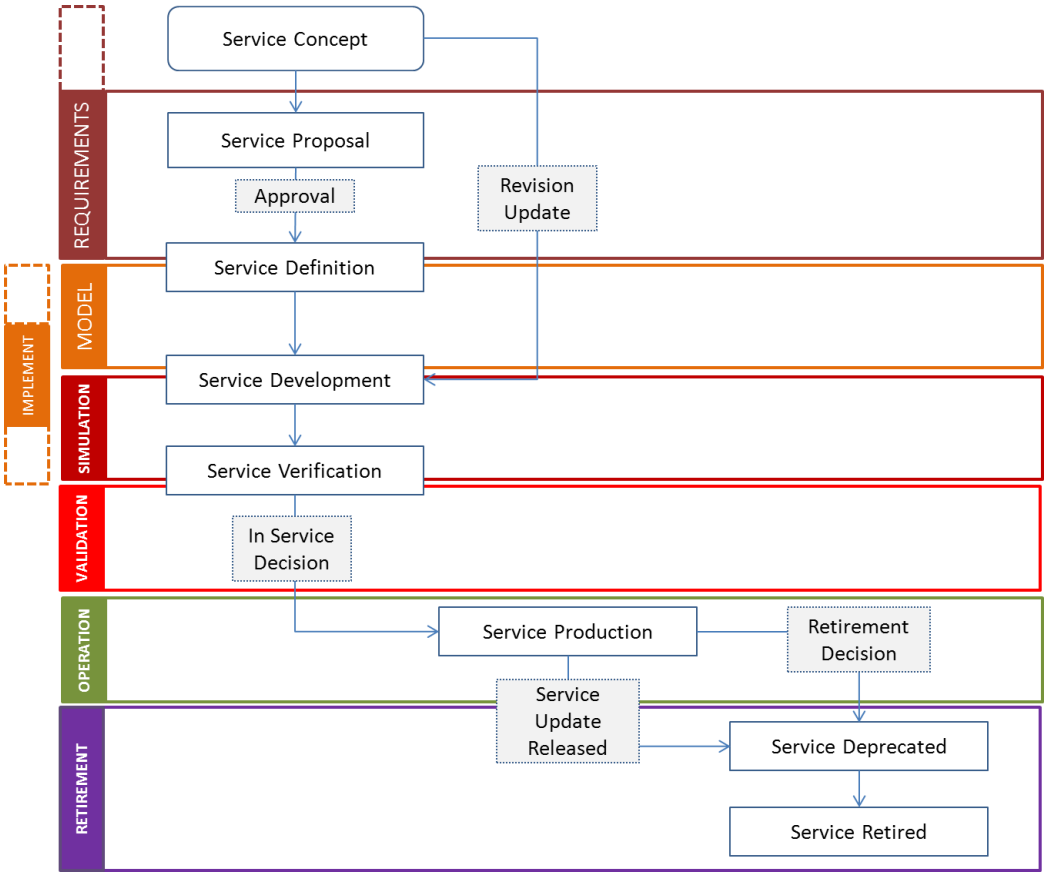
We did, however, discuss an alternative approach, e.g. to identify interaction points likely to be common and necessary irrespective of the PLC and the specifics of the UC. An analysis of commonalities between the IPs will be conducted at a later stage, as part of the work in T3.2. We also identified the need to look beyond the AQUAS demonstrators and how PLCs not practiced in AQUAS, e.g. "continuous integration" are likely to affect IPs. This effort, too, is expected to be undertaken later in the project – either in T3.2 or as part of the work on CE/PLC objectives in AQUAS.

In the following subsections we present the likely interaction points identified for some of the use cases considered in AQUAS. A more detailed analysis has been performed for the ATM and the Medial Devices use cases, whereas basic schemes of the interaction point study are provided for the

Industrial Drives and the Space Multicore Architecture use cases. We also refer to the work on IPs, reported in D2.2. A deeper study and analysis of these interaction points will be provided in the Task 3.2.

4.6.1. The ATM Use Case

The PLC for the ATM UC was presented in D2.2.1. Below we include a diagram depicting the adopted PLC for the UC.



The likely IPs identified for this UC are summarised below. They are grouped by PLC phase, which are shown in colour:

- ***Requirements*** phase is likely to include the following important IPs:
  - ***OTS selection***. This IP will concentrate on selecting the most suitable off-the-shelf (OTS) software to satisfy specific needs. D2.2 lists a number of decisions to be taken in this regard, e.g. selecting a "publish-subscribe middleware", which will be used in the ground services to deliver messages: i) from UAVs to the ground services and back to the UAV and possibly ii) among the UAV, which are in close proximity of the ground services. Among the options are Apache … and OMG DDS specification, for which a number of implementations exist both open-source and commercial. Similarly, the communication stack to be deployed on the UAVs will have to be chosen from the available OTS solutions. We expect an interaction point would include the interaction needed in order to select an OTS component for a particular need. If more than one component needs to be selected, then each will trigger a different instance of the IP: all these IPs (focused on

OTS software selection), however, are likely to be very similar and include activities such as: discussions among the teams with different area of expertise to organise a meaningful combined analysis of the candidate OTS alternatives (from the point of view of performance, security and some aspects of safety). For instance:

- TrustPort would undertake threat analysis to identify the credible threats and, possibly, vulnerabilities (e.g. via pen-testing)
- A number of partners (AbsInt and BUT) with expertise in performance evaluation would undertake performance analysis (e.g. by instrumenting the OTS software code) in "trusted" (i.e. without any cyber-attacks) and in "adverse" environment (i.e. when the OTS candidates are subjected to attacks identified as credible in the threat analysis), respectively.
- A model-based assessment (e.g. based on the work by City) will use the performance measurements to parameterise a probabilistic model and then use the model to study the likely behaviour of the OTS under variable load and adverse environment on the particular OTS, likely to occur in real operation.

The outcome of this analysis will be ranking the candidate OTS-es, available for a particular need, according to their performance in different environments (trusted vs. adverse) with variable load. This ranking then might be used by the UC owner to make a decision, which of the OS candidates to choose.

- ***Requirements feasibility and trade-off analysis***. This IP will be focused on establishing if the stated requirements related to performance and security are feasible. If not, the requirements will be traded-off until an acceptable and feasible set of requirements is achieved. The IP would include as inputs the results from threat and vulnerability analyses conducted by TrustPort on the ***entire system*** and a set of performance measurements on the entire system, which include end-to-end delays. The latter can be conducted using CHESS (Intecs), designed for such studies, which makes use of the empirical work conducted on OTS software components (see above). A combined system performance evaluation under attacks is also envisaged here. This will be conducted using state-based models under development by the City team, in which under plausible assumptions we will establish whether the stated performance requirements are achievable or not in the face of the attacks identified in the threat/vulnerability analysis. If the combined analysis reveals doubts that the required performance (e.g. end-to-end delays) can be achieved, additional security controls will be added to the system architecture, which would either reduce the likelihood of successful attacks or mitigate their consequences. The combined analysis will be repeated iteratively until the system enhanced with additional controls achieves the required performance under attacks. The combined analysis initially would be conducted using a model of the system developed by City, but this model may also serve as a guidance of how the CHESS functionality for ***dependability analysis*** should be enhanced, so that the combined analysis can be repeated entirely using the CHESS tool only.

- ***Modelling/Simulation.*** A detailed example of an interaction point including modelling/simulation is provided in D2.2.1. We also note that the IPs defined for Requirements above are, clearly based on models of system structure and behaviour (e.g. SysML for the system architecture in CHESS and probabilistic state-based models used for combined analysis). Some of these models may be "solved" via Monte-Carlo simulation.

- ***Validation.*** In validation we expect to apply a complex interaction point, in which the results from various analyses on the implemented demonstrator (performance measurement or performance model-based evaluation, e.g. using CHESS's built in functionality, BUT and AbsInt methods) and pen-testing (which reproduces the attacks identified for the demonstrator) will be fed into models of combined analysis, e.g. the City's state-based models to check if the stated performance requirements are satisfied by the demonstrator in adverse environment. We also envisage that some formal proofs (e.g. conducted by HSRM or AbsInt) of specific properties might be possible to make use of in validations. This attractive direction, however, is yet to developed in detail, which we would expect to be done in the next few months. The outcome of the combined analysis might be used by the UC owner, INTEGRASYS, helped by Tecnalia, in building an assurance case for the demonstrator. The interaction point may need to be repeated multiple times with modification of the components (debugging) or of the system architecture by adding/altering controls, as necessary, so that the validation demonstrates that the system requirements are satisfied.

  The description of the likely IPs for the ATM use case is given in the following table.

Interaction Point Identifier	Interaction Point Informal Description		Who	Is expected to do What	When	On the basis of which information (artefacts)	Attributes studied	Background Info	Status: [Certain/Possible] these are likely interation points:	Producing what kind of results (outputs)	Interactions between viewpoints: Review of Assumptions made in this analysis	Interactions between viewpoints: Review of Recommendations from this analysis	Known Gaps (i.e. what aspects of analysis will be omitted by the activities envisaged for this interaction point)	Relates to Tools
Interaction Point 1 (IAP_XX)	**Type:** Measurement-based interference analysis interaction point. **PLC Placement:** Concept Phase, software selection for the demonstrator. **Purpose:** Selecting an acceptable off-the-shelf SPLICE implementation. Selection based on measuring performance in trusted and non-trusted environment. **Activities:** Discussion of the performance measurements needed in adverse environement: all credible cyber-threats {Se1, Se2, ..., SeN, envisaged in the Se requirements} that potentially might affect the performance of a SPLICE implementation should be applied to a candidate SPLICE implementation and performance measurements should be conducted and the results recorded. The measurements results (Pe, SeX, X = 1,2, ..., N) will be reviewed and the implementation ranked according to an adopted methodology of ranking the individual threats (SE, Pe). An implementation will be selected which is ranked "sufficiently well".	Quantitative	City (assisted by TrustPort)	**Activity:** Measuring the performance of SPLICE implementations considered for selection under different cyber threats (one at a time) and ranking the implementations based on the ranking of threats (i.e. all other partners with that analysis attribute) **Method:** Direct performance measurements and statistical analysis of the results. **Output:** Se-Pe [table] of all SPLICE implementations considered for use with ATM	i) TrustPort has completed the threat analysis of SPLICE specification and provided scripts (pen-tests) to apply all credible threats to a SPLICE implementation. ii) BUT/Absint have provided the needed instrumentation to conduct performance measurements with a SPLICE implementation.	Threat Analysis (from TrustPort) report, Instrumentation for performance analysis by Absint, Instrumentation for "safety" analysis by BUT.	(Se, Pe) and also special forms of "Sa" (BUT specific "safety" analysis)	Requirements and technical demonstrator architecture from Demonstrator Architecture D2.2.1	Possible (unlikely to be used in the demonstrator since the selection will be based on other aspects, e.g. an open-source implementation)	Ranked candidate implementations of the SPLICE specifications which take into account the interference between Se and Pe.	Security analysis partner (e.g. TP) AND Performance analysis partner (Absint) AND "Safety" analysis (BUT)	Ranking off-the-shelf SPLICE implementations according to (Se, Sa, Pe)	Safety analysis is limited to "safety" in concurency control used in the SPICE implementations. System safety is not part of this analysis.	Absint tools. BUT tools. Pen-testing tools used by TrustPort
Interaction Point 2 (IAP_XX)	**Type:** Discussion based **PLC Placement:** Concept Phase, SeR inputs for the system. **Purpose:** Selecting necessary security parameters for the use-case for future evaluation with Performance/Safety **Activities:** Discussion of the security requirements for the selected use-case: all credible cyber-threats {SeR_1, SeR_2, ..., SeR_N, envisaged in the Se requirements}. Creating impact connections between Safety/Performance parameters (least to most influencing security parameters on Sa/Pe). The final security requirements for the system and for future methodology concept (analysis) should be selected.	Qualitative	Security partners (e.g. Trustport) together with Safety and Performance partners (and use-case leader)	**Activity:** Analytical work and discussion between partners **Method:** Best practice, Standards and Recommendation analysis followed by expert discussion. **Output:** Se-Pe [table] of all SPLICE implementations considered for use with ATM	(i) Each security partner has completed own SeR Table; (ii) security partners will review the requirements to each other and discuss the semi-final SeR table and add there suggested connection to Sa/Pe; (iii) other (Sa/Pe) partners will review the table and discuss with Se partners the final version of the PeR Table; (iv) Final version should be discussed and approved by use-case leader	Requirements analysis from the main standardizations, recommendations and best practice. (from security partners, e.g. Trustport) and connections will be based on the provided Sa/Pe	Se together with connections to Sa/Pe	*Requirements and technical demonstrator architecture from Demonstrator Architecture D2.2.1*	Possible	SeR table with clear connections to the SaR/PeR (excel table)	Security analysis partner (e.g. TP) will firstly review each other suggestions, the semi-final version of SeR table will be after that reviewed by other Sa/Pe partners (with closer look to the influences/connections). Last but not least, the final SeR table should be discussed and approved by use-case leader.	The Sa/Pe partners will review from their point the suggested SeR Table and provided connections to the SaR/PeR. Finally, the use-case leader should provide approval and final discussion on the SeR Table.	Safety and Performance parameters must be provided (they are not included in this IP) without clear definition of SaR/PeR the 2-dimensional connections (SeR-SaR and SeR-PeR) cannot be suggested	The SeR should be input for the modeling and analytical tools

Interaction points for the ATM use case

4.6.2. The Medical Device Use Case

For the medical device use case, the state of development of the product at the start of AQUAS collaborative work (we call it in this section "AQUAS baseline device") is

- There is a basic functional specification, and software and hardware architecture, for the essential subsystems of the future product: monitoring part, BP control algorithm, NMT control algorithm;
- There is an initial functional implementation (code) of the following parts: monitoring part (essentially the same as for the existing monitoring-only product), BP control algorithm. We call this "initial, functional implementation" because it is meant to the support soon the testing of the basic functional algorithms in simulated use conditions. So, it does not include finalised design or code for e.g. fault detection, treatment of exceptions, security protocols, user interface.

We note that for this product, as is frequent for innovative products, although final documentation must assure traceability of requirements and assurance of their satisfaction according to some variant of a V lifecycle, one should not assume that development proceeds top-down uniformly for all subsystems. Different subsystems are at different stages of development, and the detailed safety and security assurance activity has a delayed start compared to parts of implementation, because a) parts of the implementation exist as previous product(s), and b) implementation of some novel functions is needed first as proof of viability of the functional concept, and only later it becomes appropriate to analyse details of the safety/security concerns.

The ideal state of results at the end of AQUAS includes

- The "hardware in the loop (HiL) prototype" for testing of the essential subsystems of the device;
- A test plan, and results of applying it in the HiL so as to give evidence (produced by using the HiL) that a device (including one or both the above control methods) is ready for proceeding to testing in a laboratory environment;
- A set of methods for analysis of safety, security and performance issues, that can be reused as desired on similar devices; and results of application of some steps of these methods to orient design and verification of the present device so as to facilitate certification;
- The device to be tested in the laboratory environment, which may differ from the AQUAS baseline device in that it includes more complete implementation, including some safety/security-oriented design details selected through the above methods.

So, stages of development at which interaction points should occur include:

1. **Requirements definition and verification of the AQUAS baseline design**. Here the interacting activities include: some form of hazard analysis, preliminary verification that the various categories of non-functional requirements are compatible in view of the likely implementation, and that the high-level design can satisfy them. Any findings about possible conflicts and trade-offs could result in changes: to the requirements themselves, within the constraints imposed by certification requirements; to the high-level hardware/software architecture; to the user interface and specifications of intended use; to the already implemented parts. Apart from the methods applied by RGB, relevant methods applied by other partners include
   a. Analysis of requirements imposed by relevant standards (Tecnalia, Trustport);
   b. Analysis of certain human factors issues in safety (City);
   c. Preliminary analysis of potential performance issues caused by safety/security concerns (All4Tec, with use of A2K or CHESS);

d. Preliminary FMEVA (All4Tec, possibly AMT) on existing design;
e. Documentation of the requirements and design in one or more tools. This first IP will be paper-documented though it will drive specs for the planned features in A2K. Possibly there will be trial use of other tools (CHESS, Medini analyser), to study how to use tools to spot conflicts or document their absence.

Outputs at this stage include, besides refined requirements documents and hazard analysis, notes about (1) *potential* conflicts identified that should be checked at later stages of refinement of the design, (2) constraints on future refinement and documentation that arise from regulatory and certification requirements.

2. **System verification, at one or more stages of advancement of the development**, after the baseline stages. The exact stages when these interaction points will take place are TBD but one will concern the last version of the device in AQUAS. These interaction points will bring together:
   a. Code-level analyses for performance and for defects with potential safety/security implications (CEA, BUT, etc.), e.g. static code analysis (Frama-C) to verify absence of undefined behaviours which lead to potential safety risks and security vulnerabilities. The low-level properties specified at code level must be relevant to the requirements and software architectures derived at the previous stage. Thus a potential interaction activity is the translation of the requirements and architectures (derived by safety, security, and performance analysis partners) to correct low-level code properties (exploitable by code analysis partners) that shall be verified to check the (non-) satisfaction of said requirements. For this interaction, inference and code generation (Papyrus) techniques can be explored.
   b. Checks that the parts of design that have been refined from the previous stage satisfy non-functional requirements set at the earlier stage; in part though translation to tool-treatable format and in part by hand.

   Outputs of the activities in these interaction points may include documentations of conflicts arising from implementation, and thus also documentation of changes decided to resolve the conflicts.

3. **Definition of test plans** for the HiL environment. Although this is just a part of the required steps for testing, it will require finalising the specification of what properties the testing will verify and thus require combined checks to ensure that the tests are sufficient (and do not exceed too much what is necessary) for demonstrating the required properties. This will bring together expertise on:
   a. Functional requirements of the control algorithms;
   b. Disruptions to be tolerated, with possible origins in operation environment, security, hardware/software failures and human factors;
   c. Regulatory requirements;
   d. A2K will be central to the test planning; other tool requirements that may arise will be explored.

The description of the likely IPs for the Medical Device use case is given in the following table.

Interaction Point Identifier	Interaction Point Informal Description		Who	is expected to do What	When	On the basis of which information (artefacts)	Attributes studied	Background Info	Status: [Certain/Possible] these are likely interation points.	Producing what kind of results (outputs)	Interactions between viewpoints: Review of Assumptions made in this analysis	Interactions between viewpoints: Review of Recommendations from this analysis	Known Gaps (i.e. what aspects of analysis will be omitted by the activities envisaged for this interaction point)	Relates to Tools
Interaction Point X (IAP_XX)	**Type:** Discussion based **PLC Placement:** Concept Phase, SeR inputs for the system. **Purpose:** Selecting necessary security parameters for the use-case for future evaluation with Performance/Safety **Activities:** Discussion of the security requirements for the selected use-case- all credible cyber-threats (SeR_1, SeR_2, ..., SeR_N) envisaged in the Se requirements). Creating impact connections between Safety/Performance parameters (least to most influencing security parameters on Se/Pe). The final security requirements for the system and for future methodology concept (analysis) should be selected	Qualitative	Security partners (e.g. Trustport) together with Safety and Performance partners (and use-case leader)	**Activity:** Analytical work and discussion between partners **Method:** Best practice, Standards and Recommendation analysis followed by expert discussion. **Output:** Se-Pe [table] of all SPLICE implemntations considered for use with ATM	(i) Each security partner has completed own SeR Table; (ii) security partners will review the requirements to each other and discuss the semi-final SeR table and add there suggested connection to Se/Pe; (iii) other (Se/Pe) partners will review the table and discuss with Se partners the final version of the PeR Table; (iv) Final version should be discussed and approved by use-case leader	Requirements analysis from the main standardisations, recommendations and best practice. (from security partners, e.g. Trustport) and connections will be based on the provided Se/Pe	Se together with connections to Se/Pe	*Requirements and technical demonstrator architecture from Demonstrator Architecture D2.2.1*	Possible	SeR table with clear connections to the SeR/PeR (excel table)	Security analysis partner (e.g. TPI) will firstly review each other suggestions, the semi-final version of SeR table will be after that reviewed by other Se/Pe partners (with closer look to the influences/connections). Last but not least, the final SeR table should be discussed and approved by use-case leader.	The Se/Pe partners will review from their point the suggested SeR Table and provided connections to the SeR/PeR. Finally, the use-case leader should provide approval and final discussion on the SeR Table.	Safety and Performance parameters must be provided (they are not included in this IP) without clear definition of the SeR/PeR the 2-dimensional connections (SeR-SeR and SeR-PeR) cannot be suggested	The SeR should be input for modelling and analytical tools

Interaction points for the Medical Device use case

4.6.3. The Industrial Drives Use Case

The description of the likely IPs for the Industrial Drives use case is given in the following table.

Interaction Points Outline based on Interaction PointTemplateMinimal_v01.xlsx Note: This list is a draft and is based on our current understanding of tools/methods.	Note: The following is a proposal for interaction points in the Concept Phase of the UC4 PLC. The timely organization can be seen in a separate Gantt chart (see the separate Excel file).	Legend: FR... (Non-)Functional Requirements SeR ... Security Requirements SaR ... Safety Requirements PeR ... Performance Requirements	Define System Design							
Interaction Point Identifier	Interaction Point Informal Description	Analysis	Who	On the basis of which information (artefacts)	Attributes studied	Producing what kind of results (outputs)	Gaps	Comments
Interaction Point 5 (IAP_05)	**Type:** Interference analysis. **PLC Placement:** System Design. **Purpose:** Determine the interference of technical safety and security measures on SSP requirements. Goal of the analysis is to find a system design that holds on an acceptable set of SSP requirements. **Activities:** Analysis of the system design and newly introduced safety and security measures.		AMT (Medini Analyze): SysML system design models and safety/security requirements tracking. MDS (MVP) : system design in SystemC with simulation and PPS requirements tracking CITY (SAN): Quantitative system assessment wrt SSP. MTTP (TTool) system design with respect to SSP requirements.	Preliminary Architecture, Functional SaR, SeR, PeR (e.g. SLS with demanded max. failure rate), non-functional requirements (e.g. operation ambient temperature)	SSP	Safe/Secure System Design that keeps performance requirements (recommendations) Technical SaR, SeR, PeR (recommendations)		
Interaction Point 6 (IAP_06)	**Type:** Discussion-based consolidation and alignment **PLC Placement:** System Design. **Purpose:** Consolidation of the system design concept recommendations and SSP requirements comments of the various partners/tools. **Activities:** Discussion and alignment of results between the partners and creation of a common system design and set of SSP requirements.		AMT (Medini Analyze) MDS (MVP) CITY (SAN) IN TECS (CHESS) MTTP (TTool)	Safe/Secure System Design that keeps performance requirements (recommendations) Technical SaR, SeR, PeR (recommendations)	SSP	Aligned/Consolidated Safe/Secure System Design that keeps performance requirements Aligned/Consolidated Technical SaR, SeR, PeR		If no system design is found as an acceptable solution, then go back to AQUAS LC phase 1. If the newly derived technical SSP requirements lead to a change of the functional SSP requirements, then IAP_01 in the Concept Phase shall be re-triggered.
Interaction Point 7 (IAP_07)	**Type:** Interference Analysis. **PLC Placement:** System Design. **Purpose:** Consolidation of the system design concept (split into HW and SW) recommendations and SSP requirements comments of the various partners/tools. **Activities:** Discussion and alignment of results between the partners and creation of a common system design and set of SSP requirements.		IN TECS (CHESS) : determination of WCRT (worst-case response time) analysis of the current system design in conjunction with Medini Analyze. MDS (MVP) : system design in SystemC with simulation and PPS requirements tracking CITY (SAN): Quantitative system assessment wrt SSP. MTTP (TTool) system design with respect to SSP requirements.	Aligned/Consolidated Safe/Secure System Design that keeps performance requirements Aligned/Consolidated Technical SaR, SeR, PeR	SSP	Safe/Secure System Design (HW/SW architecture) that keeps performance requirements (recommendations) Technical SaR, SeR, PeR (recommendations)		
Interaction Point 8 (IAP_08)	**Type:** Discussion-based consolidation and alignment. **PLC Placement:** System Design. **Purpose:** Consolidation of the system design concept (split into HW and SW) recommendations and SSP requirements comments of the various partners/tools. **Activities:** Discussion and alignment of results between the partners and creation of a common system design and set of SSP requirements.		MDS (MVP) CITY (SAN) IN TECS (CHESS) MTTP (TTool)	Safe/Secure System Design (HW/SW split) that keeps performance requirements (recommendations) Technical SaR, SeR, PeR (recommendations)	SSP	Aligned/Consolidated Safe/Secure System Design with a split into HW/SW that keeps performance requirements Aligned/Consolidated Technical SaR, SeR, PeR		If no system design (HW/SW split) is found as an acceptable solution, then go back to AQUAS LC phase 1. If the newly derived technical SSP requirements lead to a change of the functional SSP requirements, then IAP_01 shall be re-triggered.
Interaction Point 9 (IAP_09)	**Type:** Tool interfacing for co-simulation. **PLC Placement:** Development. **Purpose:** Enable the co-simulation of AMESim (motor model) and the SystemC domain (motor control board) **Activities:** Implementation of an infrastructure that enables co-simulation of both tools		SSW (AMESim) works on the bridging to SystemC. SAG (Virtual Prototype - SystemC) provides SystemC models.	Virtual Prototype implementation in SystemC AMESim simulator with electric motor model	-	Co-simulation system for SystemC/AMESim		
Interaction Point 10 (IAP_10)	**Type:** Virtual prototype security verification. **PLC Placement:** Integration and Testing. **Purpose:** Security-related verification of the virtual prototype: Communication channels between motor control application and board are tested, as well as tests on the linux-based operating system residing on the motor control board. **Activities:** Virtual prototype security analysis.		SAG (Virtual Prototype - SystemC / AMESim) provides the implemented system for security testing. TRUSTPORT (SSDLC) handles SSDLC for security verification.	Virtual Prototype co-simulation system with SystemC models and AMESim motor model SSDLC adapted to the UC	Se	Verification Report from SSDLC (list of tested security requirements)		

Interaction points for the Industrial Drives use case

4.6.4. The Space Multicore Architectures Use Case

The description of the likely IPs for the Space Multicore Architectures use case is given in the following table.

Interaction Point Identifier	Interaction Point Informal Description		Who	Is expected to do What	When	On the basis of which information (artefacts)	Attributes studied	Background Info	Status: (Certain/Possible) these are likely interation points.	Producing what kind of results (outputs)	Interactions between viewpoints: Review of Assumptions made in this analysis	Interactions between viewpoints: Review of Recommendations from this analysis	Known Gaps (i.e. what aspects of analysis will be omitted by the activities envisaged for this interaction point)	Relates to Tools
Interaction Point 1 (IAP_XX)	Type: Performance-based design space exploration system refinement interaction point. PLC Placement: SW requirements and architecture engineering process. Purpose: Check different off-the-shell HW/SW architectures and DSE solutions. DSE led by evaluating performance using different tools and simulation activities. Activities: Discussion on the performance analysis needed to achieve and refine search steps, and (Sa, Pe) attributes that should be used to drive better the search of solution suitable for RB. The input models should be the application model, the platform model and (Sa, Pe). A deployment model will be selected which fulfill RB. TS could be enhanced in order to suggest different suitable solution for PDR and to guarantee a fast production of CDR.	Qualitative	UNIVAQ (assisted by INTECS, ALL4TEC, ITI/TRT, Axsim and SysGO)	Activity: Enhance design space exploration methodology considering (Sa, Pe) metrics taken from different partners activities (involved in performance, schedulability analysis and timing simulation). Introducing FPGA accelerators into the proposed solution, considering hypervisor system behavioral testing and features. Method: Direct inject performance measurements and simulation, enhancing platform models to consider different HW/SW technologies. Output: (Sa, Pe) of all DSE solution finded as a different implementation for uCS	I) ITI has completed code analysis with Leon3 and FPGA II) TRT has extended and adapted Time4Sys to model dynamic architecture of the case-study, so model transformation should apply to inject the model into the timing verification tool (ITI/BUT/Axsim) III) INTECS has applied CHESS methodology to the uCS. III) ALL4TEC will apply Safety Architect tool for safety Analysis	Schedulability and timing analysis (from ITI, Axsim), performance analysis (from TRT), PikeOS kernel static analysis report (from SysGO, Axsim), application models (from INTECS), Safety analysis on application model (from ALL4TEC)	(Sa, Pe)	Requirements and technical demonstrator architecture from Demonstrator Architecture D2.2.5, System Models from other partners	Possible	Deployment models (needed for A2K) and implementation solution which take into account (Sa, Pe) attributes.	System level modelling partner (INTECS) AND Performance analysis partner (TRT) AND Timing analysis partners(ITI, Axsim), Safety analysis on system level modelling (ALL4TEC)	Implementation solution according to (Sa, Pe)	Safety analysis is limited to "Safety" in partitioning and deployment activities. System safety assurance steps is not part of this analysis.	ITI tools, TRT tools, INTECS tools, SysGO PikeOS toolchains, Axsim tools, ALL4TEC tools

Interaction points for the Space Multicore Architectures use case

5 Conclusions

In this document, various methods and techniques for the assessment of safety, security and performance have been presented as reference documentation of the techniques that are provided by the AQUAS partners and planned for the use in the project. Furthermore, general considerations on implementing the concept of "interaction point" have been provided. In addition, the interaction points likely to occur in different use cases have been identified by different demonstrator teams according to the needs of each use case and the specifics of the product lifecycle adopted in the development of a particular use case.

6 References

[Bishop 2011] P. Bishop, R. Bloomfield, I. Gashi, and V. Stankovic, Diversity for Security: A Study with Off-the-Shelf AntiVirus Engines. 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE 2011), Nov. 29 2011-Dec. 2 2011 2011. 11-19.

[Gashi 2009] I. Gashi, C. Leita, O. Thonnard, and V. Stankovic, An Experimental Study of Diversity with Off-the-Shelf AntiVirus Engines. Proceedings of the 8th IEEE Int. Symp. on Network Computing and Applications (NCA), 2009. IEEE Computer Society, 4-11.

[Gashi 2004] I. Gashi, P. Popov, V. Stankovic, and L. Strigini 2004. On Designing Dependable Services with Diverse Off-The-Shelf SQL Servers. In: DE LEMOS, R., GACEK, C. & ROMANOVSKY, A. (eds.) Architecting Dependable Systems II. Springer.

[Gashi 2207] I. Gashi, P. Popov, and L. Strigini 2007. Fault tolerance via diversity for off-the-shelf products: a study with SQL database servers. IEEE Transactions on Dependable and Secure Computing, 4, 280-294

[Leita 2008] C. Leita, and M. Dacier, A Worldwide Deployable Framework to Support the Analysis of Malware Threat Models. 7th European Dependable Computing Conference (EDCC), 7-9 May 2008 2008 Kaunas, Lithuania. 99 - 109.

[Littlewood 2000] B. Littlewood, and L. Strigini 2000. A discussion of practices for enhancing diversity in software designs. DISPO project reports. London: City, University of London.

[Popov 2012] P. Popov, V. Stankovic, and L. Strigini 2012. An exploratory study about the effectiveness of “diversity-seeking decisions” based on a large population of diverse programs. DISPO project reports. London: City University London.

[Popov 2014] P. Popov T., A.A. Povyakalo, V. Stankovic, and L. Strigini Software diversity as a measure for reducing development risk. 10th European Dependable Computing Conference - EDCC, 2014 Newcastle upon Tyne, UK.

[Schneider 2012] F. Schneider, 2012. Blueprint for a Science of Cybersecurity. The Next Wave, 19, 47-57.

[Stankovic 2006] V. Stankovic, & P. Popov Improving DBMS Performance through Diverse Redundancy. 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06), 2006 Leeds, United Kingdom. IEEE Computer Society, 391-400.

[Vandiver 2007] B. Vandiver, H. Balakrishnan, B. Liskov, and S. Madden, 2007. Tolerating byzantine faults in transaction processing systems using commit barrier scheduling. Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles. Stevenson, Washington, USA: ACM.

[CHESS] A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, A. Zovi and T. Vardanega, CHESS: A Model-Driven Engineering Tool Environment for Aiding the Development of Complex Industrial Systems, Proceedings of Automated Software Engineering (ASE) International Conference, Essen, July 2012.

[CHESS FLA] B. Gallina, M. Atif Javed, F. Ul Muram and S. Punnekkat. Model-driven Dependability Analysis Method for Component-based Architectures. In proceedings of the Euromicro-SEAA Conference, Cesme, Izmir, Turkey, September, 2012.

[CHESS FLA2] B. Gallina, E. Sefer and A. Refsdal, "Towards Safety Risk Assessment of Socio-Technical Systems via Failure Logic Analysis," 2014 IEEE International Symposium on Software Reliability Engineering Workshops, Naples, 2014, pp. 287-292.

[CHESS-SBA] L. Montecchi, P. Lollini, A. Bondavalli, A reusable modular toolchain for automated dependability evaluation., VALUETOOLS 2013, Torino, Italy, Dec 2013, pp. 298-303.

[CHESS-SBA2] L. Montecchi, P. Lollini, and A. Bondavalli. Towards a MDE Transformation Workflow for Dependability Analysis. In: IEEE International Conference on Engineering of Complex Computer Systems. Las Vegas, USA, 2011, pp. 157–166.

[FoReVer] L. Baracchi, A. Cimatti, G. Garcia, S. Mazzini, S. Puri and S. Tonetta, Requirements refinement and component reuse: the FoReVer contract-based approach, in A. Bagnato, I. R. Quadri, M. Rossi and l. S. Indrusiak, Editors Industry and Research Perspectives on Embedded System Design, IGI Global, Hershey PA, USA 2014.

[MAST] MAST: Modeling and Analysis Suite for Real-Time Applications. [Online], Available: http://mast.unican.es/ [Accessed: January 22, 2016].

[OCRA] OCRA: A command-line tool for the verification of logic-based contract refinement for em-bedded systems. [Online], Available: https://es-static.fbk.eu/tools/ocra/ [Accessed: January 22, 2016].

[CHESS2] Mazzini, S., Favaro, J., Puri, S., Baracchi, L.: CHESS: an open source methodology and toolset for the development of critical systems. Third Workshop on Open Source Software for Model Driven Engineering (OSS4MDE 2016)

[CHESSML] https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf

[CONTR1] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple Viewpoint Contract-Based Specification and Design. In FMCO, pages 200–225, 2007.

[CONTR2] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K.G. Larsen. Contracts for System Design. Technical Report RR-8147, INRIA, November 2012.

[CONTR3] A. Benveniste, B. Caillaud, and R. Passerone. A Generic Model of Contracts for Embedded Systems. Technical report, INRIA, 2007.

[CONTR4] S.S. Bauer, A. David, R. Hennicker, K.G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from Specifications to Contracts in Component-Based Design. In FASE, pages 43–58, 2012.

[CONTR5] M. Bozzano, A. Cimatti, C. Mattarei, and S. Tonetta. Formal Safety Assessment via Contract-Based Design. In ATVA, 2014.

[DEEM] A. Bondavalli, I. Mura, S. Chiaradonna, R. Filippini, S. Poli, and F. Sandrini. DEEM: a tool for the dependability modeling and evaluation of multiple phased systems. In Proc. of the Int. Conference on Dependable Systems and Networks (DSN2000), pages 231.236, New York, USA, June 2000.

[SSFRT] S. Mazzini (ed.), S. Puri, X. Olive, C. Paccagnini, E. Tronci: "Guidelines for Model Based Space System Engineering ", SSFRT Report, 2009.

[MBSSE] Mazzini S., Olive X., Tronci E., A Model-Based methodology to support the Space System Engineering (MBSSE), Proc. of Embedded Real Time Software and Systems Conf. (ERTSS), 2010.

[OBSW-RA] SAVOIR-FAIRE Working Group: Space on-board software reference architecture, Proceedings of DASIA Conference, Budapest, May 2010.

[CONCERTO] Baldovin A., Zovi A., Nelissen G. and Puri S., CONCERTO: A toolset for model-based engineering of avionics applications, Proc. of 20th International Conference on Reliable Software Technologies- Ada-Europe, Madrid, June 2015.

[BUT-Atomrace] Zdeněk Letko, Tomáš Vojnar, and Bohuslav Křena. 2008. AtomRace: data race and atomicity violation detector and healer. In Proceedings of the 6th workshop on Parallel and distributed systems: testing, analysis, and debugging (PADTAD '08). ACM, New York, NY, USA, Article 7, 10 pages. DOI: https://doi.org/10.1145/1390841.1390848

[BUT-Anaconda] Fiedor J., Vojnar T. (2013) ANaConDA: A Framework for Analysing Multi-threaded C/C++ Programs on the Binary Level. In: Qadeer S., Tasiran S. (eds) Runtime Verification. RV 2012. LNCS, vol 7687. Springer, Berlin, Heidelberg.

[BUT-Predator] K. Dudka, P. Peringer, and T. Vojnar. Byte-Precise Verification of Low-Level List Manipulation. In Proc. of 20th Static Analysis Symposium (SAS'13), Seattle, USA, volume 7935 of LNCS, pages 215--237, 2013. Springer-Verlag.

[BUT-Loopus] M. Sinn, F. Zuleger, and H. Veith. Complexity and Resource Bound Analysis of Imperative Programs Using Difference Constraints. Journal of Automated Reasoning, volume 59, issue 1, pp. 3–45, 2017.

[BUT-Ranger] T. Fiedor, L. Holik, A. Rogalewicz, M. Sinn, T. Vojnar, and F. Zuleger. From Shapes to Amortized Complexity. To appear in Proc. of VMCAI'18, LNCS, Springer, 2018.

[BUT-Perun] T. Fiedor. Perun: Lightweight Performance Version System. Available online at . [Checked Nov. 29, 2017.]

[Eclipse 2017] Eclipse, 2017. Opencert [WWW Document]. Eclipse/Polarsys. URL https://polarsys.org/opencert/index/ (accessed 11.27.17).

[Hawkins 2013] Hawkins, R., Habli, I., Kelly, T., McDermid, J., 2013. Assurance cases and prescriptive software safety certification: A comparative study. Saf. Sci. 59, 55–71. doi:10.1016/j.ssci.2013.04.007

[Kelly 2004] Kelly, T., Weaver, R., 2004. The Goal Structuring Notation – A Safety Argument Notation, in: Dependable Systems and Networks - DSN.

[Larrucea 2017] Larrucea, X., Walker, A., Colomo-Palacios, R., 2017. Supporting the Management of Reusable Automotive Software. IEEE Softw. 34, 40–47. doi:10.1109/MS.2017.68

[Object Management Group 2017] Object Management Group, 2017. Structured Assurance Case Metamodel [WWW Document]. URL http://www.omg.org/spec/SACM/ (accessed 11.27.17).

[Chinneck] P. Chinneck, Pumfrey, D.J., Kelly, T., n.d. Turning up the HEAT on Safety Case Construction, in: Redmill, F., Anderson, T. (Eds.), Twelfth Safety-Critical Systems Symposium. Springer, Birmingham.

[Piètre-Cambacédès 2013] Piètre-Cambacédès, L., Bouissou, M., 2013. Cross-fertilization between safety and security engineering. Reliab. Eng. Syst. Saf. 110, 110–126. doi:10.1016/j.ress.2012.09.011

[Redmill 2008] Redmill, F., Anderson, T. (Eds.), 2008. Improvements in System Safety, the Sixteenth Safety-critical Systems. Springer London, London. doi:10.1007/978-1-84800-100-8

[Törner 2008] Törner, F., Öhman, P., 2008. Automotive Safety Case A Qualitative Case Study of Drivers, Usages, and Issues. 2008 11th IEEE High Assur. Syst. Eng. Symp. 313–322. doi:10.1109/HASE.2008.44

[Wassyng 2011] Wassyng, A., Maibaum, T., Lawford, M., Bherer, H., 2011. Software Certification : Is There a Case against Safety Cases?, in: Proceedings of the 16th Monterey Conference on Foundations of Computer Software: Modeling, Development, and Verification of Adaptive Systems. Springer-Verlag, Redmond, WA, pp. 206–227.

[Zeng 2012] Zeng, F., Lu, M., Zhong, D., 2012. Software Safety Certification Framework Based on Safety Case. 2012 Int. Conf. Comput. Sci. Serv. Syst. 566–569. doi:10.1109/CSSS.2012.147

[Barabas 2012] Barabas, M., Drozd, M., Hanáček, P.: Behavioral signature generation using shadow honeypot, In: World Academy of Science, Engineering and Technology, Vol. 2012, No. 65, US, p. 829-833, ISSN 2010-376X.

[Barabas 2013] Barabas, M., Homoliak, I., Drozd, M., Hanáček, P.: Automated Malware Detection Based on Novel Network Behavioral Signatures, In: International Journal of Engineering and Technology, Vol. 5, No. 2, 2013, Singapore, SG,p. 249-253, ISSN 1793-8236

[Safety Architect] http://www.all4tec.net/en/safety-architect

[Cyber Architect] http://www.all4tec.net/en/cyber-architect

[EBIOS] https://www.ssi.gouv.fr/guide/ebios-2010-expression-des-besoins-et-identification-des-objectifs-de-securite/

[MERgE_A4T] Julien Brunel, David Chemouil, Laurent Rioux, Mohamed Bakkali, Frédérique Vallée. A Viewpoint-Based Approach for Formal Safety & Security Assessment of System Architectures. 11th Workshop on Model-Driven Engineering, Verification and Validation, Sep 2014, Spain. pp.39-48.

[Capella-SA] Marc Sango and Régis de Ferluc. An Integrated Model-Based Tool-Chain for Safety Assessment in Early Validation of System Architectures. (Poster). IMBSA'2017

[Apvrille 2016] Ludovic Apvrille, Yves Roudier, "Designing Safe and Secure Embedded and Cyber-Physical Systems with SysML-Sec", Chapter in Model-Driven Engineering and Software Development, p293--308, Springer International Publishing, 2016

[Schm2014a] C. Schmittner, T. Gruber, P. Puschner, and E. Schoitsch, Security application of failure mode and effect analysis (FMEA), International Conference on Computer Safety, Reliability, and Security (SafeComp 2014), 2014.

[Arrow2013] The ARROWHEAD project, http://www.arrowhead.eu/.

[Schm2014b] C. Schmittner, Z. Ma and P. Smith, FMVEA for Safety and Security Analysis of Intelligent and Cooperative Vehicles, Computer Safety, Reliability, and Security: SAFECOMP, Florence, Italy, September 8-9, 2014.

[Schm2015] C. Schmittner, Christoph, et al., A case study of fmvea and chassis as safety and security co-analysis method for automotive cyber-physical systems, Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, 2015.

[Frama] https://frama-c.com

[Spark 2014] http://www.spark-2014.org/about

[Stanford] http://cvc4.cs.stanford.edu

[Alt-ego] http://alt-ergo.lri.fr

[Coq] https://coq.inria.fr

[Cl.clam] https://www.cl.cam.ac.uk/research/hvg/Isabelle

[Why3] http://why3.lri.fr

[Adacore] https://www.adacore.com

[Why3.lri] http://why3.lri.fr/doc-0.80/manual004.html

[Barrett] Satisfiability Modulo Theories by Clark Barrett et. al. From the Handbook of Satisfiability, Chapter 26 (http://disi.unitn.it/~rseba/papers/p02c12_smt.pdf)