(ITEA 3 – 17003)

PANORAMA
Boosting Design Efficiency for Heterogeneous^3 Systems

Deliverable:
Living Roadmap

Work Package:
Contribution to ITEA Living Roadmap

Task:
State of the art
Data Models, Data transformation, Analysis Methods, Visualizations and Data Traceability

Document Type:	Deliverable	**Classification:**	Public
Document Version:	Revision 1	**Contract Start Date:**	2019-04-01
Document Preparation Date:	2020-07-13	**Duration:**	2022-03-31

History

Rev.	Content	Resp. Partner	Date
1	Initial version	Project partners	2020-07-17

Contents

Contents

Contents

List of Figures

List of Tables

Summary

This deliverable is a summary of the State of the Art analysis done within the work-packages of the Panorama project. The project is split into several work-packages to cover the different aspects and needs.

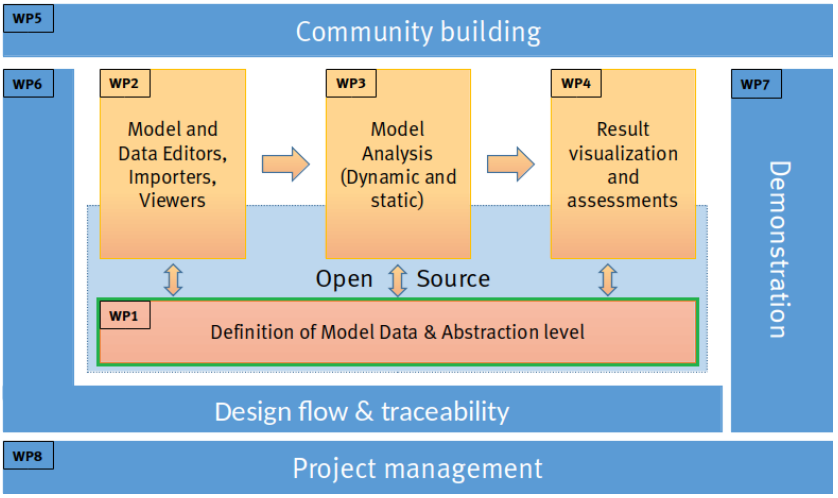


Figure 1: PANORAMA Work Package Structure

The project is organized into two groups of work packages: The technical solutions to the project objectives are developed by WP1-4 (colored in red / orange), whereas WP5-8 cover management, packaging, and dissemination related topics (colored in blue). Within the following chapters we are focusing on the technical areas wherein the state of the art analysis have been done.

1 Executive Summary

1.1 Purpose and objectives

The complexity of future solutions for mobility results in intricate and unforeseen impact of product and project decisions on systems level, even in late development phases. Especially software development in mobility, one of the main factors for innovation and value creation – and costs, must be evaluated also in system context. To cope with this fact, the early assessment of design decisions is a key factor for success. Non-recurring cost for aerospace products sees software development as main driver – especially for Unmanned Aerial Vehicles (UAV). The future mobility will be electrified, automated and connected. As a result, especially automotive and aerospace systems will undergo a radical shift in the way they are organized on software and hardware level, as well as how they are designed, leading to a threefold heterogeneity:

1. Integration of heterogeneous function domains on centralized computing platforms
2. Use of heterogeneous specialized hardware
3. Involvement of heterogeneous, collaborating parties for design and development

An early assessment of design decisions increases the development efficiency of new products. For this reason, the goal of PANORAMA is to research into model-based methods and tools to master design and development of heterogeneous systems and therewith-increasing development efficiency. Focus is automotive industry, synergies and learnings for other domains such as avionics are researched as well. Open standards for models and open source tools are the main approach to achieve a wide adoption of the PANORAMA results.

Putting the assessment of automotive and aerospace designs and projects on reliable and common foundations in all phases of development can be a major lever for better software, better products and time-to-market while reducing costs and risks due to qualified decision-making. This will enable the European automotive and aerospace industry to cope with the worldwide market. PANORAMA's results will generate new business opportunities for suppliers and customers in two of the biggest industries – automotive and aerospace – which are currently undergoing a substantial redefinition of their value creation process. In general, PANORAMA enables the established players to compete in a fast growing market while establishing new opportunities for new players (e.g. consulting or tools providers). The dissemination as open source lowers the entry barriers especially for SMEs.

Major technical outcomes of PANORAMA will be:

- Models (and meta-models) which support the (early) assessment of design decisions
- Tools, Methods and Processes that enable or improve assessment
- Means for supporting the user with domain and use-case specific views and visualization

- Demonstrators that show the effectivity and efficiency gains
- A dissemination as open source software supported by an active community

These technical goals imply three important conditions: (1) PANORAMA's approach must well integrate with existing ecosystems, (2) it is crucial that all information is stored only once at a single location, and (3) guidance is needed to efficiently address the specific design task (use-case) at hand with the right abstraction level of the performance model and visualization.

2 Project overview

2.1 Rationale of the project

2.1.1 Problem statement and market value chain

The future mobility will be electrified, automated and connected. As a result, automotive systems will undergo a radical shift in the way they are composed of software and hardware, as well as how they are designed and developed, resulting in a threefold heterogeneity:

1. Formerly separated function domains will be integrated onto centralized computing platforms leading to a heterogeneous mix of applications with different models of computation (e.g., control, stream processing, and cognition).

2. Hardware will be diverse and specialized to satisfy the tremendous increase of needed computing power (e.g., application cores, deep learning accelerators, issued across several control units and connected by networks).

3. Collaborating parties for design and development will be more heterogeneous as compared to today (e.g., experts of different domains from OEM and Tier1 jointly developing complex applications; Tier1 and semiconductor vendor jointly developing hardware).

In this context, joint and efficient development is required for products that are safe, secure, and fascinating for the end user (the car buyer), and eventually profitable for all parties involved in creating the product (for instance, Tier2, Tier1, and OEM creating a new video-based driver assistance system). To successfully compete with established and forming consortiums in Asia and North America, higher development efficiency for European partnerships is the key. Consequently, we demand new methodology, approaches, and tools that conveniently enable discussion, reasoning, and early assessment of design decisions to increase development efficiency of embedded automotive systems. Thereby, sustainable methodology and approaches must consider standards and best practices. For instance, modelling approaches may be based on open source project APP4MC, and methodology will respect relevant regulatory and industrial frameworks such as ISO26262, and the AUTOSAR & AUTOSAR adaptive, MSR, ASAM-MDX & CDF standards. The aerospace industry is facing similar challenges, especially with respect to the three areas of heterogeneity. "Automated" and "connected" are relevant trends especially for Unmanned Aerial Vehicles (UAV). Due to the ever rising share of software development within the (also rising) share of the non-recurrent cost in aerospace products similar issues and solutions compared to the automotive industry come up. Therefore, it is straightforward to address both automotive and aerospace industry within PANORAMA. Additional relevant standards (e.g. DO-178B) can be incorporated based on the competence of additional projects partners. Ideally, a sole, convenient modelling approach serves the majority of design tasks during design and development. Tools needed for efficient development, e.g., timing analysis tools, then obey this open modelling approach allowing the development parties to pick their preferred and

suitable tool chain, and to change tools during development depending on team preferences and design needs. This idealized approach—sole modelling standard, seamless methodology formed around it, rich tool ecosystem obeying the standard—paves the way for efficient development of heterogeneous systems by heterogeneous parties. In real world, a multitude of modelling approaches exists, e.g., based on SysML/UML, or defined by design guidelines based on tools such as Enterprise Architect from SparxSystems or Rational Rhapsody from IBM. Most often, different approaches are taken by different parties involved in developing the same product (even in the same company). Obviously, joint development requires exchange and communication between parties, now reduced to the least common denominator, often resulting in reasoning, discussing, and assessing on basis of spreadsheets and informal text documents or slide decks.

On the methodology side, effectiveness of timing and performance analysis for increasing development efficiency in the context of single functional domains and multi-core hardware was shown in previous projects (e.g., TIMMO, Timmo2USE, AMALTHEA, AMALTHEA4public, ARAMiS I & II). Especially non-functional performance analysis based on abstract system models proved to be suitable to assess and back-up decisions during the design of hardware/software multi-core systems. However, solutions for heterogeneous function domains on heterogeneous hardware platforms developed by diverse parties are missing. In other words: performance analysis tools lag behind current hardware trends, and solutions are missing to face the heterogeneous systems era that demands to master, e.g., accelerators and special IP blocks in addition to homogeneous multi-core chips. Introduction of complex scheduling approaches, such as POSIX-based operating systems, and hypervisors in automotive systems imply additional technological challenges.

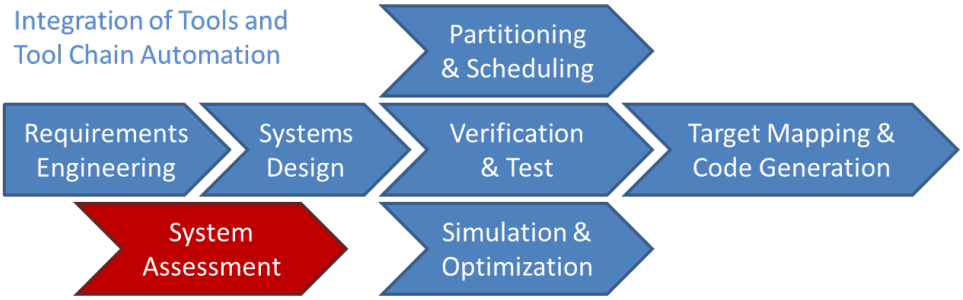


Figure 2.1: System Assessment in the early phase of SW design

The focus of PANORAMA is the qualitative assessment of the design of large automotive or aerospace systems in the early phase of the design. By providing the means in terms of models, tools, methods and processes the project aims to provide the effectivity and efficiency gains delivered by front loading the design process in terms to avoid wrong decisions, misunderstanding amongst partners, mistakes, wrong assumptions and predictions. Such problems are known to have a great influence of project and product success, on time-to-market, on product quality and on cost. This is achieved by providing tools and tools chains as well as processes and methods.

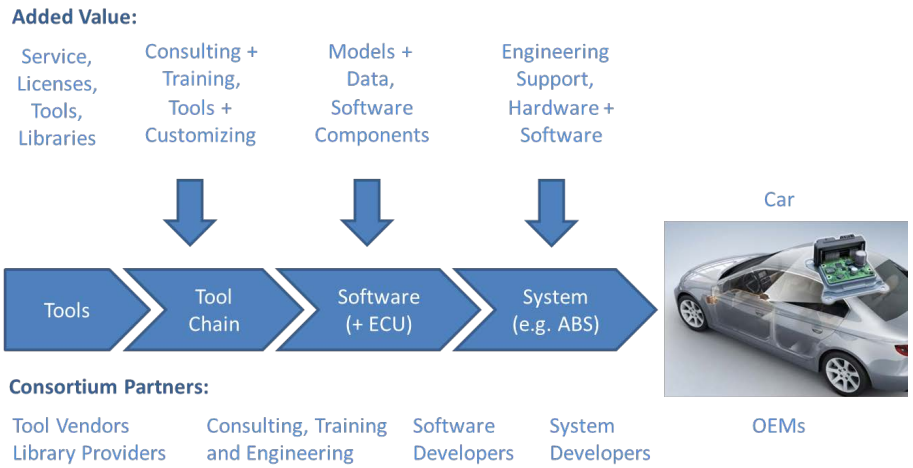


Figure 2.2: Market Value Chain of Automotive (and Aerospace) SW design

The market value chain for software development solutions in general and for the tools and processes for system assessment in particular consists of four main groups of players:

1. Tool Vendors and Library Providers sell services, licenses, and libraries to all three groups of the following steps of the market value chain. In the case of the early phase of the design these are tools e.g. for performance, safety and security aspects of automotive and aerospace systems, models and libraries with respective models. They earn money by charging license and service fees. Their market access is becoming much easier due to standardized models and interfaces to the open source tool platform.

2. Consulting, Training and Engineering companies integrate the tools and libraries into the software development tools chains. They have similar business models as the tool vendors. In addition they may charge for engineering services based on time and material. Their market access is improved by the open standards and tools, too.

3. Based on the tool and process environment provided by the first two groups the developers of automotive software can do their job. This will be supplied by 3rd parties, e.g. with navigations maps, OS suppliers, cloud services etc.. Finally, they deliver the software product to the systems developers. They benefit from better tools by achieving better quality, lower cost and faster time-to-market. Furthermore, they benefit from better interfaces and exchange of the development artefacts.

4. System developers are direct suppliers to the car makers or airplane producers. They deliver electronic control units (ECUs) together with the respective software. System developers have the challenge to give concrete offers (quality, prices and timelines) based on uncertain assumptions to their customers (the car makers and aerospace companies). Furthermore, they have to set up and manage the network of their software suppliers. The definition and setup of the joint tooling for the system development is in their responsibility and benefits from common standards and interfaces.

The PANORAMA project consortium needs partners covering all four steps in both the aerospace and automotive industry. Furthermore – to establish common European standards –

it needs access to the most relevant European markets. Finally, the consortium needs a strong open source partner for dissemination and access to important automotive OEMs and aerospace companies for evaluation and review of the results (e.g. as a steering board).

2.1.2 Project innovations and technology value chain

The goal of PANORAMA is to research into model-based methods and tools to master design and development of heterogeneous systems by heterogeneous parties, and provide best practice and guidance for development (cf. Figure 3). To that end, the main line of action is extending the scope of current system level approaches by enhancing existing abstract performance meta-models to be suitable for heterogeneous hardware, and heterogeneous function domains. We will stand on the shoulder of giants by building on the meta-model developed in the AMALTHEA and AMALTHEA4public projects, taking results from projects such as TIMMO, Timmo2USE, ARAMiS I & II into account. Thereby, the enhanced meta-model must be a common and open standard to support development by diverse parties across organizations.

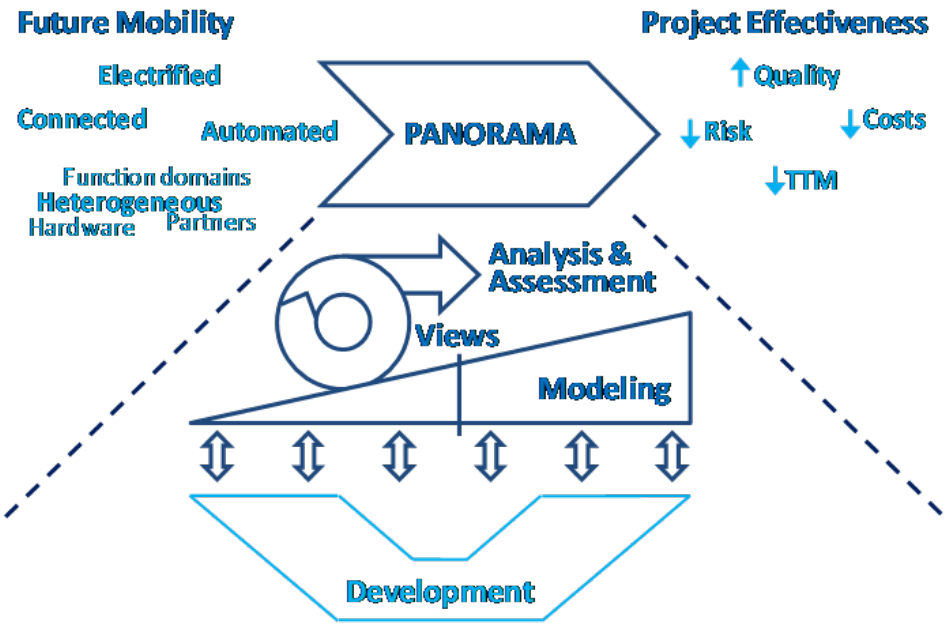


Figure 2.3: PANORAMA project overview

For integration and transition, existing modelling approaches (see Chapter 2.1.1) have to be respected and transformations into the developed meta-model may be offered. To ensure broad acceptance and justify investment by industry, the meta-model has to suit many use-cases, ideally across the complete development cycle. A meta-model that is rather wide and generic is the result. In addition, performance models of products usually grow and get enriched during the development process. These facts impose three important conditions:

1. PANORAMA's approach must well integrate with existing ecosystems,
2. it is crucial that all information is stored only once at a single location, and

3. guidance is needed to efficiently address specific design tasks with the right abstraction level of the performance model.

Therefore,

- co-existence of models proposed here with established forms of information storage and system specification such as AUTOSAR, AUTOSAR adaptive, SysML, and EAST-ADL has to be considered, industrial practice has to be cross-checked,
- co-existence of tools deployed in design, open source and commercial, has to be sought, and
- use-case specific "views" on the performance models are required that provide focus for the developer on specific design tasks at hand, and are suitable for exchange between parties.

Use-cases addressed by PANORAMA's methodology are, for example, assessment of different hardware architectures for given software, assessment of deployment alternatives in a system, or guidance for optimization of system level design decisions by visualization of analysis results. To address these use-cases, besides the underlying modelling approach, static and dynamic analysis approaches will be provided. For instance, dynamic analysis based on performance simulation is one path we will take, paying attention on combining strengths of flexible and open solutions (such as SystemC) with established and mature commercial simulators. In context of static analysis, we intend to evolve current timing and schedulability analyses for multi- and many-core platforms to consider the extra complexity of heterogeneous platforms and hierarchical scheduling approaches, provide methods and tools for checking fault isolation and path coverage. Moreover, we will employ performance analysis methods based on the theory of models of computation to develop scalable design space exploration methods for distributed multi-core real-time systems, enabling the identification of efficient implementations at an early design stage.

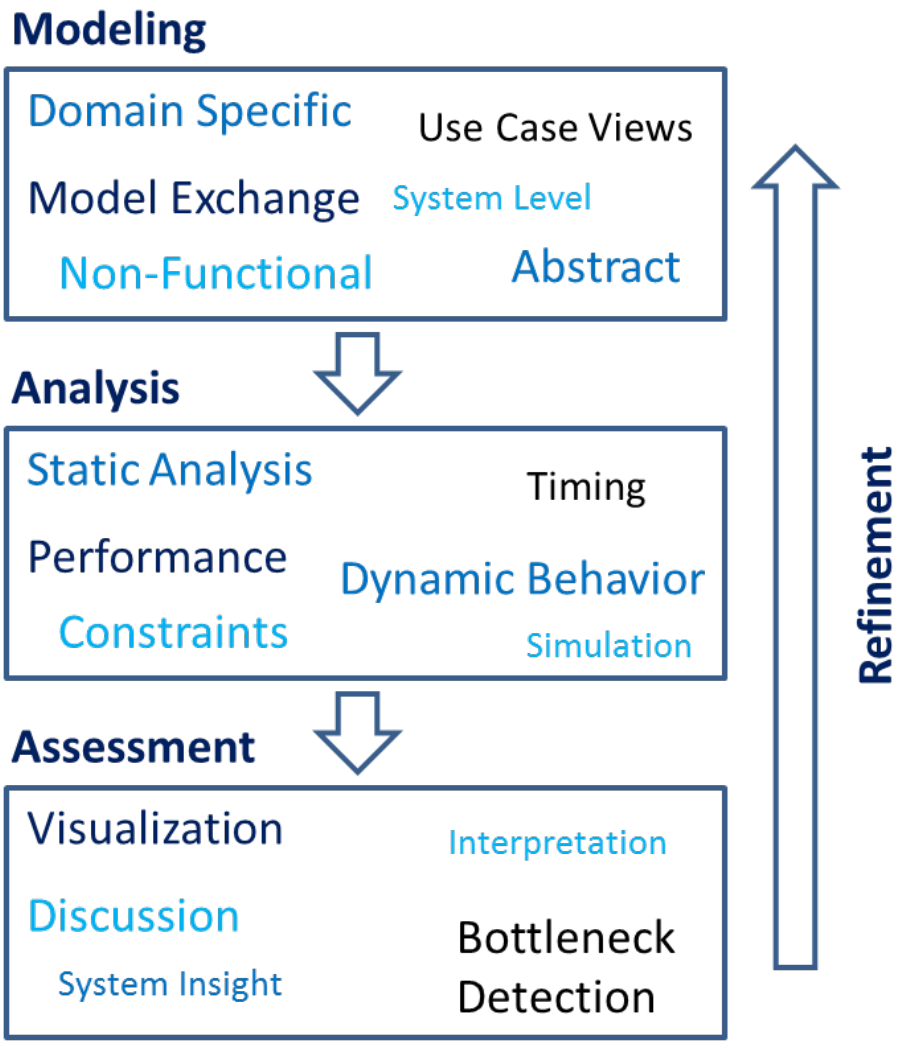


Figure 2.4: Technology Value Chain for Early Design Phase

Out of scope of PANORAMA are, for example, development of functions, development of hardware architectures, and automatic optimization framework. Functions will be provided by partners in the project, but developing or optimization of functions is not sought to be funded. While any sort of assessment and then tuning of design decision can be considered as an optimization of the target product (what is in scope of PANORAMA), PANORAMA has no intention to build feature rich frameworks to automatically optimize the system at whole. As lesson learned from related projects and industrial practice, two further aspects will be considered to pave the way of methods and tools provided by PANORAMA:

- Closed source only and non-adaptable solutions are prohibitive in the heterogeneous era; flexibility for tools and methods is required to cope with fast-paced hardware trends. It will be ensured that PANORAMA's solutions are freely available and highly adaptable. Provided solutions shall be completed with commercial tools that offer maturity and experience required for industrial deployment.

- Documentation of meta-models often lacks formality; to eliminate confusion from the beginning, our meta-model solutions are escorted with reference implementations that clearly define their semantics and provide guidance for usage.

The major technical outcomes of PANORAMA will contribute to the provision of the complete technology value chain for system assessment in the early design phase by:

- Providing new models (and meta-models) which support the early assessment of software designs by containing non-functional (e.g. qualitative) information. Such abstract models need to support exchange with tools and R&D partners, system level description and a sufficiently accurate assessment of system features (e.g. performance, required HW configuration). A consistent and comprehensive modelling environment for this purpose is new. It would put the partners on all stages of the market value chain in position to develop better systems in a faster and more cost efficient way. The modelling environment (common model definitions, tools, standards) enables the partners to set up their contributions and to connect and cooperate.

- Tools, Methods and Processes which support the analysis of the models and the modelled systems form the next step of the technology value chain. They enable the tool vendors and library providers to develop their products and provide their services. Tooling for model based systems engineering and especially for the analysis in the early design phase are a growing market and opens the way for additional tools and services.

- Tools, Methods and Processes which support the assessment of the analysis results are the final step of the overall process. Such tools need to support the user (software engineer, but also other domains like project management and product definition) with domain specific views and visualization of the analysis results. This enables discussion within the network of partners of a development project. Providers of the tools benefit from the developed technology and from the standardized models and interfaces. Furthermore, automotive and aerospace software and system developers benefit from using the tools.

- Demonstrators which show the effectivity and efficiency gains are crucial for generating market acceptance, wide spread usage and adoption of the tools, methods and processes. This is relevant for the positioning of standards and de-facto standards.

A dissemination as open source software supported by an active community will lead to continuous maintenance and further development of the tool chain and the modelling environment. This community is the basis of spreading the use and adoption and forms the fundament of the business models of the partners of the market value chain.

3 State of the art standards and data models

3.1 AMALTHEA

The foundations for a more efficient use of modern high-performance hardware were created by the ITEA projects Amalthea and Amalthea4public. The main result of the projects is a software platform for multi-core systems with the central datamodel called Amalthea. Amalthea4public extended the platform to support many-core systems and established the Eclipse project APP4MC[1]. The community continued to work on the project after the end of the research projects.

In order to investigate the (timing) behavior of systems in a multi- or many-core context, a suitably abstracted description of the factors must be available. The Amalthea system model combines various partial models, which together contain the information required for a performance simulation, leading to an analysis and optimization of the timing behaviour of a system. The descriptions include the hardware characteristics, the structure of the software, execution times and prescribed requirements. The description can vary in degree of abstraction and detail, depending on the stage of the development. For example, execution times can be estimated at the beginning of the process and later on be replaced by measurements from the real controller.

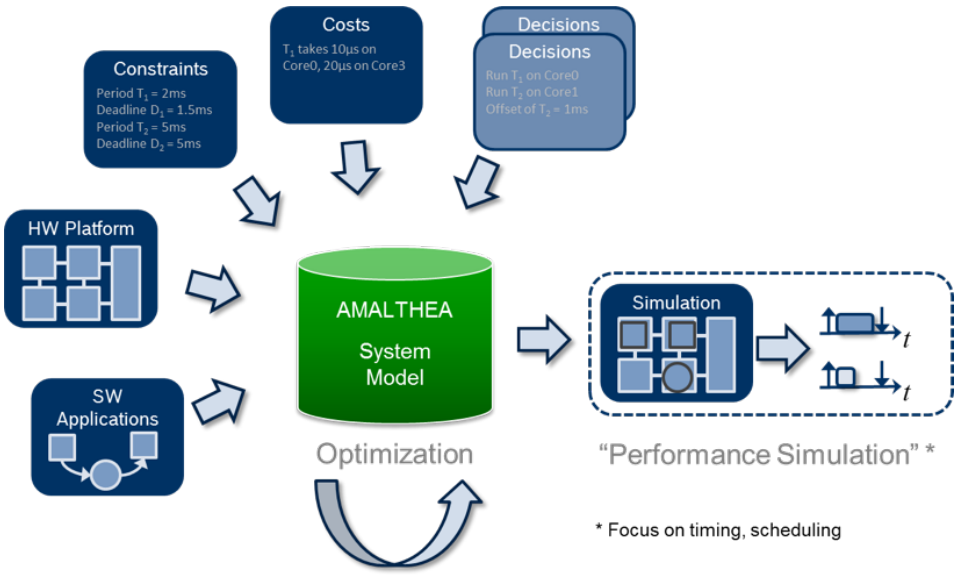


Figure 3.1: Main data in the Amalthea System model

[1] https://www.eclipse.org/app4mc/

An Amalthea model per se is a static, structural model of a hardware/software system. The basic structural model consists of software model elements (tasks, runnables), hardware model elements (processing units, memories, connection handlers), stimuli that are used to activate execution, and mappings of software model elements to hardware model elements. Semantics of the model then allows for definite and clear interpretation of the static, structural model regarding its behavior over time.

Different levels of model detail Amalthea provides a meta-model suitable for many different purposes in the design of hardware/software systems. Consequently, there is not the level of Amalthea model detail – modeling often is purpose driven. Regarding timing analysis, we exemplary discuss three different levels of detail to clarify this aspect of Amalthea. Note that we focus on level of detail of the hardware model and assume other parts of the model (software, mapping, etc.) fixed.

Essential software model elements are runnables, tasks, and data elements (labels and channels). Runnable items of a runnable specifies its runtime behavior. You may specify the runnable items as a directed, acyclic graph (DAG). Amalthea has different categories of runnable items, we focus on the following four:

- Items that *branch paths* of the DAG (runnable mode switch, runnable probability switch).
- Items that *signal* or call other parts of the model (custom event trigger, runnable call, etc.).
- Items that specify *data access* of data elements (channel receive/send, label access).
- Items that specify *execution time* (ticks, execution needs).

Tasks may call runnables in a activity graph. Note that a runnable can be called by several tasks. We then map tasks to task schedulers in the operating system model, and we map every task scheduler to one or more processing units of the hardware model. Further, we map data elements to memories of the hardware model.

This coarse level of hardware model detail - the hardware model consists only of *mapping targets*, without routing or timing for data accesses - may already be sufficient for analysis focusing on event-chains or scheduling.

As the second example of model detail, we now add access elements to all processing units of the hardware model, modeling *data access latencies or data rates* when accessing data in memories - still ignoring routing and contention. This level of detail is sufficient, for example, for optimizing data placement in microcontrollers using static timing analysis.

A more detailed hardware model, the third and last example of model detail, will contain information about *data routing and congestion handling*. Therefore we add connection handlers to the hardware model for every possible contention point, we add ports to the hardware elements, and we add connections between ports. For every access element of the processing units, we add an access path, modeling the access route over different connections and connectionHandlers. the combination of all access elements are able to represent the full address space of a processing unit. This level of detail is well suited for dynamic timing simulation considering arbitration effects for data accesses.

Structural modeling of heterogeneous platforms To master the rising demands of performance and power efficiency, hardware becomes more and more diverse with a wide spectrum of different cores and hardware accelerators. On the computation front, there is an emergence of specialized processing units that are designed to boost a specific kind of algorithm, like a cryptographic algorithm, or a specific math operation like "multiply and accumulate". As one result, the benefit of a given function from hardware units specialized in different kinds may lead to nonlinear effects between processing units in terms of execution performance of the algorithm: while one function may be processed twice as fast when changing the processing unit, another function may have no benefit at all from the same change. Furthermore the memory hierarchy in modern embedded microprocessor architectures becomes more complex due to multiple levels of caches, cache coherency support, and the extended use of DRAM. In addition to crossbars, modern SoCs connect different clusters of potentially different hardware components via a Network on Chip. Additionally, power and frequency scaling is supported by state of the art SoCs. All these characteristics of modern and performant SoCs (specialized processing units, complex memory hierarchy, network like interconnects and power and frequency scaling) are supported with the current Amalthea hardware model. The concept tries to represent hardware components in a flexible and easy way. It supports modeling of manifold hierarchical structures and also domains for power and frequencies. Furthermore, explicit cache modules are available and the possibilities for modeling the whole memory subsystem are extended, the connection between hardware components can be modeled over different abstraction layers. Only with such an extended modeling approach, a more accurate estimation of the system performance of state of the art SoCs becomes feasible.

The design of the hardware model is focusing on flexibility and variety to cover different kind of designs to cope with future extensions, and also to support different levels of abstraction. To reduce the complexity of the meta model for representing modern hardware architectures, as less elements as possible are introduced. For example, dependent of the abstraction level, a component called ConnectionHandler can express different kind of connection elements, *e.g.*, a crossbar within a SoC or a CAN bus within an E/E-architecture.

ProcessingUnits are the master modules in the hardware model. The following example shows two ProcessingUnits that are connected via a ConnectionHandler to a Memory. There are two different possibilities to specify the access paths for ProcessingUnits like it is shown for ProcessingUnit 2 in the next figure. Every time a HwAccessElement is necessary to assign the destination *e.g.*, a Memory component. This HwAccessElement can contain a latency or a data rate dependent on the use case. The second possibility is to create a HwAccessPath within the HwAccessElement which describes the detailed path to the destination by referencing all the HwConnections and ConnectionHandlers. It is even possible to reference a cache component within the HwAccessPath to express if the access is cached or non-cached. Furthermore it is possible to set addresses for these HwAccessPath to represent the whole address space of a ProcessingUnit. A typical approach would be starting with just latency or data rates for the communication between components and enhance the model over time to by switching to the HwAccessPaths.

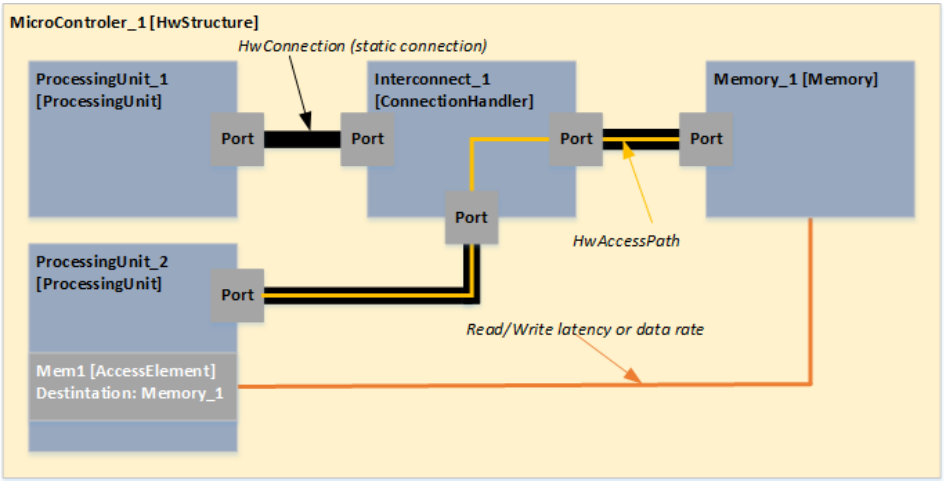


Figure 3.2: Access elements in the Amalthea hardware model

3.2 AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) [AUT18] is a standard (process/data exchange) developed and mainly used by the automotive industry. It was initiated in autumn 2003 by the need to manage the growing software complexity in electric/electronic systems. Enabling software component reuse, variability, accelerated development, cost optimization, system scalability, etc., it defines standardized highly configurable interfaces and design processes. A simplified version of that design process is depicted in Figure 3.3.

Generally, AUTOSAR can be used after the initial system design (when it is clear which parts will be software and which will be hardware), *i.e.*, AUTOSAR is targeted for software design and configuration. As a starting point for the AUTOSAR process there are some system constraints to initiate the system configuration and software component (SWC) description. Since the system configuration step relies on a SWC description the latter step is done first. Then, during the system configuration, the SWCs are deployed to computing resources (Electronic Control Units - ECUs) and the communication between them is defined. After that, the SWCs can be implemented in parallel to the ECU configurations. The system configuration is usually done by Original Equipment Manufacturers (OEMs), while a single ECU is provided by Suppliers. The interface (subject matter of the contract) between OEM and suppliers is ECU Extract for the supplier specific ECU. It contains only the communication relevant for that ECU and its SWCs. Suppliers can now extend this ECU Extract by configuring it. During the ECU configuration additional SWCs may be introduced by the supplier.

SWCs communicate via a standardized interface, the Virtual Function Bus (VFB). Refined interfaces of the VFB and their implementations are called Runtime Environment (RTE), *i.e.*, the RTE is the realization of the VFB. On the lower abstraction level of the hardware, there is the Basic Software (BSW) which takes care of hardware specifics, *e.g.*, Input/Output, low level drivers, etc. In addition to the BSW there is the operating system (OS) which governs the resource sharing (computation time and physical resources like memory) among the SWCs. Once the ECU is configured the BSW modules, the RTE implementation, and the OS can be generated. Combined with the SWC implementation, these artifacts can be compiled to a binary which can be flashed to the target device for execution/testing.

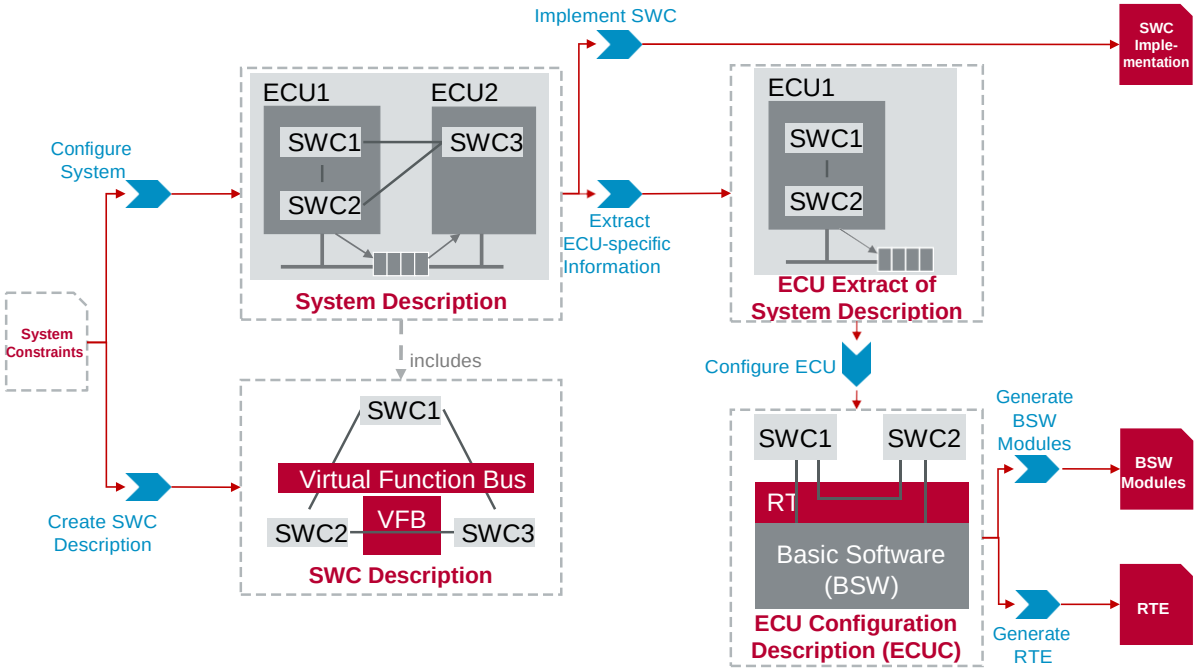


Figure 3.3: Simplified AUTOSAR development process.

There are several meta-model parts and their extensions (*e.g.*, TimingExtensions) in AUTOSAR. As indicated in Figure 3.3, there are two layers/views of interest for AUTOSAR models: System and ECU layer. On system layer, the general communication between ECUs is important to deploy functionality (SWCs) to appropriate computation nodes. Hardware resources, SWCs, and communication signals are modeled here. On ECU layer, many details can be described, it is also possible to model new parameters and assign values to them. More details about hardware resources, operating system specifics, communication protocols, partitions, etc. can be modeled here. Most of these details are modeled via parameter and value definitions.

On top of the general two layers there are crosscutting artifacts like constraints, data types, mappings, and even SWCs (they are used on both layers). SWCs can be decomposed into subcomponents, constituting an arbitrary deep compositional hierarchy. Leaf SWCs (not further decomposed) have Runnable Entities (REs) which can be combined to OS tasks (different composition) to be scheduled by an OS for efficient resource sharing. With AUTOSAR all these artifacts can be modeled and persisted in well defined AUTOSAR XML files.

3.3 SysML

SysML [OMG19] is a general-purpose systems modeling language standardized by the Object Management Group (OMG) for application of a model-based systems engineering (MBSE) methodology to development of complex multidisciplinary systems. As opposed to the conventional document-based design specification approach, this methodology aims at exploiting domain models (or model-based specifications) as the primary medium for information exchange between engineers while directly involving project stakeholders in early development stages. It

was initiated by INCOSE MBSE Initiative in January 2007.

The design complexity of embedded hardware/software systems is continually increasing, while on the other hand, due to ever growing variability of products and their optimizations, functions from different domains are getting increasingly mixed up together in different new, often beforehand unforeseen contexts of their operation which leads to potential interoperability problems for entire system. This kind of potential interoperability malfunction can rather hardly be mitigated or even detected by tests. To approach that, MBSE methodology and SysML provide means to describe a system in a top-down manner starting from an abstract system level by capturing consistent system requirements, system context, use cases, boundary conditions, functions, and their interfaces for subsystems from different interacting engineering disciplines (such as software, hardware, engine, control, power electronics, mechanics, *etc*) down to the detailed design at component level. Such a system architecture model comprises a structured set (or a graph) of logically interconnected model elements that together with an appropriate tooling provide an engineer with the ability to automatically query a context of any function with corresponding requirements and intended scenarios of its usage during any stage of a development process. In contrast to conventional document-based description approaches, a SysML model as a formalized specification has a further advantage of representing a traceable single source of information regarding requirements, structure, behavior, and parameters over the whole lifecycle. This allows for automated validation and verification of cross-disciplinary consistency between important specifications such as system requirements, boundary conditions, and interfaces for system components from different domains.

Generally, SysML, being a derivative of OMG Unified Modeling Language (UML), provides means (*i.e.*, a structured set of model elements and respective diagrams) for describing a modeled system on various levels of granularity and from different viewpoints such as requirements, structure, behavior, and parameters. However, the language itself does not provide any guidance method to adopt the MBSE methodology in a specific development process. SysML is not an approach or a software tool that guides engineering during the specification of a system. There exist several methodologies to guide engineers in adoption and use of MBSE technology, like INCOSE Object-Oriented Systems Engineering Method (OOSEM[FMS14]), SYSMOD [Wei16], Architecture Analysis and Design Integrated Approach (ARCADIA) [Voi17], IBM Rational Unified Process for System Engineering (RUP-SE) [Can03], SPES/SPES XT [BDH+12; BDK+16], and others, that can be implemented using SysML. Since OMG SysML is based on UML, the language extendability features have been preserved by means of additional domain-specific profile definitions at the meta-level of the language. This makes the language easily extendable and thus it can be integrated into a wide range of development processes of different organizations.

Although last decade has shown a steadily growing interest of different industries toward adoption of MBSE methodology [INC14], there are however some points of criticism concerning the current implementation of SysML. In particular, an extensive use of language extendability in terms of UML profile extensions, which is rather often the case for practitioners from different domains, unfortunately leads to a restricted interoperability of SysML models among different collaborating partners and different modeling tools. Furthermore, since a significant part of an exchanged model represents a visual information, the standard format for storing SysML metadata information, *i.e.*, OMG XML Metadata Interchange (XMI) [OMG], becomes rather tool and vendor specific. Due to a general nature of the language and to some extent due to syntactic and semantic overlap with UML, SysML models lack a precise unambiguous semantics to encompass methods of formal verification and model checking or at least to formally judge

about model equivalence.

OMG is currently working on the new version v2 of SysML addressing major problems of the current implementation by improving following key elements [Fri]:

- New Metamodel that is not constrained by UML (grounded in formal semantics),
- Robust visualizations based on flexible view & viewpoint specification and execution (graphical, tabular, textual),
- Standardized API to access the model.

Initial submission of the proposed v2 specification to OMG is planned by SysML v2 Submission Team in June 2020.

3.4 EAST-ADL

EAST-ADL [CFJ+10] is an Architecture Description Language (ADL) defined within the ITEA EAST-EEA. EAST-ADL aims at describing automotive electronic systems through an information model that captures engineering information in a standardised form. The EAST-ADL model is structured in abstraction levels (*i.e.*, Vehicle, Analysis, Design and Implementation level) which map to the abstraction levels given in ISO26262**[FiXme Fatal: Please add a reference!]**. What is more, the EAST-ADL model covers a variety of orthogonal concerns spanning from requirements to structure, through timing, dependability and several more. To this end, EAST-ADL provides the following packages. FiXme Fatal!

The *Structure* package defines the static structure of the instances of the system being modelled and their static relationships. This includes the instances internal structure as well as their external interfaces. Within this package, each abstraction level has a corresponding artefact as follows.

- Vehicle Level include the Technical Feature Model.
- Analysis Level includes the Functional Analysis Architecture (FAA).
- Design Level includes the Functional Design Architecture (FDA), the Hardware Design Architecture and the Allocation.
- Implementation level includes a reference to an AUTOSAR model (see section 3.2).

The Structure package is UML2 [OMG17] compliant.

The *Environment* package describes the environment of the vehicle electric and electronic architecture by means of continuous functions. The *Behavior* package describes either a function performing some computation on provided data or the execution of a service called upon by another function. Within EAST-ADL, the execution of the behaviour assumes a strict run-to-completion, single buffer-overwrite management of data. What is more, the execution is non-concurrent within an elementary function. The *Variability* packages serves for expressing variability in the FAA, FDA and implementation level architecture.

The *Requirement* package describes conditions or capabilities that must be met or possessed by a system or components to satisfy a set of (formally imposed) properties. EAST-ADL offers constructs for dealing with the highly changing nature of requirements, *e.g.*, courses, types, levels of abstraction, stakeholders, etc. The Requirement package, as the Structure one, is UML2

compliant. The *Timing* package serves for the specification of timing constraints according to TADL2**[FiXme Fatal: Please add a reference!]**. The *Dependability* package supports: FiXme Fatal!

- the definition and classifications of safety requirements through preliminary Hazard Analysis Risk Assessment,
- tracing and categorising safety requirements according to their role in the safety life-cycle,
- formalising safety requirements using safety constraints,
- formalising and assessing fault propagation through error models, and
- organising evidence of safety in a Safety Case

The Dependability package is designed to support the automotive standard for Functional Safety, ISO/DIS 26262.

Eventually the *GenericConstraints* and *Infrastructure* packages serve for the specification of properties, requirements or validation results for identified elements and for the specification of the infrastructure constructs, respectively.

3.5 Rubus Component Model (RCM)

The Rubus Component Model (RCM) is a component model for the development of distributed real-time systems that has been conceived by Arcticus Systems AB[2] together with Mälardalen University. The model targets three main activities in the design of vehicular systems, 1) design, 2) analysis, 3) synthesis [HMN+08]. RCM can be used together with architectural languages such as EAST-ADL (see section 3.4), where it focuses mainly on the design and implementation level.

In RCM, the basic unit of hierarchical decomposition is the Software Circuit (SWC). A SWC encapsulated software functions and is defined by its interfaces, internal states, and behaviour. Multiple behaviours can be implemented, where each has its own entry function. A SWC has distinct trigger and data ports that separate the flow of data from the flow of execution. When triggered for execution, a SWC follows the read-execute-write semantic, *i.e.*, first all input variables are copied, then execution is performed on the copied variables, and finally all output variables are written. Budgets or estimates of for timing attributes, and/or memory consumption (on different platforms) can be specified on the model and serve as guidelines or constraints for the function development.

Timing constraints can be specified and analysed via formal timing analysis methods [MNS+17; BCC+16]. Timing analysis requires the annotation of WCET and maximum stack usage for each SWC. Different values can be annotated for different target platforms. Most of the available timing constraints in RCM can directly be expressed, or modelled by the Timing Augmented Description Language v.2 (TADL2) [MNS+17].

The platform model in RCM allows to define Nodes (*i.e.*, ECUs), Cores, and Partitioned. The mapping of software architecture elements to the hardware platform can be specified among any two elements. Communication between different ECUs can be modelled via different networks or buses (for example CAN). A network connection transmits signals, where one or more signal can be part of a message.

[2] https://www.arcticus-systems.com

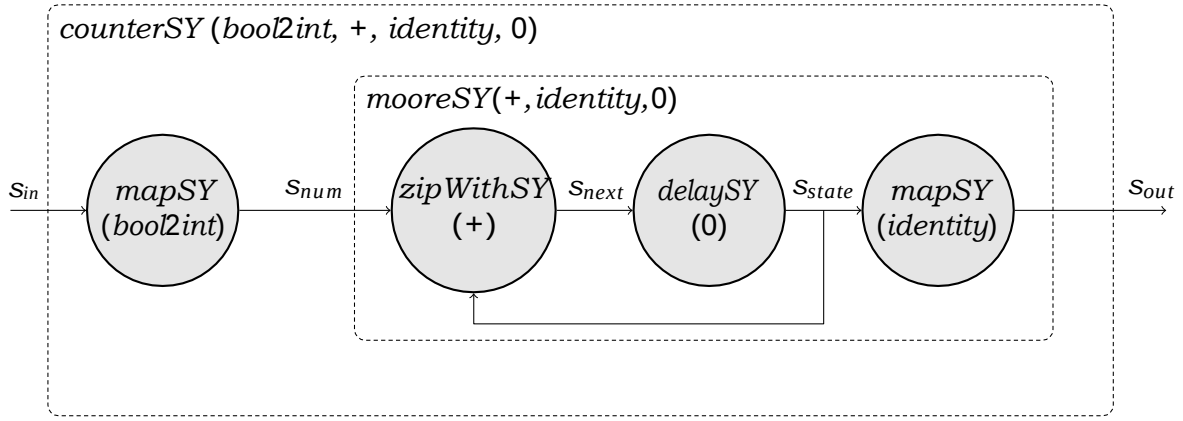


Figure 3.4: Example counter system modeled in the synchronous MoC.

3.6 ForSyDe

The ForSyDe methodology, as a mathematical system(s) design framework, has no particular data model aside from mathematical descriptions of the system(s) being modeled. As long as the tooling built around the ForSyDe methodology captures the essential ideas of signals, processes, processes constructors and Models of Computation (MoCs), it can be considered a ForSyDe data model [SJ04]. Thus, the ForSyDe methodology can loosely be considered a standard for data model definitions and tooling development. To illustrate this claim, the tooling built around the ForSyDe methodology, hereinafter referred as ForSyDe tooling, is implemented both as a set of Haskell libraries (e.g. ForSyDe-Shallow) and a a SystemC library (ForSyDe-SystemC)[3].

Moreover, the ForSyDe methodology can also be interpreted as a component based framework, where the processes are the primary blocks and their connections happen due to the interplay of connected signals. These processes, as components, can only be instantiated if they have an accompanying process constructor to clearly define their input-output relations of the mentioned connected signals. From this viewpoint, hierarchy is created by grouping processes into a single bigger process so that free signals are now the inputs and outputs of the new process.

Figure 3.4 showcases these concepts via an example where an arguably common counter is described upon the synchronous MoC without any absent values. Everytime the signal s_{in} is a true value, the counter increments and outputs the current count of true values in the signal s_{out}; considering that the count is zero initially. The functions *bool2int* outputs 1 for true values and 0 for false values ; whereas *identity* simply returns any value evaluated. *mapSY*, *zipWithSY* and *delaySY* are all process constructors for the synchronous MoC that cannot be reduced any further and that uniquely define the bigger *mooreSY* and *counterSY* process constructors, the latter which also corresponds to the desired counter system specification. Note how it is possible for *mooreSY* and *counterSY* to pass information down to their defining children, as seen usually in component-based frameworks. To finish this example, suppose that a signal $s_{in} = \{T, T, \perp, T\}$ is applied to this system design/specification. Then by the

[3] All ForSyDe tooling projects can be found in https://forsyde.github.io/tools.html

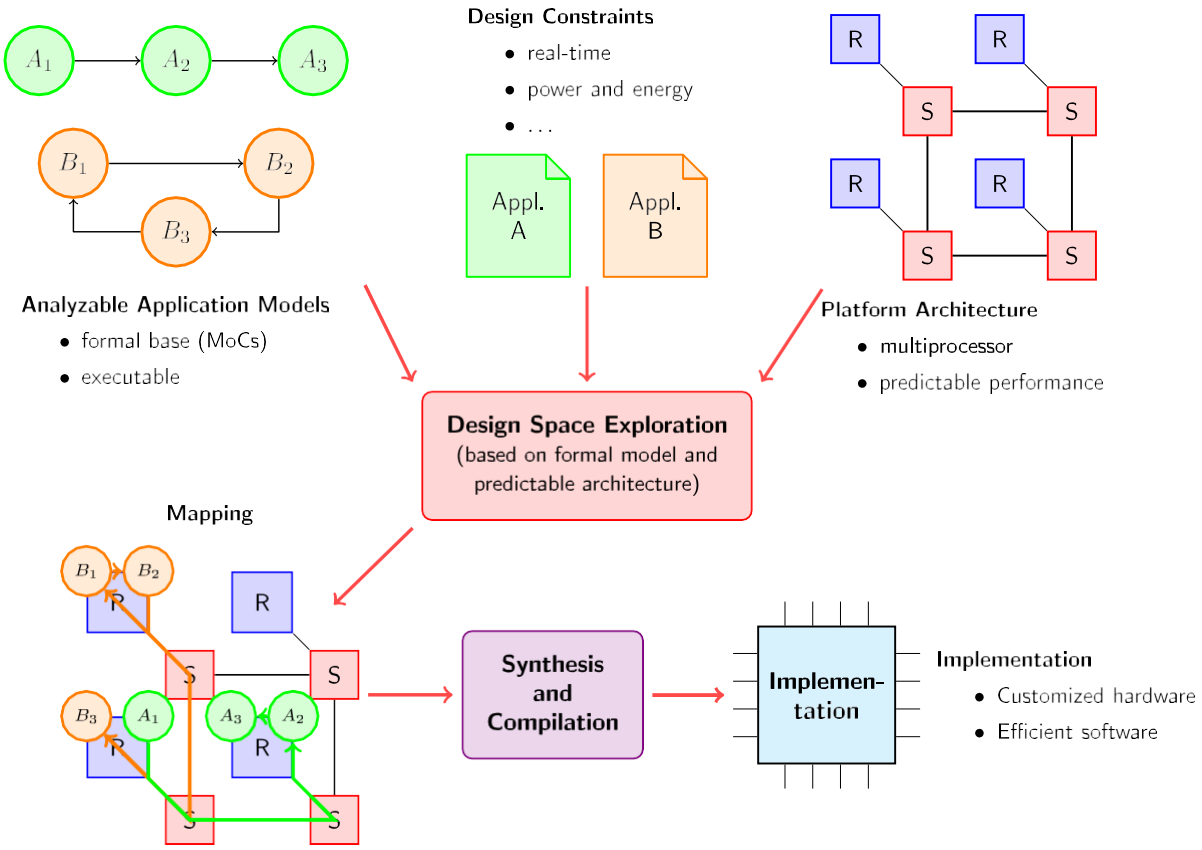


Figure 3.5: ForSyDe's design flow overview.

synchronous MoC semantics, there is only one possible outcome, which is:

$$s_{num} = \{1, 1, 0, 1\},\ s_{next} = \{1, 2, 2, 3\},$$
$$s_{state} = \{0, 1, 2, 2\},\ s_{out} = \{0, 1, 2, 2\}$$

This system model specification does not enforce any affinity for processes to be implemented completely in hardware, completely in software or somewhere in between. Rather, it is taken as the specification from which the behaviour, as in outputs, of the implementation is compared against. Typically a system to be designed will have other requirements that are not captured commonly by MoCs, such as computation taking physical time, maximum power consumption for some parts of the system etc. The ForSyDe methodology tackles these requirements by using other data models as companion representations for such extra specifications, as shown in Figure 3.5, and finding a suitable final implementation through design space exploration [RS14; RMU+18]. Examples of other potential data models for constraints and mappable platforms include ad-hoc descriptions for some specific use cases or the AMALTHEA meta-model (see section 3.1); both in XML format.

3.7 ASAM MDX

A first version of the Model Data Exchange Format (MDX) was released by the Association for Standardization of Automation and Measuring Systems (ASAM) in 2006. The standard specifies a model that enables the description of SW-components of a single ECU, their interfaces, and data elements. Since the ASAM MDX standard intents to be an exchange format for model data, it does not define its own methodology. The standard also does not provide an actual reference implementation, just a XML schema definition and a document type definition for formal file validation.

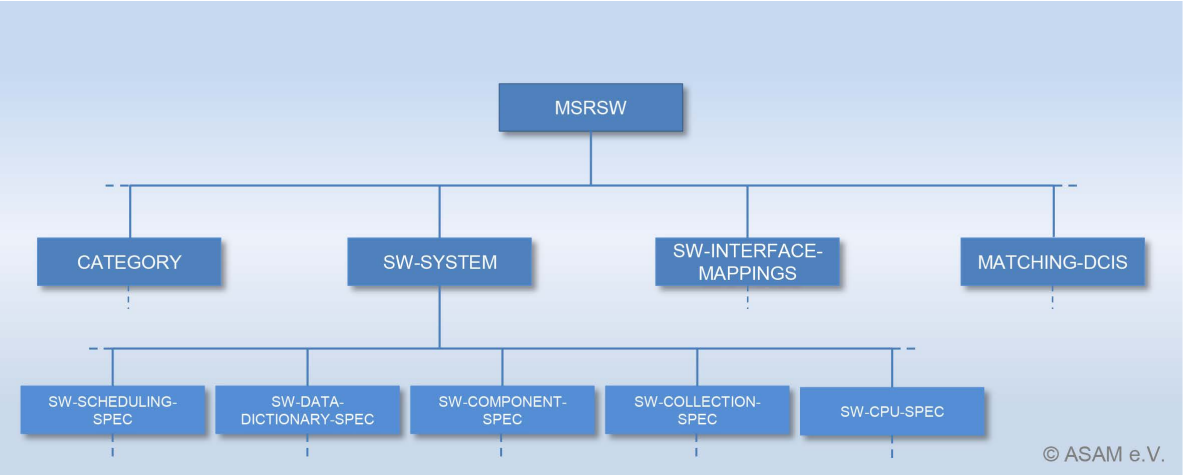


Figure 3.6: Top-level structure of an MDX-file

Source: https://www.asam.net/standards

3.8 SHIM 2.x

SHIM (Software-Hardware Interface for Multi-many-core) is an interface for software tools to extract the properties of a multicore hardware platform, which affect the software at the architectural design level. SHIM is a joint effort form academia and industry and is maintained and extended by the multicore association.

SHIM and its successor SHIM 2 should support tools which focusing on the architectural design level of multi- & many-core systems. Therefore the important characteristics like communication channels between cores(e.g. routing, message passing protocols), memory (e.g. memory size, access latency, hierarchy, topology, coherency) and the cores and accelerators itself (e.g. instructions, special execution units, address space) are described in a XML schema where the software tools can extract the relevant information. The idea is that the hardware vendor provides the SHIM XML for a dedicated hardware platform and tool vendors can use this description as an interface for their tools to us the SHIM model for optimization, analysis and system configuration (mapping of software tasks to hardware or performance analysis/prediction).

SHIM as well as SHIM 2.x do not cover any modeling regarding the software architecture, middleware or runtime environment also OS or Hypervisor related topics like scheduling are not

supported.

Improvements from SHIM 2 over SHIM are extended power consumption modelling, functional units to represents special execution units within a core, extended parameters for CPU instructions, more detailed caches, contention groups and more (see SHIM 2.o specification).

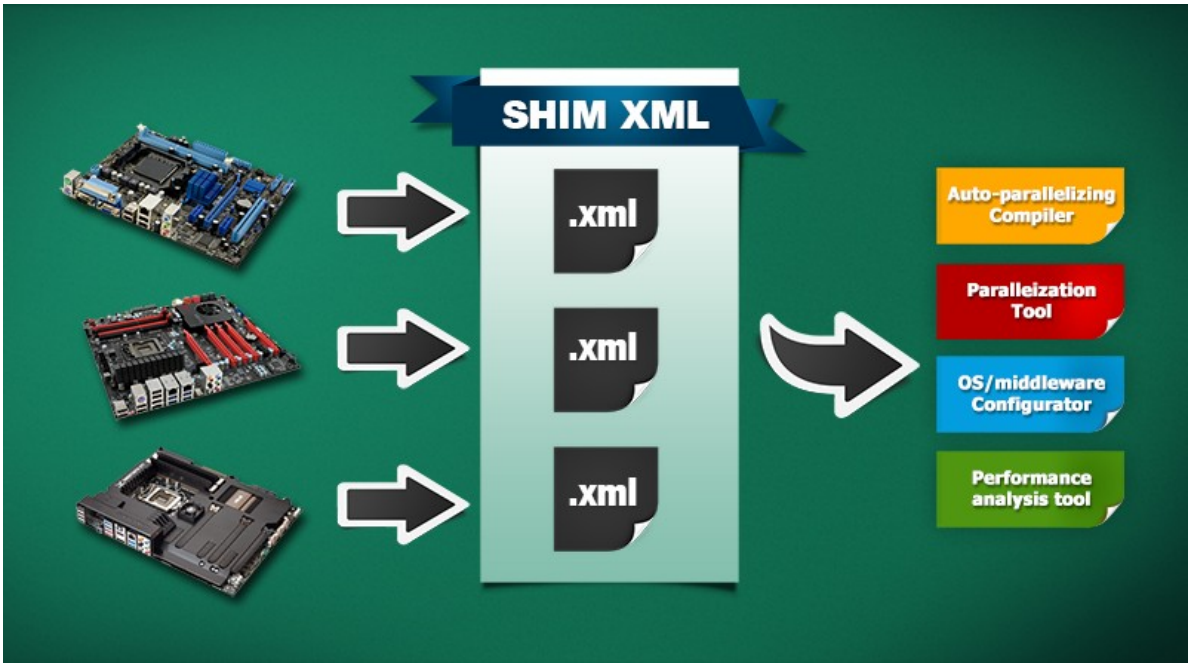


Figure 3.7: SHIM - Relationship to Multicore Tools

Source: https://www.multicore-association.org/workgroup/shim.php

References: SHIM 2.0 specification, SHIM 1.0 specification, www.multicore-association.org

3.9 AADL

The Architecture Analysis & Design Language (AADL) is an architecture description language that comes with a well defined textual and visual representation semantics [FGH06]. It was initiated in the field of avionics (AADL formerly stood for Avionics Architecture Description Language). The AADL is a Society of Automotive Engineers (SAE) standard released in November 2004 as AS5506. Targeted for software/hardware architectures of real-time, embedded systems it supports early analyses. The underlying formal modeling concepts enable an automatic architecture verification and conversion between notations. It supports specialized real-time/performance properties and software to hardware mapping. So called *annexes* that extend the capabilities of the main language are also available, *e.g.*, error model, requirements definition and analysis, ARINC653, and behavior. There is an Eclipse based open source AADL modeling tool: "Open Source AADL Tool Environment" - OSATE[4].

The language offers a number of predefined component categories with formally specified semantics. There are three of those categories:

[4] https://osate.org

- Software: Data, thread and processes
- Execution platform: Processor, memory, bus, and devices
- Software- and execution platform components: systems, groups

Each of these component categories can contain a group of features. Features represent connectors between interfaces of components forwarding control- or dataflows. There are three kinds of connectors in the AADL:

Ports represent point-to-point connections for individual data of events

Subprograms are groups of control- or dataflows (subprogram parameters)

Subcomponents-Access represents access to external data or bus components

Components and feature descriptions are enriched by a group of predefined specialized properties which offer flexible data attributes. These properties can be used to define a variety of things like a reference to source data or real-time constraints. Using property groups, these special properties can be accumulated and globally utilized.

Another feature of the AADL are so called modes attached to components. Modes represent alternative configurations of the implementation of a component. Only one mode can be set at a time so only one implementation can be active. Additionally, alternative configurations of execution plattform components can also be modeled with modes. With these modes it is possible to model dynamic changes during run–time in a static topology.

Components are defined through type and implementation declarations. A component has a type which is defined by elements contained in the component's interface and attributes that are externally visible. The definition of the component's implementation consists of the internal structure of components through subcomponents and their connections. There are also subprogramm squences, modes, flow implementations, and properties contained in the implementation. Components are grouped by the application software, execution platform, and their combinations. Furthermore, a global grouping is enabled through packaging. Extensions in the form of annexes enable the designer to expand the language specification.

According to [FGH06], the goal of the AADL was to define a standard to describe the software architecture and the execution platform of a system to design. Like in EAST-ADL (see section 3.4) there is a way to describe system components, their interfaces, and the data they interchange. Figure 3.8 depicts the structure of AADL. It is divided in global and private declarations. Private declarations are further divided as displayed in the figure.

An AADL specification file contains a package specification on the top level. Packages can have private and public components. This indicates whether the components are externaly usable or not. Sets of properties are not part of the core specification of AADL and can be viewed as a collection of types and units. Component types can be declared in packages. They can for example be a system or device. A system also has an implementation to be specified, which can contain further subcomponents and their interconnections. These connections can either be control- or dataflows. Component types also have features, which describe the input and output ports along with required bus connection access (like access to the CAN or LIN bus). Furthermore, component types contain flow specifications to describe from where to where a flow is connected. Properties can also be specified in component types. Port group types establish the identity (name), features, and properties of a collection of ports. Through annexes,

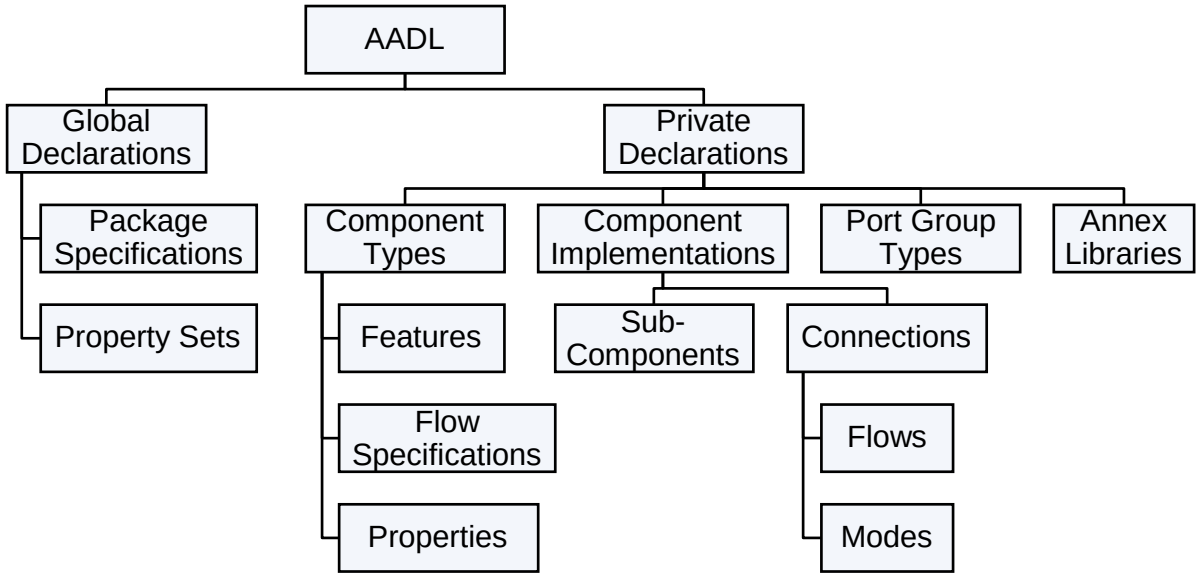


Figure 3.8: AADL structure.

AADL offers easy extensibility. Annex libraries define the name and contents of reusable AADL core-model declarations that are not part of the AADL standard.

AADL offers different ways to describe a system architecture, namely a textual, a graphical, a combined format, and a storing format defined in XML. The textual and graphical notations are part of the AADL standard.

3.10 Safety models

3.10.1 EAST-ADL Error Model

The Dependability package of *EAST-ADL (Electronics Architecture and Software Technology - Architecture Description Language)* (see Section **??**) is designed to complement the development guidelines of the ISO 26262 automotive safety standard [Int11]. Using this approach, a vehicle system's safety analysis begins with a hazard and risk analysis, where high-level hazards are identified and their risk is estimated as a combination of severity, likelihood of occurrence and driver controllability. Based on these hazards, safety objectives are defined to avoid or mitigate the hazards accordingly. The vehicle functions defined in the vehicle layer are associated with each of the hazards and are assigned appropriate functional safety objectives. As functions are implemented by systems and, in turn, by components, safety objectives are assigned accordingly based on failure analysis targeting the appropriate architectural level.

Potential error behavior identified by engineers can be captured through an *EAST-ADL Error Model* and then used to perform tool-supported analysis. Each Error Model (Type) (EMT) identifies potential error behavior of a system and characterizes it with properties such as anomaly types (e.g., internal fault, input/output failure), propagation links to other EMTs and how the output failures of the system/component function are related to its input ports or internal components. ISO 26262 Safety Integrity Levels (ASILs) can also be incorporated

into the EMT description. Different EMTs are composable via Prototypes, which apply a given EMT to a specific context, such as linking to a nominal system design [CMW+13].

The EAST-ADL Behavior Description Annex also supports the definition of state machines. State machines enable, among other uses, the incorporation of temporal/dynamic features into the system design and analysis. State machines are particularly relevant to the HiP-HOPS models methodology for its temporal fault tree analysis extensions [Wal09] and [Mah12]. These allow the definition and analysis of fault trees where the likelihood of system failure depends, wholly or partially, on the sequence of two or more failure events.

3.10.2 AADL Error Model Annex (AADL EA)

The *Architecture Analysis & Design Language (AADL)* is an SAE standard [Com17] that supports system architecture modeling. Its application domain focuses on the aerospace industry. The language supports the specification and analysis of software and hardware system architecture. The language follows a component-based paradigm, viewing systems as collections of software components mapped onto a (hardware) execution platform.

AADL is extensible, allowing a more detailed or specialized description of system behavior aspects to be defined through such extensions. One key example is the *AADL Error Model Annex (AADL EA)*, which allows component error models and relevant properties to be captured in addition to the base system architecture. This feature delivers many of the benefits of Model-Based Safety Analysis [JHMW06]; most notably, early, rapid and repeatable dependability analysis.

Under the AADL EA, error models and properties can be defined for each AADL component. Properties can include fault/repair assumptions, fault propagation behavior and fault tolerance policies. Error models describe component behavior in the presence of local (i.e. , internal to the component) failure/repair events and failure propagations as input deviations of the component. Input deviations can be propagated from other components as output deviations from the latter. Error propagation is specified by rules along system dataflow paths or explicitly by the designer.

AADL and its application in dependability modeling have been described in detail in [FR07; JH07; RKK07]. Further support has been added for deriving static fault trees [JVB07], dynamic fault trees [DD08], generalized stochastic petri nets [RKK08] and HiP-HOPS models [MBPB12].

3.10.3 SysML Failure Logic Extension

In the *COMPASS* project, SysML was chosen for the development of systems-of-systems (SoS) models which were later analyzed using the project's techniques and tools. In an outline of the vision for COMPASS [CML+12], SysML is employed alongside UML by the modeling tool Artisan Studio to model and generate software. Afterwards, the remaining tools connect either directly or through file exchange, the files being defined in the project's shared modeling language, CML. The remaining tools provide static & dynamic analysis (theorem proving and Fault Tree Analysis (FTA) [VGRH81], among others) as well as simulation of the generated software. The use of contracts aimed to address confidentiality issues between collaborating users. Contracts provide abstract behavior definitions of the systems that compose the SoS in a formal way [ABPK14].

Fault modeling in COMPASS enables the generation of fault trees from annotated system models, which can then be exported to external tools such as HiP-HOPS [PM99] for analysis. The *Fault Modeling Architectural Framework (FMAF)* aims to provide support for structured

fault modeling and is designed based on the taxonomy from [ALRL04a]. The FMAF is defined as a SysML profile from which a tool generates the corresponding HiP-HOPS XML input [IAPP14; AFPR13]. The FMAF can also be leveraged to model and analyse faults in SoS contracts [ABPK14]. For a more in-depth description of how HiP-HOPS is leveraged by this process, we refer the reader to [14].

3.10.4 SafeML

Another approach that aims at extending SysML with a viewpoint for modeling the artifacts of the safety engineering lifecycle is the modeling language *SafeML* [BSK16]. SafeML[5] is a SysML profile for integrating safety information with system design information, as an aid to information consistency and communication between development teams and among members of a team. It can be used for:

- Tracing from hazards through the safety measures used to the verification steps taken to test those measures
- Documenting the analyzed hazards and their safety measures to certification authorities
- Communicating from safety engineers to system engineers the hazards that must be considered while designing to meet requirements, as identified through the hazard and safety analysis processes
- Communicating from system engineers to safety engineers the hazards that the system is designed to manage, including the safety measures used

The goal of SafeML is to allow the intuitive documentation of hazard and safety analysis results and safety measures in the system model. This can improve consistency among multiple analyses and aid in communicating the results of analyses. SafeML focuses on making this information visible in the system design. SafeML is designed to be used in conjunction with SysML. SysML provides the diagrams and element types necessary for design modeling, while SafeML provides the element types used to add safety information to the model (see Figure 3.9).

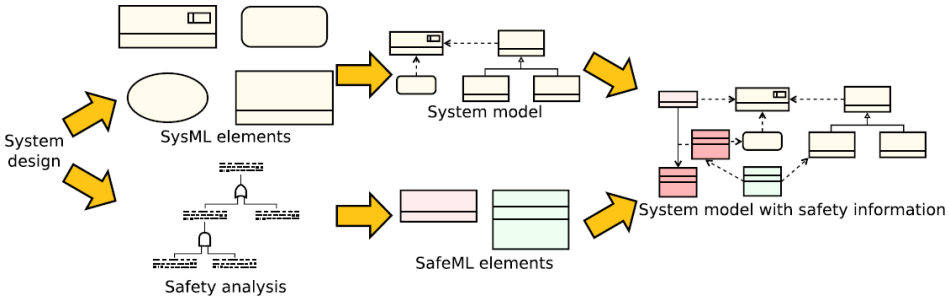


Figure 3.9: SafeML usage concept

The modeling approach is organized in two parts. The first deals with hazards, harms and the context necessary for harms to occur. The second deals with safety measures, which in SafeML are targeted at preventing the hazardous event necessary for a hazard to lead to harm, or mitigating harm should a hazardous event occur.

[5] https://astahblog.com/2016/01/28/safeml-intro/

One of the most remarkable aspects of the SafeML approach is that it is a concrete modeling language. It defines modeling elements and relationships and, more over it relies on SysML, which is by itself also a modeling language.

The purpose of SafeML is to extend SysML models with safety information, by focusing on the definition of hazards and measures. This covers however just a portion of the actual information needs for dependability systems. One could even consider the language as incomplete, since there are no means to perform a complete hazard and risk analysis. For instance, there are no means for identifying malfunctions. Therefore, this process should basically occur somewhere else and only the resulting hazards will be documented.

3.10.5 Common Assurance & Certification Metamodel (CACM)

In the AMASS project[6], a *Common Assurance & Certification Metamodel (CACM)* has been developed. CACM is a cluster of metamodels that captures concepts in various aspects of system assurance, such as system specification, argumentation, evidence, process, standards and the mapping between process and standards. The overlap between AMASS and DEIS is quite observable, therefore, it is best practice to re-use part of the CACM from AMASS to model process and certification. CACM has a subset of metamodels called Compliance Management Metamodel, which contains the following aspects:

- **Assurance project definition**. It is used to define the assets produced during the development, assessment and justification of a safety-critical system;
- **Process Definition**. It is used to model general reference processes (e.g., Waterfall Process, Agile Process, V-model process), or company-specific processes (e.g., the Thales process to develop safety-critical systems).
- **Standard Definition**. It is used to model standards (IEC 61508 [Int98], ISO 2626 [Int11], DO-178C [Rad11], EN 50126 [Eur99], etc.) and any regulations (either as additional Requirements or model elements in a given model representing a standard or a new reference standard). For the implementation another metamodel is added, the *Baseline Metamodel*, to capture what is planned to be done or to be compiled with a defined standard, in a concrete assurance project.
- **Vocabulary Definition**. It is used to provide a Thesaurus-type vocabulary, which defines and records key concepts relevant to safety assurance within the target domains and the relationships between them.
- **Mapping Definition**. It is used to capture the nature of the vertical and horizontal mappings between the different levels of model in the AMASS Framework and between the concepts and vocabulary used in these models. There are two types of mapping: equivalence mapping that maps process models with models of standards; and process mapping, which maps process models to project specific models.

3.10.6 Open Dependability Exchange Meta-model (ODE)

The *Open Dependability Exchange Meta-model (ODE)* is the meta-model for the core concept of the *Digital Dependability Identity (DDI)* and thus a central artifact of the H2020 DEIS project[7].

[6] https://www.amass-ecsel.eu/
[7] http://deis-project.eu/

The second version of the ODE (ODEv2) consists of two part: the *Structured Assurance Case Metamodel (SACM)* [Obj19; WKD+19] and the ODE itself. The ODE is clearly distinguished in its role as a collection of 'Product' metamodels. In contrast, the SACM provides a generic, higher-level structure for encapsulating assurance claims and argumentation and can reference other models, including those derived from Product metamodels such as the ODE v2.

Figure 3.10 presents an overview of both the SACM (highlighted in purple) and the ODE (highlighted in green). The overview indicates that while there has been some reduction and simplification, the ODE (and SACM) remain quite complex metamodels, spanning across a plethora of metamodeling elements. The ODE is open source and can be found on GitHub https://github.com/DEIS-Project-EU/ODEv2.

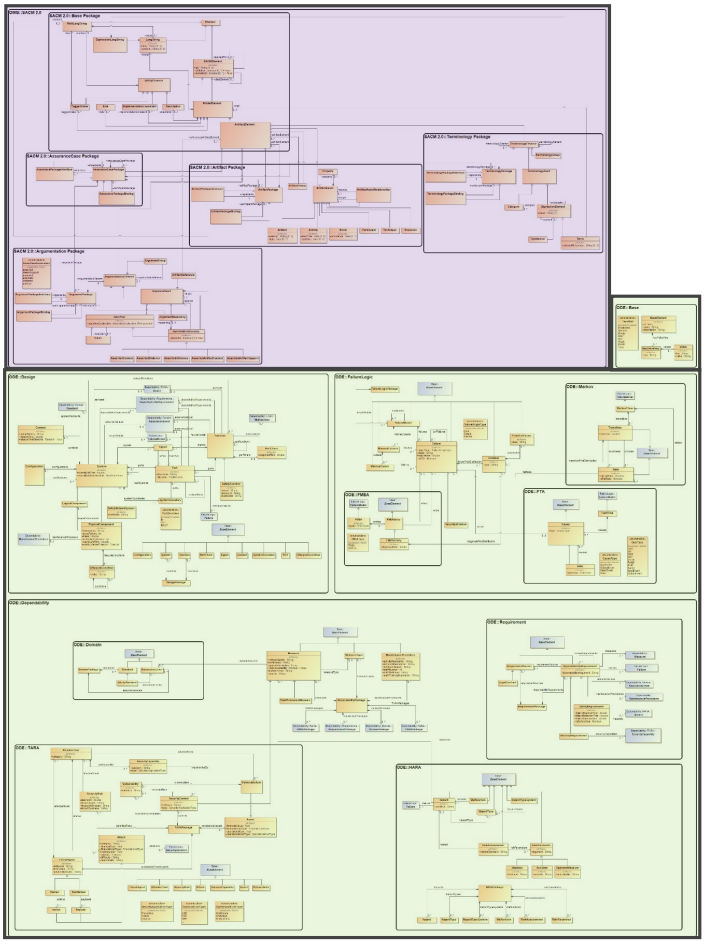


Figure 3.10: Overview of the ODE v2 Metamodel

Structured Assurance Case Meta-Model (SACM) 2.0

The *Structured Assurance Case Meta-Model (SACM)* [Obj19; WKD+19] is a modeling language specified by the Object Management Group (OMG) to create model-based assurance cases. SACM supports existing system assurance case approaches such as the *Goal Structuring Notation (GSN)* [KW04] and *Claims-Arguments-Evidence (CAE)* [BB00].

An assurance case is a set of auditable claims, arguments and evidence created to support the claim that a defined system/service will satisfy typical requirements such as safety and/or security. An assurance case in this context is a machine-readable model that facilitates the exchange of information between various systems stakeholders such as suppliers and integrators, and between the operator and regulator, with the knowledge related to the safety and security of the system being communicated in a clear and defendable way. Each assurance case should communicate the scope of the system, the operational context and the safety and/or security arguments, along with the corresponding evidence.

In general, the SACM enables the user to create assurance cases by combining structured argument(s) into *ArgumentPackage(s)* with their corresponding evidence defined in *Artifact-Package(s)*, as well as the controlled vocabularies used within the scope of the assurance case with regards to the information of the system/service for which the assurance case provides assurance for, in *TerminologyPackage(s)*

The SACM meta-model consists of the following parts:

- The SACM takes into consideration that machine-readable assurance cases can be created. The **Base component** of the SACM provides the necessary means such that not only names/descriptions can be described in natural language, they can also be described in computer languages (e.g., formal notations) to ena ble automated argument reasoning in future. At the same time, the SACM provides various facilities allowing the user to define necessary constraints, notes, additional attributes etc.

- The **SACM Argumentation component** provides the facilities for creating structured argumentations. The user of the SACM can make a number of different types of claims which provide means of assertion, context, assumption and justification. The user can also make use of the *Artifact* component to refer to corresponding evidence (internal/external to the SACM model) to support the claims. There are various types of *AssertionRelationships* to link claims to sub-claims, contexts, assumptions, justifications etc.

- The **SACM Artifact component** provides the facilities to maintain evidence such as *Resource*, *Artifact* as well as *Activities*, *Event*, *Participant* and *Technique*,. SACM enables the user to point to external files/URLs of the related artifacts via the use of Property. In this sense, the SACM provides the necessary abstraction, as it does not demand the use of *models* for argumentation evidence. This abstraction provides a significant degree of openness regarding its adoption in open systems (i.e., Cyber-Physical Systems). However, the SACM does provide the necessary means for the Artifacts to be linked to model elements in the sense that constraints (described in model querying languages such as the Object Constraint Language) can be embedded into each one of the Artifacts, which, in turn, would be executed at runtime and retrieve the value of the referenced model elements.

- The **Terminology component** provides the necessary means for defining controlled vocabularies which in turn link system information to the structured argumentation in the *ArgumentPackages*. Concerning system information, the user can define *Terms*, *Expressions* and *Categories*, which are the terminologies in the system for which the assurance case provides assurance. At this point SACM also provides the necessary abstraction so that external system information (such as system models, failure logic models, FMEA models, FTA models etc.) can be referenced.

ODE v2 Product Metamodels

This section provides an overview of the overall structure of the ODE meta-model.

Design Package The *Design package* contains elements that are necessary for describing architectural aspects of the system under development. These aspects model both the structure and behavior of the design, including functional, performance and design attributes. Note that *ODE::Design* is not intended to provide modeling elements that are required for engineering a system architecture completely, but rather only those required as input for the dependability lifecycle.

The *System* is the central root element of the design package.A *System* can comprise sub-systems and ports. *Ports* represent the explicitly defined interface through which the *System* communicates with external systems or sub-systems via signals. A *Signal* can be described as a connection between ports through which information (e.g., data flows) is passed between different *Systems*. Depending on the direction of the signal (incoming, outgoing, both), a *Port* has an assigned *PortDirection*. Independent of the specific modeling aspect, a *System* as a hierarchical representation of the architectural structure will always have a set of *Functions* representing the behavior the *System* should realize. Attached to a function to be realized are the required *PerfChars* (=performance characteristics) emerging from the functional requirements of the *System*. In addition, the (typically embedded) *System* will always operate within a certain *Context*, which might contain relevant information about the *System's* operation, usage or environment.

FailureLogic Package The *FailureLogic package* contains the meta-model elements which describe the potential causes of failure of the system. These causes are derived from the failure analysis (Markov modeling, Failure Modes and Effects Analysis (FMEA) [Int91] or Fault Tree Analysis [VGRH81; Int90]) captured under a specific *FailureModel*.

The *FailureModel* contains most of the other elements of the package, providing access to various types of *Failures* to be shared through it. *FailureLogicPackages* are also composable, which provides support for modular and hierarchical analysis techniques such as Component Fault Trees (CFTs) [KLM03; KSA+18; HJZ+18] and HiP-HOPS [PM99]. The *Failure* element aims to abstract common characteristics of the various failures that can be identified within functions, systems or components. *Failures* attributable to a specific architectural element can be categorized into *InterfaceFailures*, which can be *InputFailures* or *OutputFailures*, or *InternalFailures*. This distinction is based on the viewpoint of the analysis of the failure. For example, if the analysis of a system identifies failures originating from within its boundary, those are *InternalFailures*. Furthermore, the abstraction is useful for composing heterogeneous failure analysis results of hierarchical models. Other types of *Failure* that can be modeled are so-called *Common Cause Failures (CCFs)*, which can trigger other failures and are captured via the CCF element. The *MinimalCutSet* element describes the combinations of *Failures* that can lead to *OutputFailures*.

HARA Package In the *HARA* package, the essential root data elements used in Hazard and Risk Analyses (HARAs) and their relationships are modeled.

The root of this meta-model is the element *Hazard*, which linking the data elements *Failures* and *Measure* and where the hazard analysis takes place based on the corresponding malfunction. *Malfunction* is modeled as a separate data element which references the identified function

(*Function*). It is further referenced by the *Hazard* element, where the hazard analysis takes place based on the corresponding malfunction. A *Hazard* can be associated with zero or more *Failures* since hazards can also be caused by a combination of failures. The identified hazard is then referenced by:

- A *RiskAssessment*, for conducting quantitative risk analysis of the corresponding hazard
- A *SafetyRequirement*, for the derivation of the safety goal and safety requirement
- A *SafetyFunction*, for the derivation of the safety function;

ODE::Dependability::Requirements Package The focus with respect to dependability-requirement modeling rests on safety requirement modeling in the ODE meta-model. A safety requirement is derived from a *Measure* and the corresponding *RiskAssessment*. It refers to what the *SafetyRelatedSystem* shall do as well as to what it shall not do, in order to ensure the *SafeState* of the designated system and its environment, and the quantitative or semi-quantitative quality and integrity requirements that the designated system shall fulfill. Except for the security requirements, all the other dependability requirements are derived from hazard and failure analysis techniques. Functional hazard analyses, FME(C/D)A, FTA and Markov chains all lead to the fault tolerance measure and the risk assessment. The respective dependability requirements could be derived as follows:

- Safety requirement based on the identified measure and its associated integrity level, which are the results of the hazard analysis and risk assessment
- Reliability requirement based on the probability and failure rate (as a time function) calculated by means of a quantitative hazard/failure analysis, such as fault tree analysis
- Availability requirement based on the availability calculated by means of quantitative fault tree analysis
- Maintenance requirement based on Mean Down Time (MDT), also calculated by means of quantitative fault tree analysis

ODE::TARA Package The TARA package models the results of a Threat And Risk Analysis (TARA) for security. *ThreatAgents* are either *Human* or *NonHuman* (typically electronic) sources of *Attacks*. While individual *Attacks* may serve many purposes, *ThreatAgents* will also feature some higher-level *AttackerGoal*, representing the overall goal of the attacker. The *AttackerGoal* revolves around negatively impacting the *Assets* being considered for security, often being the system's operation and its data for example. *Attacks* exploit *Vulnerabilities* of the system.

Cumulatively, the above elements contribute towards the *SecurityRisk* posed by the various threats identified during the TARA. To combat these threats and reduce risk, *SecurityCapabilities* and *SecurityControls* are established. Respectively, these are high-level and low-level security safeguards/counter-measures.

SecurityCapabilities are directly associated with *SecurityRequirements*. *SecurityControls* are instead directly associated with *Measures*. After applying these measures, risk is reduced accordingly.

3.11 Event traces

A trace is defined as a sequence of events. Events depict a change in the state of a system and can be represented on different levels of abstraction. For the timing analysis of embedded multi-core real-time systems a trace on system level is required.

Tools that analyze or visualize traces must be able to interpret the recorded events. For example, the software that interacts with hardware trace devices must be able to understand the hardware events that are generated on-chip. Otherwise it is not possible to transform the hardware events into higher level software events. For that reason a well-defined format for events is required for further processing of recorded traces.

Depending on the goal pursued with a trace measurement, one level of abstraction can be more appropriate than another. On the one hand, a software engineer who implements a feedback control system is mainly interested in the functions and variables that correspond to that particular task. A system engineer on the other hand, who integrates a variety of different modules into a single application, is not interested in the details of each individual module. Instead the functionality of the system as a whole is of interest. There are several trace data exchange formats, in the following, we will give brief introductions about some of them.

3.11.1 Better/Best Trace Format

A trace on system level can be used to analyze timing, performance, and reliability of an embedded system. The Better Trace Format is a simple text-based trace format which was developed to support these use cases. It is intended to ease the exchange of traces between measurement tools, simulators and analysis tools. Its simple structure based on columns and separators allows the implementation of own scripts as well as the import into MS Excel and other spreadsheet programs.

The Better Trace Format was defined in 2009 by Continental and INCHRON. Since then, the format has been changed slightly and was renamed "Best Trace Format". It was also provided as a trace format by Timing Architects (now Vector) to the ITEA2 project AMALTHEA and was released by the Eclipse Auto IWG [Tim14]. The following details about BTF are mostly sourced from [Mar15].

BTF assumes a signal processing system where one entity influences another entity in the system. This means an event does not only contain information about which system state changes but also the source of that change. For example, an observed event on system level could be the activation of a task with the corresponding timestamp. Then a BTF event additionally contains the information that the task activation was triggered by a certain alarm.

Let k be an index in $\mathbb{N}_0$ denoting an individual event occurrence then a BTF event can be defined as an octuple:

$$b_k = (t_k,$$

$$\in$$

Field	Meaning
time (t)	Timestamp relative to a certain point in time.
source	Entity that caused an event.
source instance (ψ)	Entity instance that caused an event.
target type (ι)	Type of the entity that is influenced by an event.
target (T)	Entity that is influenced by an event.
target instance (τ)	Entity instance that is influenced by an event.
action (a)	The way in which target is influenced by source.
note (υ)	An optional field that is used for certain events.

Table 3.1: A BTF event consists of eight fields. An event describes the way in which one system entity is influenced by another one.

A BTF event can be represented textually as a comma-separated list where each field maps to an element as shown in the following listing:

```
1 | 12891 , TASK_200MS, 3 , SIG , EngineSpeed , 0 , write , 42
```

The first field (12891) represents the timestamp of the event. A BTF trace contains the chronological order of events that occurred in a system. Therefore, for each timestamp $t_k \in \mathbb{N}_0$ in a trace it holds that $t_k \leq t_{k+1}$. All timestamps within the same trace must be specified relative to a certain point in time, that can be chosen arbitrarily. Hence, neither trace nor system start must occur at $t_0 = 0$. The time period between two events b_k and b_{k+1} can be calculated as $t = t_{k+1} - t_k$. If not specified otherwise, the unit for time is nanoseconds.

A BTF event represents the notification of one entity by another. Each entity has an unique name. In the previous example, the source entity has the name TASK_200MS and the target entity T is called EngineSpeed.

The fourth field SIG is the short representation of the target entity type ι. Table 3.2 gives an overview of all entity types and their corresponding short ID. In this example, the target entity EngineSpeed is a signal. The source entity type is not part of a BTF event.

Some entities, tasks, isrs, runnables, and stimuli have a lifecycle. This means at a certain point in time an entity becomes active in the system and eventually it leaves the system. For example, the lifecycle of a task starts with its activation and ends when it terminates. If multiple task activations are allowed for an application, it is possible that multiple *instances* of a task are active at the same time. For those cases where multiple instances of an entity are currently active, it is consequently not clear to which instance of the entity the event refers.

Instance counter fields ψ and τ are used to distinguish between multiple instances of the same entity. The counters are integer values $\psi, \tau \in \mathbb{N}_0$ that are incremented for each new entity becoming active in the system. The first instance of an entity gets the counter value 0. TASK_200MS has an instance counter value of 3 which means the event refers to the fourth instance of this entity. For entities that do not have a lifecycle like signals, the counter field is not relevant and 0 can be used as a placeholder value.

The seventh field a represents the way in which the target entity is influenced by the source entity. In this example TASK_200MS writes a new value to the signal entity EngineSpeed.

```
version 2.1.3
creator BTF Writer (15.01.0.537)
creation Date 2015-02-18T14:18:20Z
timeScale ns
     0,       Sim,   0, STI,       S_1MS,  0, trigger
     0,    S_1MS,    0,   T,    T_1MS_0,   0, activate
   100,   Core_0,    0,   T,    T_1MS_0,   0, start
   100, T_1MS_1,     0,   R,  Runnable_0,  0, start
 25000, T_1MS_1,     0,   R,  Runnable_0,  0, terminate
 25100,   Core_1,    0,   T,    T_1MS_0,   0, terminate
```

Listing 3.1: A BTF trace file consists of two sections. A meta section at the beginning of a file includes information such as creator, creation date and time unit. It is followed by a data section that contains one event per line. Comments are denoted by a number sign followed by a space.

Depending on source and target entity type, different actions are allowed by the specification.

For signal write events the note field v is used to denote the value that is written to the signal in this case 42. The note field is only required for specific events.

A BTF trace can be persisted in a BTF trace file. This file contains two parts: a meta and a data section. The meta section is written at the beginning of the file. It contains general information on the trace such as BTF version, creator of the trace file, creation date, and time unit used by the time field. Each meta attribute uses a separate line, starting with a , followed by the attribute name, a space, and the attribute definition.

In the data section one BTF event is written per line in chronological order. The first event of a trace is located directly after the meta section and the last event at the end of the file. Comments are denoted by a followed by a space. Listing 3.1 shows an example trace file.

As shown in Table 3.2 BTF specifies fourteen entity types that can be classified into five categories: environment, software, hardware, operating system, and information. The actions or in other words the way in which one entity can be influenced by another are defined for each entity type.

Category	Entity Type	Type ID
Environment	Stimulus	STI
Software	Task	T
	Isr	I
	Runnable	R
	Instruction Block	IB
Hardware	Electronic Control Unit	ECU
	Processor	Processor
	Core	C
	Memory Module	M
Operating System	Scheduler	SCHED
	Signal	SIG
	Semaphore	SEM
	Event	EVENT
Information	Simulation	SIM

Table 3.2: BTF entity types can be divided into five categories.

The current version of the BTF specification [Tim14] is in some definitions not clear and unambiguous enough, so a reworked version is currently being prepared.

3.11.2 SQL database

Tracing in text-based formats, as BTF, offers an easy way for interoperability between different tools, but there are several downsides. Textfiles or comma separated value files have usually a lower density of data, because e.g. numbers are mostly stored as character strings. This especially slow down the creation of such files in simulations or while the execution of real systems. Moreover, for the analysis of such files, they mostly need to be parsed completely into memory or you use line based file operations, which are usually slow. Besides these technical issues, the proposed structure of an ordered list of events is a natural and beneficial way to describe trace events. Therefore, the APP4MC project also includes the *AMALTHEA Trace Database* (ATDB) specification.

Basically, the ATDB describes the usage of a relational database for the same information as stored in BTF. In APP4MC the usage of SQLite is proposed, but the database model can also be used in other SQL-based systems. The traced events are stored in EventTables, which mainly correspond to the data section of BTF: a list of occurred events with the corresponding timestamp and involved entities. The referenced entities in EventTables are described in Entity tables. Further information about event-entity relations are stored in EntityInstance and EntitySource tables.

Besides the more efficient storage and faster processing of a relational database, the usage of an SQL-based data storage allow the definition of standardized analysis queries, by the definition of SQL-Queries. For example, a query to determine the latency for the completion of a Runnable. Such queries allow a formal and unambiguous analysis objectives and may be defined within the PANORAMA project.

4 Abstraction Levels

During the design process of an embedded system there are several *versions* of the model representation of said system. These versions are intermediate states of the system model(s) e.g. after taking a design decision, after a handover between different departments, after adding specific details, etc. There are two dimensions in which these versions can be classified: Abstraction levels and views. All views of a system combined, constitute the whole system model - they must be consistent. Whereas each abstraction level represents a whole system model (which may differ between levels). This abstraction level/view matrix has been used and adapted in several other research projects SPES2020 [BDH+12], SPES_XT [BDK+16], CESAR [RW13], CRYSTAL [CRY16], and others. There is also a standard about architecture views and viewpoint definitions: [IEE11]. It served as a reference for the abstraction levels and views.

4.1 ISO/IEC/IEEE 42010-2011

This standard defines a set of terms and notions to better understand architecture descriptions in general. Among these notions are: Architecture, Architecture Description, Architecture View, Architecture Viewpoint, Concern, Model Kind, and Stakeholder. It then relates the terms to each other, *e.g.*, an architecture description *expresses* an architecture which a system *exhibits*; a stakeholder *has interest in* a system; stakeholders *have* various concerns. The standard does not define abstraction levels, however, abstraction levels can be seen as a concern.

Figure 4.1 shows an excerpt of the relevant artifacts and their relations in the scope of an architecture description. An architecture description (which expresses an architecture) identifies relevant stakeholders and concerns. The stakeholders themselfes have these concerns which, in turn, are framed by one or more architecture viewpoints. Architecture viewpoints govern architecture view, *i.e.*, the viewpoints define what is visible or not in a specific view. Architecture viewpoints have one or more model kinds for representation purposes. An architecture view has one or more architecture models which are governed by the respective model kinds of their architecture viewpoints.

With these architecture description definitions we can define abstraction levels as follows: Having the possibility to subdivide the system design process artifacts into different abstraction levels is a concern. Two such abstraction levels could be called “System Design” and “Software Design”. These two are architecture viewpoints. Within each of these viewpoints we define what model kinds can be used to model the architecture on this abstraction level. For example: A package diagram is only to be used in the viewpoint “System Design” whereas a class diagram is only to be used in the viewpoint “Software Design”.

There are also some additional *Meta* artifacts that relate to all of the above architecture description elements: Correspondence, Correspondence Rules, and Architecture Rationale. Correspondences (which are governed by Correspondence Rules) can be used to enforce relations between concrete architecture description elements. For example, we can enforce that models on the “System Design” abstraction level must exist befor we can start creating models on the “Software Design” level. We can then enforce that each “System Design” artifact must have

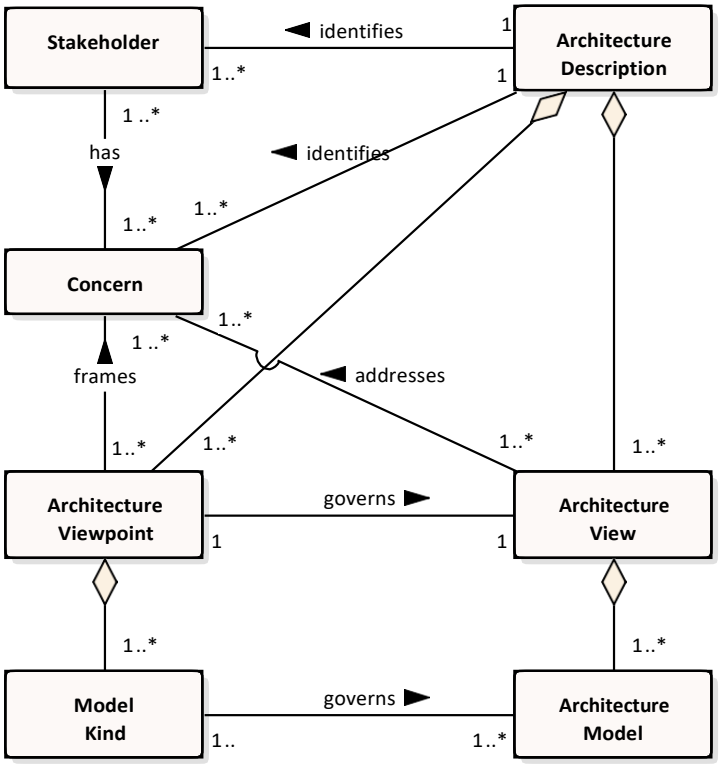


Figure 4.1: Architecture Views and Viewpoints (excerpt from [IEE11]).

at least one "Software Design" pendant which it was refined into. As mentined, there are also Architecture Rationales which can be used to motivate the existence of certain architecture elements. These rationales can serve as a basis upon which specific decisions can be made during the development. For example, when transitioning from "System Design" to a lower abstraction level, it may be necessary to decide if a "Hardware Design" level is needed at all. The description of the rationale for the "Hardware Design" abstraction level can then be used to justify the decision.

The ISO/IEC/IEEE 42010 standard "Systems and software engineering – Architecture description" gives a broad overview of artifacts to consider during the system development process. *How* this architecture description is structured and implemented is highly dependent on the development process of that system. For example, if we know we want to develop an embedded system it may be a good idea to first design the system before subdividing the design into soft- and hardware. The relevant chosen abstraction levels and the order in which to traverse them thus is intertwined with the process. The following section will shortly outline how this standard was used as a reference in earlier research projects.

4.2 Abstraction Levels in other Projects

This section shortly describes two previous research projects which relate to abstracion levels and viewspoints. There are the two BMBF-funded projects "Software Plattform Embedded Systems 2020" - SPES2020 and the followup "Software Plattform Embedded Systems XT" - SPES_XT. On european level there are the projects CESAR - "Cost-Efficient methods and

processes for SAfety Relevant embedded systems" and the followup CRYSTAL - "CRitical sYSTem engineering AcceLeration"

4.2.1 SPES2020 and SPES_XT

During the two SPES projects abstraction layers and perspectives were defined, resulting in a *design matrix*. In this matrix each cell within a row represented a different viewpoints of the same system, and the conjunction of these viewpoints constitutes a consistent system. There are four of these viewpoints defined: requirements, functional, logical, and technical viewpoint. Whenever the designer forwards into another abstraction layer or viewpoint there are some conditions and steps to be taken into account (*e.g.*, to ensure proper traceability). The resulting two-dimensional matrix is depicted in Figure 4.2. The development process usually starts with the requirements phase in which the system under design is considered as a black box that should perform certain functions. This requirements viewpoint (see top left hand side of the figure) is described in the next paragraph.

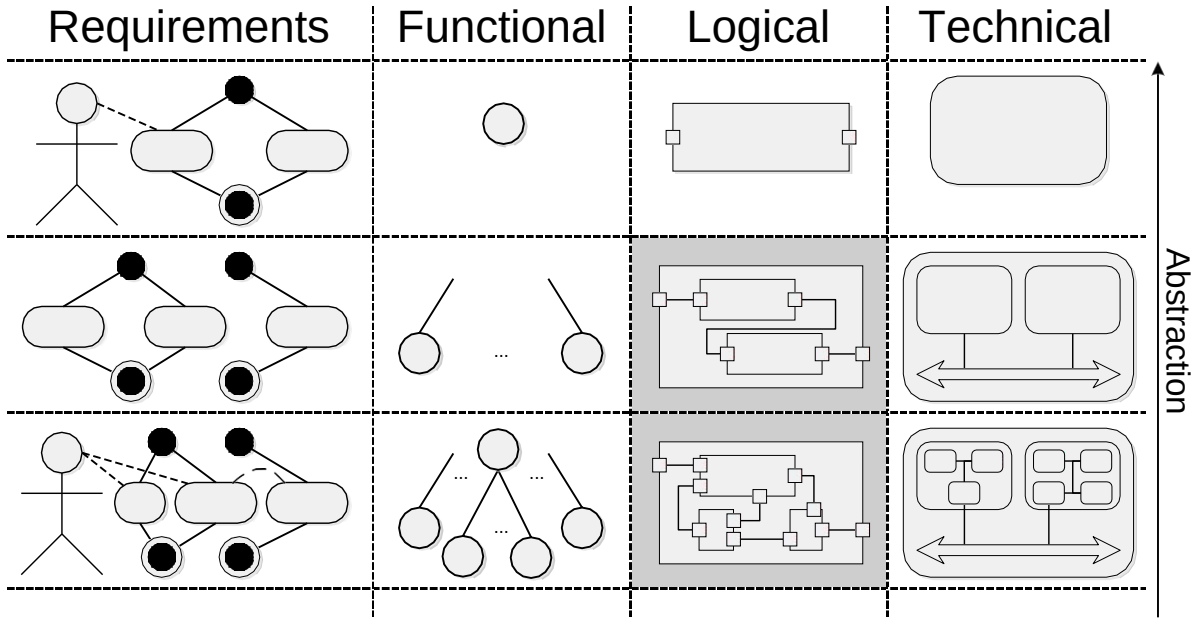


Figure 4.2: SPES2020 design process: Two-dimensional matrix.

Requirements Viewpoint

Classic systems design starts with numerous natural language requirements towards the system that is to be developed. In model based design it is beneficial to *model* these requirements since this gives a more unambiguous meaning, enables a better understanding, and allows for a higher degree of automation early in the design process. For example, one possibility to model requirements is to use the Unified Modeling Language (UML) [OMG17] use case diagrams, activity diagrams, or sequence charts. The latter may be automatically transformed into initial functional interfaces [DWP14].

Functional Viewpoint

With the help of elicited details about the requirements towards the system under design the resulting functional structure is further refined in this viewpoint. With this step the system engineer now enters the solution space and refines the internal functional structure of the functions derived from the corresponding requirements. This process also involves deriving more technical requirements (also called constraints) from the initial ones elicited in the requirements viewpoint.

Logical Viewpoint

After defining the functional structure we now need to take a deeper look into the logical structure which does not have to be the same as the functional one. Since we now explicitly consider the communication between the functions it might be a good idea to group some functions to a logical component and to optimize the structure in a logical way. Consider for example two functions both calculating something based on the same sensory input. To minimize the overall communication in such a scenario on the one hand we could combine both functions to one logical component. On the other hand we have to ensure that such a logical component does not meld too much behavior such that it is impossible to be executed on a target platform.

Technical Viewpoint

Once we described the size and frequency of communication messages between logical components in the logical viewpoint we can now decide on which technical architecture this logical architecture is to be executed. One of the most important decisions to be made in this viewpoint is how to realize a logical componet. This decision alone for each logical component is already a research topic of the last decades. The technical viewpoint uses more abstract notions. It mainly involves resource providers (*i.e.*, computation resources) and resource consumers (*i.e.*, features or applications to execute) [WRHS12]. It also employs various concepts of resource sharing and their inherent necessity of a resource provider to have a scheduler. This is particularly important when validating hard real-time constraints against the model in the technical viewpoint.

4.2.2 CESAR and CRYSTAL

CESAR and CRYSTAL take a somewhat different approach to viewpoints and layers. They first define development activities like requirements engineering, architecture exploration, and component-based development. Then, the Reference Technology Platform is introduced which brings these activities together into a tool prototype with a meta-model, a general process, and domain specific instanciations. Another general idea is to connect tools with each other via a common understanding (meta-model) to build tool-chains that support the general process (this idea was later picked up by SPES_XT). In CESAR, the abstraction levels, viewpoints (called perspectives), and how to traverse them are defined in more detail as in SPES2020. Nonetheless, these levels and viewpoints are very similar to SPES2020. The main difference is that in CESAR the abstraction levels are not arbitrary.

Figure 4.3 shows the four perspectives and abstraction levels addressed in CESAR. As can be seen, apart from the operational perspective, the viewpoints are the same as in SPES2020. There are four abstraction levels: Operational, Functional Need, System Composition, and

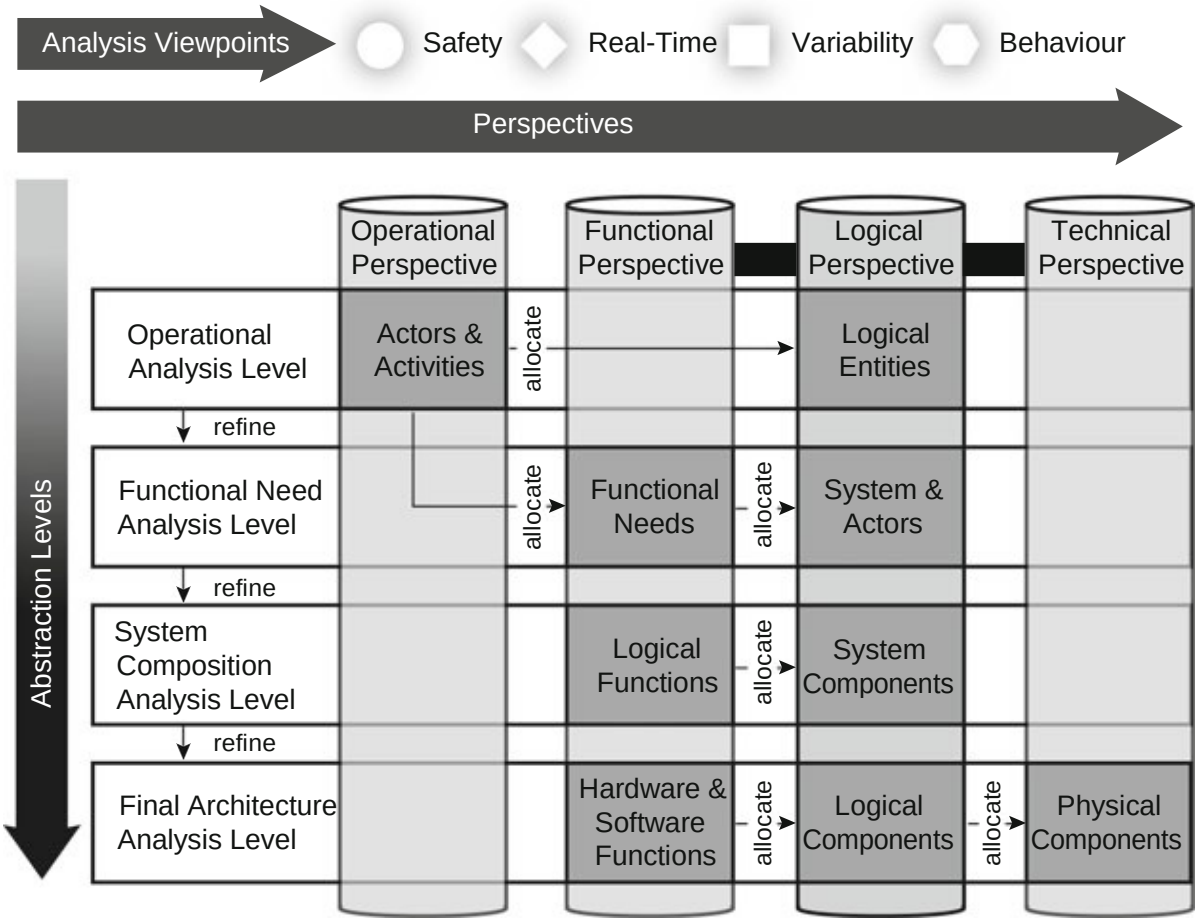


Figure 4.3: CESAR Engineering multi-views (from [RW13, p. 226]).

Final Architecture Analysis Level. Intersecting the rows with the columns shows how these cells should be traversed during the development process. The levels and perspectives have been inspired by EAST-ADL (see section 3.4).

5 Static Analysis

This Chapter describes static analysis methods, which are based on offline analysis of the expected behavior of the system focus. These methods stipulate assumptions on the, usually worst-case, behavior of the system, to analyze, if non-functional requirements can be met.

The Chapter starts with the description of the approaches to determine the behavior of the system, in what concerns timing properties. Section 5.1 discusses the approaches to offline determine the execution times of applications, whilst Sections 5.2 and 5.3 provide two different approaches to determine if the execution of the application will meet its end-to-end timing requirements.

Afterwards, the joint consideration of time and energy is analyzed in Section 5.4, which discusses a method to minimize the energy consumption of an application, using Dynamic Voltage and Frequency Scaling (DVFS), whilst guaranteeing the timing requirements of applications.

Finally, the Chapter focuses in approaches to analysis safety properties of the system. Section 5.5 describes approaches to guarantee that the deployment of an application into the underlying platform meets the segregation requirements mandated to ensure system safety. Section 5.6 then describes approaches to analyze fault-propagation, whilst Section 5.7 focus on a specific approach (CFT).

5.1 Static Timing Analysis

Most timing analysis methodologies focus on determining the worst-case execution time (WCET) of a program, a thread or a task, in order to use this value to determine a safe upper bound (Figure 5.1) on the execution time, to be able to verify if the software will meet specific timing deadlines imposed by the application (using other analysis such as schedulability analysis – e.g. utilization tests or response-time analysis).

Although several different techniques for determining WCET are available, they can be basically divided in two main blocks. Static timing analysis is based on using the information available on the structure of the program, and a model of the underlying hardware, to determine the WCET, without actually needing to execute the application. Measurement-based timing analysis (MBTA) is based on executing the application, with multiple inputs steps, to obtain the maximum observed execution time. This section will provide some information on static timing analysis, with MBTA (and hybrid approaches) being described in the next chapter.

Static timing analysis is usually performed in three conceptual and possibly overlapping phases [NYP15]:

1. A flow analysis phase in which information about the possible program execution paths is derived. This step builds a control flow-graph from the given program with the aim of identifying the worst path (in terms of execution time).

2. A low-level analysis phase during which information about the execution time of atomic parts of the code (e.g., instructions, basic blocks, or larger code sections) is obtained from a model of the target architecture.

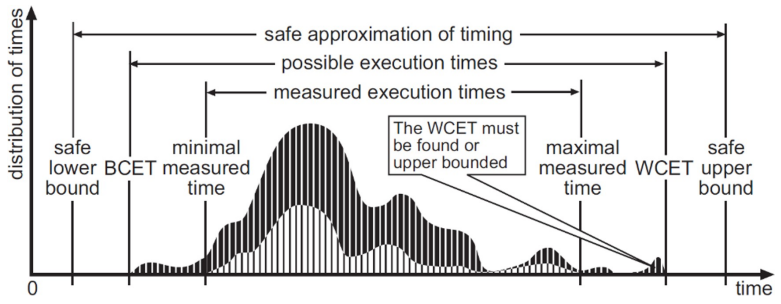


Figure 5.1: Example distribution of execution time (picture taken from [EE07]).

3. A final calculation phase in which the derived flow and timing information are combined into a resulting WCET estimate.

Flow analysis needs to consider both loop bounds (requiring upper-bounds on the number of iterations) and maximum recursion depth. Another purpose of flow analysis is to identify infeasible execution paths, which are paths that are executable according to the control-flow graph but are not feasible when considering the semantics of the program and the possible input data values.

Low-level analysis methods typically use models of all the hardware components and their arbitration policies, including CPU caches, cache replacement policies, write policies, instruction pipeline, memory bus and their arbitration policies, etc. These models are typically expressed in the form of complex mathematical abstraction.

The use of static analysis lends itself to formal proofs in guaranteeing timing response of applications, with several WCET tools available, with examples being aiT [aiT] on the commercial side and OTAWA [OTA] in the research domain.

Static timing analysis relies on the existence of an accurate model of the timing behavior of the underlying hardware, a difficult but possible task for simpler platforms, but that is very challenging when considering parallel and heterogeneous platforms. Although the bulk of work which has been performed to derive schedulability analysis for predictable parallel execution [BBB10], the multiple interferences and inter-dependencies between parallel execution (together with the many times absence of details of commercial platforms), make that designing an accurate hardware model is currently very challenging [NYP+14]. Moreover, solutions tend to model each hardware resource separately, deriving independent worst-case values, which are then composed together to form the final WCET bound, inherently extremely pessimistic.

Tools are lagging hardware evolution, and solutions are not coping with the emergence of heterogeneous systems (e.g., accelerators and specialized IP blocks), and multiple layers of underlying software, including complex OSes and hypervisors. Therefore, static timing analysis is not considered in the scope of PANORAMA.

5.1.1 Memory Management

On multi-core systems allocation and distribution of data and code to microprocessor memories *RAM*, *Flash* of an ECU plays an important role. Because, each reading and writing operation have different latencies depending on the memory location. Therefore, the following aspects need to be covered for improving the runtime efficiency of the system. **Variable access times** The

read/write access times for accesses from SW running on a specific core to a variable allocated in RAM varies on a microprocessor with NUMA architecture.

Instruction Fetch time The instruction fetch time to load an instruction from a Flash Bank to the Core local program cache of a specific core depends on the bus situation.

So methodologies should be developed based on static analysis techniques to identify the access statistics for variables and data and re-allocate them according the core usage.

5.2 Schedulability Analysis

Methods for schedulability analysis allow to calculate safe bounds for the possible timing due to scheduling. All possible and allowed schedules, activation and execution variations are covered to calculate worst-case boundaries of e.g. the response times of functions. Furthermore, performance indicators like the possible utilizations of resources, the worst-case filling of queues and especially the worst-case response times on end-to-end chains for data processing (event chains) are of interest. There are multiple general methods to conduct a schedulability analysis. The first method was developed to calculate utilization-bounds, which is required to answer the question whether process executions always finish within a certain period of time [LL73]. The response-time analysis [JP86] allows to calculate worst-case response times for processes based on a fix-point calculation. The analysis was initially defined for fixed-priority scheduling and a periodic occurrence of events. Later on its scope was successfully extended to analyze period task sets with jitter [Leh90] [LSST91], distributed systems [TC94], task-set with offsets based on transactions [PGG97], as well as other task models [Bar03].

Another approach is the real-time calculus [TCN00] [Wan06] which is based on the network calculus [LT01], a method developed for predicting network traffic and message queues in communication systems. Based on the available capacity functions (service curves) and the requested demand functions (arrival curves) of a processing unit, the method calculates the remaining capacity curves after processing the demand and, if applicable, the resulting densities of outgoing activation events (outgoing event curves). The calculated remaining capacity may then be used as available capacity for the next priority level. The outgoing event curve is the incoming activation curve for a following task. The method is based on min-plus and max-plus algebra. Events, processing demand and available and used capacities are described by a set of interval-based functions providing for each interval length upper bounds on the worst-case and lower bounds on the best-case behavior. To allow an efficient calculation of the functions a flexible approximation is provided in [Alb11].

The common goal of all approaches for schedulability analysis is to calculate safe upper and lower bounds for the timing behavior, mainly the response times. The bounds cover all scheduling variations, event densities and execution demands. If the modeling is done correctly, the calculated bounds will not be exceeded in the later system execution. Academic and commercial tools are available for performing schedulability analysis for various scheduling methods and models. An academic example is the tool MAST[GGPD01] provided by the University of Cantabria. It implements the generalized response-time analysis and a lot of extensions developed in the last decades. For the real-time calculus the Matlab RTC toolbox [Matlab], implements the RTC calculations [RTCToolbox]. The tool chronVAL [chronVAL] provided by INCHRON is a commercial analysis tool based on the Real-Time Calculus but it also uses parts of the response-time analysis.

5.2.1 Event chain analysis

Effect or event chains in the context of multi-core system is an important aspect, depending on their type event can trigger different kind of communication pattern in the system. For example, as chains of processes or tasks connected by signals starting at the inputs and ending at the outputs, while being processed sequentially by executable software units *task*, *thread*.

So we need to analyze the behavior of events and triggered chains based on their timing as well as data characteristics.

5.3 Mapping and scheduling of Synchronous Dataflow Applications

There exists a large amount of model-based frameworks that can be used to make deductions at design time for systems described in them [SY15; DB11]. For example, in a general sense, the basic independent periodic task model [Liu00] provides to the designer ability to conclude if, for a certain scheduling algorithm, the system as a task set will meet its deadlines or not. Unsurprisingly, different models are better suited for different challenges of systems design than others. The task model just aforementioned is useful for deriving timing guarantees of a system that is scheduled at runtime. On the other hand, complex data dependencies require refinement of the model into another one of higher complexity, accompanied by a matched refinement of the analysis method [SY15; DB11].

Models of Computations (MoCs) are a family of formal models focusing on input-output relations between data. The relations need not to transform data, for instance, the Synchronous Dataflow (SDF) MoC [LM87; SB17] is a MoC where the input-output relations of interest are communication input-outputs between computing actors. An example where data transformation is considered is the Synchronous MoC [BCE+03]. The ForSyDe methodology itself is based on MoCs so that a system modeled in it becomes an unambiguous executable specification [SJ04]. The trade off of this approach is that timing properties are not part of most MoCs: they are accessible only after a ForSyDe-modeled system is considered mapped into a hardware element, so that then it makes sense to say that the input-output relations occurs for some physical time. For this reason, Design Space Exploration (DSE) within ForSyDe is defined as the exploration of combinations between hardware (platform) elements with the application elements (MoCs/Software) as to produce a final fusion (mapping and scheduling) that respects the MoC relations and timing constraints given; optionally optimizing the fusion for metrics such as energy or throughput [RS17; RMU+18].

Although the tooling built around the ForSyDe methodology, hereinafter referred as ForSyDe tooling, already sports a variety of MoCs to be simulated, including the two examples mentioned; the DSE segment, DeSyDe, currently focuses solely in SDF applications. Moreover, DeSyDe currently supports only TDMA-based buses and Buffer-less NoCs platforms, as platforms that provide analytical bounds in the form of worst case execution times (WCETs) and worst case communication times (WCCTs) for DeSyDe to reason with. Note that these platforms have no middle-ware between the application and themselves, i.e. the result is a bare-metal order-based schedule implementation [RS17; RKUS17; RMU+18]. Figure 5.2 illustrates DeSyDe's DSE flow.

Thus, we define DeSyDe's static analysis methods as ones that check if a feasible SDF application mapping and schedule exists. This schedule encompasses both execution and communication. If such a mapping and schedule exists, optimizations are performed for either

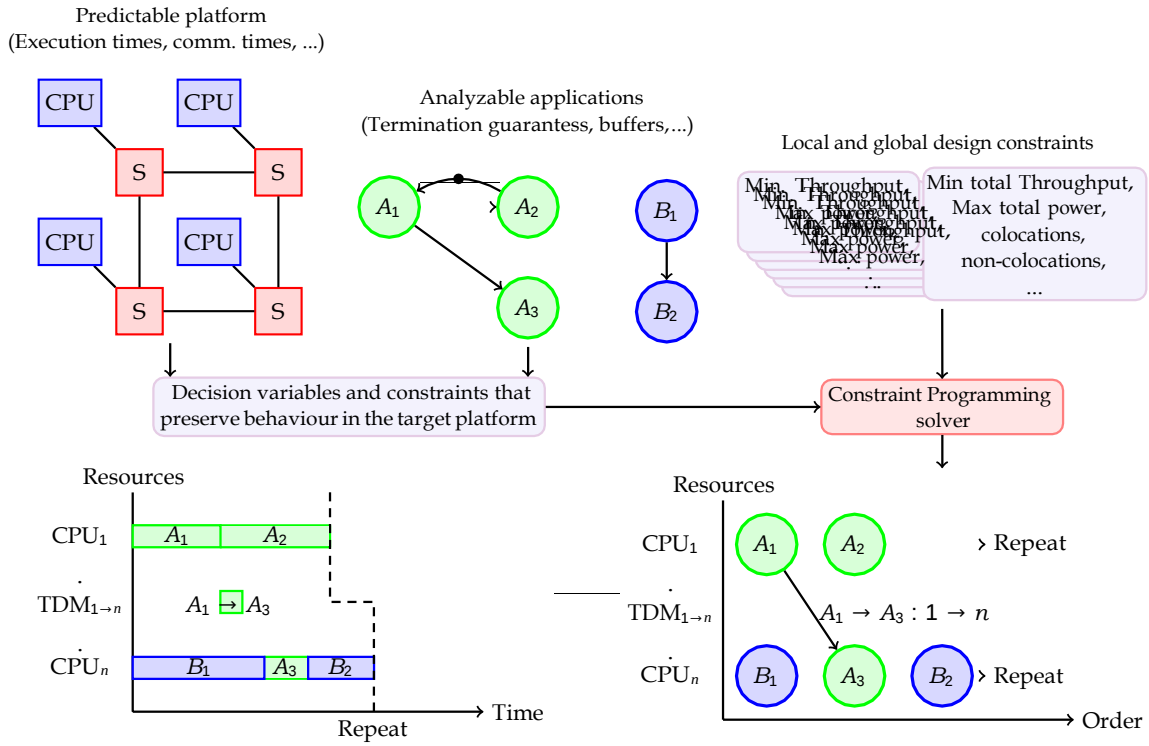


Figure 5.2: DeSyDe's flow overview

power or throughput. Generally, the search space where this map and schedule combination can be found is very large, more so if one considers that almost any formulation of this problem ends up as a bin-packing and job scheduling variation [GJ09], so accordingly, exponential in size. To tackle this challenge, DeSyDe's methods make heavy use of constraint programming as to enable direct benefit of a mature programming discipline that deals specially well with combinatorial problems [RvBW06]. As an example, the mapping and communication of SDF actors into the platform can be partially instantiated as a Hamiltonian cycle problem [GJ09], for which many efficient search pruning techniques exist and are promptly available for use in constraint programming solvers. Additionally, constraint programming solutions are complete ones: to the limit of the provided model, if no solution is found, it means that none exists [RvBW06].

Alternative similar methods include the use of Mixed Integer Linear Programming (MILP) to find the optimal SDF mapping and schedule in terms of latency and throughput [LSGE11]; Nature-inspired meta-heuristics, genetic algorithms in particular [Pim17]; and specific purpose heuristics that search for a mapping and schedule simultaneously [SBGC07].

The end result, as illustrated at the bottom in Figure 5.2, is the mapping of each SDF actor onto the platform's computing units and a order-based schedule for execution and communication. The platform is assumed to provide blocking writes and reads. Blocking write is necessary so that the fastest repeating computing unit does not overflow the communication channel with any other computing unit during runtime. If the platform does not provide such communication features, a time-triggered schedule can be derived from the order-based schedule computed through the provided WCETs and WCCTs.

5.4 Energy Analysis and Minimization

A rudimentary approach for analyzing and optimizing (e.g. minimizing) the energy consumption of an embedded system can be achieved by applying DVFS techniques and assigning a fixed number of cycles an executable should be computed for each voltage mode. The approach described in the following has already been published in [KWF15] and [AMA14].

An approach that aims at minimizing the energy consumption of variable voltage processors executing real time dependent tasks is provided by Zhang et al. [YXC02]. This method is implemented as a two phase approach which integrates

- **Task assignment**: allocating each task to a core
- **Task ordering**: ordering of tasks in due consideration of their constraints and deadlines
- **Voltage selection**: selecting a slower but less energy consuming processor mode in order to save energy without harming any constraints, such as deadlines

In the first phase, opportunities for energy minimization are revealed by ordering real-time dependent tasks and assigning them to processors on the respective target platform.

- On **single processor platforms**, the ordering of tasks is performed by applying Earliest Deadline First (EDF) scheduling. A further allocation of tasks to processors becomes needless, as only one allocation target exists.
- On **multi processor platforms**, a priority based task ordering is performed. The allocation of tasks to processors is determined by a best fit processor assignment.

Once the scheduling is created, there will be time frames between the end of one task and the start of another during which the processor is not being utilized (so-called slacks). These slacks the prerequisites for the second phase, which performs the voltage selection. This phase aims at determining the resp. (optimal) processor voltage for each of its task executions without harming the constraints and eventually minimizing the total energy consumption of the system. In order to determine these voltages, the task scheduling is transformed into a directed acyclic graph (DAG) that is used to model the selection problem as integer programming (IP) problem. Once the model has been set up, it is optimized by a mathematical solver.

5.4.1 Switching Capacitance per Processor Cycle

The *switching capacitance* per cycle (C_u) for a specific task u is typically [IY98] used in energy consumption calculations. Usually, it contains an unique value for each of the executed tasks. However, it also depends on several device related parameters at circuit level, which usually are not part of publicly available material and therefore hard to obtain. Henceforth, we assume a common switching capacitance Cinstead the processor individual switching capacitance C_u. Thus, the constant C represents a linear scale in calculating the total energy consumption (see Equation 5.2). This fact consequently allows to set this value within this method's context to any random value $= 0$ without harming the correctness of the energy minimization process, allowing us to neglect the constant in further calculations.

5.4.2 Adaptated ILP Formulation

The ILP formulation for assigning voltage modes to distributed tasks is listed in Equations (5.1)-(5.7) with D_x describing a node (task) x's start time. This notation is based on the work in [YXC02] and slightly adjusted towards an efficient implementation into the ILP Solver Oj!Algo that is used by App4MC.

$$minimize \quad \sum_{u \in V} E_u \tag{5.1}$$

The objective of the formulation is shown in (5.1) and aims at minimizing the total power consumption of all tasks.

$$subject\ to \quad E_u = C_u \left(\sum_{i=}^{m} N_{u,i} (V_i^2 - V_h^2) + N_u V_h^2 \right) \forall u \in V \tag{5.2}$$

The power consumption E_u for each single Task u is described in equation (5.2). It is the sum of the respective voltages V_i times the number of cycles $N_{u,i}$ during a voltage level i, with N_u being the total number of cycles, V_h the highest available voltage level, and C_u a task's switching capacitance.

$$D_{Out} - D_{In} \leq T_{Con} \tag{5.3}$$

Equation (5.3) ensures, that the path from the first node of a graph D_{In} to it's last node D_{Out}, i.e. a single activation (call, execution) of the application, is executed within the time limit specified by T_{Con}.

$$D_v - D_u - \sum_{i=1}^{m} N_{u,i} (CT_i - CT_h) \geq T_u \ \forall e(u, v) \in E \tag{5.4}$$

Equation (5.4) is used to prevent the start of a successor v before its predecessor u is finished. The sum term represents the amount of time the task u is delayed by being executed at lower voltage cycles, with CT_i being the cycle time[1] at voltage level i, whereas CT_h represents the cycle time for the highest voltage level. The initial task execution time, i.e. the time task u requires if being executed only at the highest voltage level, is represented by T_u.

$$\sum_{i=1}^{m} N_{u,i} (CT_i - CT_h) \geq 0, \ int, \forall u \in V \tag{5.5}$$

$$D_u \geq 0, \ N_{u,i} \geq 0, \ int, \forall u \in V \tag{5.6}$$

$$\sum_{i=1}^{m} N_{u,i} \leq N_u \ \forall u \in V \tag{5.7}$$

Finally, (5.5)-(5.7) are used to constraint the values for the tasks duration (5.5), constraint start times D_u as well as cycles for each voltage level $N_{u,i}$ to positive integer values (5.6), and prevent $N_{u,i}$ from exceeding a task u's total number of cycles N_u (5.7).

[1] Time required for a single cycle

5.5 Spatial Segregation Analysis for Software Deployment

In a typical model-based development process, a major part of the system development is the decomposition of the top-level functional architecture into smaller components [SDP12]. For safety-critical systems, during functional decomposition and logical component design not only partitioning of functionality takes place, but also measures are developed and introduced into the design that ensure system safety. These so-called safety mechanisms are additional functionality to detect and mitigate failures during system operation and maintain a safe state. In order to be effective, sufficient *independence* of the safety mechanism from the primary component has to be ensured. A common approach to protect the system against failure of one hardware component is to duplicate the entire functionality. In this case, independence means that the software implementing the primary and secondary system function must not share certain hardware resources. For example, they must not be executed on the same core or share the same memory.

In the ARAMiS II project, OFFIS has developed an analysis for this kind of spatial segregation requirements between software components. The analysis prototype serves two tasks:

1. It checks if a set of spatial segregation requirements is consistent wrt. a given hardware architecture, i.e., if there exists a mapping of components to hardware that satisfies all segregation constraints.

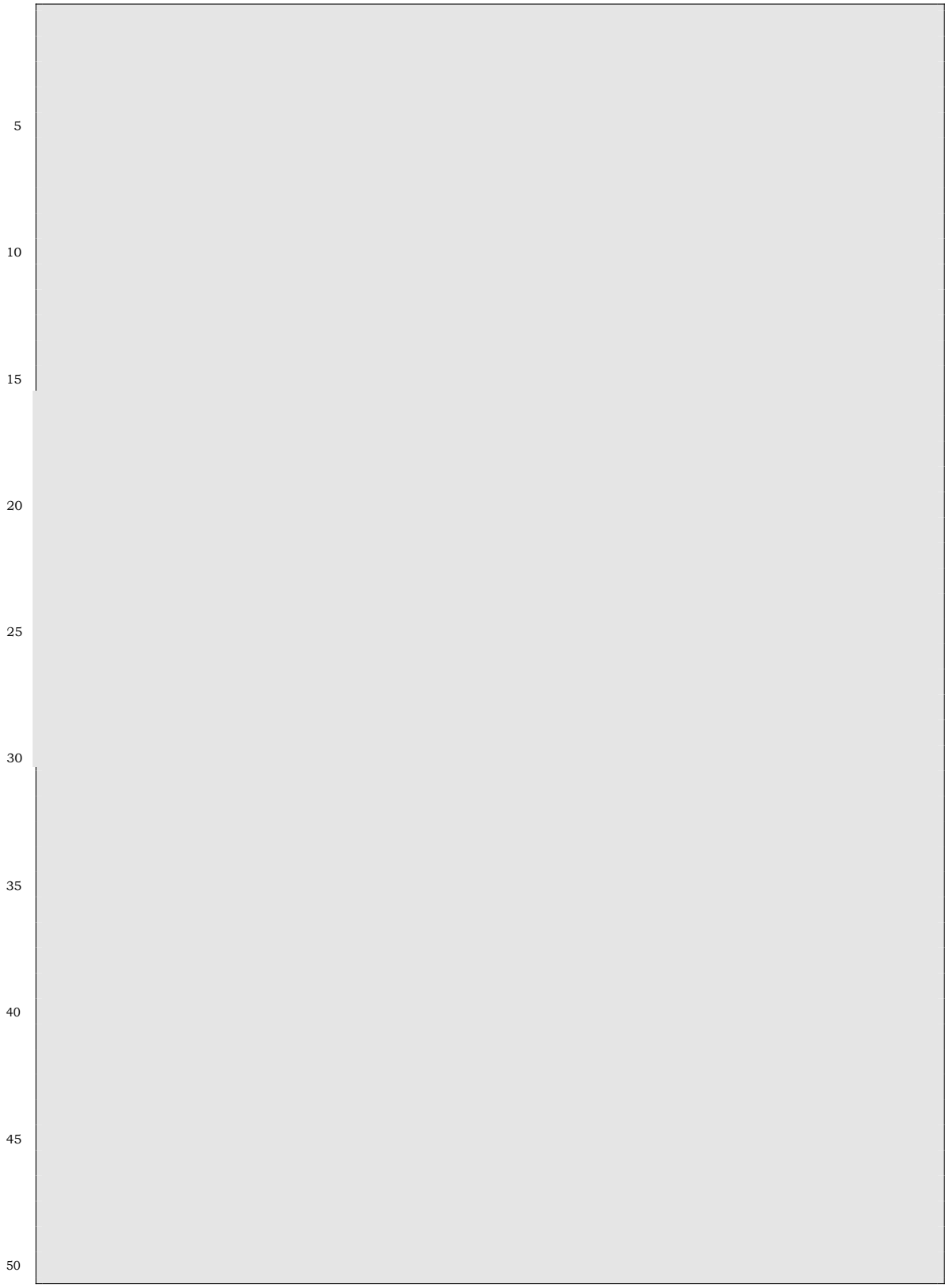
2. Given a concrete software model and an allocation of the software to hardware, it checks if the segregation constraints are satisfied.

In the following, the analyses are described in detail.

5.5.1 Methodical Background

As stated above, during functional decomposition, segregation requirements are introduced that describe the needed independence between components. During development of the logical architecture, it is typically known which kind of hardware resource a component needs (e.g., shared memory and communication resources), and which resources of the needed type are available on the target platform. But it is usually not known, to which part of the hardware the component will be mapped finally. For example, a function might require access to a specific type of communication port. On the target hardware, three ports of that type may exist and two cores where the function may be mapped to. Depending on the deployment configuration, the function will finally use one of each. Additionally, accessing the communication port from the core may be possible only via shared hardware access paths on the board that need to be considered when assessing independency.

To overcome this problem, service oriented design principles [BBP+18] are used. The idea is to allow access to hardware resources only via services of the operating system. In other words, the access to resources is encapsulated in that services. Services and resources may be concrete or abstract. When defining the logical architecture, it is described which abstract services are required for each function. When allocating the logical architecture to the hardware, the abstract services are mapped to concrete services provided by the operating system. The abstract services needed by each function and the amount of resources needed for each call to a concrete service is expressed in a textual language. The language that has been implemented in the RTAna tool suite during the ARAMiS II project [The19a] is a variant of the pattern-based RSL [RSRH11; BBB+11] that has been developed and improved in different projects (i.a. SPES 2020, CESAR, MULTIC, ARAMiS II) over the last decade.



Listing 5.1: Pattern language used by the RTAna tool suite

The *reaction pattern* is the basic pattern to describe the behavior that is visible at a component's ports. It relates the input and output of a component in form of an if-then relation on timed events. The *periodic event pattern* describes recurring events and is mostly used to describe events coming from the environment (i.e., assumptions of the system under design). The semantics of these patterns is close to that defined by the TIMEX extension of AUTOSAR [RSRH11]. The *service request patterns* are variants of the reaction and periodic patterns that describe which services are requested by a component, and how many requests are made. The *resource usage patterns* in turn are used to describe the amount of resources needed for one service request. The *spatial segregation pattern* finally is used to describe spatial segregation requirements.

As an example, consider the hardware architecture outlined in Figure 5.3. The example is taken from [BBP+18] and describes the main board of the "multirotor" research prototype, which is a remote-controlled quadrocopter. It is equipped with a camera and intended to autonomously record soccer games. The board consists of an FPGA that implements two softcores. The quadrodopter's flight control function shall be implemented redundantly on both cores on the FPGA, and a third instance as a backup on the ARM core next to the FPGA. The sensors and the motorboard are connected via I^2C. Figure 5.5 shows the system functions (left) and the services for the I^2C access (right). Each of the softcores (MicroBlaze and Leon are considered to be identical here) a service for accessing I^2C is provided. This service implements wto abstract I^2C service (e.g., one service for normal speed and one for high speed mode). The Flight1 and Flight2 components each have a service specification

1 times I2C1.request occurs and 4 times I2C2.request occurs every 10ms

which states that within each activation of the function – which happens each 10ms – the I2C1 service is requested once and the I2C2 service is requested four times. The I^2C service implementation in turn has a resource usage specification attached to it:

whenever I2C.request occurs then Softcore is used for 2us and then AXI_Bus is used for 3us and then I2C is used for 100us and then AXI_Bus is used for 4us and then Softcore is used for 2us

This specification describes which resources (the softcore, the AXI bus and the I2C bus) are used during each service call. The segregation requirement on Flight1 and Flight2 is specified as

Flight1 and Flight2 shall not use the same resources MicroBlaze, LEON

which states that Flight1 and Flight2 must not both use resources on LEON or both use resources on MicroBlaze.

5.5.2 Analysis Prototype

Input

The analysis input is an AMALTHEA (APP4MC version 0.9.3) model[2]. As not all information that is necessary for the analysis can be modeled natively within AMALTHEA, general-purpose elements – so called *custom properties* and *custom elements* – are used instead. The input model contains at least the following elements:

[2]Documentation available online: https://www.eclipse.org/app4mc/help/app4mc-0.9.3-help.zip

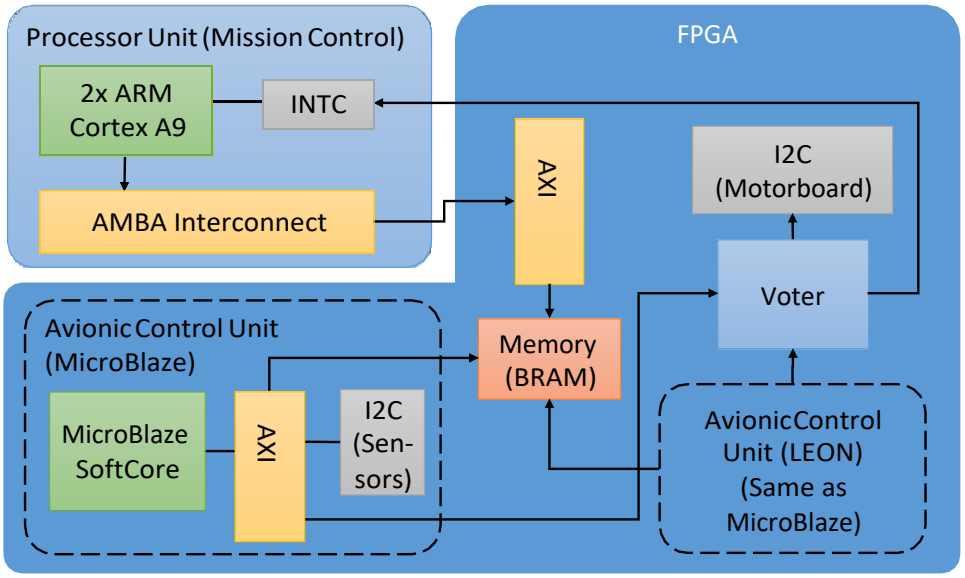


Figure 5.3: Multirotor main board architecture [BBP+18]

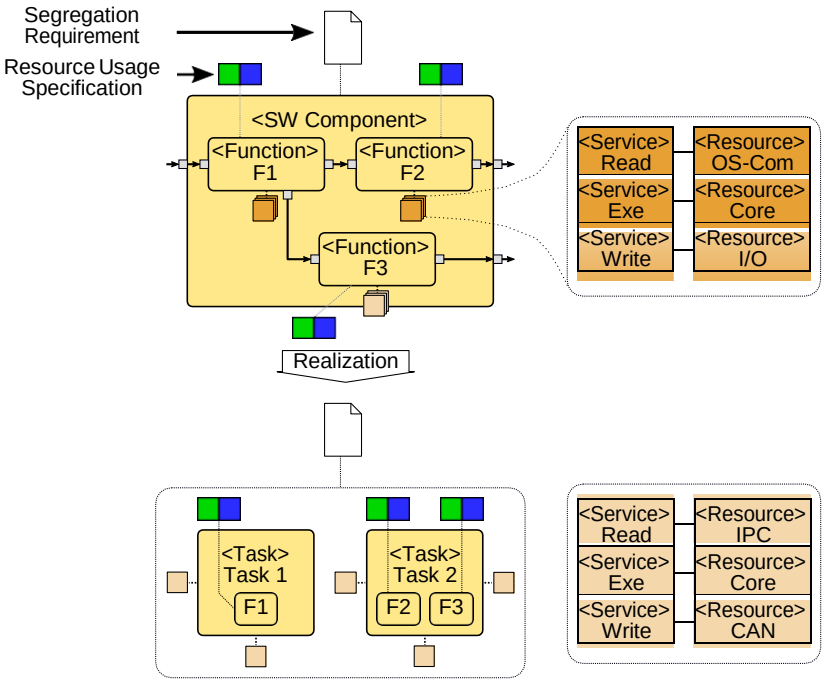


Figure 5.4: Mapping functions with services [The19b]

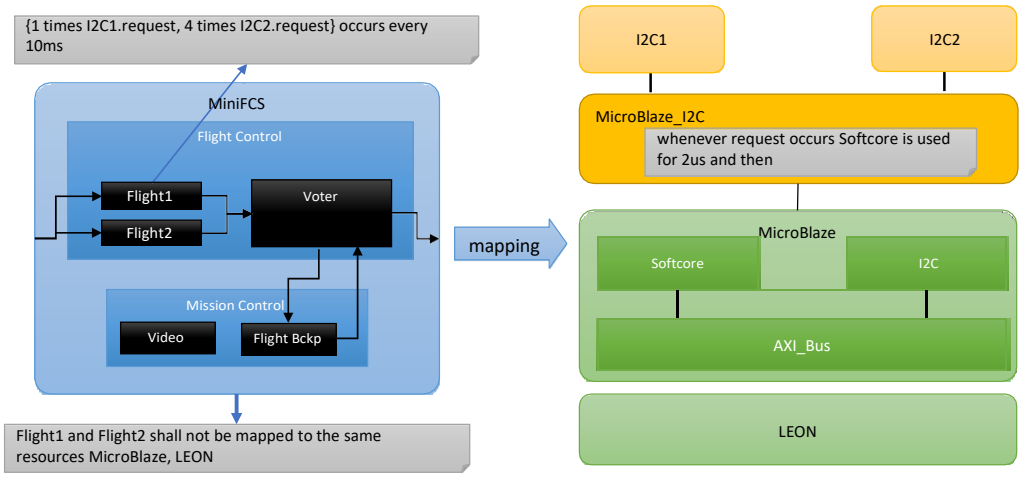


Figure 5.5: Example model for the segregation analysis

- A *Component Model* that describes the logical architecture. The components are equipped with textual specifications in the above pattern language. Custom properties are used to attach the specifications to the components.
- A *Hardware Model* that describes the target hardware for the logical architecture. The hardware elements (cores, buses, memories, …) are interpreted as concrete resources by the analysis.

For the segregation consistency analysis, in addition abstract and concrete services need to be modeled. For the services custom elements are used in AMALTHEA that are attached to the hardware via custom properties.

The segregation check, opposed to the segregation consistency analysis, does not use a service model, but instead directly checks a software to hardware mapping against the segregation constraints. Therefor the AMALTHEA model needs to contain the following information:

- A *software model* describing software tasks that implement the components. The tasks must be linked to the components.
- A *mapping model* that describes an allocation of the tasks to cores, and a mapping of software elements (task bytecode and labels[3]) to memory.

Software integration

Both analyses integrated into APP4MC IDE as a model validation. They can be started from the APP4MC IDE's graphical user interface.

Output

The analysis prototypes generate simple HTML reports. They contain a textual description of the analysis results as well as error and warning messages. The following analysis results are provided:

[3]In AMALTHEA, a label is an abstraction of a memory address

- For the *consistency analysis*: A statement whether a mapping of the components to hardware is possible or not.
  - Result "mapping possible": A candidate mapping is printed into the report that guarantees at least the spatial segregation constraints. Because the analysis approximates the timing behavior, the mapping may still be unfeasible.
  - Result "mapping impossible": A mapping is definitely not possible. As additional information, the analysis produces a minimal subset of the requirements that leads to the conflict.
- For the *segregation check*: For every segregation requirement, a statement is made whether the requirement is satisfied (Yes) or not (No). If the answer is No, a short reason for that is given in the report.

Additionally to the report, the analysis results are presented as markers in the AMALTHEA model editor. Violated/conflicting requirements as well as elements that caused an analysis error are marked with a red X.

5.6 Failure Propagation Analysis

When designing a product, the sooner potential risks can be identified, the more costs can be saved because it is easier to modify a project in its early stages [KG07]. There are several methods for analyzing risks in a system, but all require mature design. This task becomes more important when we refer to safety critical systems.

Developing safety critical systems requires ensuring that the system does not harm people and the environment, even if some system components fail [Lev95; Lut00]. The related warranty process, known as safety analysis, consists of a risk and hazard analysis phase. The purpose of risk analysis is to identify potential risks that may occur during the life of the system and to determine their tolerable hazard rates (THR) or probabilities (THP) [Lev95; GCW07]. A combination of a formally specified hazard condition, along with its probability/tolerable hazard rate is a prerequisite for formulating a safety requirement. More specifically, a safety requirement is formulated as the denial of a hazard condition combined with THP/THR [Lev95]. A comprehensive list of these safety requirements is the end result of the hazard analysis process.

The set of all safety requirements identified in the risk analysis process becomes an input to the risk analysis process. The purpose of this process is to evaluate whether a system design meets its safety requirements. Traditionally, manual methods such as Fault Tree Analysis (FTA) [Sta03; VGRH81] and Failure Mode and Effects Analysis (FMEA) [GCW07] are used to create evidence that the system meets its safety requirements. In addition to these traditional methods, model-driven safety analysis techniques have gained increasing attention from researchers and practitioners [Gru07].

Model-oriented approaches (applied in the design phase architecture) are used to automatically produce fault trees and FMEA tables based on an annotated architecture design specification with information on the failure behavior of architecture components. Example languages for these annotations are: Failure Propagation and Transformation Notation (FPTN) [FMNP94; FM93], Component Fault Trees (CFTs) [KLM03] (see detailed description in Section 5.7), State Event Fault Trees (SEFTs) [GKP05; KGF07], Failure Propagation and Transformation Calculation (FPTC) [Wal05] and the tabular failure annotation of the HiP HOPS methodology [PM01; PMSH01; PPG04].

Within the context of fault propagation, this study aims to research and summarize some works related to static failure analysis. We briefly describe the topics and approaches in heterogeneous systems. We also survey available methods and tools.

5.6.1 Approach

FMEA is a risk assessment tool that mitigates potential failures in systems, processes, projects or services and has been used in a wide range of industries [LLL13]. Like FMEA, FTA is an important approach to fault propagation analysis for engineering systems. Using these techniques, our approach is similar to that presented in [AKB04], as we focus on the static diagnosis of hardware and software components through timeless fault propagation models. Initially, the method will be applied to two heterogeneous systems types: Insulin Pump Systems and a Remote Interface Unit.

Therefore, in the context of the PANORAMA project, we are proposing fault propagation modeling, because the use of static analysis techniques we focus on are FMEA and FTA. In this approach, we will use the APP4MC[4] tool in conjunction with other Eclipse Environment plugins as shown in Figure 5.6.

For failure modeling, we start from the logic proposed by Project DEIS[5], called Open Dependability Exchange (ODE) metamodel[6], which proposes an entity model and supports FMEA and FTA. We adopt these as the formats for the outputs of our static analysis. Moreover, ODE allows defining functions and dependency relationships among them, which map to the component concept we defined for deliverable D1.1. This way, the propagation modeling is performed using ODE as a foundation.

To relate each function with its failure modes, defined from the ODE model to an AMALTHEA component, we will use Capra's traceability function[7]. Our approach should be similar to the work [LL14] that used Altarica Models to build Fault Trees. We also intend to use PlantUML[8] to write Fault Propagation Models just as Capra uses these models to link components. One possibility is to have an individual mapping between the executables defined in AMALTHEA and the functions defined for ODE. The objective is to model the most critical software component/function/failure mode relationships of the system in order to identify each executable affected by a failure directly or indirectly (if propagation occurs).

5.7 Component Fault Tree (CFT) Methodology

5.7.1 Overview

With *Component Fault Trees (CFTs)* there is a model- and competent-based methodology for fault tree analysis [Int06], which supports reuse by a modular and compositional safety analysis strategy. Component Fault Trees are Boolean models associated with system development elements such as components [KLM03; KSA+18; HJZ+18]. It has the same expressive power as classic fault trees, which are described for instance in [VGRH81]. Like classic fault trees, CFTs are used to model failure behavior of safety-relevant systems. This failure behavior, including

[4] https://www.eclipse.org/app4mc/
[5] http://www.deis-project.eu/
[6] https://github.com/DEIS-Project-EU/DDI-Scripting-Tools/tree/master/ODE_Metamodel
[7] https://projects.eclipse.org/proposals/capra
[8] https://plantuml.com/

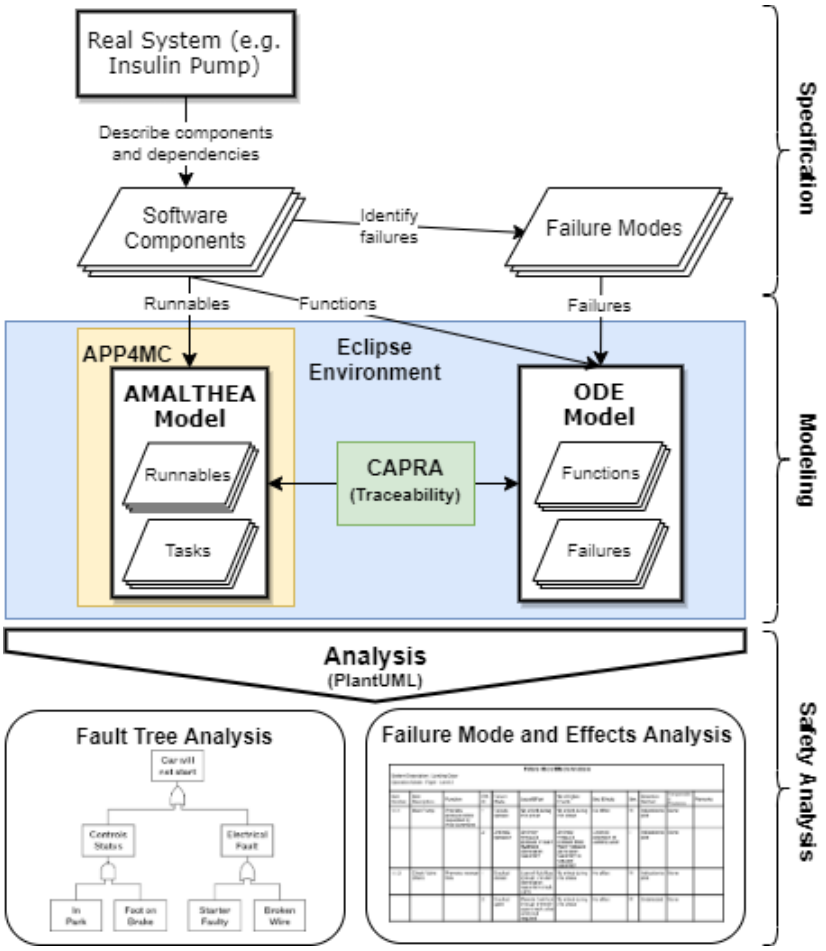


Figure 5.6: Failure propagation modeling approach.

their appearance rate, is used to document the absence of unreasonable risk of the overall system. In addition, it can also be used to identify drawbacks of the design of a system, e.g. by the propagation of failures or by the failure rates.

In CFTs, a separate so-called *CFT element* is related to a component [KLM03; ADH+11]. Failures that are visible at the outport of a component are modeled using Output Failure Modes which are related to the specific outport. To model how specific failures propagate from an inport of a component to the outport, Input Failure Modes are used. The internal failure behavior that also influences the output failure modes is modeled using Boolean gates such as OR, AND and M-out-of-N as well as so-called Basic Events. Basic Events model failure modes that originate within a component. Each Basic Event can be assigned a failure rate, e.g. the *Mean Time Between Failures (MTBF)* or the *Failure In Time (FIT)*. In case of an OR gate a failure propagates if at least one of the inputs is active, while an AND gate propagates failures only if all input failures are active.

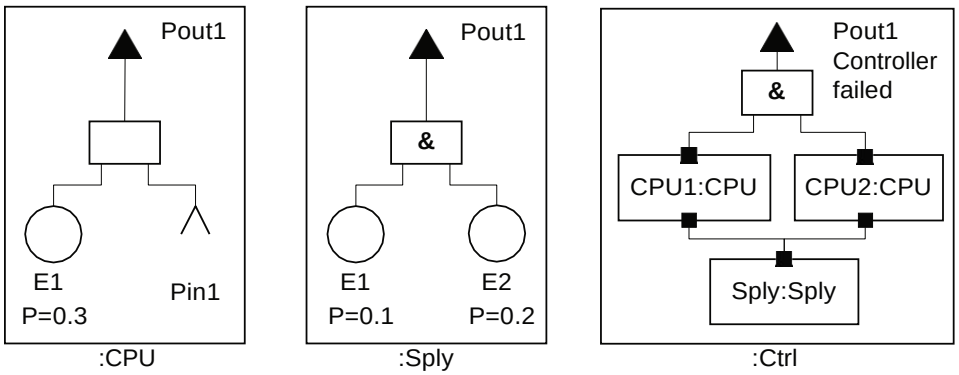


Figure 5.7: Example of a simple CFT [KSA+18]

5.7.2 Example

A small example of a CFT was presented in [KSA+18] (see Fig. 5.7). The example shows an exemplary controller system *Ctrl*, including two redundant *CPU* s (i.e. two instances of the same component type) and one common power supply *Sply* (which would be a repeated event in traditional fault tree). The controller is unavailable if both CPUs are in the state "failed". The inner fault tree of the component type CPU is shown on a separate screen; as the CPUs are of identical type, they only have to be modeled once and are then instantiated twice. The failure of a CPU can be caused by some inner basic event "E1" (the repetition of the ID "E1" in several components is not a problem, as each component constitutes its own name space). The failure of the CPU can also be caused by an external failure cause which is connected via an input port. As both causes result in a CPU failure, they are joined via a 2-input OR gate. The power supply is modeled as a separate component. Let us assume that the power supply is in its failed state if two separate basic failures are present (for example having two redundant batteries). Hence, instead of a single large fault tree, the CFT model consists of small, reusable and easy-to-review components.

5.7.3 Input

Input for the Component Fault Tree methodology is the

- list of hazards identified during the Hazard and Risk Assessments (HARA)
- description of the system architecture in any model-based system architecture description language (e.g. UML/SysML, EAST-ADL, AADL, Capella, etc.)
- information about failure rates of components (e.g. taken form standards such as MIL-HDBK-217, SN29500, etc.) and repair times, etc.

5.7.4 Output

Fault Tree Analysis is a top-down, deductive failure analysis to identify the causes which lead to a system hazard. The output of the Fault Tree Analysis is either a qualitative or a quantitative analysis:

- *Minimal Cut Set (MCS) Analysis*: Qualitative analysis of a hazard which determines the minimal cut set. A cut set is a combination of component failures which lead to the hazard (the top event of the fault tree).
- *Quantitative Analysis*: If the events in a fault tree are associated with statistical probabilities, the probability of the occurrence of a hazard (the top event of the fault tree) is calculated.

The results of these analyses help the prove that the safety requirements are fulfilled by a certain architecture or help to improve the architecture.

5.7.5 Advantages of the Component Fault Tree Methodology

The CFT approach has multiple advantages compared to classic fault tree analysis. It eases the creation and especially the maintenance of safety analysis due to the divide-and-conquer strategy applied when creating component fault trees. Since CFT elements are related to their development artifacts, they can be reused along with them. Moreover, it is possible to automatically compose system-wide failure propagation based on the CFT elements of each component[MZH+16; MBZ+17]. Hence, it is possible to create a library, which contains CFT elements for all system components. Based on this library, different CFTs by can be created by just changing the assembly of the CFT elements according to the system architecture. Moreover, with the concept presented in [ZM18] Markov Chains can be integrated in Component Fault Tree models combined in an arbitrary way.

Since Component Fault Trees provide a relation between the ports (i.e. the interfaces of components) in the system design and the failure modes in the failure propagation specification, it is possible to derive test data or input for the fault injection into simulations [ZH15; RZH+17].

Every CFT can be transformed to a classic fault tree by removing the input and output failure mode elements. The CFT is just another representation of the information in the fault tree. Therefore, the same algorithms can be used for qualitative (minimal cut set analysis) and quantitative fault tree analysis as for classic fault trees.

These advantages of the CFT methodology have been proven in different evaluations [JJH+13; GZOH18]. In addition, several industrial case studies showed the advantages of using CFTs for the safety or reliability analysis of large-scale complex systems in different industrial application domains [MNZ18; KZS18; HJZ+18; ZK19].

6 Dynamic Analysis

This Chapter describes analysis methods, which focus on the dynamic behavior of digital electronic systems. These methods often investigate functional and extra-functional aspects of the developed systems, which depend on the run-time behavior of hardware and software.

This Chapter roughly divides into two general methodologies: simulation-based and measurement based approaches. The contributions in Sections 6.1 and 6.2 describe methodologies, which are based on the simulation framework SystemC as discrete event based simulator. Both also partly use AMALTHEA models as input, but they focus on different technologies, as in section 6.2 virtual integration tests are performed and in section 6.1 model transformations are utilized. The Sections 6.3 and 6.4 present commercial simulation tools and both mainly focus on timing and schedulability analysis. Both can work on a variety of different input models and different abstraction layers.

The last two Sections 6.5 and 6.6 describe approaches, which use actual hardware. In section 6.5, a fault-injection methodology is described, which allow the analysis of the resilience against hardware and software faults. The section 6.6 describes an approach based on timing measurements on actual hardware to analyze timing and schedulability characteristics of systems.

6.1 SystemC Simulation

The design and development with a model-based process creates the need for early evaluation of the created models. Depending on the abstraction level and underlying semantics of the meta-model an evaluation based on simulation can be beneficial, as such executable specifications can serve many demands. In the first design stages, a simulation can serve as an early tool to investigate functional aspects and to give hints for design decisions. For later design phases, the same simulation can be used to test existing implementations.

In the Electronic Design Automation (EDA) community, *SystemC* is one of the de facto standards for system-level simulation and the development of virtual prototypes. *SystemC* is defined in the IEEE 1066 standard [SystemC12] as a C++ class library, which includes modeling primitives and a discrete event-based simulation kernel. The modeling primitives can be divided into the following parts: structural, behavior, communication, data types.

Structural Modeling

The main model primitive of *SystemC* is the class sc_module. A module can contain other modules to create a hierarchical structure. Moreover, a module describes the interface for communication to other modules with ports, where a port describes the exchanged data and the channel, which should be connected to the port. Besides the functional properties of such structures, SystemC also serves a hierarchical naming system for modules, to allow an easy debugging while the execution of simulations.

Behavior Model

The behavioral description of *SystemC* modules is organized in processes. There exist multiple types of processes, but the main are sc_method and sc_thread. A sc_method have the semantics of most Register-Transfer-Level models, like VHDL or Verilog, and are functions, which are completely executed as a consequence of an event occurrence from the set of predefined events of thesc_method's sensitivity list. Contrary to that, a sc_thread normally do not have a predefined sensitivity list, but is executed once at the begin of the simulation, and can dynamically interact with blocking communication calls, await events with the wait call, or uses a wait call with a time object to release the execution thread to the simulation kernel. Since a sc_thread is executed only once, a common pattern is to use an endless loop reacting on dynamic inputs, which is also a common pattern for modern software. The *SystemC* reference simulation kernel employs a cooperative scheduler, which executes only one process at the moment, which allows the simulation of many parallel executing processes, without the need for thread-safe algorithms. The obvious downside is the missing exploitation of parallel computing resources of modern computers. Depending on the abstraction level, models may mix the usage of both process types.

Communication

The communication between sc_modules is abstracted in communication sc_channels. Predefined channels exists for the semantics of simple traces (sc_signal) with arbitrary width or buffering FIFO channels (sc_fifo). To describe more complex communication protocols, e.g. to model bus accesses and arbitration, *SystemC* offers *Transaction Level Modeling (TLM)*, which is often used for high-level system models. In general, a transaction describes the information to issue an access to a memory, e.g. address, length, read or write action. Moreover, *SystemC TLM* also defines a standard protocol state machine and is generically designed to map easily to existing protocols, like PCI or other on-chip Buses.

6.1.1 Application of SystemC

Due to its generic concepts, *SystemC* covers a variety of abstraction levels transfer level, for instance, register transfer level, describing the signal flow between registers and logical gates, but also abstract models of the system behavior. *SystemC* is for example often used for the simulation of virtual prototypes, where instruction set simulators execute the actual target code of a platform and hardware modules are described in *SystemC*.

The nature of *SystemC* as C++ library offers the power and degree of freedom, that a general-purpose and multi paradigms programming language supports. The downside of that is less analyzability and tool-assisted semantic guiding of modelers. Therefore, it is beneficial to use more restricted and domain-specific meta-models to describe an embedded system and then transform the model to *SystemC* for simulation. There exist many system-level design meta-models, which cover many aspects of the developed system. The paper [NSTW04] describes a methodology for the metamodeling language *UML*, which transforms *UML* models, especially behavior describing state machines, to *SystemC* and simulate them. In [PMPV10] a quite similar approach with the *MARTE* profile of *UML* is proposed. A similar methodology was proposed in [Abd16] which used *SysML*, another offspring of *UML*, and use simulations for verification purposes. The work in [HFK+07] describes another approach, which uses *SysteMoC*,

a template library based on *SystemC*, for the modeling of streaming applications. The resulting simulation models are used in a guided design space exploration as performance evaluation.

Although a *SystemC* simulation model does not offer intrinsic formal methods for model checking, there are several methodologies for analysis. Firstly, all source code methods for the C++ language are also applicable for *SystemC*. Moreover, methods exploiting the object-oriented aspects of C++ or respectively *SystemC* class library are beneficial. Secondly, simulation-based statistical model checking, for instance, proposed in [NLQ16], can also be applied, if the model as statistical-based execution semantics.

The *AMALTHEA* meta-model, which is one of the proposed input models within this project, allows an extensive description of many aspects of heterogeneous systems, as it includes representation for hardware and software components and corresponding models for the mapping of applications. The software model of *AMALTHEA* describes most of the application-specific behavioral aspects of the designed system. Although the meta-model does not describe explicitly a formal execution semantics for all components, it is easy to derive from the meta-model description. The hardware model of *AMALTHEA* represents the hierarchical structure of the underlying system. The HwStructure describes the system hierarchy including its interfaces to other structures. It can contain HwModules of the type ProcessingUnit, Memory, Cache, HwConnection, ConnectionHandler and other HwStructures. The interfaces between all HwModules are HwPorts, which allow a one-to-one connection via HwConnection. The ConnectionHandler are used to model n-to-n semantics, e.g. in communication buses. Other important aspects of the *AMALTHEA* meta-model are the OS-model, which describes the scheduling of software model elements, and the mapping model. The latter is used to describe the connection between hardware, software and OS-model.

After a comparison of *SystemC* and *AMALTHEA*, it can be recognized, that the hardware model has obvious similarities. This makes *AMALTHEA* a suitable candidate for a model transformation to *SystemC* and subsequent simulation. The simulation can then be used to investigate the given constraints within the *AMALTHEA* model, like latency requirements. As the *AMALTHEA* model also involve statistical distribution within the modeling elements of the software model, suitable statistical methods for the *SystemC* model need to be developed. A *SystemC* simulation may also be used to create traces, which include exemplary event chains, e.g. of software task state changes, which can be used for further analysis methods. Therefore, suitable tracing formats need to be defined.

6.2 Integration Testing for Timing Requirements

During development of safety-critical embedded systems, the system functions are successively decomposed into more elementary functions which are finally to be implemented in software and hardware. In this section, analysis approaches and tools are presented that OFFIS has developed in recent research projects and that help in validating functional decompositions wrt. timing aspects.

6.2.1 Methodical Background

In contract-based design [Ben+18], a component is specified by so-called contracts. A contract consists of two parts, the *assumption* and the *guarantee*. The assumption describes some aspect of the environment in that the component shall operate, and the guarantee describes in the what the obligations of the component are in such an environment. Contracts may describe functional

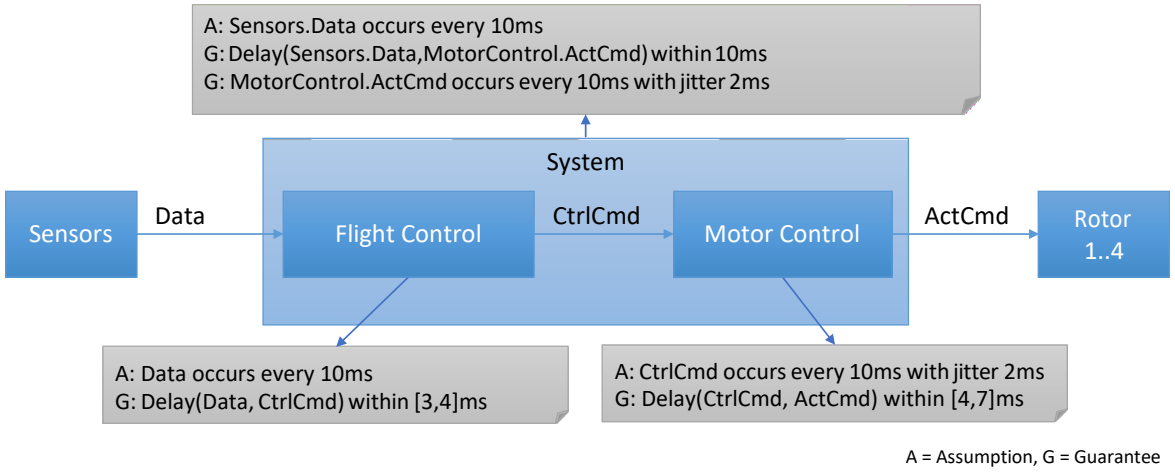


Figure 6.1: Example: Functional decomposition with timing contracts

and non-functional aspects of the systems. In this section, especially timing properties are considered.

During functional system design, the system functions are successively decomposed into more elementary functions. The functions on lowest level are finally implemented into software tasks. In a contract-based development process, each of the functions is described by contracts. As an example, consider the top-level decomposition of the flight function of the quadcopter example from section 5.5. It is shown in Figure 6.1. The behavior of the system functions is specified as timing contracts using the pattern language from section 5.5. The assumptions describe the events that are expected at the component inputs, and the guarantees describe which events shall occur in response to these inputs. In sum, the contracts at the *Sensors*, *Flight Control*, and *Motor Control* components describe an event chain from input to output of the system. The contract at the system component specifies that this event chain may not exceed 10ms. Virtual integration testing can be used to check – based on the contracts – if the decomposition of the system function into Flight Control and Motor Control is correct. Formally this is done by checking if the contracts at the sub-level components together imply the contracts of the top-level component. More information on the formal background of virtual integration testing can be found in [Ben+18; DHJ+11].

When implementing the system functions in software, it needs to be verified that the software also satisfies the timing properties specified in the functional architecture. In ARAMiS II, OFFIS has developed a satisfaction check for timing properties that also considers the effect of fault handling. Consider again the quadcopter example. As it has been introduced in section 5.5, the quadcopter's flight control function relies on a operating system service to read sensor data via I2C. Because the data transmitted via I2C may be corrupted, it is protected with a check sum. Figure 6.2 shows the control flow of an implementation of the flight control function (more precisely the part that acquires data from the sensors) and the system service for accessing I2C. If the checksum of the data reveals a transmission fault, the service is called a second time in order to request the data again.

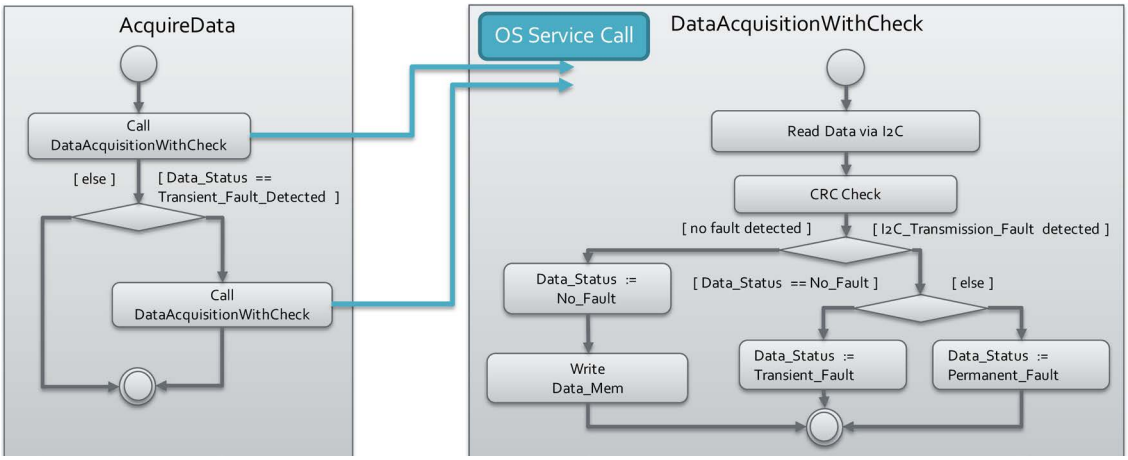


Figure 6.2: Example: Service call with HW/SW safety mechanism

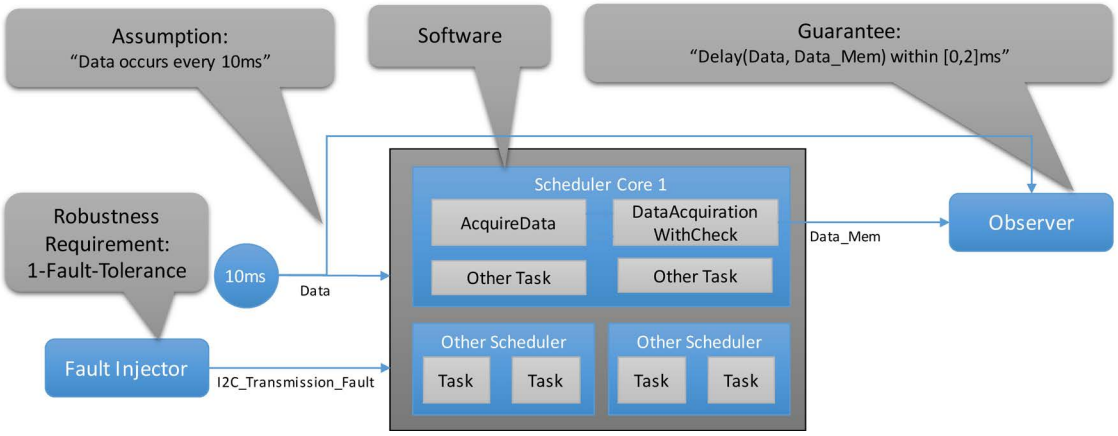


Figure 6.3: Analysis model for satisfaction check

6.2.2 RTAna2

In the ARAMiS II project, a virtual integration analysis and the satisfaction check outlined in the last section have been prototypically implemented and integrated into the APP4MC platform. The analysis relies on a successor of the $RTAna_2$ model checker [SRGB13]. The analysis approach is to model the top-level assumptions and sublevel guarantees as a task network that simulates the behavior specified in the contracts. The guarantee of the top-level contract is instead implemented as a (virtual) observer that monitors violations of the guarantee. The model checker uses a variant of stopwatch automata to model task networks. In some cases it is possible to achieve complete analysis results by unrolling and examining the complete state space. The satisfaction check uses the same model checker to simulate an AMALTHEA software model. The control flow graphs of the software tasks – that can be modeled quite detailedly in AMALTHEA – are annotated with information on fault handling. Finally, a fault injector component is inserted into the analysis model.

Input

Input of the VIT and the satisfaction analysis is an AMALTHEA model that contains at least a component model as described in subsection 5.5.2. The satisfaction check additionally requires an AMALTHEA software model and a mapping to hardware that contains stimuli for all the software tasks. The analysis interprets write accesses to AMALTHEA label elements as events on the component ports.

Integration

The VIT and satisfaction check are integrated into the APP4MC platform the same way as the OFFIS segregation analysis (see subsection 5.5.2) is.

Output

The VIT and satisfaction check produce HTML reports in the same way as the segregation analyses (see subsection 5.5.2). The result of the VIT and satisfaction check is either "pass" or "fail" for every (top-level) guarantee. Additionally some statistics are reported such as the run time of the analysis and the state space size. It is also reported if the state space has been completely unrolled. In this case, the analysis results are complete.

6.2.3 MULTIC Tooling

The virtual integration test developed in the MULTIC and MULTIC Tooling projects implements contracts in a similar way to the ARAMiS II VIT prototype. The important difference is that in the MULTIC approach SystemC is used for simulation.

Input

The input of the MULTIC Tooling VIT analysis is a Papyrus SysML component model with requirements. The SysML requirement elements contain contracts that are expressed in a pattern language that is close to that presented in section 5.5, but more expressive. An example taken from [DEG+19] is shown in Figure 6.4

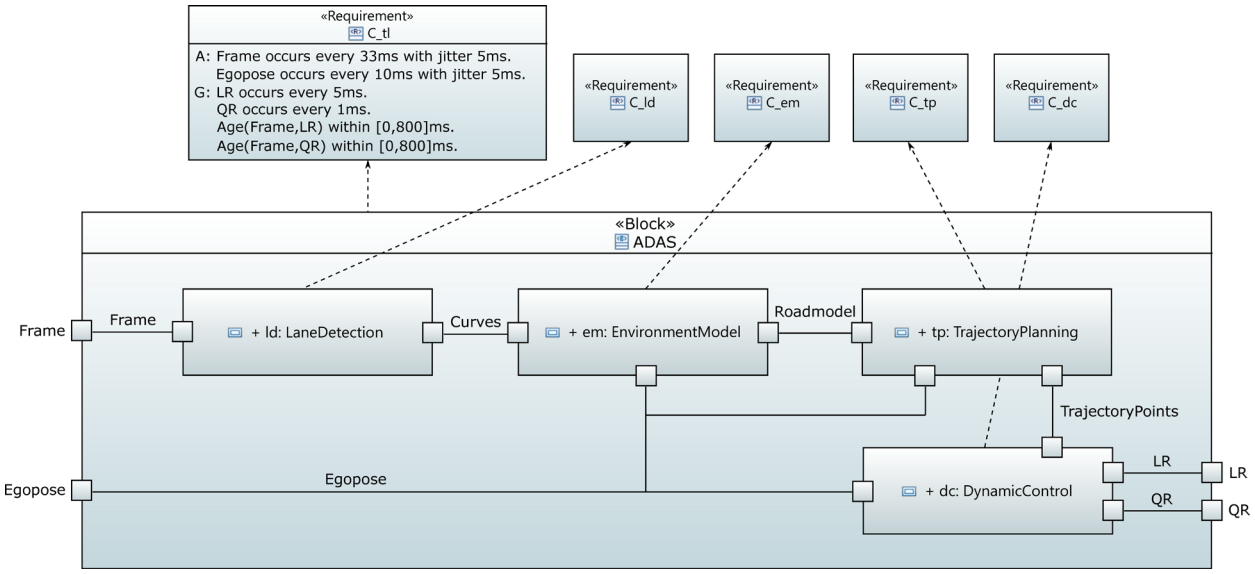


Figure 6.4: Example SysML model with requirements (taken from [DEG+19])

Integration

The analysis can be started from the Papyrus IDE's graphical user interface. An overview on the MULTIC Tooling architecture is given in Figure 6.5.

Output

The MULTIC tools display the analysis results as annotations in the Papyrus model editor. Traces for the simulation runs are also provided.

6.3 Timing Simulation and Evaluation

The Vector Timing Architecture Tool Suite (TATS) [Vec20] incorparates a model-based, event-based simulator and evaluator. The simulator uses a Monte-Carlo based Discrete Event Simulator to generate an event trace out of a given system model. The TATS then provides the possibility to evaluate the timing behavior based on that trace (either generated from the simulation or measured from a tracing hardware). The simulator mainly considers homogenous processor platforms. There is some support for heterogeneous processor cores, but specialized/custom hardware like GPUs, ASICs, or FPGAs is not supported.

The TATS can be used for the following purposes:

- In-depth evaluation of system and component timing behavior, as well as resource consumption
- Evaluating hardware resource consumption of the application software and the operating system
- Performance analysis and evaluation of different hardware platforms and software designs
- Shared resource interference and cause-effect analysis

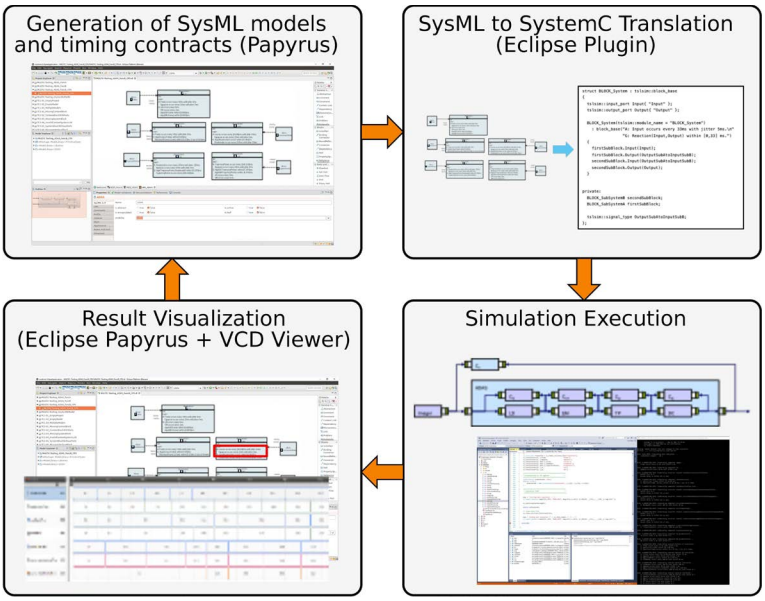


Figure 6.5: MULTIC Tooling architecture (taken from [DEG+19])

- Validation and comparison of design decisions in an early phase of the development process

The key features of the TATS are:

- Detailed simulation of single- and multicore architectures using generic or detailed vendor specific processor and operating system models
- End-to-end timing simulation and analysis of multicore and multi-processor systems
- Evaluation of the performance, resource contention, and caching effects for different hardware architectures, as well as operating system overhead
- Analysis of different multicore synchronization and resource sharing mechanisms
- Ability to define a range of requirements and evaluation criteria for the in-depth analysis of event-chains, processes, runnables, and software components

Various (multicore) scheduling algorithms are supported, including:

- OSEK (fixed-priority)
- AUTOSAR (fixed-priority)
- EDF (Earliest Deadline First)
- Linux (deadline based, priority based, and fair scheduling)
- *etc.*

The TATS interfaces with external tools by importing and exporting model files, including AUTOSAR description files, OSEK implementation language (OIL) files, AMALTHEA model files, and ASAM MDX files. The interfaces are used for creating the Timing Architecture Model (TAM), which is used as input model for scheduling simulation. Likewise, event traces generated outside of the simulator can be imported into the TATS through those mentioned file formats (*e.g.*, BTF).

6.3.1 Development Process

The simulator is used during the schedule validation/verification step to evaluate timing requirements and the typical system behavior with the given scheduling parameters and deployment. The result can be visualized via Gantt charts, report tables, and others.

6.3.2 Input

The following input artifacts are considered by the TATS:

Hardware cores, frequencies, memories, networks, *etc.*

Operating System scheduling, OS overhead, events, semaphores

Application Software tasks, interrupt service routines, functions, function execution tree, data variables and data communications/accesses, OS task configurations, inter process activations, execution times as fixed values or as distribution functions

Architecture system description, software components and compositions, ports

Mapping deployment and communication information

Stimulation activating events specification from external sources

Requirements timing requirements such as deadlines, end-to-end duration of event-chains, *etc.*

6.3.3 Output

The following output is generated by the simulator and evaluator:

Execution Trace as Gantt chart or BTF format

Evaluation results various metrics and statistics in Gantt charts, tables, histograms, bar charts, *etc.*

Reports configurable result summary documents in XML or HTML format

A typical simulation output in the TATS is depicted in Figure 6.6. Here, task executions in a Gantt chart, evaluation results of requirements, metrics, and the system load is visualized.

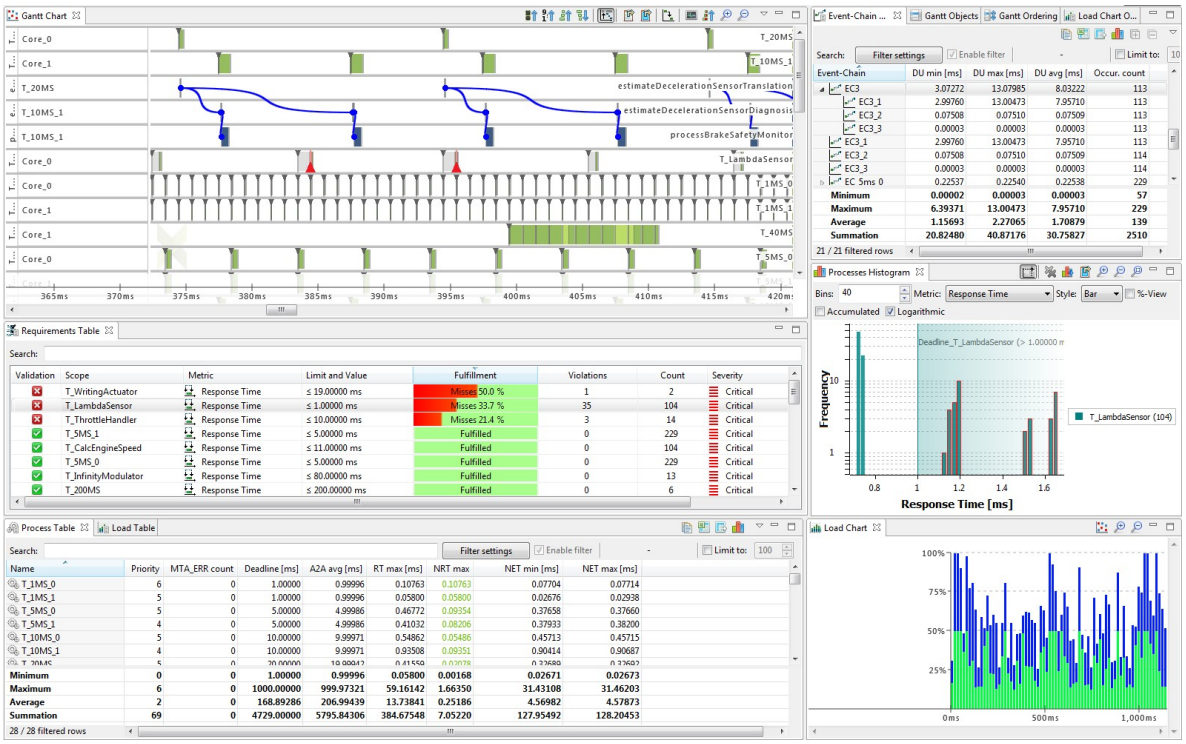


Figure 6.6: Simulation results in the TATS simulator view.

6.4 chronSIM

6.4.1 Simulation

A simulation is meant to recreate a dynamic process in a system with the help of an experimental model in order to obtain knowledge about the behavior of the real system with respect to this process. In comparison to performing experiments with the real system, simulation can offer several advantages like easier accessibility of complex system behaviors, reduced time and cost efforts, enabling of frontloading or in-creased safety at the expense of system abstraction.

The simulation tool chronSIM [chronSIM] is a multiple resource simulation tool with the strong focus on timing. The model consists of functions, interrupts and their interconnections and communications, their mapping on multiple ECUs which can be of different type and have different operating systems or scheduling. chronSIM allows a timing accurate simulation of the model, taking the operations systems, scheduling and even the drift of clocks for the different ECUs into account. The model allows hierarchical scheduling with some basic schedulers as well as complete operation systems with their timing and scheduling specific specialties, all already built into the simulator. The models for chronSIM can be abstract description models including the elements above (and a lot more). Such a model can be enriched and combined with C-code describing functions, tasks and their communication. Timing relevant information is added to the C-Code by various specific macros. The functions, tasks and messages defined in the code are connected to the abstract model so that the simulation of the model can decide when a function is activated and executed (in the context of the simulation time). The code of a function decides what is done, thereby considering how long the execution of the function will take. The code can be used as a modeling language allowing a large flexibility, or implementation code can be executed within the simulation. The execution is functional-aware. Hence, the complete functionality defined within the C-code is executed at the appropriate simulation time. With this approach non-functional models can be combined with partially functional models or full functional models. The simulation tool chronSIM allows to model different clocks for the different resources in the system. So the stimulus and the execution time does not necessarily depend on a global time base but on a local time base. The time bases can be correlated to each other, especially there can be an offset, drift, etc. between different clocks [AADG12]. For example the FlexRay bus has its own time base which is derived from the different time bases of the ECUs connected to the bus. Such systems can be modeled accurately and it allows finding effects originating from not completely synchronized time bases.

6.4.2 Optimization

The optimization is based on methods for timing evaluation with means of simulation and schedulability analysis. The special characteristic for the optimization is to generate the new candidate solutions based on the existing results from a timing analysis or timing simulation using the already calculated knowledge of the system. It includes functions to re-investigate a certain change of the candidate solution (move) based on the already calculated evaluation results of the candidate solution. In case the move is selected due to this estimation of the changes a complete simulation or analysis is performed to take all dependencies into account. The resulting accurate evaluation result will then be finally the base for the heuristic to either accept or reject the move and the resulting candidate solution. Some of the potential optimization goals are listed below:

- Calculation of realistic values for the optimized timing behavior like response times and jitter values
- Calculation of suitable scheduling parameters like priorities, message priorities, selectable offsets
- Calculation of a meaningful distribution and execution order of runnable entities (functions) on tasks.

One possibility to solve the optimization problem is to use specialized heuristic algorithms designed for the specific problem. Candidates are heuristics to generate a good static schedule or to distribute priorities. Another possibility is to use meta-heuristics. That are general optimization algorithms designed to solve optimization problems. Candidates are algorithms like Genetic Algorithms, Simulated Annealing, Tabu-Search, Ant-optimization [AntColonyOpt] and so on [Mic10]. These optimization algorithms follow in general the same scheme. They starts with one or more given or generated initial candidate solutions and then have the following general steps:

- Evaluation of the candidate solutions. They are evaluated resulting in some kind of comparable quality measurements.
- Comparing the evaluated quality with the quality of other candidate solutions or the previous candidate solutions.
- Selecting candidate solutions to proceed.
- Modifying these candidate solutions in the expectation to improve them.

The meta-heuristics can be divided into two groups, round-based and pool-based heuristics.

Round-based Heuristics

In a round-based heuristic one candidate solution is considered in each step. The candidate solution is modified. The modification can be for example by changing one certain parameter like a priority, by switching the position of two tasks, changing an offset or other things. The group of candidate solutions which can be reached by one move from a certain candidate solution is called neighborhood. The modified candidate solution is than evaluated and compared with the previous solution. In case the evaluation detects a progress, the move will be accepted. Otherwise, depending on the heuristic, another move in the neighborhood is evaluated. A basic round-based heuristic is hill-climbing [HillClimbing] It accepts a move when it has a better evaluation. The heuristic leads to a static progress and to a locally optimal solution. The problem is that a local optimum is not necessary globally optimal or even near a global optimum. So it is necessary for heuristics to accept, from time-to-time, also candidate solutions which do not directly lead to an improvement. Simulated Annealing therefore accepts also such back-stepping moves with a certain probability. The key-point is that this probability depends on the progress of the optimization. It is high in early phases of the optimization to allow a more global search and is lower in later stages to step to an (local) optimum at the end. It relies on the assumption that good local optima have a large foundation on good closely related solutions (same as with high hills in the mountains). So it is very likely to end at good local

optimal solution. Tabu-Search solves the problem with local optima by allowing back-steps in cases a local optimum is reached [Tabu]. To avoid that the optimization returns immediately to the same local optima, the move is put into a list of not allowed moves. The heuristic depends on the assumption (and a corresponding model) that local optima are not far from each other. chronOPT part of the INCHRON Tool-Suite [INCHRON] that provides a framework for optimization uses Tabu-Search meta-heuristic . There are a lot of other round-based heuristic. Depending on the heuristics either one modified candidate solutions is considered or several moves out of the neighborhood.

Pool-base Heuristics

Some heuristics are pool-base (genetic algorithms, ant optimization), considering in each round a set of different candidate solutions. In one step of the optimization, the best candidate solutions of the pool will be selected as the base solutions for the moves. Other candidate solutions with less success will be removed from the pool. Genetic algorithms uses two kind of moves, modification and recombination. The modification changes one or a few parameters of one candidate solution. The re-combination tries to combine parts of two different (good) candidate solutions [Mic96]. The challenge for many pool-based heuristics is to develop an adequate modeling which takes advantage of the special strategy of the heuristics. For example for genetic algorithms a modeling is necessary which supports a meaningful re-combination operator. The resulting candidate solutions need to include meaningful characteristics of both previous candidate solutions. A problem with pool-based heuristics is that for one step of the optimization many evaluations are necessary. One step for a pool-base heuristic with 50 active solutions in the pool would require as much computation time as 50 steps of a round–base heuristic. But the pool-based heuristics might have advantages if it is possible to parallelize and distribute the computation on a grid-computer.

6.5 Fault Propagation Analysis for Hardware

In the era of autonomous driving, automotive software is growing in both size and complexity. Simultaneously, the unremitting effort of semiconductors down-scaling its feature sizes following the well-known Moore's Law to diminishing design margins and stringent power constraints. This orientation lead to more dependence on commercial off the-shelf (COTS) hardware [GL08; OM19], that provides high performance on one hand, whilst rising their sensitivity against random hardware faults due to external causes such as radiation effects or electromagnetic interference [Bor05; Bau05].

Dependable systems include attributes such as reliability, availability, safety and security. The design of dependable systems and software rely on the systematic scrutiny of potential faults, their subsequent effect and the countermeasures of detecting and recovering them. Fault injection (FI) is widely adopted dependability assessment technique that ISO26262 strongly recommend to validate the functional and technical safety mechanism are properly achieved [OLSM18].

PyFI (Python backend for Fault Injection), is a fault injection mechanism utilizing *iSystem iC5000 on-chip Analyzer* to inject faults to the components of application at microarchitectural level (e.g. register and memory locations) or to application level that are accessible at its assembly instructions. It captures program execution traces and applies fault-space reduction

algorithms to reduce the overall execution time of the conducted campaigns. The reaction to of the injected faults campaign are recorded during the experiment and evaluated afterwards [OM19].

6.5.1 Fault Model

Faults can manifest in all system levels, from Hardware Faults (e.g. bit flips) to Software Faults (e.g. control flow faults). Hardware faults can be categorized - regarding their duration - into *permanent*- and *transient* -faults. Permanent Hardware faults reflects durable deficiency in the system's hardware due to manufacturing process variations, aging or decay [SPW09; KK10]. Transient Hardware Faults on the other hand, is a result of decrease of feature sizes of the System-of-Chip (SoC) which makes them more vulnerable to electromagnetic interference and other sources of electrical noise [SWK+05; OLSM18]. Hardware faults impact the system operational behavior either directly by arsing on the system components, such as memory or registers and clock values, or indirectly due to the effect of propagated error on further abstraction levels of the systems emerging safety and dependability threats [ALRL04b].

Since that the space of possible faults can not be thoroughly foreseen; FI is a crucial step for the development and design of dependable embedded systems, and it considered an essential validation tool for the system's functional safety against fault. The evaluation of the fault properties of an application is usually accomplished by applying Software-Implemented Fault Injection (SWiFI) techniques which emulate hardware faults at software level[SFB+00; QHXL09]. SWiFI techniques typically operate at the assembly or machine level of the application to emulate hardware faults at the low level which propagate up to the application level [OLSM18]. In the context of fault injection experiments, a fault model is characterized by the fault location (where to inject), fault timing (when to inject) and fault pattern (what to inject) [HTI97].

Figure 6.7 illustrate the abstraction level of fault model. These models are adopted in PyFI and represent the exploration space of PyFI for fault injections and analysis:

- Microarchitectural Level (MA) [LRK+09].
  Faults in MA consists of three main elements:
  - Central Processing Unit (CPU)
  - Memory
  - Interconnecting Bus
- Instruction Set Architecture Level (ISA) [KF15].
  ISA Faults can be categorized mainly into three types:
  - Operand error
  - Operator error
  - Arithmetic error
- Operating System Level (OS) [SFB+00].
  Fault that propagate from ISA to OS, can be categorized into:
  - Data Error
  - Program Flow Error

- Application Level (APP) [Frt16].
  Fault that propagate from ISA to APP, can be categorized into:
  - Data error
  - Program flow error
  - Access error
  - Timing error

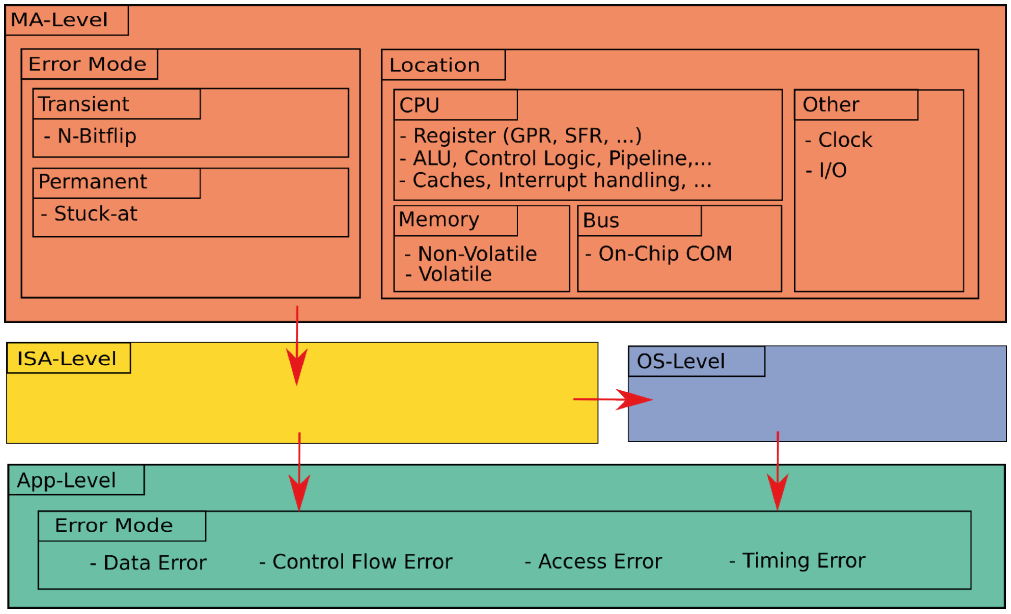


Figure 6.7: Fault Model adopted in PyFI. The model is divided into four levels of abstraction [OLSM18].

6.5.2 PyFI - Workflow and Design

The PyFI architecture comprises of three phases [OLSM18] (see Figure 6.8):

- pre-injection analysis
- fault injection campaign
- post-injection analysis

Pre-Injection Analysis

This phase consists of three operations:

1. **Experiment Configurator**: An experiment configuration file contains the setup that tailors the fault injection mechanism. This configuration file determines the:
   - *Location* of faults to inject at the MA-level (address of memory (volatile and non-volatile) or CPU register)

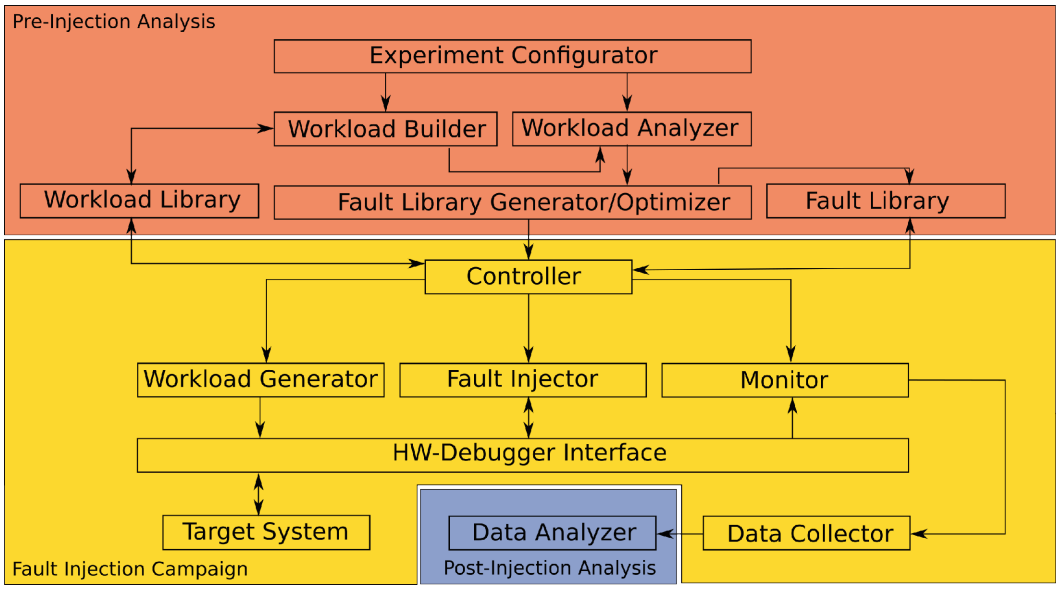


Figure 6.8: PyFI Architecture [OLSM18].

- *Mode* of injected error (bit-flip, stack-at)

2. **Workload Analyzer**: For the purpose of improving fault injection campaign's efficiency, the workload analyzer:
   a) Traces the *gold run* execution of disassembled ELF-file of the application (with no faults injected), by utilizing the iSystem On-chip Analyzer. During the execution of the golden run, statistical values are collected for the memory and register usage. This provides the static behavior of the system under investigation.
   b) In addition to tracing static behavior, the gold run execution provide insight for the dynamical behavior as well. This is fulfilled by collecting information whether the executed instructions perform a memory or register access, the current program counter, the accessed register and memory addresses and the type of each access, i.e., if a read or write access occurred.
   c) The obtained information is afterwards made available to the next *fault library generator/optimizer*.

3. **Fault Library Generator/Optimizer**: The fault library generator/optimizer uses the data set provided by the workload analyzer to create an optimized fault library according to the experiment configuration. The optimization include reducing the space of test to be performed by eliminating faults that have no possibility of activation. This so-called *Fault Collapsing* methods can be split into two main groups:
   a) Reduction of the number of possible faults to only effective ones
   b) Reduction of the duration for a singe experiment.

Fault Injection Campaign

The *controller* coordinates the fault injection campaign under the guidance of the *fault library*, generated in the *pre-injection analysis* step. It then parameterize the *fault injector* with the:

- fault type
- fault location
- trigger for the injection.

The fault injector then injects the specified fault using the API of the iSystem On-Chip Analyzer which is connected to the targeted controller for investigation.

During the fault injection, the *monitor* logs the time, location and type of the fault. Also is observes the system's behavior for, e.g., trap occurrences or timeouts, and traces the memory and register contents by triggering the data collector which stores a log file for post-injection analysis.

Post Injection Analysis

While the *data collector* redirects the output of the monitor to a file to save logging information for further analysis, the data analyzer generates detailed statistic about the number of injected and detected faults as well as silent data corruptions. Furthermore, the deviations from a normal system behavior such as traps, crash and timeouts can be extracted from the logs. On basis of the obtained information the data analyzer is able to calculate different metrics such as diagnostic coverage.

6.6 Measurement-Based Timing Analysis

The traditional and most common method in industry to determine program timing is by measurements (MBTA – Measurement-Based Timing Analysis). The approach is to use the actual hardware as the model for analysis: the code is deployed and executed in the hardware, providing different inputs, and actual execution is measured (usually by instrumenting the source code at different points). To obtain statistical validity, multiple executions of the code must be done, for the same set of inputs, to capture variations in execution time. The main challenge is basically to determine the inputs that leads to the worst (observed) execution time.

Compared with static analysis, measurements have the advantage of being performed on the actual hardware, which avoids the need to construct a hardware model and hence reduce the overall cost of deriving the estimates.

The use of MBTA is nevertheless hindered by several factors [NYP15]. First, measurements require that the hardware is actually available, and that the environment (important in the embedded domain) acts as the final system. Moreover, most of the time the code requires instrumentation, which means that the code is changed from analysis to deployment (although in most cases instrumentation code can be left in the deployed application).

The main issue, however, is, as noted above, that it is required to guarantee that the input values which lead to the worst-case are known. However, the number of possible execution paths is too large to guarantee exhaustive testing and therefore, measurements are carried out only for a subset of the possible input values. This leads to a safety margin being added, in the hope

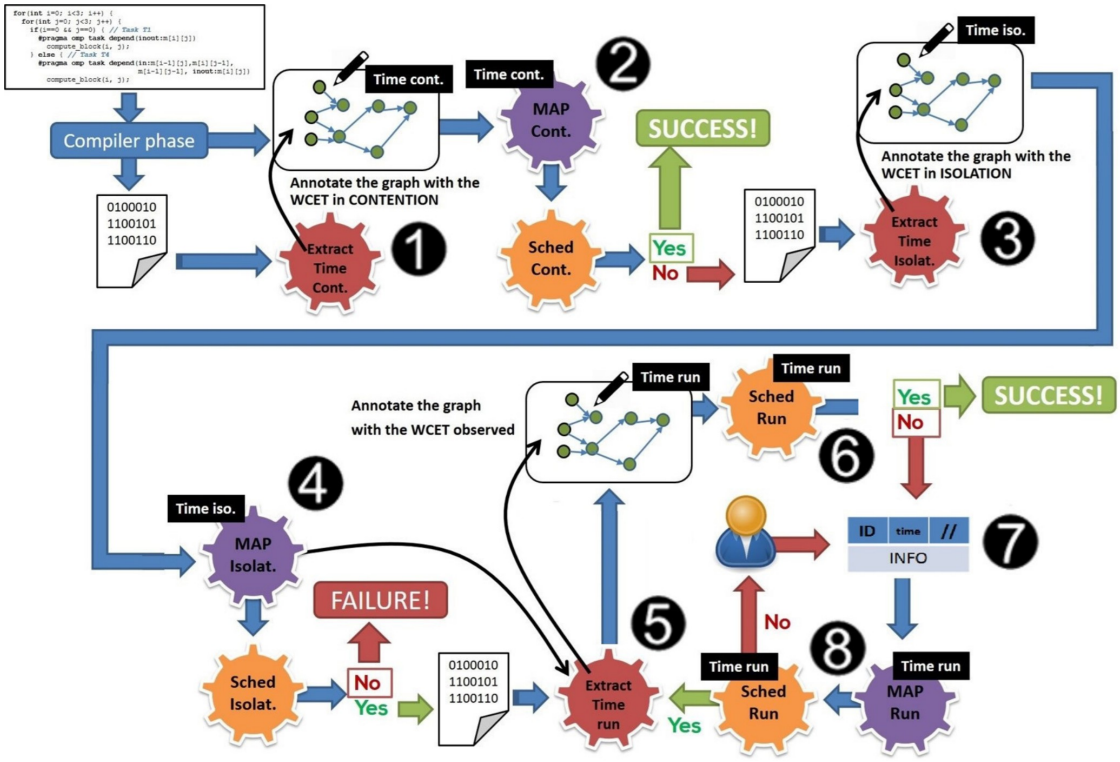


Figure 6.9: UpScale Analyser flow (picture taken from [NYP17]).

that the actual WCET lies below the resulting WCET estimate [WEE+08]. The main issue is whether the extra safety margin provably provides a safe bound, since it is based on some informed estimates. A very high margin will result in resource over-dimensioning, leading to very low utilization while a small margin could lead to an unsafe system [NYP15].

Addressing this challenge requires that the semantics of the program, and the architecture of the hardware are actually known, to be able to guide the selection of the program inputs to test. This leads to what is called the hybrid approaches [Pet00]: which combines static and measurement-based timing analysis.

Hybrid approaches, borrow the flow analysis phase from static methods to construct a control flow-graph of the given program and identify a set of feasible and potentially worst execution paths (in terms of execution time). Next, unlike static methods that use mathematical models of the hardware components, hybrid tools borrow their second phase from measurement-based techniques and determine the execution time of those paths by executing the application on the target hardware platform (or by cycle-accurate simulators) to collect execution traces. These traces are a sequence of time-stamped values that show which parts of the application has been executed. Finally, hybrid tools produce performance metrics for each part of the executed code and, by using the performance data and knowledge of the code structure, they allow to estimate the worst-case execution time of the program. This maps well to the approach in the project, as the flow-graph of the application is available in the system models. Example of tools include Rapitime [Rap] in the commercial domain, and the UpScale Analyzer in the research domain [Ana].

Figure 6.9 depicts the analysis flow of the latter [NYP17]. First, the application is compiled and its parallel graph of computation is annotated with the maximum execution time of every parallel task when it suffers a maximum interference on the shared resources. This execution time, the MEET (Maximum Extrinsic Execution Time), is measured for each independent code block by enforcing the "worst" interference conditions it may suffer (as much interference as possible from other tasks and applications running concurrently accessing shared memory and network). This allows to derive an initial mapping of parallel tasks to the hardware. Schedulability analysis for parallel DAGs can be thus used to determine if this worst-case condition meets application deadlines (see [Fon19] for a discussion on schedulability and mapping of parallel DAGs). If it does, no more processing is required.

If no mapping is possible that leads to a guaranteed execution, a second step is taken, where the graph is annotated with the execution time of each code block in isolation, the "MIET" of the tasks (Maximum Intrinsic Execution Time), which means there is no interference. Obviously this is the best case scenario, so if no solution exists in this case, then the system is never possible.

If a solution is possible in this case, then an iterative process of actual execution of the application, with the normal contention between its parallel blocks, is used to determine the maximum actual/observed execution time (MAET) for each block. In each iteration, schedulability analysis is used to assess validity of the solution.

The approach combines well with frameworks to assess the statistical validity of the obtained traces, as well as deriving probabilistic analysis on the WCET [SMDJ14]. Specifically, in the UpScale approach, the traces of execution times collected at runtime are fed into a statistical framework, called DiagXtrm [Dia], in which they are subjected to a set of tests to verify basic statistic hypothesis (such as stationarity, independence, . . .) which determines if the usual Extreme Value Theory (EVT) can be applied. Even more, the framework provides an approach to assess how "trustworthy" EVT estimations can be (to evaluate the quality of the estimations and find out whether confidence can be placed into the analysis).

7 Visualization Techniques

This chapter discusses the techniques for visualizing data coming from e.g. analysis results in order to support assessment. Section 7.1 provides a general overview on existing techniques that can be used for e.g. large amounts of data. It summarizes the variety of graphs, charts, plots, and other visualization techniques along with proper examples on how those can be applied for assessing analysis results during the development of embedded systems in e.g. the automotive or avionic domain. Finally, Section 7.2 extends this description by an in-depth discussion of assessment techniques for safety based on a failure propagation use case.

7.1 Data Visualization

The visualization of data is an important part of every analysis. Data is gathered or produced during an analysis and has to be filtered and represented in a way that a person can understand and explore the information.

There are five steps that need to be considered during the visualization of data (see [Maz09] for more information):

1. Define the problem: The problem defines for what a visualization is used. It could be the representation of a hypotheses, finding new information in a data set, or communicate information between people.

2. Examine the nature of the data: There are different types of data and every type is suitable for different visualizations. The data could be quantitative (e.g. integers), ordinal (days of a week), or categorical / nominal (city names).

3. Number of dimensions: The number of dimensions is important for choosing the visualization. The dimensions can be independent or dependent. The dependent variables vary and their behavior is analyzed compared to the independent variables.

4. Data structures: The data can be linear structured (e.g. tables), temporal (changing over time), spatial or geographical (e.g. a map), and network (e.g. relationships between entities).

5. Type of interaction: A visualization can be static (e.g. figure), transformable (a user can control the process of date visualization, e.g. change the scale), or manipulable (the user can modify parameters of the visualization, e.g. zoom on details).

Fig. 7.1 shows a summary of important variables that need to be considered during the process of visualize data.

Different examples of graphical elements that can be used to visualized data are represented in Fig. 7.2. The different elements can be combined to represent different characteristics of data. The color could be used to identify different variables or categories in a set of points.

Problem	Data type	Dimensions	Data structure	Type of interaction
Communicate	Quantitative	Univariate	Linear	Static
Explore	Ordinal	Bivariate	Temporal	Transformable
Confirm	Categorical	Trivariate	Spatial	Manipulable
		Multivariate	Hierarchical	
			Network	

Figure 7.1: Variables to consider when designing visual representations [Maz09]

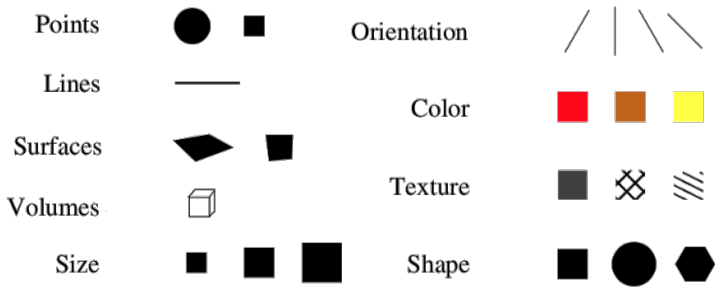


Figure 7.2: Example of graphical elements [Maz09]

In Fig. 7.1 the effectiveness of graphical elements or variables for the visualization of the three types of data quantitative, ordinal, and nominal is shown. For example the length is very good to represent quantitative data (like a set of Integers) but is less effective to represent ordinal (weak days) or nominal data. Colors should be used to separate the different categories of nominal data, but is less effective for quantitative data.

The next section represents different charts that are used to visualize the different types of data sets.

7.1.1 Charts

This section gives an overview of different charts and what kind of data is suitable for a specific chart.

In a **scatter plot** different variables are visualized. Every data point is represented as a point between the axis. The scatter plot helps to identify correlations between two different variables or identify clusters and outliers. Fig. 7.3 shows an example of a scatter plot. The plot represents the results of an optimization for a combustion system. The goal of the optimization was to achieve similar soot and NOx emissions to that of the baseline case but with a 10% fuel consumption improvement.

The **line chart** is used to visualize quantitative data as a position on quantitative scale. The points are connected to form a line- or curve-segment. Points between to data points can be interpolated. The interpolation helps to visualize trends, locale structures, and the general distribution of the data. The line chart helps to visualize groups of data points with a continuous domain. One line chart can consists of more than one data visualization. The different variables need to be on the same scale to be integrated in a single line chart. The numbers of combined line charts should not be higher then 3 or 4. Fig. 7.4 shows an example for a line chart diagram illustrating the periodic activation pattern of three tasks over time scale of $20ms$.

A **bar chart** can be vertical or horizontal. The data is represented as a bar instead of a

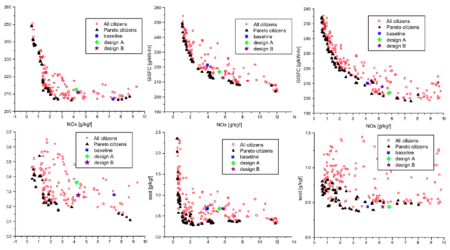


Figure 7.3: Example of a Scatter Plot

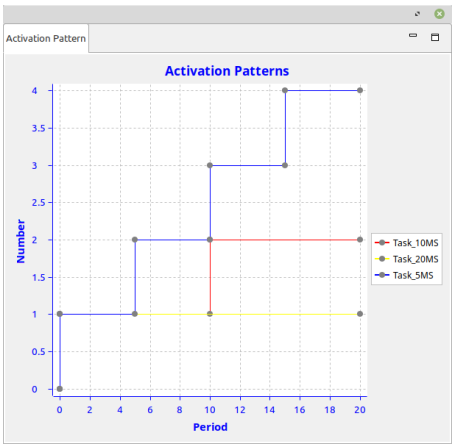


Figure 7.4: Example of a Line Chart, illustrating the number of triggering events (activations) for a given task set.

line. It can be used to represent nominal, discrete, quantitative, and dependent data. Normally, the horizontal axis contains the independent dimension and the vertical axis the associated dependent dimension. The Integration of several bar charts in a single chart is possible to represent several variables. A 3D visualization in a 3D-coordination system can be used if a second independent variable is available as a third dimension. Fig. 7.5 shows an example of a bar chart indicating the number of memory accesses per task from a modern engine management system.

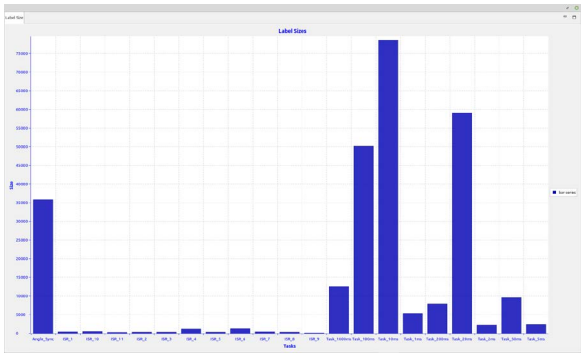


Figure 7.5: Bar Chart used for graphically representing the number of memory accesses for a given task set.

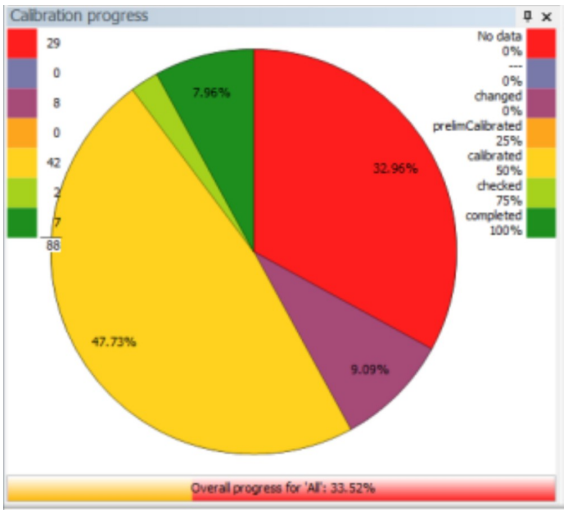


Figure 7.6: Example of a Pie Chart, Vector [Vec]

The **pie chart** is used to visualize quantitative characteristics over a nominal and independent variable. The characteristics a represented as different colored or textured segment of a circle.

A pie chart implies, that the segments can be summed up to a basic population. The size of a pie charts is important because in small circles it is difficult to compare the surfaces of segments. Fig. 7.6 shows a pie chart that indicates the current status of an automated calibration process, including information on how much work has already been completed (green area), along with the status of the remaining activities.

An **histogram chart** is a special bar chart or line chart. The chart is used to visualize the frequency of occurrence of a data point and not the data point itself. For the representation of quantitative data, the values are classified. Fig. 7.7 illustrates a bar chart histogram depicting the number of activation events for two tasks that were extracted from multiple traces with individual time spans. An example of a line chart histogram showing the distribution of e.g. the execution time of a task following a gaussian distribution is shown in Fig. 7.8.

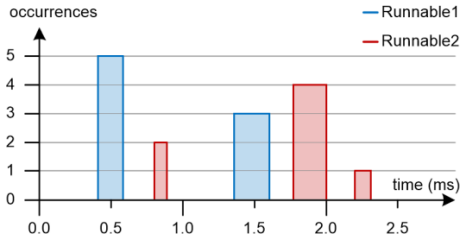


Figure 7.7: Column Histogram representing the number of activations of two tasks for a limited number of traces with different time spans [App]

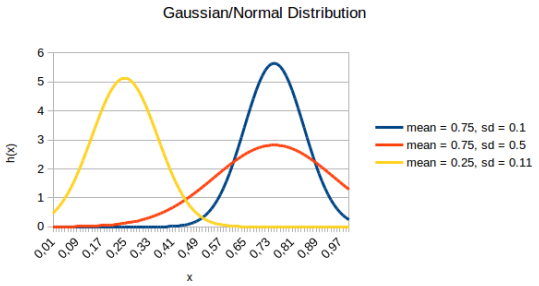


Figure 7.8: Line Histogram visualizing three Gaussian distributions with different values for mean and standard deviation [App]

Examples of different possible curve forms for a histogram are illustrated in Fig. 7.9 representing the following distributions:

- a) Normal distribution
- b) Bimodal distribution (indicates, that characteristics of two different populations exist)
- c) Multi-modal distribution (indicates, that characteristics of several different populations exist)
- d) skew distribution (positive)
- e) skew distribution (negative)
- f) Upset distribution (indicates a very concentrated distribution)
- g) Very flat curve (indicates values from different populations)
- h) cropped curve (indicates, that a part of the population is missing or was deleted)
- i) Upset distribution with a peak (indicates, that all elements after a specific threshold are combined)

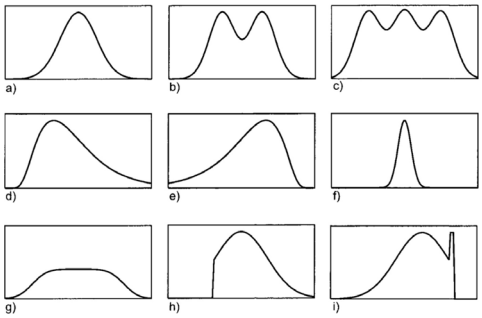


Figure 7.9: Possible Curve Forms of a Histogram, Schumann & Mueller [SM00]

The extension of a line or curve chart in the 3 dimensional space is the **surface chart**, represented in Fig. 7.10. Points in the 3 dimensional space are connected to build a surface. The surface structure provides information about the distribution and local trends, such as the fitness landscapes representing the quality of a solution space resulting from an optimization approach.

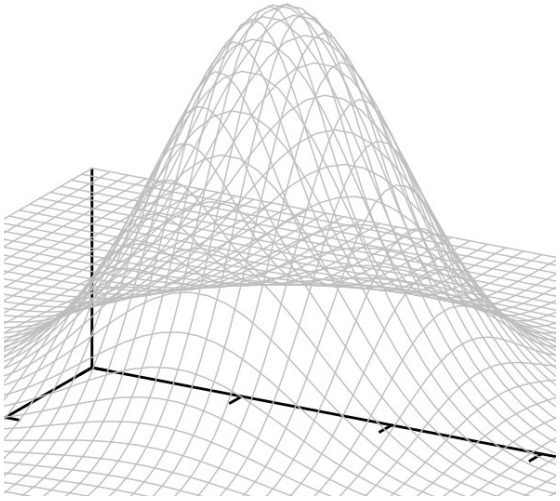


Figure 7.10: Surface Chart representing the fitness landscape of an optimization solution space [AM15]

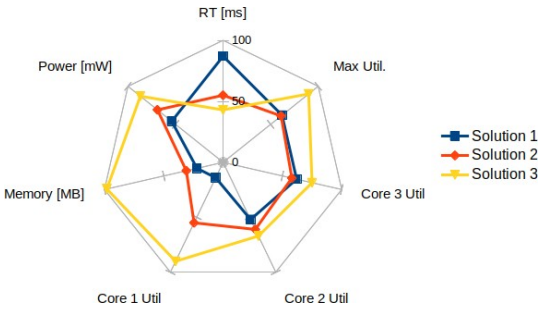


Figure 7.11: Spider Chart visualizing various metrics of three possible implementation candidates

A spider chart can be used to compare different strengths and weaknesses of multivariate data with three or more quantitative variables. The axis represent different characteristics. The values are represented by lines between the axis. An example of a spider chart comparing various quality metrics of three possible implementation candidates is illustrated in Fig. 7.11.

The parallel plot represents the different characteristics as vertical parallel bars and not as a circle like the spider chart. It is used to compare different strengths and weaknesses of multivariate data with three or more quantitative variables.

The **box and whisker plot** is used to visualize the distribution of a data set by categories. The median and first and third quartiles are represented in a box. The whiskers represent

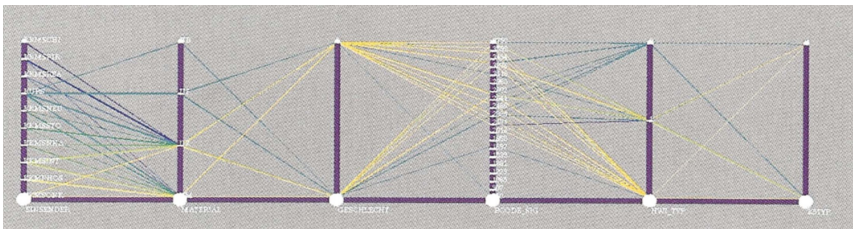


Figure 7.12: Example of a Parallel Plot, Schumann & Mueller [SM00]

the minimum and maximum of a data set. Fig. 7.13 shows an example of a box and whisker plot. The plot represents the emission factors for individual plume analysis separated between periods with no influence from trucks (red) and periods with at least one passing truck (black). Horizontal lines represent the median values, boxes represent the 75th percentile and whiskers represent the 90th percentile.

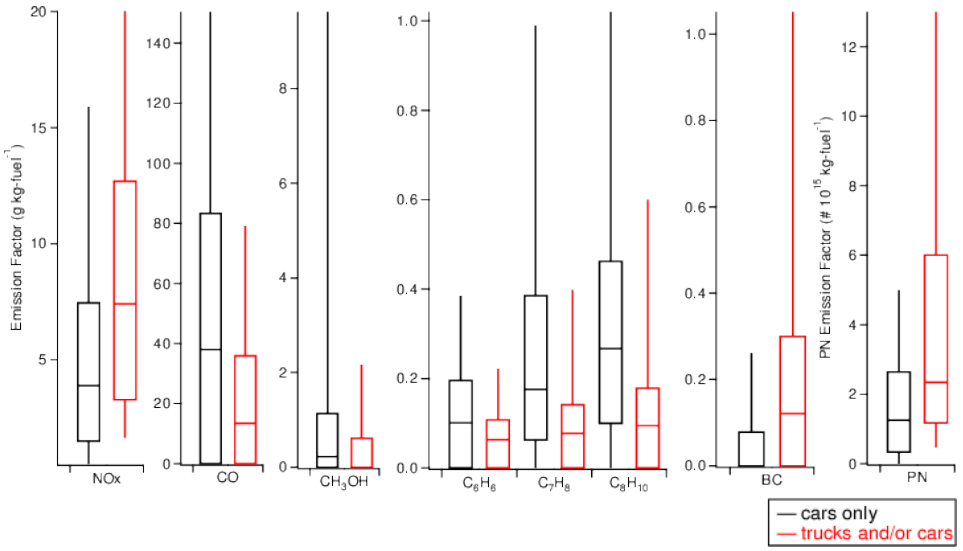


Figure 7.13: Example of a Box and Whisker plot, Wang et al. [WJZ+15]

The identification of the best chart to represent data depends on many different variables as mentioned in section 7.1. The Fig. 7.14 represents the "*chart chooser*" from Abella [Abe]. The chart chooser can be used as a starting point to choose a chart. Beginning in the center different ways based on the data types and attributes of the data a chart can be chosen .

7.1.2 Graphs

This section shows how information can be visualize using graphs. A graph is an abstract structure and can be used to represent any information that can be modeled as objects and the relationship between the objects. Objects are represented as nodes and relations between the objects as edges.

The first and major challenges for visualize structured information lies in the *representation* of a graph. The challenge is to visualize the graph covering most information but make it easy to read and to interpret. The second challenge is the *scaling* problem. The algorithm that are

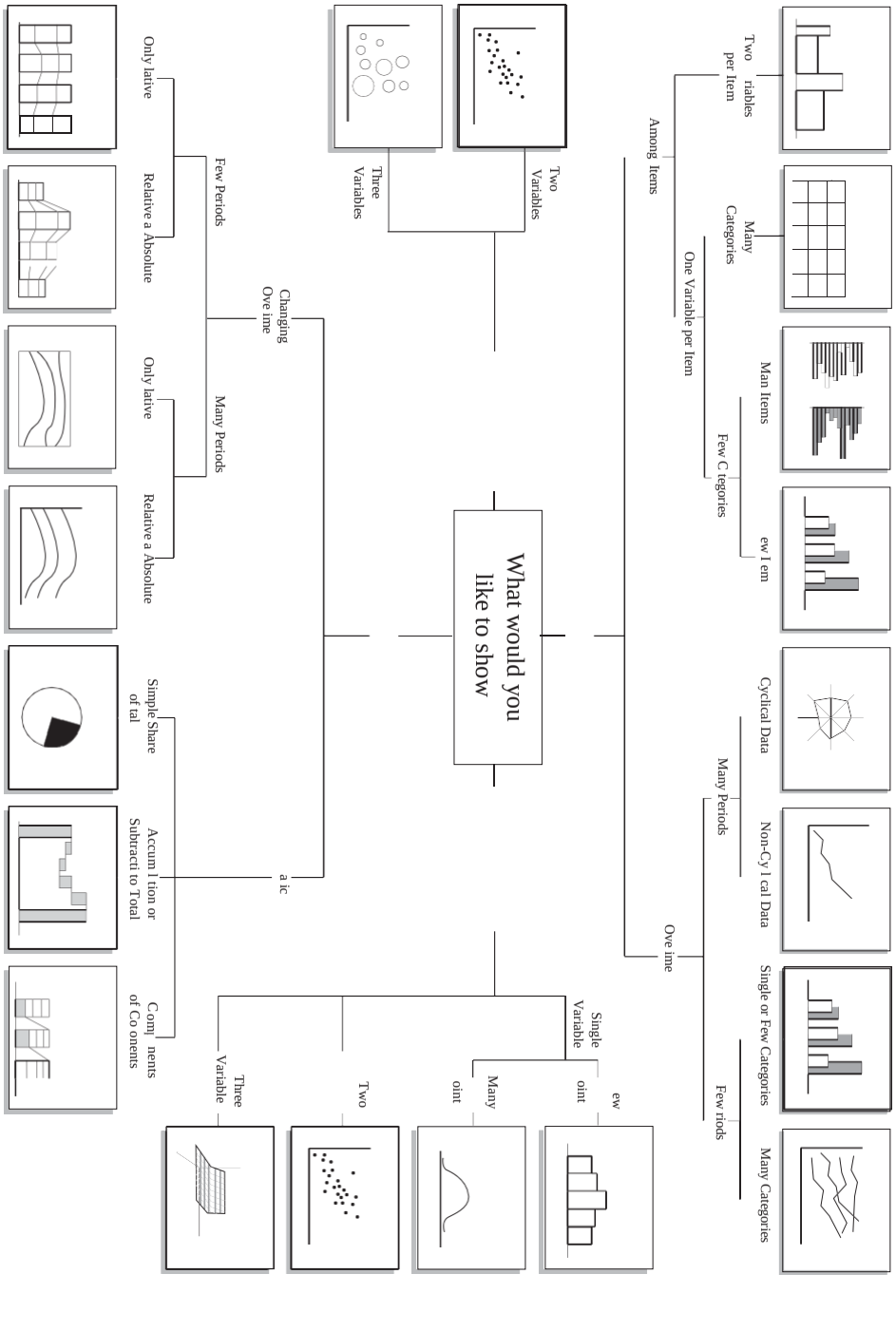


Figure 7.14: Chart Chooser, Abella [Abe]

used for the layout of the representation must be able to process the amount of information in a given time. Another scaling problem is the limited real estate of display area.

In a graph different information can be visualized by using colors, sizes, forms, or shapes for the nodes and edges.

Next, different graph types are described and some example graph layouts to handle the challenges regarding graph visualization

Graph Types

All Graphs have general attributes that classify their type. A graph consists of a nonempty set of nodes (vertices or points) and a set of edges that represents the relation between the nodes. The edges can be unweighted or weighted (nominal or ordinal quantitative). The weight is sometimes referred to as the *cost* of an edge. Examples for weights are a measure of length of a route, the energy required to move between to locations, etc. Some graphs can be traversed to form a path. This path consists of all traversed nodes and a sequence of edge to reach the nodes. A simple path has no repeated nodes within the path. In a cycle path the initial nodes is also the end node of the path. A graph without any cycle path is called acyclic.

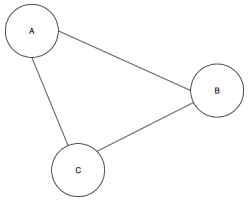


Figure 7.15: Example of an Undirected Graph

Undirected Graphs. An undirected graph is a graph, which only contains bi-directed edges. Fig. 7.15 shows an example of an undirected graph.

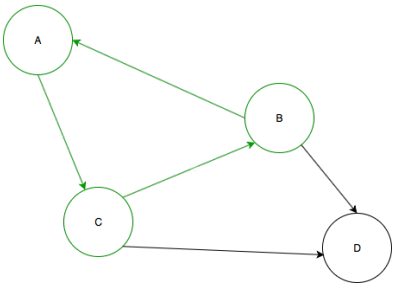


Figure 7.16: Example of directed cyclic graph

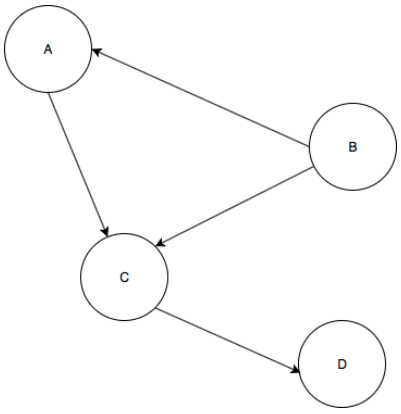


Figure 7.17: Example of directed acyclic graph

Directed Graphs. A directed graph hast direction for every edge. The direction is normally visualized by adding arrows to an edge. A path can be build by traversing between the nodes via the directed edges. As mentioned before, a distinction is made between directed cyclic graphs (see Fig. 7.16) and directed acyclic graphs (see Fig. 7.17).

Tree. A tree is a special type of graph. It contains no cycles, is usually directed, and has a single node as starting point, which is called root. The tree is a hierarchical structure that starts at the root node. The end nodes of a tree are called leaves. Fig. 7.23 shows and example of a tree. The tree represents an Amalthea system model including its nested sub-models (software, hardware, ...) and model elements.

Figure 7.18: Example of Tree represented as Indented List

Network. A Network is a special type of directed graph. It has usually weighted edges, but in contrast to a tree, it has no topological restrictions. The graph in Fig. 7.19 shows a Network on Chip (NoC) as an example of a network graph. A NoC has a specific topology which describes the structure of the Network, in this case a mesh structure. The graph shows the connection between the cores via network interfaces and routers.

Graph Layout

There are many different layouts for the visualization of graphs. In the following paragraphs, three example layouts are described.

Radial. The nodes are arranged in circles around a focus node. It is usually used in an interactive visualization, where a user can choose the focus node. Fig. 7.20 shows an example for the radial layout.

Circular. In a circular layout, the nodes are arranged in a circles. The space between the nodes is usually evenly. Fig. 7.21 shows an example of the circular layout.

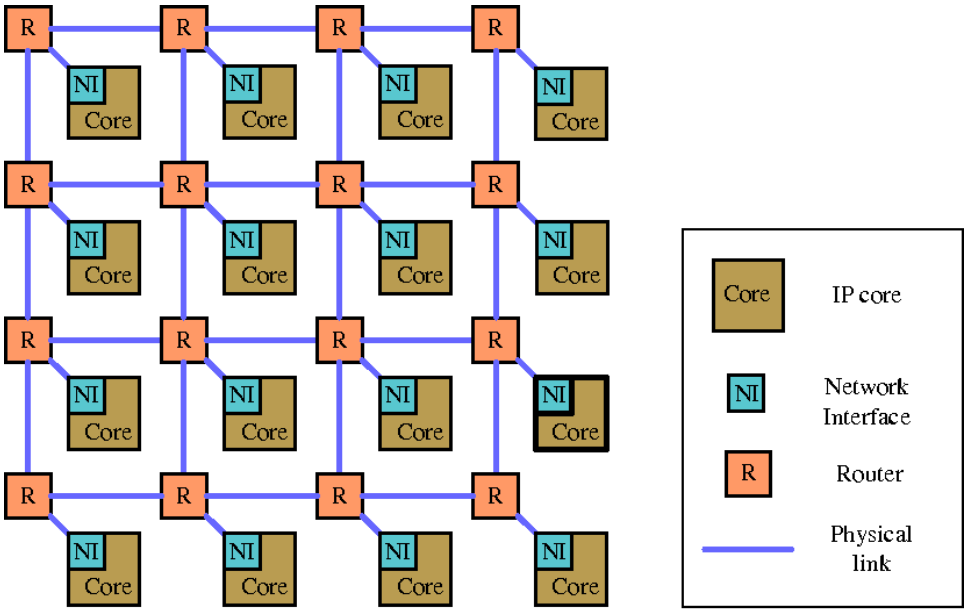


Figure 7.19: Example of a Network, Liu et al. [LGY12]

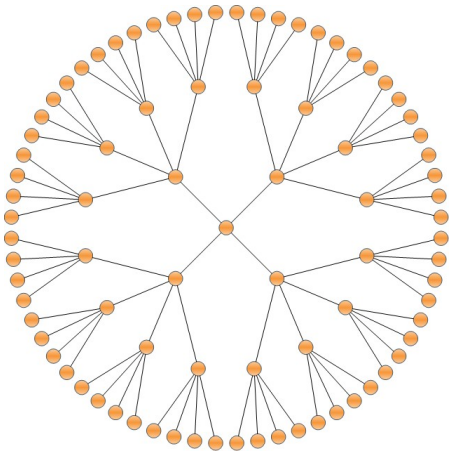


Figure 7.20: Example of Radial Graph Layout

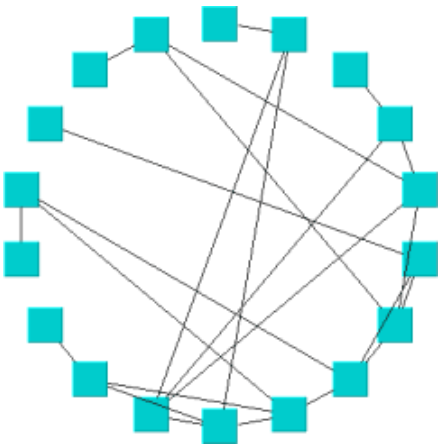
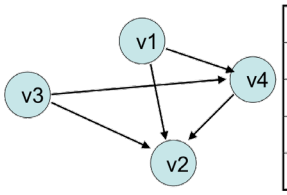


Figure 7.21: Example of Circular Graph Layout

Adjacency Matrix. The adjacency matrix represents a graph as matrix. The rows represent edges leaving the node and the columns represents edges entering a node. The structure is used for storing and processing a graph on a computer. Fig. 7.22 shows an example adjacency matrix.



	v1	v2	v3	v4
v1	0	1	0	1
v2	0	0	0	0
v3	0	1	0	1
v4	0	1	0	0

Figure 7.22: Example of an Adjacency Matrix

Tree Layouts. Trees are hierarchal structures with a root node. Every node can have multiple child nodes and with every refinement of a node the tree becomes larger. The following three layouts are examples to visualize trees. Fig. 7.24 shows a **note diagram**, which is a simple visualization of a tree. The starting point is the root node, which is refined vertically or horizontally layer for layer. A more complex visualization is the **tree-map** shown in Fig. 7.18. A tree-map is used to visualize a hierarchical structure. The different nodes are represented as nested rectangles. The size of the rectangles represents the value of the data element. A tree-map can be used to compare different structured information by the size of data elements. The last example in Fig. 7.23 shows a tree represented as **indented lists**. The Tree is build vertical and every new layer is indented on the horizontal. The examples in the Figure represents an Amalthea model used in PANORAMA.

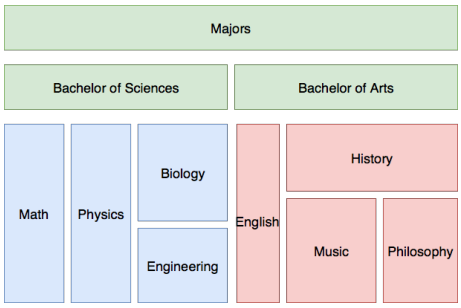


Figure 7.23: Example of Tree represented as Tree Map

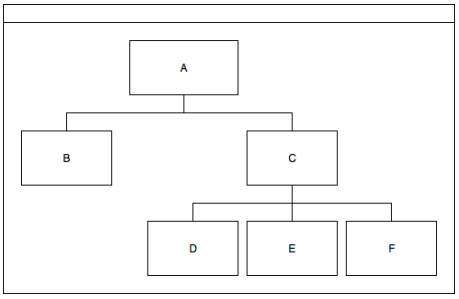


Figure 7.24: Example of a Node Tree

7.1.3 Tools for Assessment

A variety of tools exist that implement e.g. graphs for assessing analysis results, executions, dumps, application behavior, and other aspects that are relevant in developing multi- and many core systems. Two tools that integrate a variety of views, charts, graphs, diagrams, metrics, and various other visualization techniques to extract information from traces and logs is realized by Eclipse Trace Compass (cf. Fig 7.25) and the App4MC Task Visualizer (cf. Fig 7.26).

Eclipse Trace Compass [Ecl20] is a Java-based open-source tool that allows displaying and analyzing any kind of logs or traces. It provides support for a large number of trace formats,

Figure 7.25: Example of Trace Compass usage for kernel analysis [Ecl20]

such as Common Trace Format (CTF), allowing to inspect Linux LTTng kernel traces as well as bare metal traces, GDB traces, and hardware traces. Especially its support for the Best Trace Format (BTF) for OSEK, along with features for e.g. Latency and Critical Path analysis along with Real-Time deadline investigation makes it especially applicable for the automotive domain.

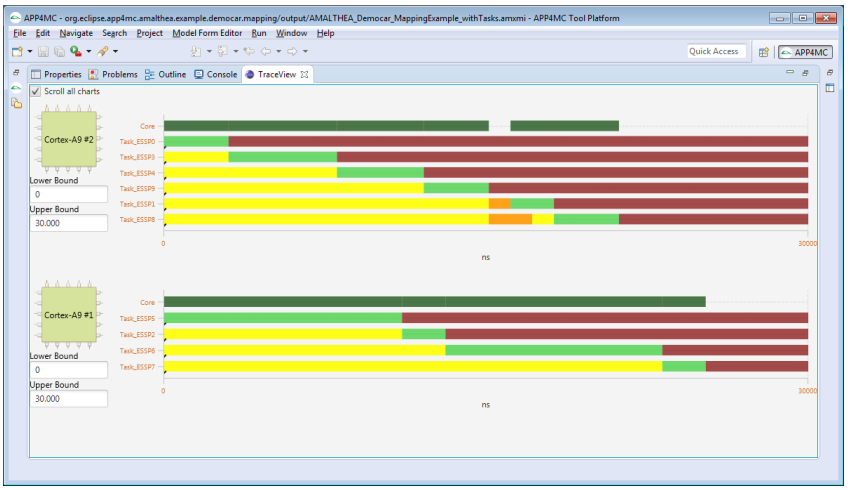


Figure 7.26: Gantt-Chart illustrating the execution of 10 tasks on a dual-core ECU

The App4MC Task Visualizer [App] is a tool for visualizing the execution of tasks along with their states and state changes on the resp. executing cores. In order to execute the task visualizer, it is necessary to describe the overall system in terms of an AMALTHEA Model file. The minimal amount of information consists of a Software Model, Hardware Model, and Mapping Model denoting the specifying the deployment of software to hardware, allowing its usage in early design phases without any dine-grained knowledge of implementation details.

For an overview on additional tools that can also be used for assessing e.g. analysis results, such as the commercial INCHRON or Vector Tools, we refer to the State of the Art section in

[PAN20].

7.2 Failure Propagation Visualization

As described in delivery D3.1, in the context of PANORAMA, we are also focusing on fault propagation modeling and analysis, specifically applying the FMEA and FTA techniques. The goal in this step is to create one or more views that represent FMEA, FTA as well as the propagation of modeled system failures. Figure 7.27 shows graphical examples of the outcome of the system safety analysis process.

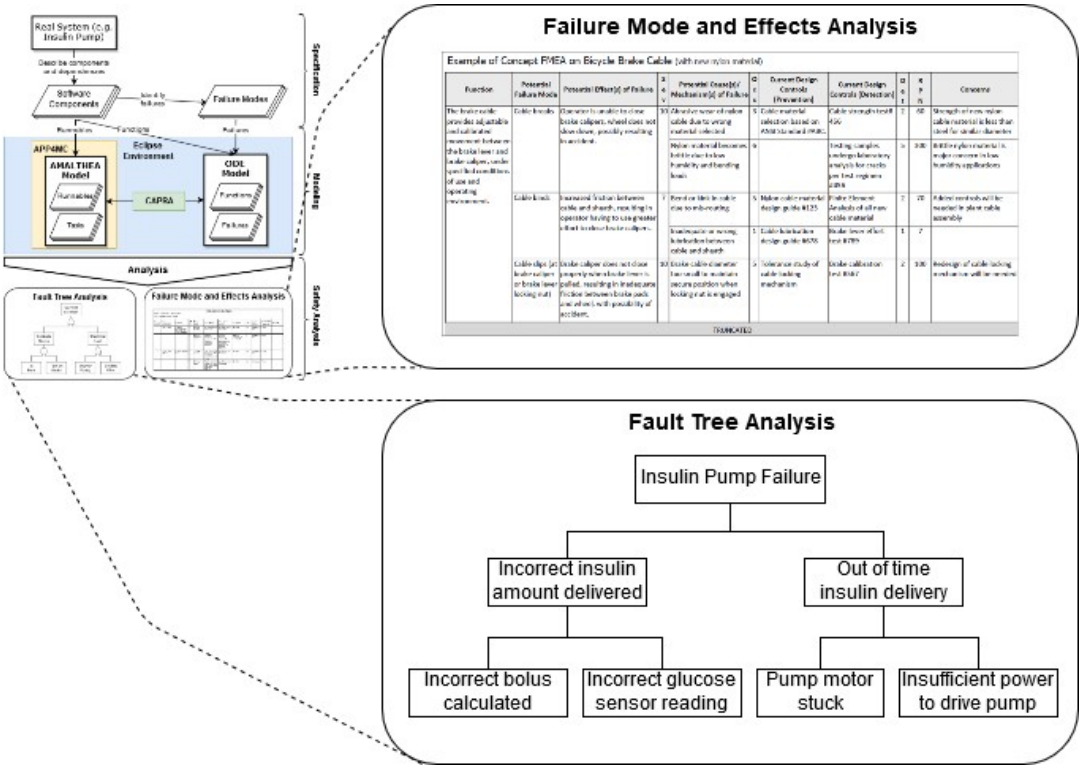


Figure 7.27: System Safety Analysis Process Representation

7.2.1 Safety Modelling Process

In our Safety Analysis process proposal the cycle starts with the construction of the ODE and AMALTHEA models with the specification of Critical Functions, Failures and Runnables.

Figure 7.28 represents an ODE model for the Insulin Pump case study and Figure 7.29 refers us to the mapping of Runnables within the AMALTHEA model.

FTA Using the ODE FailureLogic model we can create an abstract view of an FTA through the FaultTree, FailureModel, Cause, and Failure classifiers as shown in Figure 7.30.

FMEA Although an FMEA is commonly represented through a table for this work, our visualization approach consists of ODE model class diagrams with a list of failure modes. Figure

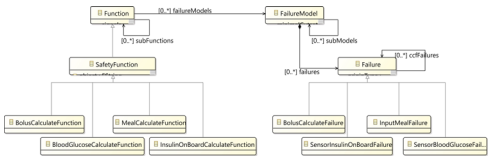


Figure 7.28: Insulin Pump Systems ODE Model

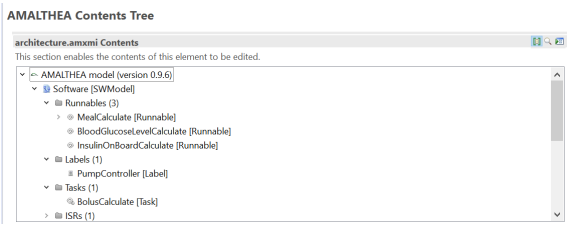


Figure 7.29: Insulin Pump Amalthea Software Model

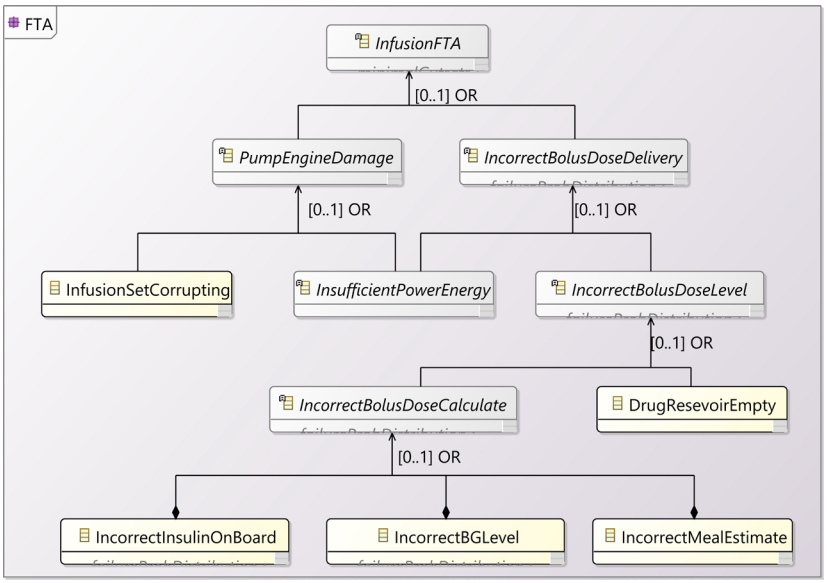


Figure 7.30: Insulin Pump - Fault Tree Analysis

7.31 shows a generic case study of an Insulin Pump System.

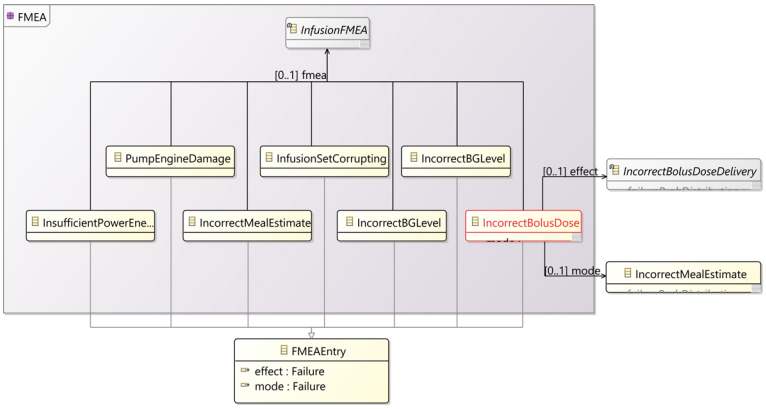


Figure 7.31: Insulin Pump - Failure Mode and Effects Analysis

7.2.2 Conclusion

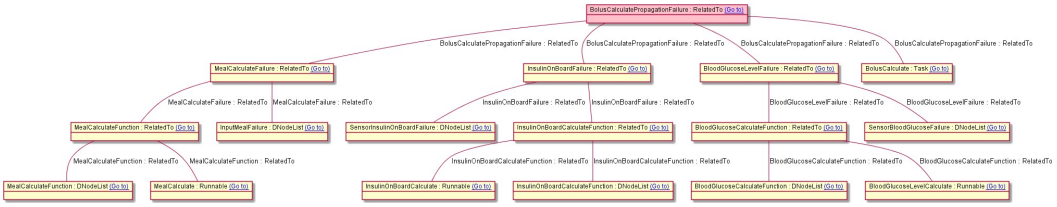


Figure 7.32: Insulin Pump - Failure Propagation Model

Our approach uses the APP4MC tool in conjunction with other Eclipse Environment plugins like Capra (Figure 7.32) to enable the construction and analysis of critical heterogeneous systems. Here our main focus is not on building an Amalthea model, but on the relationship of its components to the critical functions of a system for mapping and representing failure modes and their propagation analysis. For this, we use ODE component modeling. Importantly, at this time of prospecting, we still have models at a high level of abstraction. However, as future work, we will make possible a more definite view of modes and fault propagation through graphs and tables representing FMEAs and FTAs.

8 State-of-the-Art of Collaborative Development Processes

In this section, we provide an overview of the state of the art for collaborative development processes. Thus, we summarise how these processes have been described in the scientific literature and in other projects. After providing a general high-level view of software development processes in section 8.1 and extending it to safety-critical systems in section 8.2, we report on the processes developed in DEIS in section 8.4 and ARAMIS II in section 8.5. Finally, we also introduce the general workflow supported by Siemens Polarion in section 8.6.

8.1 Software Development Standard Process

In practice, every development process is adaptable, depending on the system context. However, the process generally follows the life cycle of Figure 8.1, which can cover most software development projects on heterogeneous systems:

System Engineering consists in the initial specification of the system/software to be developed including the features list, architectural design (high-level design) and – if applicable - first user interaction solution previews (form and behaviour).

- **Kick-off Meeting (KOM)** is the formalization of the beginning of the project. This usually has as input the initial version of the Technical Specification contains a high-level definition of the features / functions, main components, which may include User Interfaces (Mock-ups), and high-level solution and design of the system to be developed.

Requirements Engineering consists in the specification of the software to be developed including the software requirements, architectural design (high-level design) and user interaction solution previews (form and behaviour).

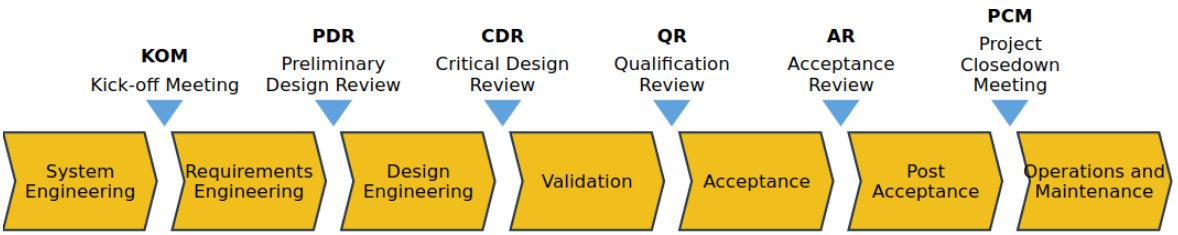


Figure 8.1: Critical Software – Software Development Generic Life Cycle

- The software requirements engineering phase is completed by the **Preliminary Design Review (PDR)**. The inputs to this milestone are: the Software Requirements Specification, system architecture, the preliminary interface control document and Solution Previews (e.g. Mock-ups, lo-fi and/or hi-fi Prototypes), all part of the Technical Specification.

Design Engineering produces the detailed design and source code in parallel allows the design to be generated from source code using reengineering tools. Producing the unit testing in parallel with the coding allows the errors to be identified and corrected earlier.

- The results of this phase are the input to the **Critical Design Review (CDR)**, which signals the end of the design phase. The state of the software project after critical design review is called Defined State.

Validation verifies the end-to-end functionality of the system in satisfying all requirements and specifications (mainly system testing), including system usability verifications.

- The validation phase includes a **Qualification Review (QR)**. The state of the software project after qualification review is called qualified state.

Acceptance demonstrates that the system meets your requirements in the operating environment through testing conducted under the supervision of an independent acceptance testing team and follows the procedures specified in the Acceptance Test Plan.

- The acceptance phase includes an **Acceptance Review (AR)**. The state of the software project after acceptance review is called the accepted state.

Operations and Maintenance process is activated when the software product undergoes any modification to code or associated documentation as a result of correcting an error, a problem or implementing an improvement or adaptation.

8.1.1 PANORAMA Context

The PANORAMA project aims to provide modeling tools that will support mainly the design engineering phase and the validation phase. For instance, we aim to support modeling of safety related models such as Faults Tree Analysis (FTAs) models and Failure Modes and Effects Analysis (FMEA) models. We also aim to facilitate the use of analysis results from the different analysis tools for improvement of the models. Since various modeling languages are used in the different steps, in the PANORAMA project aims to extend the AMALTHEA model so that it can be integrated with the various commonly used modeling tools to allow for smoother collaborative work.

8.2 Safety-critical Systems Development Process

In section 8.1 we have described a generic process for software development. In this section, we describe a typical development design flow for safety critical systems that includes both hardware and software [**TreiEtAl2016**]. This process describes the functional steps as well as

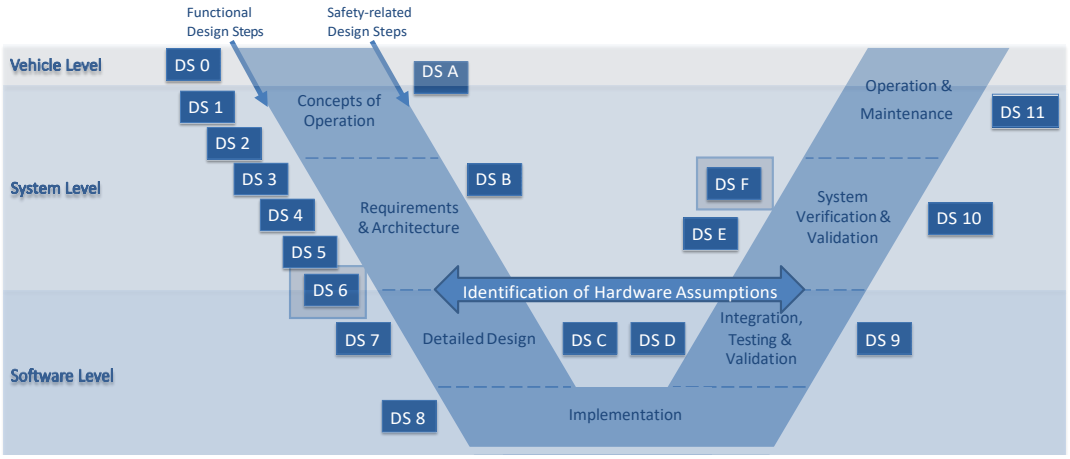


Figure 8.2: Safety-critical Systems Development Life Cycle [**TreiEtAl2016**]

Table 8.1: Overview of the Identified Design Steps

Functional Design Steps	
DS 1: System Requirements Engineering	DS 7: Variant Configuration
DS 2: System Architecture Design	DS 8: Implementation
DS 3: Software Requirements Engineering	DS 9: Validation and Testing
DS 4: Derivation of Product Variants	DS 10: System Integration
DS 5: Definition of Software Architecture	DS 11: Handover
DS 6: Behaviour Modelling	
Safety-related Design Steps	
DS A: Derivation of the Functional Safety Concept	DS B: System Safety Requirements Engineering
DS C: Software Safety Requirements Engineering	DS D: Verification of Software Safety Requirements
DS E: Safety Validation	DS F: Functional Safety Assessment

safety-related steps required to design a whole system. This design flow is a result from the project AMALTHEA4Public, and is compatible with the ISO 26262 standard. It is depicted in Figure 8.2. Step **DS 0** to **DS 11** are functional design steps while steps **DS A** to **DS F** are safety related design steps. A summary of these design steps is given in Table 8.1. Additionally, more details on how this fits into collaborative systems engineering process are given in **??**

8.3 Collaborative Work in Tool Platforms

The development of heterogeneous embedded systems involves coming together of different organizations with expertise in different domains. Realizing such systems requires concrete collaboration between specific groups of different organizations. Hence for efficient collaboration between different organizations, interoperable tool support is necessary. Essential aspects of collaborative work in tool platforms may include data management, information management,

data privacy, data security, and tool usability [**crest**].

Emails are one of the most commonly used communication tools to share information among organizations. However, emails may not be well suited for specific tasks such as working on a draft that involves the input of multiple parties. Essential information such as comments, context information, and managing of documents are challenging to track and can get mixed up in a long chain of email history. Version control systems (VCS) improve the document editing process, as every partner has a view of current state and complete version history, including tags for special versions. The VCS such as GIT [**git**], SVN [**svn**] also offers integrated diff tool that highlights the differences between different versions. The VCS also provides the possibility to resolve conflicts before a new version is created. Although VCS offers a better solution to manage data; however, context information such as task sharing among partners needs to be tracked using a separate tool such as kanban board.

8.3.1 Document-centric Collaboration

Document-centric collaboration tools keep track of the collaboration metadata in addition to the document. Comments (metadata) created are highlighted in the document and are visible to everybody. Hence the problem of missing context information is addressed by keeping track of the metadata. Cloud collaboration tools share the data in a public cloud. Therefore, documents can be shared with external organizations by taking appropriate security measures. The editing possibilities of the documents are supported by desktop and web applications. The web services offered by cloud collaboration tools enable interoperability with other web services using public APIs. Real-time collaboration tools are a special type of cloud collaboration solution that allows multiple users to work together at the same time. Ad hoc discussions can be started with the help of integrated chat and video conferencing capabilities. Hence, real-time collaboration tools enable all collaborators to see the changes made by each other instantaneously and offer the possibility to react immediately or later.

8.3.2 Artifact-centric Collaboration

Collaborative development between different organizations not only involves sharing of documents but also involves sharing of other artifacts such as models. The artifact-centric collaboration process facilitates the exchange of well-defined artifacts between partner organizations. Essential aspects of artifact-centric collaboration tools are described as follows [**crest**]:

- Supports synchronization of the artifact data such that all collaborators can see the updated version as soon as possible.
- Avoids introducing new bugs to the artifacts during the process of editing to ensure the safety of the artifact exchanged.
- Enables automatic notifications of artifact changes to all the partners involved.
- Enables ad-hoc communication between different partners.
- Supports version management of artifacts.
- Enable legacy support for importing and exporting artifacts

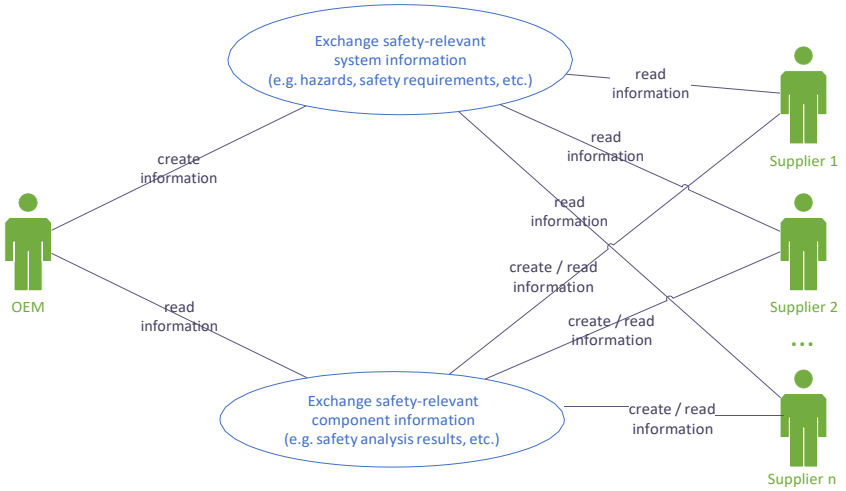


Figure 8.3: DEIS Collaboration Scenario 1

8.4 Distributed Dependable Systems Development

The following collaboration scenarios were specified as part of the H2020 research project DEIS[1]. In industrial practice, the development of complex, safety-critical systems is scattered across different partners (e.g., OEM, Tier-1 suppliers). This is the case for instance in automotive, avionics, or railway domain. Thereby, the suppliers need to take over an increasing share of the risk from the OEM. Moreover, the OEMs builds their safety case based on the information provided by the suppliers.

A high amount of alignment activities is required for ubiquitous feature development and several iterations are needed to align the interdependent function development. Moreover, lots of assumptions and constraints for development of elements-out-of-context are made by the suppliers. Hence, interdependent functions are avoided as much as possible, therefore innovative functionalities are hampered. All partners involved in the development process interchange safety- but also reliability-related information. Thereby, all involved participants along the supply chain use different methodologies and tools for engineering functionalities. Today, the information is exchanged using documents.

8.4.1 Collaboration Scenario: Requirement-driven Design

The OEM provides the safety requirements which must be taken into account by the suppliers. The individual component suppliers can base their assumptions on the exchanged information and update the context their sub-system/component. The suppliers themselves provide safety information (e.g., about the conducted safety analyses as well as their safety concept) in addition to the component/sub-system they deliver to the OEM (see Figure 8.3). This information is exchanged with other suppliers and the OEM (during the integration stages of the development life-cycle). Based on this the OEM is able to generate a safety case for the target product.

[1] http://deis-project.eu/

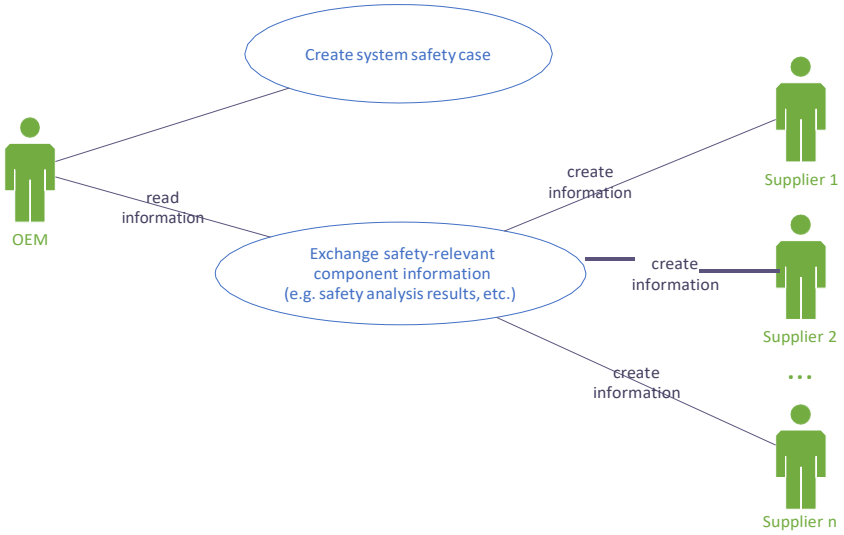


Figure 8.4: DEIS Collaboration Scenario 2

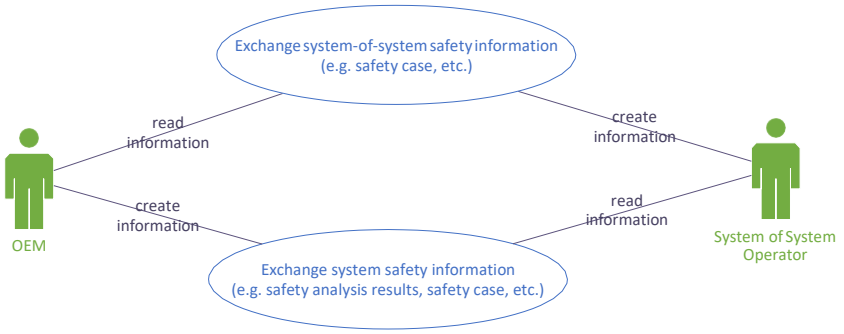


Figure 8.5: DEIS Collaboration Scenario 3

8.4.2 Collaboration Scenario: Components-of-the-Shelf

The OEM builds its system based on pre-existing components of the suppliers. Thereby, the OEM does not provide safety requirements to the suppliers. However, the suppliers need to deliver safety information to the OEM in addition to the component/sub-system (see Figure 8.4). Based on this information the OEM builds its safety case for the overall system.

8.4.3 Collaboration Scenario: System-of-systems Integration

The OEM provides all necessary dependability information related to a product/system which must be integrated into a (pre-existing) system-of-system (see Figure 8.5). Hence, the system-of-system operator can create an safety case for its system-of-system.

8.4.4 Challenges

Seamless interchange of safety information enables the creation of safety cases and/or the assurance of correct integration for systems/products. Moreover, the safety requirements provided by the OEM helps the supplier to develop a dependable sub-system/component. Since

safety-related information are currently exchanged in a document-based way, it is too much effort for the involved parties to read and understand the information and to enter the data into the own custom tool chains. A machine-readable format is required to formalize the information and to automate the exchange. Moreover, since different methodologies and tools are used by the partners along the supply chain for the various engineering activities, a tool-independent exchange format is required.

8.5 ARAMiS II Generic Process

Within the ARAMiS II project [**Aramis**], a common and generic design process that covers a variety of structured development processes used in industrial practice of multiple domains such as automotive, avionics, and industrial automation has been defined. It includes comprehensive expertise and know-how from manufacturers (e.g., Audi, Bosch, Continental, Denso, Airbus, Diehl, Liebherr, Siemens, Wika), research institutes (e.g., DLR, OFFIS, Fraunhofer, KIT, fortiss), and tool providers (e.g., Timing Architects, Vector, AbsInt, Symtavision, Silexica).

While the focus is on the development of multicore systems, it is based on the commonly used V-model as a state-of-the-art process model. In addition, the generic design process is aligned with a variety of international standards such as *Functional Safety for Road Vehicles* (ISO 26262), *Software Considerations in Airborne Systems and Equipment Certification* (DO-178C), *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems* (IEC 61508), *Systems and Software Engineering – System Life Cycle Process* (ISO 15288), and *Systems and Software Engineering – Software Life Cycle* (ISO 12207).

The description of the generic design process consists of tasks/activities and corresponding work products. The individual parts are defined in terms of the *Software Process Engineering Metamodel* (SPEM). The basic process is divided into five key activities, which are depicted in Figure 8.6 and will be examined more closely in the following sections. Work products are those that have been identified to be relevant among all industrial project partners.

8.5.1 User and System Requirements Engineering

As a first step, user and customer needs are collected in the *User and System Requirements Engineering* (REQ) activity. Moreover, the problems to be solved are formulated as a set of requirements. Subsequently, these requirements are transformed into system requirements that have to be fulfilled by the system under development.

Inputs Customer Needs

Outputs System Definition and Interfaces, System Requirements and Constraints

8.5.2 System Architecture

Based on these first results, the analysis of the system requirements is carried out in the *System Architecture* (SYS) activity. Following the design of the system architecture, which identifies the system elements and their relationship as well as the mapping between requirements and elements, it is decided which elements are realized by hardware, software, and/or mechanics.

Inputs System Definition and Interfaces, System Requirements and Constraints

Outputs System Architecture, Software Architecture Requirements and Constraints

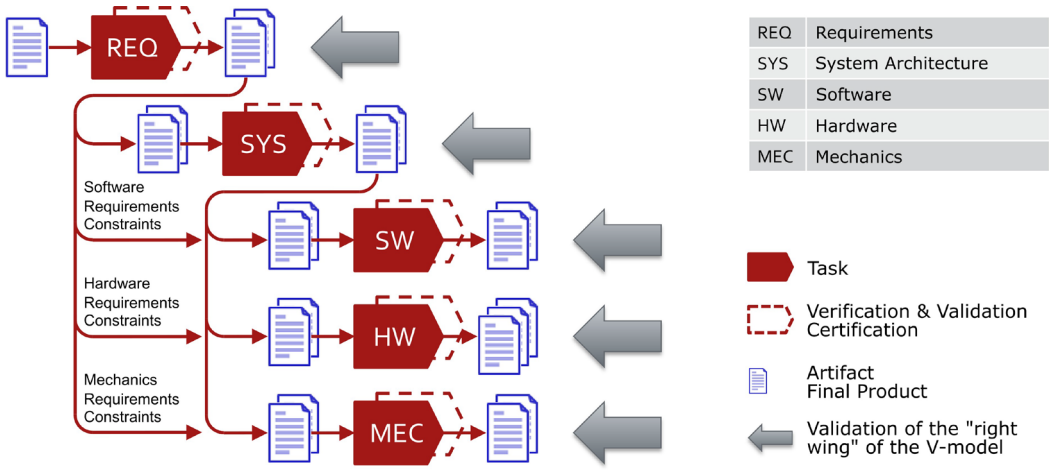


Figure 8.6: ARAMiS II Generic Process [**AramisGenericProcess**]

8.5.3 Software Development

The *Software Development* (SW) activity comprises the whole software development process, which is summarized in Figure 8.7. Besides the analysis of the system architecture and the development of the software architecture in the *Software Architecture* (SWA) activity, this step also includes all *Software Design* (SWD) and *Software Implementation* (SWI) activities.

8.5.4 Hardware Development

In analogy to the SW activity, the *Hardware Development* (HW) covers the whole hardware development process, which is illustrated in Figure 8.8. It is subdivided into the *Hardware Architecture* (HWA), *Hardware Design* (HWD), and *Hardware Implementation* (HWI) activities.

8.5.5 Mechanics Development

The last step to be mentioned is the *Mechanics Development* (MEC) activity.

8.5.6 Verification and Validation

As indicated in Figures 8.7 and 8.8, the corresponding verification and validation steps are performed along the individual tasks. Besides the elicitation of relevant process phases a common structured terminology has been defined.

8.5.7 Importance for PANORAMA

In the context of PANORAMA, the generic design process will be used as a tool to structure the requirement elicitation process and to align the design process with all standards mentioned above. It is important to note that the process does not cover all aspects of the development

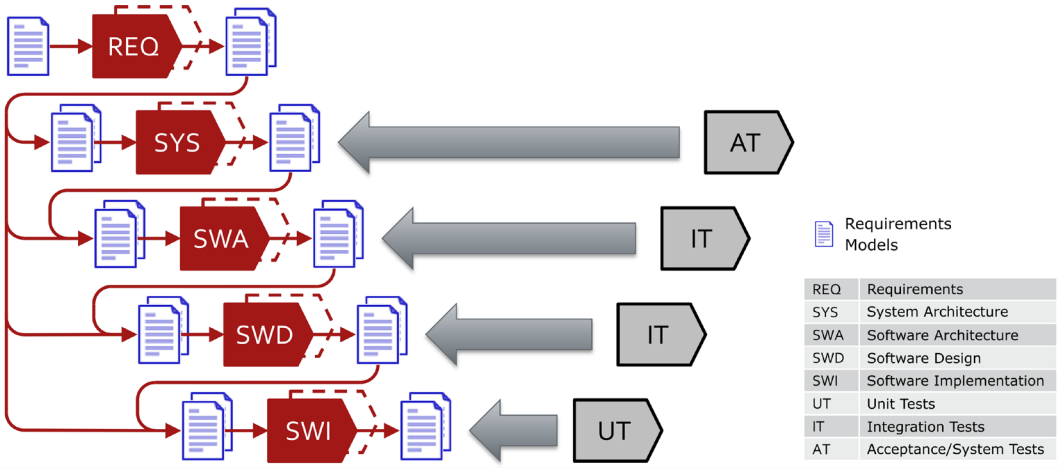


Figure 8.7: ARAMiS II Generic Software Development Process [**AramisGenericProcess**]

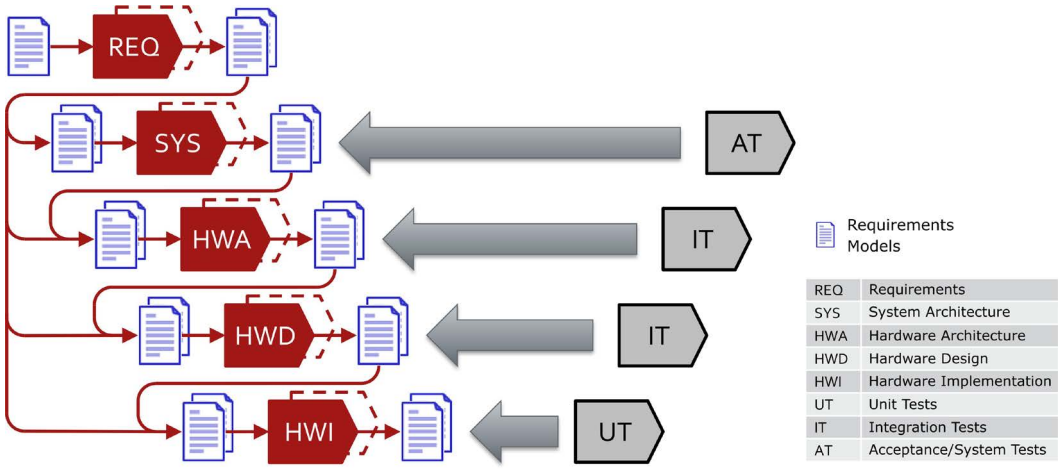


Figure 8.8: ARAMiS II Generic Hardware Development Process [**AramisGenericProcess**]

process. Just to name a few examples from ISO 26262, missing things are the planning of safety activities such as validation plans (clause 4-6), integration and testing plans and specifications (clause 4-8) as well as verification plans and specifications (clauses 6-6, 6-9, 6-10, 6-11).

8.6 Collaboration Traceability Workflow

This section describes how the Polarion platform [**Polarion**] allows to find an effective way to organize the collaborative process across different teams and to manage multiple projects along the development process stages depending on the project specifics [**PolarionAAI-WP**].

Web-based Collaboration The Polarion platform is the browser-based front-end. Thanks to the always-up-to-date online environment with live dashboards as well as access-controlled threaded commenting the tool is designed to eliminate emails and meetings in order to keep all

Figure 8.9: Key Features of the Polarion Platform

the information on specification, change requests, design decisions and etc. in one system. One centralized repository at the core of all activities serves the single source of truth. The centralized nature of information exchange enables development teams sitting in different locations to effectively convey ideas, much faster than the use of email, instant messaging and teleconference.

Team Collaboration Stakeholders can collaborate and communicate on various levels. For discussions and collaboration at higher levels, Polarion features a built-in wiki with default wiki spaces and documents for the repository and each project. This provides a highly flexible communication medium accessible to everyone with access rights to the repository, project or document.

More granular collaboration and communication takes place in comments on individual work items. Discussions on multiple threads can occur among project team members. Comment visibility can be optionally controlled and limited; for example, some comments may be visible only to managers.

Interchange between OEMs and Suppliers Domain experts who want to stay in their familiar environments can do so and still be tied into the centralized repository. The Polarion software's native integration with MATLAB®, for example, enables customers to include Simulink Model-Based Design workflows as an integral part of their application lifecycle. Bidirectional traceability facilitates navigation from Simulink model elements to associated Polarion work items and vice versa. Versioning aids collaborative design, opening up the assets for easy re-use and variant management across an entire automotive portfolio.

Another native integration that is popular among automotive customers is the round-trip

for Requirements Interchange Format (RIF/ReqIF) through which traceability across multiple documents or tools is maintained. The Object Management Group's (OMG's) standard for requirements exchange, a widely used Extensible Markup Language (XML) file format and workflow to support lossless exchange between partners, brings OEMs and their suppliers together around the globe.

Sharing and Reviewing Documents/Work Products Polarion also enables data modification, including approval of requirements via Word documents. Using the Polarion unique Round-trip for Word capability, documents containing managed artifacts can be exported to a Word document, which can then be shared with and reviewed by people who don't have access to Polarion. After changes (the type of which can be optionally restricted during export), the Word document can be re-imported to Polarion, where the changes it contains are incorporated into the online document, and the document history is updated.

Development Workflow Polarion allows to establish a workflow among diverse groups within and outside the organization working together. A customized solution can be established in the beginning, or a template for most common methodologies, as for example, V-Model, Agile Software Project, can be chosen and configured to map to specific business scenarios.

Traceability Comprehensive traceability allows developers to refer back to the software requirements that underlie their assigned tasks, and to reach out to the respective authors when they have questions. The same applies to the testers that verify whether the requirements have been met. All activities and decisions are automatically tracked, with collaboration history available to reveal how decisions were made every step along the way. Formal approval processes with compliant e-signatures complete the information exchange.

ISO 26262/IEC 61508 Qualification by TÜV NORD Siemens PLM Software is the first ALM vendor to receive ISO 26262/IEC 61508 qualification by TÜV NORD for the Polarion suite of products. The qualification at the highest Automotive Safety Integrity Level (ASIL-D) as defined in ISO 26262 is based on evidence that Polarion's software development processes can be reliably implemented and replicated. Due to the nature of the qualification, any software and hardware systems developed using Polarion's processes is also deemed to meet the functional safety requirements of ISO 26262, in turn radically reducing compliance efforts.

Future Goal within PANORAMA Project As it was mentioned, the Polarion platform allows to organize a workflow and to customize the traceability links according to a specific methodology by defining an extension[**PolarionExt**]. Such an extension would allow Polarion customers to arrange a collaboration process in a faster and in a more effective way. It is planned that an extension established within the PANORAMA Project covers a collaborative workflow for timing analysis compliant with PANORAMA model (support timing requirements and constraints using compatible information models).

A future PANORAMA extension would involve template specification, which is basically definition of semantics of the basic Polarion elements. As for example, work item plays a role of a basic element and can be related to anything you want to track in the project [**PolarionALG-WP**]. Regularly, a work item turns into requirements, activities, change request and test cases. The relationship between working items can be customized as well. Continuing this way, the Polarion

platform gives a lot of opportunities to customize a working flow. Additionally, it is planned to integrate with architecture modeling tools, such as Mentor Capital Systems and Mentor Capital Software Designer, where bidirectional traceability across requirements/tests/risks/etc. and modeling elements is already implemented.

Bibliography

[14] *Modelling and analysis of faults in sysml with application to systems of systems*, Tutorial held at IMBSA 2014, https://cse.cs.ovgu.de/imbsa2014/files/tutorial8/files/Fault%20Modelling%20and%20Analysis%20Tutorial.pdf, 2014.

[AADG12] S. Anssi, K. Albers, M. Doerfel, and S. Gerard, "Chronval/chronsim: A tool suite for timing verification of automotive applications," *ERTS*, 2012.

[Abd16] A. A. Abdulhameed, "Combining SysML and SystemC to Simulate and Verify Complex Systems," Theses, Université de Franche-Comté, 2016. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01562826.

[Abe] A. Abela, *Chart suggestions - a tought-starter*, www.ExtremePresentation.com.

[ABPK14] Z. Andrews, J. Bryans, R. Payne, and K. Kristensen, "Fault modelling in system-of-systems contracts," *arXiv preprint arXiv:1404.7775*, 2014.

[ADH+11] R. Adler, D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J.-P. Schwinn, and M. Trapp, "Integration of component fault trees into the uml," in *Models in Software Engineering*, ser. Lecture Notes in Computer Science, J. Dingel and A. Solberg, Eds., vol. 6627, Springer Berlin Heidelberg, 2011, pp. 312–327, isbn: 978-3-642-21209-3. doi: 10.1007/978-3-642-21210-9_30. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21210-9_30.

[AFPR13] Z. Andrews, J. Fitzgerald, R. Payne, and A. Romanovsky, "Fault modelling for systems of systems," in *2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS)*, Mar. 2013, pp. 1–8. doi: 10.1109/ISADS.2013.6513445.

[aiT] aiT, https://www.absint.com/ait/index.htm.

[AKB04] S. Abdelwahed, G. Karsai, and G. Biswas, "System diagnosis using hybrid failure propagation graphs," in *The 15th International Workshop on Principles of Diagnosis*, Citeseer, 2004.

[Alb11] K. Albers, "Approximative real-time analysis," PhD thesis, Universität Ulm, 2011. doi: 10.18725/OPARU-1771. [Online]. Available: https://oparu.uni-ulm.de/xmlui/handle/123456789/1798.

[ALRL04a] A. Avizienis, J. .-.-. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004, issn: 2160-9209. doi: 10.1109/TDSC.2004.2.

[ALRL04b] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004. doi: 10.1109/TDSC.2004.2. [Online]. Available: https://bit.ly/35itMXo.

[AM15] A. Aleti and I. Moser, "Fitness landscape characterisation for constrained software architecture optimisation problems," Dec. 2015, pp. 11–20. doi: 10.1109/ICECCS.2015.12.

[AMA14] AMALTHEA, *Deliverable 3.4: Prototypical implementation of selected concepts*, Apr. 2014.

[Ana] U. Analyser, https://github.com/upscale-sdk/UpScale/tree/master/analysis_flow/Analyzer.

[AntColonyOpt] *Ant colony optimization*. [Online]. Available: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms.

[App] App4MC consortium, *App4mc documentation*, https://www.eclipse.org/app4mc/help/app4mc-0.9.6/.

[AUT18] AUTOSAR initiative, *AUTOSAR Website - Standards*, 2018. [Online]. Available: https://www.autosar.org/standards/classic-platform.

[Bar03] S. Baruah, "Dynamic- and static-priority scheduling of recurring real-time tasks," *Real-Time Systems*, vol. 24, pp. 93–128, Jan. 2003. doi: 10.1023/A:1021711220939.

[Bau05] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005. doi: 10.1109/MDT.2005.69. [Online]. Available: https://bit.ly/35gnXJZ.

[BB00] P. Bishop and R. Bloomfield, "A methodology for safety case development," *Safety and Reliability*, vol. 20, no. 1, pp. 34–42, 2000. doi: 10.1080/09617353.2000.11690698.

[BBB+11] A. Baumgart, E. Böde, M. Büker, G. E. Werner Damm, T. Gezgin, S. Henkler, H. Hungar, B. Josko, M. Oertel, T. Peikenkamp, P. Reinkemeier, I. Stierand, and R. Weber, "Architecture modeling," OFFIS, Tech. Rep., Mar. 2011. [Online]. Available: http://ses.informatik.uni-oldenburg.de/download/bib/paper/OFFIS-TR2011_ArchitectureModeling.pdf.

[BBB10] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*. Springer, 2010.

[BBP+18] B. Bauer, J. S. Becker, T. Peikenkamp, C. Schlaak, and I. Stierand, "Entwurfsabsicherung für eingebettete mehrkernsysteme im kontext der iso 26262," in *SEERTS 2018: Workshop on Software Engineering for Applied Embedded RealTime Systems*, 2018.

[BCC+16] A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, and M. Sjödin, "A metamodel for the rubus component model: Extensions for timing and model transformation from east-adl," *Journal of IEEE Access*, vol. 5, no. 1, Dec. 2016.

[BCE+03] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, Jan. 2003, issn: 1558-2256. doi: 10.1109/JPROC.2002.805826.

[BDH+12] M. Broy, W. Damm, S. Henkler, K. Pohl, A. Vogelsang, and T. Weyer, "Introduction to the SPES Modeling Framework," in *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*, K. Pohl, H. Hönninger, R. Achatz, and M. Broy, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 31–49, isbn: 978-3-642-34614-9. doi: 10.1007/978-3-642-34614-9_3.

[BDK+16] W. Böhm, M. Daun, V. Koutsoumpas, A. Vogelsang, and T. Weyer, "SPES XT Modeling Framework," in *Advanced Model-Based Engineering of Embedded Systems: Extensions of the SPES 2020 Methodology*, K. Pohl, M. Broy, H. Daembkes, and H. Hönninger, Eds. Cham: Springer International Publishing, 2016, pp. 29–42, isbn: 978-3-319-48003-9. doi: 10.1007/978-3-319-48003-9_3.

[Ben+18] A. Benveniste *et al.*, "Contracts for system design," *Foundations and Trends in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[Bor05] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005. doi: 10.1109/MM.2005.110. [Online]. Available: https://bit.ly/2LJUCzK.

[BSK16] G. Biggs, T. Sakamoto, and T. Kotoku, "A profile and tool for modelling safety information with design information in sysml," *Software & Systems Modeling*, vol. 15, no. 1, pp. 147–178, Feb. 2016, issn: 1619-1374. doi: 10.1007/s10270-014-0400-x. [Online]. Available: https://doi.org/10.1007/s10270-014-0400-x.

[Can03] M. Cantor, "Rational unified process for systems engineering: Part 1," *Journal of Systems Architecture - JSA*, 2003.

[CFJ+10] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M.-O. Reiser, A. Sandberg, D. Servat, R. Tavakoli Kolagari, M. Törngren, and M. Weber, "The EAST-ADL Architecture Description Language for Automotive Embedded Software," in *Model-Based Engineering of Embedded Real-Time Systems: International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers*, H. Giese, G. Karsai, E. Lee, B. Rumpe, and B. Schätz, Eds. Springer Berlin Heidelberg, 2010, pp. 297–307, isbn: 978-3-642-16277-0. doi: 10.1007/978-3-642-16277-0_11.

[chronSIM] *Chronsim*. [Online]. Available: https://www.inchron.com/tool-suite/chronsim/.

[chronVAL] *Chronval*. [Online]. Available: https://www.inchron.com/tool-suite/chronval/.

[CML+12] J. W. Coleman, A. K. Malmos, P. G. Larsen, J. Peleska, R. Hains, Z. Andrews, R. Payne, S. Foster, A. Miyazawa, C. Bertolini, and A. Didierk, "COMPASS tool vision for a system of systems collaborative development environment," in *2012 7th International Conference on System of Systems Engineering (SoSE)*, Jul. 2012, pp. 451–456. doi: 10.1109/SYSoSE.2012.6384150.

[CMW+13] D. Chen, N. Mahmud, M. Walker, L. Feng, H. Lönn, and Y. Papadopoulos, "Systems modeling with east-adl for fault tree analysis through hip-hops*," *IFAC Proceedings Volumes*, vol. 46, no. 22, pp. 91–96, 2013, 4th IFAC Workshop on Dependable Control of Discrete Systems, issn: 1474-6670. doi: https://doi.org/10.3182/20130904-3-UK-4041.00043. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667015339938.

[Com17] S. E. C. S. Committee, *Architecture analysis & design language (aadl) as5506c*, https://www.sae.org/standards/content/as5506c, 2017.

[CRY16] CRYSTAL Project Partners, *CRYSTAL - CRitical sYSTem engineering AcceLeration*, Available from: http://www.crystal-artemis.eu/, 2013 - 2016.

[DB11] R. I. Davis and A. Burns, "A Survey of Hard Real-time Scheduling for Multiprocessor Systems," *ACM Comput. Surv.*, vol. 43, no. 4, 35:1–35:44, Oct. 2011, issn: 0360-0300. doi: 10.1145/1978802.1978814. [Online]. Available: http://doi.acm.org/10.1145/1978802.1978814 (visited on 12/15/2018).

[DD08] J. Dehlinger and J. B. Dugan, "Analyzing dynamic fault trees derived from model-based system architectures," *Nuclear Engineering and Technology*, vol. 40, no. 5, pp. 365–374, 2008.

[DEG+19] W. Damm, G. Ehmen, K. Grüttner, P. Ittershagen, B. Koopmann, F. Poppen, and I. Stierand, "Multi-layer time coherency in the development of adas/ad systems: Design approach and tooling," in *Proceedings of the Workshop on Design Automation for CPS and IoT*, ACM, 2019, pp. 20–30.

[DHJ+11] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand, "Using contract-based component specifications for virtual integration testing and architecture design," in *Design, Automation and Test in Europe, DATE 2011, Grenoble, France, March 14-18, 2011*, 2011, pp. 1023–1028. doi: 10.1109/DATE.2011.5763167. [Online]. Available: http://dx.doi.org/10.1109/DATE.2011.5763167.

[Dia] DiagXtrm, https://forge.onera.fr/projects/diagxtrm.

[DWP14] M. Daun, T. Weyer, and K. Pohl, "Validating the functional design of embedded systems against stakeholder intentions," in *Model-Driven Engineering and Software Development (MODELSWARD), 2014 2 International Conference on*, 2014, pp. 333–339.

[Ecl20] Eclipse Foundation. (Jan. 31, 2020). Trace compass, [Online]. Available: https://www.eclipse.org/tracecompass/.

[EE07] A. Ermedahl and J. Engblom, "Handbook of real-time and embedded systems," in, I. Lee, J. Leung, and S. Son, Eds. Chapman & Hall, 2007, ch. Execution Time Analysis for Embedded Real-Time Systems.

[Eur99] European Committee for Electrotechnical Standardization (CENELEC), *CENELEC EN 50126: Railway Applications – The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*, 1999.

[FGH06] P. Feiler, D. Gluch, and J. Hudak, "The Architecture Analysis & Design Language (AADL): An Introduction," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, US, Tech. Rep. CMU/SEI-2006-TN-011, 2006. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7879.

[FM93] P. Fenelon and J. A. McDermid, "An integrated tool set for software safety analysis," *Journal of Systems and Software*, vol. 21, no. 3, pp. 279–290, 1993.

[FMNP94] P. Fenelon, J. A. McDermid, M. Nicolson, and D. J. Pumfrey, "Towards integrated safety analysis and design," *ACM SIGAPP Applied Computing Review*, vol. 2, no. 1, pp. 21–32, 1994.

[FMS14] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language, Third edition*, 3rd ed., ser. MK/OMG Press. Morgan Kaufmann, 2014, isbn: 978-0-12-800202-5.

[Fon19] J. Fonseca, "Multiprocessor scheduling and mapping techniques for real-time parallel applications," PhD thesis, University of Porto, 2019.

[FR07] P. H. Feiler and A. Rugina, "Dependability Modeling with the Architecture Analysis & Design Language (AADL)," Jul. 2007. doi: 10.1184/R1/6572996.v1. [Online]. Available: https://kilthub.cmu.edu/articles/Dependability_Modeling_with_the_Architecture_Analysis_Design_Language_AADL_/6572996.

[Fri] S. Friedenthal, *SysML v2 Submission Team (SST) SysML v2 Update*, INCOSE. [Online]. Available: http://www.omgwiki.org/.

[Frt16] J. Frtunikj, "Safety framework and platform for functions of future automotive E/E systems," *Automotive and Engine Technology*, vol. 1, no. 1-4, pp. 93–105, 2016. doi: 10.1007/s41104-016-0007-z. [Online]. Available: https://bit.ly/2tcReaf.

[GCW07] L. Grunske, R. Colvin, and K. Winter, "Probabilistic model-checking support for fmea," in *Fourth International Conference on the Quantitative Evaluation of Systems (QEST 2007)*, IEEE, 2007, pp. 119–128.

[GGPD01] M. Gonzalez Harbour, J. J. Gutierrez Garcia, J. C. Palencia Gutierrez, and J. M. Drake Moyano, "Mast: Modeling and analysis suite for real time applications," in *Proceedings 13th Euromicro Conference on Real-Time Systems*, Jun. 2001, pp. 125–134. doi: 10.1109/EMRTS.2001.934015.

[GJ09] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 27. print, ser. A Series of Books in the Mathematical Sciences. New York [u.a]: Freeman, 2009, 338 pp., OCLC: 551912424, isbn: 978-0-7167-1044-8 978-0-7167-1045-5.

[GKP05] L. Grunske, B. Kaiser, and Y. Papadopoulos, "Model-driven safety evaluation with state-event-based component failure annotations," in *International Symposium on Component-Based Software Engineering*, Springer, 2005, pp. 33–48.

[GL08] J. S. Gansler and W. Lucyshyn, "Commercial-off-the-shelf (cots): Doing it right," MARYLAND UNIV COLLEGE PARK CENTER FOR PUBLIC POLICY and PRIVATE ENTERPRISE, Tech. Rep., 2008. [Online]. Available: https://bit.ly/2PC0tIt.

[Gru07] L. Grunske, "Early quality prediction of component-based systems–a generic framework," *Journal of Systems and Software*, vol. 80, no. 5, pp. 678–686, 2007.

[GZOH18] T. Gonschorek, M. Zeller, F. Ortmeier, and K. Höfig, "Fault trees vs. component fault trees: An empirical study," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, B. Gallina, A. Skavhaug, E. Schoitsch, and F. Bitsch, Eds., Springer International Publishing, 2018, pp. 239–251.

[HFK+07] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, and J. Teich, "A systemc-based design methodology for digital signal processing systems," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, p. 047 580, 2007.

[HillClimbing] *Hill climbing*. [Online]. Available: https://en.wikipedia.org/wiki/Hill_climbing.

[HJZ+18] K. Höfig, A. Joanni, M. Zeller, F. Montrone, M. Rothfelder, R. Amarnath, P. Munk, and A. Nordmann, "Model-based reliability and safety: Reducing the complexity of safety analyses using component fault trees," in *2018 Annual Reliability and Maintainability Symposium (RAMS)*, 2018.

[HMN+08] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K. Lundback, "The rubus component model for resource constrained real-time systems," in *2008 International Symposium on Industrial Embedded Systems (SIES)*, Jun. 2008, pp. 177–183.

[HTI97] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *IEEE Computer*, vol. 30, no. 4, pp. 75–82, 1997. doi: 10.1109/2.585157. [Online]. Available: https://bit.ly/2qLwB49.

[IAPP14] C. Ingram, Z. Andrews, R. Payne, and N. Plat, "SysML fault modelling in a traffic management system of systems," in *2014 9th International Conference on System of Systems Engineering (SOSE)*, Jun. 2014, pp. 124–129. doi: 10.1109/SYSOSE.2014.6892475.

[IEE11] IEEE, "ISO/IEC/IEEE Systems and software engineering – Architecture description," *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pp. 1–46, Dec. 2011. doi: 10.1109/IEEESTD.2011.6129467.

[INC14] INCOSE, *System Engineering Vision 2025*, International Council on Systems Engineering, 2014. [Online]. Available: https://www.incose.org/docs/default-source/aboutse/se-vision-2025.pdf.

[INCHRON] *Inchron*. [Online]. Available: https://www.inchron.com/tool-suite.

[Int06] International Electrotechnical Commission (IEC), *IEC 61025: Fault Tree Analysis (FTA)*, 2006.

[Int11] International Organization for Standardization (ISO), *ISO 26262: Road vehicles – Functional safety*, 2011.

[Int90] International Electrotechnical Commission (IEC), *IEC 61025: Fault Tree Analysis (FTA)*, 1990.

[Int91] ——, *IEC 60812: Analysis Techniques for System Reliability - Procedure for Failure Mode and Effects Analysis (FMEA)*, 1991.

[Int98] ——, *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety related systems*, 1998.

[IY98] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the International Symposium on Low Power Design*, 1998. doi: 10.1145/280756.280894.

[JH07] A. Joshi and M. P. E. Heimdahl, "Behavioral fault modeling for model-based safety analysis," in *10th IEEE High Assurance Systems Engineering Symposium (HASE'07)*, Nov. 2007, pp. 199–208. doi: 10.1109/HASE.2007.58.

[JHMW06] A. Joshi, M. Heimdahl, S. Miller, and M. Whalen, "Model-based safety analysis," NASA Langley Research Center, Tech. Rep., 2006.

[JJH+13] J. Jung, A. Jedlitschka, K. Höfig, D. Domis, and M. Hiller, "A controlled experiment on component fault trees," in *Computer Safety, Reliability, and Security*, F. Bitsch, J. Guiochet, and M. Kaâniche, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 285–292, isbn: 978-3-642-40793-2.

[JP86] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, Jan. 1986, issn: 0010-4620. doi: 10.1093/comjnl/29.5.390. eprint: http://oup.prod.sis.lan/comjnl/article-pdf/29/5/390/1314410/290390.pdf. [Online]. Available: https://doi.org/10.1093/comjnl/29.5.390.

[JVB07] A. Joshi, S. Vestal, and P. Binns, "Automatic generation of static fault trees from AADL models," in *DSN Workshop on Architecting Dependable Systems DSN07-WAD*, 2007.

[KF15] D. Kuvaiskii and C. Fetzer, " -encoding: practical encoded processing," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE, 2015, pp. 13–24. doi: 10.1109/DSN.2015.20. [Online]. Available: https://bit.ly/2RLetSR.

[KG07] D. Krus and K. Grantham Lough, "Applying function-based failure propagation in conceptual design," in *ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers Digital Collection, 2007, pp. 407–420.

[KGF07] B. Kaiser, C. Gramlich, and M. Förster, "State/event fault trees—a safety analysis model for software-controlled systems," *Reliability Engineering & System Safety*, vol. 92, no. 11, pp. 1521–1537, 2007.

[KK10] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Elsevier, 2010. [Online]. Available: https://bit.ly/35hnqqX.

[KLM03] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software - Volume 33*, ser. SCS '03, 2003, pp. 37–46.

[KSA+18] B. Kaiser, D. Schneider, R. Adler, D. Domis, F. Möhrle, A. Berres, M. Zeller, K. Höfig, and M. Rothfelder, "Advances in component fault trees," in *Safety and Reliability – Safe Societies in a Changing World, Proceedings of ESREL 2018*, 2018, pp. 815–823.

[KW04] T. Kelly and R. Weaver, "The goal structuring notation–a safety argument notation," in *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, Citeseer, 2004, p. 6.

[KWF15] L. Krawczyk, C. Wolff, and D. Fruhner, "Automated Distribution of Software to Multi-core Hardware in Model Based Embedded Systems Development," in *Communications in Computer and Information Science*, vol. 538, 2015, pp. 320–329, isbn: 9783319247694. doi: 10.1007/978-3-319-24770-0_28. [Online]. Available: http://link.springer.com/10.1007/978-3-319-24770-0%7B%5C_%7D28.

[KZS18] S. Klabes, M. Zeller, and J.-P. Schwinn, "Safety assessment of ETCS using component fault trees:a case study," in *INCOSE EMEA Sector Systems Engineering Conference (EMEASEC) / TdSE*, 2018.

[Leh90] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *[1990] Proceedings 11th Real-Time Systems Symposium*, Dec. 1990, pp. 201–209. doi: 10.1109/REAL.1990.128748.

[Lev95] N. G. Levenson, *System safety and computers*. Addison Wesley Boston, 1995.

[LGY12] F. Liu, H. Gu, and Y. Yang, "Dtbr: A dynamic thermal-balance routing algorithm for network-on-chip," *Comput. Electr. Eng.*, vol. 38, no. 2, pp. 270–281, 2012, issn: 0045-7906. doi: 10.1016/j.compeleceng.2011.12.006. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2011.12.006.

[Liu00] J. W. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ: Prentice Hall, 2000, 610 pp., isbn: 978-0-13-099651-0.

[LL14] S. Li and X. Li, "Study on generation of fault trees from altarica models," *Procedia Engineering*, vol. 80, Dec. 2014. doi: 10.1016/j.proeng.2014.09.070.

[LL73] C. Liu and J. Layland, *Scheduling algorithms for multiprogramming in a hard-real-time environment*, 1973.

[LLL13] H.-C. Liu, L. Liu, and N. Liu, "Risk evaluation approaches in failure mode and effects analysis: A literature review," *Expert systems with applications*, vol. 40, no. 2, pp. 828–838, 2013.

[LM87] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 24–35, Jan. 1987, issn: 0018-9340. doi: 10.1109/TC.1987.5009446.

[LRK+09] M.-L. Li, P. Ramachandran, U. R. Karpuzcu, S. K. S. Hari, and S. V. Adve, "Accurate microarchitecture-level fault modeling for studying hardware faults," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, IEEE, 2009, pp. 105–116. doi: 10.1109/HPCA.2009.4798242. [Online]. Available: https://bit.ly/2E9Ieox.

[LSGE11] J. Lin, A. Srivatsa, A. Gerstlauer, and B. L. Evans, "Heterogeneous multiprocessor mapping for real-time streaming systems," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 1605–1608. doi: 10.1109/ICASSP.2011.5946804.

[LSST91] J. P. Lehoczky, L. Sha, J. K. Strosnider, and H. Tokuda, "Fixed priority scheduling theory for hard real-time systems," in *Foundations of Real-Time Computing: Scheduling and Resource Management*, A. M. van Tilborg and G. M. Koob, Eds. Boston, MA: Springer US, 1991, pp. 1–30, isbn: 978-1-4615-3956-8. doi: 10.1007/978-1-4615-3956-8_1. [Online]. Available: https://doi.org/10.1007/978-1-4615-3956-8_1.

[LT01] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Heidelberg: Springer-Verlag, 2001, isbn: 354042184X.

[Lut00] R. R. Lutz, "Software engineering for safety: A roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, ACM, 2000, pp. 213–226.

[Mah12] N. Mahmud, "Dynamic model-based safety analysis: From state machines to temporal fault trees by," PhD thesis, The University of Hull, 2012.

[Mar15] F. Martin, "Transformation of hardware traces to system traces for embedded multi-core real-time systems," Master Thesis, Ostbayerische Technische Hochschule Regensburg, 2015.

[Matlab] *Matlab*. [Online]. Available: https://de.mathworks.com/products/matlab.html.

[Maz09] R. Mazza, *Introduction to Information Visualization*, 1st ed. Springer Publishing Company, Incorporated, 2009, isbn: 1848002181, 9781848002180.

[MBPB12] Z. Mian, L. Bottaci, Y. Papadopoulos, and M. Biehl, "System dependability modelling and analysis using aadl and hip-hops," *IFAC Proceedings Volumes*, vol. 45, no. 6, pp. 1647–1652, 2012, 14th IFAC Symposium on Information Control Problems in Manufacturing, issn: 1474-6670. doi: https://doi.org/10.3182/20120523-3-RO-2023.00334. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667016333870.

[MBZ+17] F. Möhrle, K. Bizik, M. Zeller, K. Höfig, M. Rothfelder, and P. Liggesmeyer, "A formal approach for automating compositional safety analysis using flow type annotations in component fault trees," in *Proceedings of the 27th European Safety and Reliability Conference (ESREL): Safety and Reliability – Theory and Applications*, 2017.

[Mic10] Z. Michalewicz, *How to Solve It: Modern Heuristics 2e*. Berlin, Heidelberg: Springer-Verlag, 2010, isbn: 3642061346.

[Mic96] ——, *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)* Berlin, Heidelberg: Springer-Verlag, 1996, isbn: 3540606769.

[MNS+17] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, "Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints," *International Journal on Software and Systems Modeling*, pp. 1–31, Jan. 2017.

[MNZ18] J. Menu, M. Nicolai, and M. Zeller, "Designing fail-safe architectures for aircraft electrical power systems," in *2018 AIAA/IEEE Electric Aircraft Technologies Symposium, AIAA Propulsion and Energy Forum (AIAA 2018-5032)*, 2018. doi: https://doi.org/10.2514/6.2018-5032.

[MZH+16] F. Möhrle, M. Zeller, K. Höfig, M. Rothfelder, and P. Liggesmeyer, "Automating compositional safety analysis using a failure type taxonomy for component fault trees," in *Risk, Reliability and Safety: Innovating Theory and Practice: Proceedings of ESREL 2016*, L. Walls, M. Revie, and T. Bedford, Eds., 2016, pp. 1380–1387.

[NLQ16] V. C. Ngo, A. Legay, and J. Quilbeuf, "Statistical model checking for systemc models," in *2016 IEEE 17 International Symposium on High Assurance Systems Engineering (HASE)*, Jan. 2016, pp. 197–204. doi: 10.1109/HASE.2016.24.

[NSTW04] K. D. Nguyen, Z. Sun, P. S. Thiagarajan, and W.-F. Wong, "Model-driven soc design via executable uml to systemc," in *25 IEEE International Real-Time Systems Symposium*, Dec. 2004, pp. 459–468. doi: 10.1109/REAL.2004.32.

[NYP+14] V. Nelis, P. M. Yomsi, L. M. Pinho, J. Fonseca, M. Bertogna, E. Quinones, R. Vargas, and A. Marongiu, "The challenge of time-predictability in modern many-core architectures," in *14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, 2014.

[NYP15] V. Nelis, P. M. Yomsi, and L. M. Pinho, "Methodologies for the WCET Analysis of Parallel Applications on Many-core Architectures," in *The Euromicro Conference on Digital System Design (DSD 2015)*, 2015.

[NYP17] ——, "The P-SOCRATES timing analysis methodology for parallel real-time applications deployed on many-core platforms," in *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*, 2017.

[Obj19] Object Management Group (OMG), *Structured Assurance Case Metamodel (SACM), Version 2.1*, https://www.omg.org/spec/SACM, 2019.

[OLSM18] L. Osinski, T. Langer, M. Schmid, and J. Mottok, "PyFI-fault injection platform for real hardware," in *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*, VDE, 2018, pp. 1–7. [Online]. Available: https://bit.ly/35hQgI2.

[OM19] L. Osinski and J. Mottok, "S^{3}DES-Scalable Software Support for Dependable Embedded Systems," in *International Conference on Architecture of Computing Systems*, Springer, 2019, pp. 15–27. doi: 10.1007/978-3-030-18656-2_2. [Online]. Available: https://bit.ly/2tcZgjt.

[OMG] OMG, *XML Metadata Interchange (XMI) Specification*, Object Management Group. [Online]. Available: http://www.omg.org/spec/XMI/.

[OMG17] ——, *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5.1*, Object Management Group, 2017. [Online]. Available: http://www.omg.org/spec/UML/2.5.1.

[OMG19] ——, *Systems Modeling Language (SysML), Version 1.6*, Object Management Group, 2019. [Online]. Available: https://www.omg.org/spec/SysML/1.6/.

[OTA] OTAWA, http://www.otawa.fr/.

[PAN20] PANORAMA Consortium, "Panorama d2.1 - state-of-the-art analysis on editors, viewers and transformationtools used in the automotive and avionic industry," ITEA3 Project PANORAMA, Deliverable, Jan. 31, 2020.

[Pet00] S. Petters, " Bounding the execution time of real-time tasks on modern processors," in *7th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'00)*, 2000.

[PGG97] J. C. Palencia Gutierrez, J. J. Gutierrez Garcia, and M. Gonzalez Harbour, "On the schedulability analysis for distributed hard real-time systems," in *Proceedings Ninth Euromicro Workshop on Real Time Systems*, Jun. 1997, pp. 136–143. doi: 10.1109/EMWRTS.1997.613774.

[Pim17] A. D. Pimentel, "Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration," *IEEE Design Test*, vol. 34, no. 1, pp. 77–90, Feb. 2017, issn: 2168-2356. doi: 10.1109/MDAT.2016.2626445.

[PM01] Y. Papadopoulos and M. Maruhn, "Model-based synthesis of fault trees from matlab-simulink models," in *2001 International Conference on Dependable Systems and Networks*, IEEE, 2001, pp. 77–82.

[PM99] Y. Papadopoulos and J. A. McDermid, "Hierarchically performed hazard origin and propagation studies," in *Computer Safety, Reliability and Security*, M. Felici and K. Kanoun, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 139–152, isbn: 978-3-540-48249-9.

[PMPV10] P. Peñil, J. Medina, H. Posadas, and E. Villar, "Generating heterogeneous executable specifications in systemc from uml/marte models," *Innovations in Systems and Software Engineering*, vol. 6, no. 1, pp. 65–71, Mar. 1, 2010, issn: 1614-5054. doi: 10.1007/s11334-009-0105-4. [Online]. Available: https://doi.org/10.1007/s11334-009-0105-4.

[PMSH01] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner, "Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure," *Reliability Engineering & System Safety*, vol. 71, no. 3, pp. 229–247, 2001.

[PPG04] Y. Papadopoulos, D. Parker, and C. Grante, "Automating the failure modes and effects analysis of safety critical systems," in *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.*, IEEE, 2004, pp. 310–311.

[QHXL09] B. Qu, Y. Huang, X. Xie, and Y. Lu, "A Developed Dynamic Environment Fault Injection Tool for Component Security Testing," in *2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement*, IEEE, 2009, pp. 417–422. doi: 10.1109/SSIRI.2009.9. [Online]. Available: https://bit.ly/35aRWTv.

[Rad11] Radio Technical Commission for Aeronautics (RTCA) and European Organisation for Civil Aviation Equipment (EUROCAE), 2011.

[Rap] RapiTime, https://www.rapitasystems.com/products/rapitime.

[RKK07] A. Rugina, K. Kanoun, and M. Kaâniche, "An architecture-based dependability modeling framework using AADL," *CoRR*, vol. abs/0704.0865, 2007. arXiv: 0704.0865. [Online]. Available: http://arxiv.org/abs/0704.0865.

[RKK08] A. Rugina, K. Kanoun, and M. Kaâniche, "The adapt tool: From aadl architectural models to stochastic petri nets through model transformation," in *2008 Seventh European Dependable Computing Conference*, May 2008, pp. 85–90. doi: 10.1109/EDCC-7.2008.14.

[RKUS17] K. Rosvall, N. Khalilzad, G. Ungureanu, and I. Sander, "Throughput Propagation in Constraint-Based Design Space Exploration for Mixed-Criticality Systems," in *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. RAPIDO '17, New York, NY, USA: ACM, 2017, 4:1–4:8, isbn: 978-1-4503-4840-9. doi: 10.1145/3023973.3023977. [Online]. Available: http://doi.acm.org/10.1145/3023973.3023977 (visited on 11/21/2018).

[RMU+18] K. Rosvall, T. Mohammadat, G. Ungureanu, J. Öberg, and I. Sander, "Exploring Power and Throughput for Dataflow Applications on Predictable NoC Multiprocessors," Aug. 1, 2018, pp. 719–726. doi: 10.1109/DSD.2018.00011.

[RS14] K. Rosvall and I. Sander, "A constraint-based design space exploration framework for real-time applications on MPSoCs," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2014, pp. 1–6. doi: 10.7873/DATE.2014.339.

[RS17] K. Rosvall and I. Sander, "Flexible and Tradeoff-Aware Constraint-Based Design Space Exploration for Streaming Applications on Heterogeneous Platforms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 2, 21:1–21:26, Nov. 2017, issn: 1084-4309. doi: 10.1145/3133210. [Online]. Available: http://doi.acm.org/10.1145/3133210 (visited on 11/12/2018).

[RSRH11] P. Reinkemeier, I. Stierand, P. Rehkop, and S. Henkler, "A pattern-based requirement specification language: Mapping automotive specific timing requirements.," in *Software Engineering (Workshops)*, vol. 184, 2011, pp. 99–108.

[RTCToolbox] *Real-time calculus (rtc) toolbox*. [Online]. Available: http://www.mpa.ethz.ch/Rtctoolbox.

[RvBW06] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. Elsevier, Aug. 18, 2006, 977 pp., isbn: 978-0-08-046380-3. Google Books: Kjap9ZWcKOoC.

[RW13] A. Rajan and T. Wahl, *CESAR - Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*. Springer Wien Heidelberg New York Dordrecht London, 2013, isbn: 978-3-7091-1386-8. doi: 10.1007/978- 3-7091-1387-5.

[RZH+17] S. Reiter, M. Zeller, K. Höfig, A. Viehl, O. Bringmann, and W. Rosenstiel, "Verification of component fault trees using error effect simulations," in *Model-Based Safety and Assessment - 5th International Symposium, IMBSA 2017*, 2017, pp. 212–226.

[SB17] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization, Second Edition*. 2017, OCLC: 1004359691, isbn: 978-1-4200-4802-5. [Online]. Available: http://www.crcnetbase.com/isbn/9781315219752 (visited on 12/04/2018).

[SBGC07] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal, "Multiprocessor Resource Allocation for Throughput-constrained Synchronous Dataflow Graphs," in *Proceedings of the 44th Annual Design Automation Conference*, ser. DAC '07, New York, NY, USA: ACM, 2007, pp. 777–782, isbn: 978-1-59593-627-1. doi: 10.1145/1278480.1278674. [Online]. Available: http://doi.acm.org/10.1145/1278480.1278674 (visited on 11/23/2018).

[SDP12] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming dr. frankenstein: Contract-based design for cyber-physical systems," *European journal of control*, vol. 18, no. 3, pp. 217–238, 2012.

[SFB+00] D. T. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R. K. Iyer, "NFTAPE: a framework for assessing dependability in distributed systems with lightweight fault injectors," in *Proceedings IEEE International Computer Performance and Dependability Symposium. IPDS 2000*, IEEE, 2000, pp. 91–100. doi: 10.1109/IPDS.2000.839467. [Online]. Available: https://bit.ly/34gcfh9.

[SJ04] I. Sander and A. Jantsch, "System Modeling and Transformational Design Refinement in ForSyDe," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 1, pp. 17–32, Jan. 2004, issn: 0278-0070. doi: 10.1109/TCAD.2003.819898. [Online]. Available: http://ieeexplore.ieee.org/document/1256053/ (visited on 09/11/2019).

[SM00] H. Schumann and W. Müller, *Visualisierung: Grundlagen und allgemeine methoden*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, isbn: 978-3-642-57193-0. doi: 10.1007/978- 3- 642- 57193- 0_3. [Online]. Available: https://doi.org/10.1007/978-3-642-57193-0_3.

[SMDJ14] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart, "On the Sustainability of the Extreme Value Theory for WCET Estimation," in *14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, 2014.

[SPW09] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: a large-scale field study," in *ACM SIGMETRICS Performance Evaluation Review*, ACM, vol. 37, 2009, pp. 193–204. [Online]. Available: https://bit.ly/2qPYuIo.

[SRGB13] I. Stierand, P. Reinkemeier, T. Gezgin, and P. Bhaduri, "Real-time scheduling interfaces and contracts for the design of distributed embedded systems," in *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, IEEE, 2013, pp. 130–139.

[Sta03] I. Standard, "Iec 60812," *Analysis Techniques for System Reliability-Procedure for Failure Mode and Effects Analysis (FMEA)*, 2003.

[SWK+05] G. P. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An experimental study of soft errors in microprocessors," *IEEE micro*, vol. 25, no. 6, pp. 30–39, 2005. doi: 10.1109/MM.2005.104. [Online]. Available: https://bit.ly/2tbDBrT.

[SY15] M. Stigge and W. Yi, "Graph-based models for real-time workload: A survey," *Real-Time Systems*, vol. 51, no. 5, pp. 602–636, Sep. 2015, issn: 0922-6443, 1573-1383. doi: 10.1007/s11241-015-9234-z. [Online]. Available: http://link.springer.com/10.1007/s11241-015-9234-z (visited on 06/20/2019).

[SystemC12] "IEEE Standard for Standard SystemC Language Reference Manual - Redline," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005) - Redline*, pp. 1–1163, 2012.

[Tabu] *Tabu*. [Online]. Available: https://en.wikipedia.org/wiki/Tabu_search.

[TC94] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 40, pp. 117–134, Apr. 1994. doi: 10.1016/0165-6074(94)90080-9.

[TCN00] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, May 2000, 101–104 vol.4. doi: 10.1109/ISCAS.2000.858698.

[The19a] The ARAMiS II Consortium, *ARAMiS II deliverable E2.3: Platform architecture characterization and mitigation / technical architecture models*, Not publically available, 2019.

[The19b] ——, *ARAMiS II deliverable E3.3: Partitioning of software components*, Not publically available, 2019.

[Tim14] Timing Architects Embedded Systems GmbH, *BTF-Specification*, https://wiki.eclipse.org/images/e/e6/TA_BTF_Specification_2.1.3_Eclipse_Auto_IWG.pdf, Norm, 2014.

[Vec] Vector, *Canape*, https://assets.vector.com/cms/content/products/canape/Docs/CANape_ProductInformation_EN.pdf.

[Vec20] Vector Informatik GmbH, *Vector TA Tool Suite*, Jan. 2020. [Online]. Available: https://www.vector.com/int/en/products/products-a-z/software/ta-tool-suite.

[VGRH81] W. Vesely, F. Goldberg, N. Roberts, and D. Haasl, *Fault Tree Handbook*. US Nuclear Regulatory Commission, 1981.

[Voi17] J.-L. Voirin, *Model-based system and architecture engineering with the arcadia method*. 2017, pp. 1–368, isbn: 978-1-78548-169-7.

[Wal05] M. Wallace, "Modular architectural representation and analysis of fault propagation and transformation," *Electronic Notes in Theoretical Computer Science*, vol. 141, no. 3, pp. 53–71, 2005.

[Wal09] M. D. Walker, "Pandora: A logic for the qualitative analysis of temporal fault trees," PhD thesis, The University of Hull, 2009.

[Wan06] E. Wandeler, "Modular performance analysis and interface based design for embedded real time systems," 2006.

[WEE+08] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, 2008.

[Wei16] T. Weilkiens, *SYSMOD - The Systems Modeling Toolbox - Pragmatic MBSE with SysML, 2nd edition*. 2016, isbn: 978-3-9817875-8-0.

[WJZ+15] J. Wang, C.-H. Jeong, N. Zimmerman, R. Healy, D. Wang, F. Ke, and G. Evans, "Plume-based analysis of vehicle fleet air pollutant emissions and the contribution from high emitters," *Atmospheric Measurement Techniques*, vol. 8, pp. 3263–3275, Aug. 2015. doi: 10.5194/amt-8-3263-2015.

[WKD+19] R. Wei, T. P. Kelly, X. Dai, S. Zhao, and R. Hawkins, "Model based system assurance using the structured assurance case metamodel," *Journal of Systems and Software*, vol. 154, pp. 211–233, 2019, issn: 0164-1212. doi: https://doi.org/10.1016/j.jss.2019.05.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121219301062.

[WRHS12] R. Weber, P. Reinkemeier, S. Henkler, and I. Stierand, "Technical Viewpoint," in *Model-Based Engineering of Embedded Systems*, K. Pohl, H. Hönninger, R. Achatz, and M. Broy, Eds., Springer Berlin Heidelberg, 2012, pp. 95–106, isbn: 978-3-642-34613-2. doi: 10.1007/978-3-642-34614-9_7.

[YXC02] Yumin Zhang, Xiaobo Hu, and D. Chen, "Task scheduling and voltage selection for energy minimization," in *Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324)*, IEEE, 2002, pp. 183–188, isbn: 1-58113-461-4. doi: 10.1109/DAC.2002.1012617. [Online]. Available: http://ieeexplore.ieee.org/document/1012617/.

[ZH15] M. Zeller and K. Höfig, "Confetti – component fault tree-based testing," in *Safety and Reliability of Complex Engineered Systems: Proceedings of the 25th European Safety and Reliability Conference (ESREL)*, L. Podofillini, B. Sudret, B. Stojadinovic, E. Zio, and W. Kröger, Eds., 2015, pp. 4011–4017.

[ZK19] M. Zeller and S. Klabes, "Iterative and incremental development of reliable systems," in *Proceedings of the 29th European Safety and Reliability Conference (ESREL)*, 2019.

[ZM18] M. Zeller and F. Montrone, “Combination of component fault trees and markov chains to analyze complex, software-controlled systems,” in *2018 3rd International Conference on System Reliability and Safety (ICSRS)*, 2018, pp. 13–20. doi: 10.1109/ICSRS.2018.8688854.